Федеральное агентство по образованию

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ

Кафедра электронных приборов (ЭП)

Е.С. Шандаров

ИНФОРМАЦИОННЫЕ СИСТЕМЫ НА БАЗЕ ТЕХНОЛОГИЙ ИНТЕРНЕТ

Учебное пособие по ГПО

2006

СОДЕРЖАНИЕ

1. Понятие информационных систем. Базовые принципы построения информационных систем на основе технологий Интернет

Современные информационные системы - это комплексные решения , интегрирующие телекоммуникационные и информационные технологии, различные среды передачи информации и ее носители. Информационные системы окружают нас повсюду, они помогают бухгалтерам в начислении зарплаты и расчете налогов, продают авиабилеты и позволяют вносить платежи за мобильную связь, обеспечивают бесперебойную и безаварийную работу технологических установок и даже пытаются управлять целыми производственными комплексами.

При проектировании и создании современных информационных систем, в дальнейшем ИС, могут быть использованы совершенно различные подходы. В данном пособии мы постараемся рассмотреть круг вопросов, связанных с проектированием и созданием информационных систем на базе технологий Интернет. Как нам представляется, именно этот подход на сегодняшний день является наиболее прогрессивным и динамично развивающимся.

Попробуем определить несколько базовых принципов построения информационных систем на основе технологий Интернет.

В первую очередь отметим, что любая ИС содержит в себе следующий набор обязательных компонентов:

- интерфейс пользователя
- подсистема обработки пользовательской информации
- подсистема хранения данных
- канал передачи данных между центром процессинга и приложением пользователя

Технологии Интернет предлагают нам обширный инструментарий для реализации любого из вышеуказанных компонентов. Для обслуживания канала передачи данных воспользуемся семейством протоколов TCP/IP, в качестве подсистемы обработки пользовательской информации возьмем HTTP-сервер

Apache с подключенным языком написания скриптов PHP. С задачей хранения структурированных данных прекрасно справится один из многочисленных SQL-серверов. В качестве пользовательского интерфейса возможно использование как специально написанного программного обеспечения на языке высокого уровня, так и набор стандартных компонентов, предлагаемый нам любым Интернет-броузером и языком гипертекстовой разметки документов HTML.

При этом наша информационная система будет доступна пользователю из любой точки земного шара (во всяком случае если там есть Интернет или работает спутниковая связь), обеспечит высокую скорость обработки информации (благодаря технологии распределенных систем) и сможет защитить наши данные от постороннего вмешательства.

В чем на наш взгляд состоят основные преимущества использования технологий Интернет для построения современных информационных систем?

Выделим несколько ключевых причин

- открытые стандарты
- поддержка любой вычислительной платформы
- возможность использования программного обеспечения с открытым кодом
- относительно простая и эффективная организация многопользовательской работы
- относительно простой способ решения задачи актуализации программного обеспечения
- доступность информационной системы из любой точки мира

Безусловно, для проектирования и создания эффективных информационных систем необходимо изучить основные компоненты технологий Интернет.

2. Интернет. Краткое историческое введение

Около 20 лет назад Министерство Обороны США создало сеть, которая явилась предтечей Интернет, - она называлась ARPAnet. ARPAnet была экспериментальной сетью, - она создавалась для поддержки научных исследований в военно-промышленной сфере, - в частности, для исследования методов построения сетей, устойчивых к частичным повреждениям, получаемым, например, при бомбардировке авиацией и способных в таких условиях продолжать нормальное функционирование. Это требование дает ключ к пониманию принципов построения и структуры Интернет. В модели ARPAnet всегда была связь между компьютером-источником и компьютером-приемником (станцией назначения). Сеть *a priori* предполагалась ненадежной: любая часть сети может исчезнуть в любой момент.

На связывающиеся компьютеры - не только на саму сеть - также возложена ответственность обеспечивать налаживание и поддержание связи. Основной принцип состоял в том, что любой компьютер мог связаться как равный с равным с любым другим компьютером.

Передача данных в сети была организована на основе протокола Internet - IP. Протокол IP - это правила и описание работы сети. Этот свод включает правила налаживания и поддержания связи в сети, правила обращения с IP-пакетами и их обработки, описания сетевых пакетов семейства IP (их структура и тому подобное). Сеть задумывалась и проектировалась так, чтобы от пользователей не требовалось никакой информации о конкретной структуре сети. Для того, чтобы послать сообщение по сети, компьютер должен поместить данные в некий «конверт», называемый, например, IP, указать на этом «конверте» конкретный адрес в сети и передать получившиеся в результате этих процедур пакеты в сеть.

Эти решения могут показаться странными, как и предположение о «ненадежной» сети, но уже имеющийся опыт показал, что большинство этих решений вполне разумно и верно. Пока Международная Организация по

Стандартизации (Organization for International Standartization - ISO) тратила годы, создавая окончательный стандарт для компьютерных сетей, пользователи ждать не желали. Активисты Internet начали устанавливать IP-программное обеспечение на все возможные типы компьютеров. Вскоре это стало единственным приемлемым способом для связи разнородных компьютеров. Такая схема понравилась правительству и университетам, которые проводят политику покупки компьютеров у различных производителей. Каждый покупал тот компьютер, который ему нравился и вправе был ожидать, что сможет работать по сети совместно с другими компьютерами.

Примерно 10 лет спустя после появления ARPAnet появились Локальные Вычислительные Сети (LAN), например,такие как Ethernet и др. Одновременно появились компьютеры, которые стали называть рабочими станциями. На большинстве рабочих станций была установлена Операционная Система UNIX. Эта ОС имела возможность работы в сети с протоколом Internet (IP). В связи с возникновением принципиально новых задач и методов их решения появилась новая потребность: организации желали подключиться к ARPAnet своей локальной сетью. Примерно в то же время появились другие организации, которые начали создавать свои собственные сети, использующие близкие к IP коммуникационные протоколы. Стало ясно, что все только выиграли бы, если бы эти сети могли общаться все вместе, ведь тогда пользователи из одной сети смогли бы связываться с пользователями другой сети.

Одной из важнейших среди этих новых сетей была NSFNET, разработанная по инициативе Национального Научного Фонда (National Science Foundation - NSF). В конце 80-х годов NSF создал пять суперкомпьютерных центров, сделав их доступными для использования в любых научных учреждениях. Было создано всего лишь пять центров потому, что они очень дороги даже для богатой Америки. Именно поэтому их и следовало использовать кооперативно. Возникла проблема связи: требовался способ соединить эти центры и предоставить доступ к ним различным пользователям. Сначала была сделана попытка использовать коммуникации ARPAnet, но это

решение потерпело крах, столкнувшись с бюрократией оборонной отрасли и проблемой обеспечения персоналом.

Тогда NSF решил построить свою собственную сеть, основанную на IP технологии ARPAnet. Центры были соединены специальными телефонными линиями с пропускной способностью 56 Kbps. Однако, было очевидно, что не стоит даже и пытаться соединить все университеты и исследовательские организации непосредственно с центрами, так как проложить такое количество кабеля не только очень дорого, но практически невозможно. Поэтому решено было создавать сети по региональному принципу. В каждой части страны заинтересованные учреждения должны были соединиться со своими ближайшими соседями. Получившиеся цепочки подсоединялись к суперкомпьютеру в одной из своих точек, таким образом суперкомпьютерные центры были соединены вместе. В такой топологии любой компьютер мог связаться с любым другим, передавая сообщения через соседей.

Это решение было успешным, но настала пора, когда сеть уже более не справлялась с возросшими потребностями. Совместное использование суперкомпьютеров позволяло подключенным общинам использовать и множество других вещей, не относящихся к суперкомпьютерам. Неожиданно университеты, школы и другие организации осознали, что получили в свое распоряжение море данных и мир пользователей. Поток сообщений в сети (трафик) нарастал все быстрее и быстрее пока, в конце концов, не перегрузил управляющие сетью компьютеры и связывающие их телефонные линии. В 1987 году контракт на управление и развитие сети был передан компании Merit Network Inc., которая занималась образовательной сетью Мичигана совместно с IBM и MCI. Старая физически сеть была заменена более быстрыми (примерно в 20 раз) телефонными линиями. Были заменены на более быстрые и сетевые управляющие машины.

Процесс совершенствования сети идет непрерывно. Однако, большинство этих перестроек происходит незаметно для пользователей. Включив компьютер, вы не увидите объявления о том, что ближайшие полгода Internet не будет

доступна из-за модернизации. Возможно даже более важно то, что перегрузка сети и ее усовершенствование создали зрелую и практичную технологию. Проблемы были решены, а идеи развития проверены в деле.

Важно отметить то, что усилия NSF по развитию сети привели к тому, что любой желающий может получить доступ к сети. Прежде Internet была доступна только для исследователей в области информатики, государственным служащим и подрядчикам. NSF способствовал всеобщей доступности Internet по линии образования, вкладывая деньги в подсоединение учебного заведения к сети, только если то, в свою очередь, имело планы распространять доступ далее по округе. Таким образом, каждый студент четырехлетнего колледжа мог стать пользователем Internet.

И потребности продолжают расти. Большинство таких колледжей на Западе уже подсоединено к Internet, предпринимаются попытки подключить к этому процессу средние и начальные школы. Выпускники колледжей прекрасно осведомлены о преимуществах Internet и рассказывают о них своим работодателям. Вся эта деятельность приводит к непрерывному росту сети, к возникновению и решению проблем этого роста, развитию технологий и системы безопасности сети.

Что составляет Internet?

В действительности Internet не просто сеть, - она есть структура, объединяющая обычные сети. Internet - это «Сеть сетей»'. Что включает Internet? Вопрос непростой. Ответ на него меняется со временем. Вначале ответ был бы достаточно прост: «все сети, использующие протокол IP, которые кооперируются для формирования единой сети своих пользователей». Это включало бы различные ведомственные сети, множество региональных сетей, сети учебных заведений и некоторые зарубежные сети (за пределами США).

Чуть позже привлекательность Internet осознали и некоторые не-IP-сети. Они захотели предоставить ее услуги своим клиентам и разработали методы подключения этих «странных» сетей (например, Bitnet, DECnet и т.д.) к Internet.

Сначала эти подключения, названные шлюзами, служили только для передачи электронной почты. Однако, некоторые из них разработали способы передачи и других услуг. Являются ли эти сети частью Internet? И да, и нет. Все зависит от того, хотят ли они того сами.

Административное устройство Internet

Internet по организации во многом напоминает церковь. Это организация с полностью добровольным участием. Управляется она чем-то наподобие совета старейшин, однако, у Internet нет патриарха, президента или Папы. Составляющие сети могут иметь своих президентов или аналогичных вождей, но это совсем другое дело; в Internet нет единственной авторитарной фигуры. Высшая власть, где бы Internet ни была, остается за ISOC (Internet Society). ISOC - общество с добровольным членством. Его цель - способствовать глобальному обмену информацией через Internet. Оно назначает совет старейшин, который отвечает за техническую политику, поддержку и управление Internet.

Совет старейшин представляет собой группу приглашенных добровольцев, называемую IAB (Совет по архитектуре Internet). IAB регулярно собирается, чтобы «благословить» стандарты и распределить ресурсы, такие, например, как адреса. Internet работает, поскольку имеются стандартные способы общения между компьютерами и прикладными программами. Это позволяет компьютерам разного типа связываться без особых проблем. IAB ответственен за стандарты; он решает, когда стандарт необходим и каким ему следует быть. Когда требуется стандарт, совет рассматривает проблему, принимает стандарт и по сети оповещает о нем мир. IAB также следит за различными номерами (и другими вещами), которые должны оставаться уникальными. Например, каждый компьютер в Internet имеет свой уникальный 32-разрядный двоичный адрес никакой другой компьютер не имеет такого же. Как присваивается этот адрес? IAB заботится о такого рода проблемах. Он не присваивает адресов самолично, но разрабатывает правила, как эти адреса

присваивать.

Пользователи Internet высказывают свои жалобы и предложения на встречах IETF (Оперативного инженерного отряда Internet). IETF - это другая добровольная организация; также собирается регулярно, чтобы обсудить текущие эксплуатационные и назревающие технические проблемы. При обсуждении достаточно важной проблемы IETF создает рабочую группу для ее дальнейшего исследования. (На практике «достаточно важная» обычно означает, что для рабочей группы находится достаточное количество добровольцев). Посещать встречи IETF и состоять в рабочих группах могут все, главное, чтобы люди работали, дело-то добровольное. Рабочие группы имеют различные функции: это может быть выпуск документации, выработка стратегии действий при возникновении проблем, стратегические исследования, разработка новых стандартов и протоколов, доработка уже существующих (например, изменение значений отдельных полей). Рабочая группа обычно выпускает доклад. В зависимости от вида рекомендации, это может быть просто документацией и быть доступной для любого желающего, что может быть принято добровольно как здравая идея, или же это может быть послано в IAB и быть объявленной стандартом.

Если некая сеть принимает учение Internet, присоединяется к ней и считает себя ее частью, тогда она и является частью Internet. Возможно ей многое покажется неразумным, странным, сомнительным - она может поделиться своими сомнениями с IETF. Некоторые жалобы-предложения могут оказаться вполне разумными и, возможно, Internet соответственно изменится. Что-то может показаться просто делом вкуса или традиции, тогда эти возражения будут отклонены. Если сеть делает что-либо, что может навредить Internet, она может быть исключена из сообщества до тех пор, пока она не исправится.

Сейчас Internet состоит десятков тысяч объединенных между собой сетей.

Финансы

За Internet никто централизовано не платит; нет такой организации как Internet Inc., которая собирает плату со всех сетей Internet или пользователей. Вместо этого каждый платит за свою часть. NSF платит за содержание NSFNET. NASA платит за Научную Сеть NASA (NASA Science Internet). Представители сетей собираются вместе и решают, как им соединяться друг с другом и содержать эти взаимосвязи. Колледж или корпорация платит за ее подключение к некоторой региональной сети, которая в свою очередь платит за свой доступ сетевому владельцу государственного масштаба.

Как структура Internet сказывается на Пользователе ?

То, что Internet не сеть, а собрание сетей, мало как сказывается на конкретном пользователе. Для того, чтобы сделать что-нибудь полезное (запустить программу или добраться до каких-либо единственных в своем роде данных), пользователю не надо заботиться о том, как эти составляющие сети содержатся, как они взаимодействуют и поддерживают межсетевые связи.

Рассмотрим для наглядности телефонную сеть - тоже в некотором роде Internet. Министерство Связи России, Pacific Bell, AT&T, MCI, British Telecom, Telefon's de Mexico и т.д., - все это отдельные корпорации, которые обслуживают разные телефонные системы. Они же заботятся о совместной работе, о создании объединенной сети; все, что вам нужно сделать, где бы на планете вы ни находились и куда бы вы ни звонили, - это набрать номер. Если забыть о цене и рекламе, вам должно быть совершенно все равно, с кем вы имеете дело: с MCI, AT&T или Министерством Связи. Снимаете трубочку, нажимаете кнопочки и говорите.

Вас, как пользователя, заботит только, кто занимается вашими заявками, когда появляются проблемы. Если что-либо перестает работать, только одна из соответствующих компаний может исправить это. Они общаются друг с другом по проблемным вопросам, но каждый из владельцев сетей ответственен за проблемы, возникающие на его собственном участке системы, за сервис, который эта сеть предоставляет своим клиентам.

Это же верно и для Internet. Каждая сеть имеет свой собственный сетевой эксплуатационный центр (NOC). Каждый такой рабочий центр связан с другими и знает, как разрешить различные возможные проблемы. Ваш регион имеет соглашение с одной из составляющих сетей Internet и ее забота состоит в том, чтобы люди вашего региона были довольны работой сети. Так что, если что-то испортится, NOC и есть та самая организация, с кого за это спросят, кого за это будут бить.

Архитектура сетевых протоколов TCP/IP, на базе которых построена Internet, предназначена специально для объединенной сети. Сеть может состоять из совершенно разнородных подсетей, соединенных друг с другом шлюзами. В качестве подсетей могут выступать самые разные локальные сети (Token Ring, Ethernet, пакетные радиосети и т.п.), различные национальные, региональные и специализированные сети, а также другие глобальные сети, такие, например, как Bitnet или Sprint. К этим сетям могут подключаться машины совершенно разных типов. Каждая из подсетей работает в соответствии со своими специфическими требованиями и имеет свою природу связи, сама разрешает свои внутренние проблемы. Однако, предполагается, что каждая подсеть может принять пакет информации и доставить его по указанному адресу в этой конкретной подсети. Все же не требуется, чтобы подсеть гарантировала доставку пакетов и имела надежный сквозной протокол (протокол работы сети в качестве посредника при передаче сообщений между двух внешних сетей). Таким образом, две машины, подключенные к одной подсети, могут напрямую обмениваться пакетами, а если возникает необходимость передать сообщение машине в другой подсети, то вступают в силу межсетевые соглашения, для чего подсети используют свой межсетевой язык - протокол IP; они передают сообщение по определенной цепочке шлюзов и подсетей, пока оно не достигнет нужной подсети, где оно и будет доставлено непосредственно получателю. Другими словами, пользователя вся эта кухня совершенно не заботит. Как и в примере с телефонной сетью, которая представляется ему единой большой сетью, а не множеством сетей, для него все

это пестрое сборище разнородных и иногда несовместимых между собой сетей представляется одной сетью - «Сетью сетей» – Internet.

Потенциальные пользователи

Кому же может быть столь полезна Internet и каким образом? Что так способствует ее развитию?

Полезность Internet повышалась вместе с развитием вычислительной техники с запаздыванием примерно в 10 лет. В конце 80-х годов появление персональных компьютеров перенесло информатику из царства знатоков к широкой публике. Internet в ходе своего развития и повсеместного распространения занимается именно таким переносом.

Internet, как и вычислительная техника, совершила переход от забавы экспертов к инструменту ежедневного пользования. И сам процесс перехода был совершенно аналогичен. Сеть постепенно становилась проще в использовании, частично потому что оборудование стало лучше, а частично потому, что сама стала скорее и надежнее. И самые смелые из тех, кто сначала не решались связываться с Internet, начали ее использовать. Эти новые пользователи породили огромную потребность в новых ресурсах и лучшем инструментарии. Улучшались старые средства, появлялись новые, предназначенные для доступа к новым ресурсам, что облегчало использование сети. И вот уже другая группа людей стала понимать пользу Internet. Процесс повторялся. Этот круговорот продолжает развиваться и по сей день.

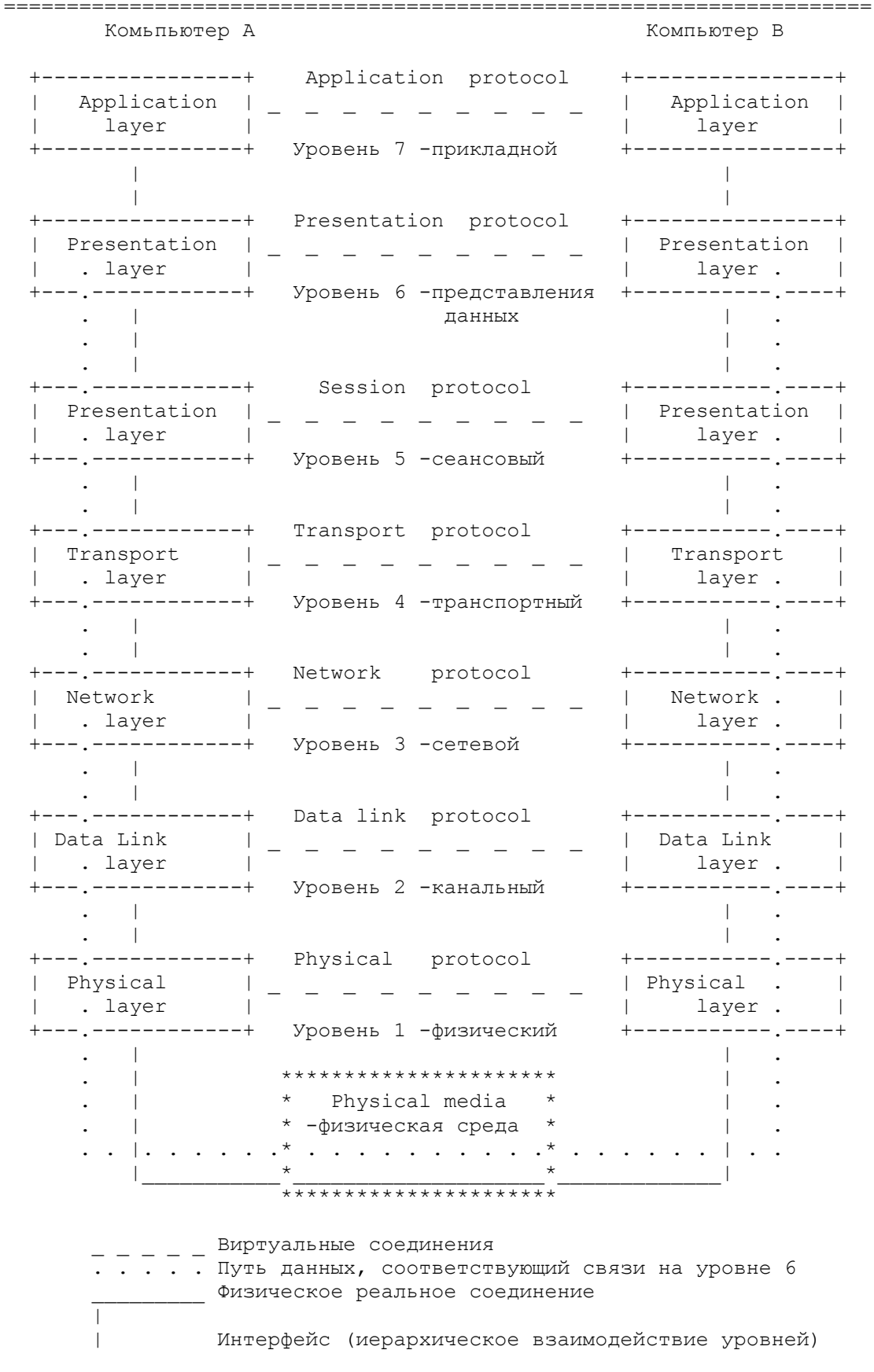
В общем, все пользователи Internet ищут одного: общения и информации. И они находят это среди людей и компьютеров.

3. Работа Интернет. Организация, структура, методы

Эталонная модель ISO OSI

Структура функционирования сети

Современные сети построены по многоуровневому принципу. Чтобы организовать связь двух компьютеров, требуется сначала создать свод правил их взаимодействия, определить язык их общения, то есть определить, что означают посылаемые ими сигналы и так далее. Эти правила и определения называются протоколом. Для работы сетей необходимо запастись множеством различных протоколов: например, управляющих физической связью, установлением связи по сети, доступом к различным ресурсам и т.д. Многоуровневая структура спроектирована с целью упростить и упорядочить это великое множество протоколов и отношений. Продемонстрируем принципы этой структуры на диаграмме. На рисунке показана общепринятая семиуровневая структура согласно ISO. Эта модель известна как «эталонная модель ISO OSI». Она позволяет составлять сетевые системы из продуктов - модулей программного обеспечения - выпущенных разными производителями.


Комьпьютер А
Компьютер В
Application protocol
Application layer
Уровень 7 -прикладной
Presentation protocol
Presentation layer
Уровень 6 -представления данных
Session protocol
Уровень 5 -сеансовый
Transport protocol
Transport layer
Уровень 4 -транспортный
Network protocol
Network layer
Уровень 3 -сетевой
Data link protocol
Data Link layer
Уровень 2 -канальный
Physical protocol
Physical layer
Уровень 1 -физический
Physical media
-физическая среда
Виртуальные соединения
Путь данных, соответствующий связи на уровне 6
Физическое реальное соединение
Интерфейс (иерархическое взаимодействие уровней)

Взаимодействие уровней в этой модели - субординарное. Каждый уровень может реально взаимодействовать только с соседними уровнями (верхним и нижним), виртуально - только с аналогичным уровнем на другом конце линии.

Под реальным взаимодействием мы подразумеваем непосредственное взаимодействие, непосредственную передачу информации, например, пересылку данных в оперативной памяти из области, отведенной одной программе, в область другой программы. При непосредственной передаче данные остаются неизменными все время. Под виртуальным взаимодействием мы понимаем опосредованное взаимодействие и передачу данных; здесь данные в процессе передачи могут уже определенным, заранее оговоренным образом видоизменяться.

Такое взаимодействие аналогично схеме цепи посылки письма одним директором фирмы другому. Например, директор некоторой фирмы пишет письмо редактору газеты. Директор пишет письмо на своем фирменном бланке и отдает этот листок секретарю. Секретарь запечатывает листок в конверт, надписывает конверт, наклеивает марку и передает почте. Почта доставляет письмо в соответствующее почтовое отделение. Это почтовое отделение связи непосредственно доставляет письмо получателю - секретарю редактора газеты. Секретарь распечатывает конверт и, по мере надобности, подает редактору. Ни одно из звеньев цепи не может быть пропущено, иначе цепь разорвется: если отсутствует, например, секретарь, то листок с письменами директора так и будет пылиться на столе у секретаря.

Здесь мы видим, как информация (лист бумаги с текстом) передается с верхнего уровня вниз, проходя множество необходимых ступеней - стадий обработки. Обрастает служебной информацией (пакет, адрес на конверте, почтовый индекс; контейнер с корреспонденцией; почтовый вагон, станция назначения почтового вагона и так далее), изменяется на каждой стадии обработки и постепенно доходит до самого нижнего уровня - уровня почтового транспорта (автомобильного, железнодорожного, воздушного,...), которым реально перевозится в пункт назначения. В пункте назначения происходит

обратный процесс: вскрывается контейнер и извлекается корреспонденция, считывается адрес на конверте и почтальон несет его адресату (секретарю), который восстанавливает информацию в первоначальном виде, - достает письмо из конверта, прочитывает его и определяет его срочность, важность, и в зависимости от этого передает информацию выше. Директор и редактор, таким образом, виртуально имеют прямую связь. Ведь редактор газеты получает в точности ту же информацию, которую отправил директор, а именно - лист бумаги с текстом письма. Начальствующие персоны совершенно не заботятся о проблемах пересылки этой информации. Секретари также имеют виртуально прямую связь: секретарь редактора получит в точности то же, что отправил секретарь директора, а именно - конверт с письмом. Секретарей совершенно не волнуют проблемы почты, пересылающей письма. И так далее.

Аналогичные связи и процессы имеют место и в эталонной модели ISO OSI. Физическая связь реально имеет место только на самом нижнем уровне (аналог почтовых поездов, самолетов, автомобилей). Горизонтальные связи между всеми остальными уровнями являются виртуальными, реально они осуществляются передачей информации сначала вниз, последовательно до самого нижнего уровня, где происходит реальная передача, а потом, на другом конце, обратная передача вверх последовательно до соответствующего уровня.

Модель ISO OSI предписывает очень сильную стандартизацию вертикальных межуровневых взаимодействий. Такая стандартизация гарантирует совместимость продуктов, работающих по стандарту какого-либо уровня, с продуктами, работающими по стандартам соседних уровней, даже в том случае, если они выпущены разными производителями. Количество уровней может показаться избыточным, однако же, такое разбиение необходимо для достаточно четкого разделения требуемых функций во избежание излишней сложности и создания структуры, которая может подстраиваться под нужды конкретного пользователя, оставаясь в рамках стандарта.

Дадим краткий обзор уровней эталонной модели ISO OSI.

Уровень 0 связан с физической средой - передатчиком сигнала и на самом деле не включается в эту схему, но весьма полезен для понимания. Этот почетный уровень представляет посредников, соединяющих конечные устройства: кабели, радиолинии и так далее. Кабелей существует великое множество различных видов и типов: экранированные и неэкранированные витые пары, коаксиальные, на основе оптических волокон и т.д. Так как этот уровень не включен в схему, он ничего и не описывает, только указывает на среду.

Уровень 1 - физический. Включает физические аспекты передачи двоичной информации по линии связи. Детально описывает, например, напряжения, частоты, природу передающей среды. Этому уровню вменяется в обязанность поддержание связи и прием-передача битового потока. Безошибочность желательна, но не требуется.

Уровень 2 - канальный. Связь данных. Обеспечивает безошибочную передачу блоков данных (называемых кадрами (frame)) через уровень 1, который при передаче может искажать данные. Этот уровень должен определять начало и конец кадра в битовом потоке, формировать из данных, передаваемых физическим уровнем, кадры или последовательности , включать процедуру проверки наличия ошибок и их исправления. Этот уровень (и только он) оперирует такими элементами, как битовые последовательности, методы кодирования, маркеры. Он несет ответственность за правильную передачу данных (пакетов) на участках между непосредственно связанными элементами сети. Обеспечивает управление доступом к среде передачи. В виду его сложности, канальный уровень подразделяется на два подуровня: MAC (Medium Access Control) - Управление доступом к среде и LLC (Logical Link Control) - Управление логической связью (каналом). Уровень MAC управляет доступом к сети (с передачей маркера в сетях Token Ring или распознаванием конфликтов (столкновений передач) в сетях Ethernet) и управлением сетью.

Уровень LLC, действующий над уровнем MAC, и есть собственно тот уровень, который посылает и получает сообщения с данными.

Уровень 3 - сетевой. Этот уровень пользуется возможностями, предоставляемыми ему уровнем 2, для обеспечения связи двух любых точек в сети. Любых, необязательно смежных. Этот уровень осуществляет проводку сообщений по сети, которая может иметь много линий связи, или по множеству совместно работающих сетей, что требует маршрутизации, то есть определения пути, по которому следует пересылать данные. Маршрутизация производится на этом же уровне. Выполняет обработку адресов, а также и демультиплексирование.

Основной функцией программного обеспечения на этом уровне является выборка информации из источника, преобразование ее в пакеты и правильная передача в точку назначения.

Есть два принципиально различных способа работы сетевого уровня. Первый - это метод виртуальных каналов. Он состоит в том, что канал связи устанавливается при вызове (начале сеанса (session) связи), по нему передается информация, и по окончании передачи канал закрывается (уничтожается). Передача пакетов происходит с сохранением исходной последовательности, даже если пакеты пересылаются по различным физическим маршрутам, то есть виртуальный канал динамически перенаправляется. При этом пакеты данных не включают адрес пункта назначения, так как он определяется во время установления связи.

Второй - метод дейтаграмм . Дейтаграммы - независимые , они включают всю необходимую для их пересылки информацию. В то время, как первый метод предоставляет следующему уровню (уровню 4) надежный канал передачи данных, свободный от искажений (ошибок) и правильно доставляющий пакеты в пункт назначения, второй метод требует от следующего уровня работы над ошибками и проверки доставки нужному адресату.

Уровень 4 - транспортный. Регламентирует пересылку пакетов сообщений между процессами, выполняемыми на компьютерах сети. Завершает

организацию передачи данных: контролирует на сквозной основе поток данных, проходящий по маршруту, определенному третьим уровнем: правильность передачи блоков данных, правильность доставки в нужный пункт назначения, их комплектность, сохранность, порядок следования. Собирает информацию из блоков в ее прежний вид. Или же оперирует с дейтаграммами, то есть ожидает отклика-подтверждения приема из пункта назначения, проверяет правильность доставки и адресации, повторяет посылку дейтаграммы, если не пришел отклик. В рамках транспортного протокола предусмотрено пять классов качества транспортировки и соответствующие процедуры управления. Этот же уровень должен включать развитую и надежную схему адресации для обеспечения связи через множество сетей и шлюзов. Другими словами, задачей данного уровня является довести до ума передачу информации из любой точки в любую во всей сети.

Транспортный уровень скрывает от всех высших уровней любые детали и проблемы передачи данных, обеспечивает стандартное взаимодействие лежащего над ним уровня с приемом-передачей информации независимо от конкретной технической реализации этой передачи.

Уровень 5 - сеансовый. Координирует взаимодействие связывающихся пользователей: устанавливает их связь, оперирует с ней, восстанавливает аварийно оконченные сеансы. Этот же уровень ответственен за картографию сети - он преобразовывает региональные (доменные) компьютерные имена в числовые адреса , и наоборот. Он координирует не компьютеры и устройства, а процессы в сети, поддерживает их взаимодействие - управляет сеансами связи между процессами прикладного уровня.

Уровень 6 - уровень представления данных. Этот уровень имеет дело с синтаксисом и семантикой передаваемой информации, то есть здесь устанавливается взаимопонимание двух сообщающихся компьютеров относительно того, как они представляют и понимают по получении передаваемую информацию. Здесь решаются, например, такие задачи, как перекодировка текстовой информации и изображений, сжатие и распаковка,

поддержка сетевых файловых систем (NFS), абстрактных структур данных и так далее.

Уровень 7 - прикладной. Обеспечивает интерфейс между пользователем и сетью, делает доступными для человека всевозможные услуги. На этом уровне реализуется, по крайней мере, пять прикладных служб: передача файлов, удаленный терминальный доступ, электронная передача сообщений, служба справочника и управление сетью. В конкретной реализации определяется пользователем (программистом) согласно его насущным нуждам и возможностям его кошелька, интеллекта и фантазии. Имеет дело, например, с множеством различных протоколов терминального типа, которых существует более ста.

Следует понимать, что подавляющее большинство современных сетей в силу исторических причин лишь в общих чертах, приближенно, соответствуют эталонной модели ISO OSI. Рассмотрим некоторые из указанных уровней в их проекции на Internet подробнее. Следует не забывать, что проекция искажена в силу несоответствия технологии Internet стандарту эталонной модели - перемешаны некоторые уровни и функции уровней

Уровни работы сети

Пересылка битов

Пересылка битов происходит на физическом уровне схемы ISO OSI. Увы, здесь всякая попытка краткого и доступного описания обречена на провал. Требуется введение огромного количества специальных терминов, понятий, описаний процессов на физическом уровне и так далее. И потом, существует столь великое разнообразие приемо-передатчиков и передающих сред, - трудно даже и обозреть этот океан технологий. Для понимания работы сетей этого и не требуется. Считайте, что просто имеется труба, по которой из конца в конец перекачиваются биты. Именно биты, безо всякого деления на какие-либо

группы (байты, декады и тому подобное).

Пересылка данных

Об организации блочной, символьной передачи, обеспечении надежности пересылки поговорим на других уровнях модели ISO OSI. То есть функции канального уровня в Internet распределены по другим уровням, но не выше транспортного. В этом смысле Internet не совсем соответствует стандарту ISO. Здесь канальный уровень занимается только разбиением битового потока на символы и кадры и передачей полученных данных на следующий уровень. Обеспечением надежности передачи он себя не утруждает.

Сети коммутации пакетов

Настала пора поговорить об Internet именно как о сети, а не паутине линий связи и множестве приемо-передатчиков. Казалось бы, Internet вполне аналогична телефонной сети, и модель телефонной сети достаточно адекватно отражает ее структуру и работу. В самом деле, обе они электронные, обе позволяют вам устанавливать связь и передавать информацию. И Internet тоже состоит, в первую очередь, из выделенных телефонных линий. Но увы! Картина эта неверна и приводит ко многим заблуждениям относительно работы Internet, ко множеству недоразумений. Телефонная сеть - это так называемая сеть с коммутацией каналов, то есть когда вы делаете вызов, устанавливается связь и на все время сеанса связи имеется физическое соединение с абонентом. При этом вам выделяется часть сети, которая для других уже не доступна, даже если вы молча дышите в трубку, а другие абоненты хотели бы поговорить по действительно неотложному делу. Это приводит к нерациональному использованию очень дорогих ресурсов - линий связи. Internet же является сетью с коммутацией пакетов, что принципиально отличается от сети с коммутацией каналов.

Для Internet более подходит модель, которая поначалу может не внушать доверия: почта, обыкновенная государственная почтовая служба. Почта

является сетью пакетной связи. Нет никакой выделенной вам части этой сети. Ваше послание перемешивается с посланиями других пользователей, кидается в контейнер, пересылается в другое почтовое отделение, где снова сортируется. Хотя технологии сильно разнятся, почта является прекрасным и наглядным примером сети с коммутацией пакетов. Модель почты удивительно точно отражает суть работы и структуры Internet. Ею мы и будем пользоваться далее.

Протокол Internet (IP)

По проводу можно переслать биты только из одного его конца в другой. Internet же умудряется аккуратно передавать данные в различные точки, разбросанные по всему миру. Как она это делает? Забота об этом возложена на сетевой (межсетевой) уровень в эталонной модели ISO OSI. О нем и поговорим.

Различные части Internet - составляющие сети - соединяются между собой посредством компьютеров, которые называются «узлы»; так Сеть связывается воедино. Сети эти могут быть Ethernet, Token Ring, сети на телефонных линиях, пакетные радиосети и тому подобные. Выделенные линии и локальные сети суть аналоги железных дорог, самолетов почты и почтовых отделений, почтальонов. Посредством их почта движется с места на место. Узлы - аналоги почтовых отделений, где принимается решение, как перемещать данные («пакеты») по сети, точно так же, как почтовый узел намечает дальнейший путь почтового конверта. Отделения или узлы не имеют прямых связей со всеми остальными. Если вы отправляете конверт из города Асино (Томская область) в Уфу (Башкирия), конечно же, почта не станет нанимать самолет, который полетит из ближайшего к Асино аэропорта (Богашево) в Уфу, просто местное почтовое отделение отправляет послание на подстанцию в нужном направлении, та в свою очередь, дальше в направлении пункта назначения на следующую подстанцию; таким образом письмо станет последовательно приближаться к пункту назначения, пока не достигнет почтового отделения, в ведении которого находится нужный объект и которое доставит сообщение

получателю. Для работы такой системы требуется, чтобы каждая подстанция знала о наличествующих связях и о том, на какую из ближайших подстанций оптимально следует передать адресованный туда-то пакет. Примерно также и в Internet: узлы выясняют, куда следует ваш пакет данных, решают куда его дальше отправить и отправляют.

На каждой почтовой подстанции определяется следующая подстанция, куда будет далее направлена корреспонденция, то есть намечается дальнейший путь (маршрут) - этот процесс называется маршрутизацией. Для осуществления маршрутизации каждая подстанция имеет таблицу, где адресу пункта назначения (или индексу) соответствует указание почтовой подстанции, куда следует посылать далее этот конверт (бандероль,). Их сетевые аналоги называются таблицами маршрутизации. Эти таблицы рассылаются почтовым подстанциям централизовано соответствующим почтовым подразделением. Время от времени рассылаются предписания по изменению и дополнению этих таблиц. В Internet, как и любые другие действия, составление и модификация, таблиц маршрутизации (этот процесс тоже является частью маршрутизации и называется так же) определяются соответствующими правилами - протоколами ICMP (Internet Control Message Protocol), RIP (Routing Internet Protocol) и OSPF (Open Shortest Path First). Узлы, занимающиеся маршрутизацией, называются маршрутизаторами.

А откуда сеть знает, куда назначен ваш пакет данных? От вас. Если вы хотите отправить письмо и хотите, чтобы ваше письмо достигло места назначения, вы не можете просто кинуть листочек бумаги в ящик. Вам следует уложить его в стандартный конверт и написать на нем не «на деревню дедушке», как Ванька Жуков, а адрес получателя в стандартной форме. Только тогда почта сможет правильно обработать ваше письмо и доставить его по назначению. Аналогично в Internet имеется набор правил по обращению с пакетами - протоколы. Протокол Internet (IP) берет на себя заботы по адресации

или по подтверждению того, что узлы понимают, что следует делать с вашими данными по пути их дальнейшего следования. Согласно нашей аналогии, протокол Internet работает так же как правила обработки почтового конверта. В начало каждого вашего послания помещается заголовок, несущий информацию об адресате, сети. Чтобы определить, куда и как доставить пакет данных, этой информации достаточно.

Адрес в Internet состоит из 4 байт. При записи байты отделяются друг от друга точками: 123.45.67.89 или 3.33.33.3 . В действительности адрес состоит из нескольких частей. Так как Internet есть сеть сетей, начало адреса говорит узлам Internet, частью какой из сетей вы являетесь. Правый конец адреса говорит этой сети, какой компьютер или хост должен получить пакет (хотя реально не все так просто, но идея такова). Каждый компьютер в Internet имеет в этой схеме уникальный адрес, аналогично обычному почтовому адресу, а еще точнее - индексу. Обработка пакета согласно адресу также аналогична. Почтовая служба знает, где находится указанное в адресе почтовое отделение, а почтовое отделение подробно знает подопечный район. Internet знает, где искать указанную сеть, а эта сеть знает, где в ней находится конкретный компьютер. Для определения, где в локальной сети находится компьютер с данным числовым IP-адресом, локальные сети используют свои собственные протоколы сетевого уровня. Например, Ethernet для отыскания Ethernet-адреса по IP-адресу компьютера, находящегося в данной сети, использует протокол ARP - протокол разрешения(в смысле различения) адресов.

Числовой адрес компьютера в Internet аналогичен почтовому индексу отделения связи. Первые цифры индекса говорят о регионе (например, 45 - это Башкирия, 63 – Томская область и т.д.), последние две цифры - номер почтового отделения в городе, области или районе. Промежуточные цифры могут относиться как к региону, так и к отделению, в зависимости от территориального деления и вида населенного пункта. Аналогично существует

несколько типов адресов Internet (типы: A, B, C, D, E), которые по-разному делят адрес на поля номера сети и номера узла, от типа такого деления зависит количество возможных различных сетей и машин в таких сетях.

По ряду причин (особенно, - практических, из-за ограничений оборудования) информация, пересылаемая по сетям IP, делится на части (по границам байтов), раскладываемые в отдельные пакеты. Длина информации внутри пакета обычно составляет от 1 до 1500 байт. Это защищает сеть от монополизации каким-либо пользователем и предоставляет всем примерно равные права. Поэтому же, если сеть недостаточно быстра, чем больше пользователей ее одновременно пользует, тем медленнее она будет общаться с каждым.

Протокол IP является дейтаграммным протоколом, то есть IP-пакет является дейтаграммой. Это совершенно не укладывается в модель ISO OSI, в рамках которой уже сетевой уровень способен работать по методу виртуальных каналов.

Одно из достоинств Internet состоит в том, что протокола IP самого по себе уже вполне достаточно для работы (в принципе). Это совершенно неудобно, но, при достаточных аскетичности, уме и упорстве удастся проделать немалый объем работы. Как только данные помещаются в оболочку IP, сеть имеет всю необходимую информацию для передачи их с исходного компьютера получателю. Работа вручную с протоколом IP напоминает нам суровые времена доперсональной компьютерной эры, когда пользователь всячески угождал ЭВМ, укрощая свои тело, дух и эстетические чувства. Об удобстве пользователя никто и не собирался думать, потому что машинное время стоило во много раз дороже человеческого. Но сейчас в аскетизме надобности уже нет. Поэтому следует построить на основе услуг, предоставляемых IP, более совершенную и удобную систему. Для этого сначала следует разобраться с некоторыми

жизненно важными проблемами, которые имеют место при пересылке информации:

- Большая часть пересылаемой информации длиннее 1500 символов. Если бы почта пересылала только почтовые карточки и отказывалась бы от пересылки чего-либо большего, мы бы, например, лишились увлекательнейшего литературного жанра - эпистолярного. Не говоря уже о том, что практической пользы от такой почты было бы очень немного;
- Возможны и неудачи. Почта, нередко бывает, письма теряет; сеть тоже, бывает, теряет пакеты или искажает в пути информацию в них. В отличие от почты, Internet может с честью выходить из таких затруднительных положений;
- Пакеты могут приходить в последовательности, отличной от начальной. Пара писем, отправленных друг за другом на днях, не всегда приходит к получателю в том же порядке; то же верно и для Internet.

Таким образом, следующий уровень Internet должен обеспечить способ пересылки больших массивов информации и позаботиться об «искажениях», которые могут возникать по вине сети.

Протокол управления передачей (TCP) и протокол пользовательских дейтаграмм (UDP)

Transmission Control Protocol - это протокол, тесно связанный с IP, который используется в аналогичных целях, но на более высоком уровне - транспортном уровне эталонной модели ISO OSI. Часто эти протоколы, по причине их тесной связи, именуют вместе, как TCP/IP. Термин «TCP/IP» обычно означает все, что связано с протоколами TCP и IP. Он охватывает целое семейство протоколов, прикладные программы и даже саму сеть. В состав семейства входят протоколы TCP, UDP, ICMP, telnet, FTP и многие другие. TCP/IP - это технология межсетевого взаимодействия, технология internet. Сеть,

которая использует технологию internet, называется internet.

Сам протокол TCP занимается проблемой пересылки больших объемов информации, основываясь на возможностях протокола IP. Как это делается? Вполне здраво можно рассмотреть следующую ситуацию. Как можно переслать книгу по почте, если та принимает только письма и ничего более? Очень просто: разодрать ее на страницы и отправить страницы отдельными конвертами. Получатель, руководствуясь номерами страниц, легко сможет книгу восстановить. Этим же простым и естественным методом и пользуется TCP.

TCP делит информацию, которую надо переслать, на несколько частей. Нумерует каждую часть, чтобы позже восстановить порядок. Чтобы пересылать эту нумерацию вместе с данными, он обкладывает каждый кусочек информации своей обложкой - конвертом, который содержит соответствующую информацию. Это и есть TCP-конверт. Получившийся TCP-пакет помещается в отдельный IP-конверт и получается IP-пакет, с которым сеть уже умеет обращаться.

Получатель (TCP-модуль (процесс)) по получении распаковывает IP-конверты и видит TCP-конверты, распаковывает и их и помещает данные в последовательность частей в соответствующее место. Если чего-то не достает, он требует переслать этот кусочек снова. В конце концов информация собирается в нужном порядке и полностью восстанавливается. Вот теперь этот массив пересылается выше к пользователю (на диск, на экран, на печать).

В действительности, это слегка утрированный взгляд на TCP. В реальности пакеты не только теряются, но и могут искажаться при передаче из-за наличия помех на линиях связи. TCP решает и эту проблему. Для этого он пользуется системой кодов, исправляющих ошибки. Существует целая наука о

таких кодировках. Простейшим примером такового служит код с добавлением к каждому пакету контрольной суммы (и к каждому байту бита проверки на четность). При помещении в TCP-конверт вычисляется контрольная сумма, которая записывается в TCP-заголовок. Если при приеме заново вычисленная сумма не совпадает с той, что указана на конверте, значит что-то тут не то, - где-то в пути имели место искажения, так что надо переслать этот пакет по-новой, что и делается.

Для ясности и полноты картины, необходимо сделать здесь важное замечание: Модуль TCP разбивает поток байтов на пакеты, не сохраняя при этом границ между записями. То есть, если один прикладной процесс делает 3 записи в -порт, то совсем не обязательно, что другой прикладной процесс на другом конце виртуального канала получит из своего -порта именно 3 записи, причем именно таких (по разбиению), что были переданы с другого конца. Вся информация будет получена исправно и с сохранением порядка передачи, но она может уже быть разбита по другому и на иное количество частей. Не существует зависимости между числом и размером записываемых сообщений с одной стороны и числом и размером считываемых сообщений с другой стороны. TCP требует, чтобы все отправленные данные были подтверждены принявшей их стороной. Он использует ожидания (таймауты) и повторные передачи для обеспечения надежной доставки. Отправителю разрешается передавать некоторое количество данных, не дожидаясь подтверждения приема ранее отправленных данных. Таким образом, между отправленными и подтвержденными данными существует окно уже отправленных, но еще не подтвержденных данных. Количество байт, которое можно передавать без подтверждения, называется размером окна. Как правило, размер окна устанавливается в стартовых файлах сетевого программного обеспечения. Так как TCP-канал является , то есть данные могут одновременно передаваться в обоих направлениях, то подтверждения для данных, идущих в одном направлении, могут передаваться вместе с данными, идущими в

противоположном направлении. Приемники на обеих сторонах виртуального канала выполняют управление потоком передаваемых данных для того, чтобы не допускать переполнения буферов.

Таким образом, протокол TCP обеспечивает гарантированную доставку с установлением логического соединения в виде байтовых потоков. Он освобождает прикладные процессы от необходимости использовать ожидания и повторные передачи для обеспечения надежности. Наиболее типичными прикладными процессами, использующими TCP, являются ftp и telnet.

Большие возможности TCP даются не бесплатно, реализация TCP требует большой производительности процессора и большой пропускной способности сети. Когда прикладной процесс начинает использовать TCP, то начинают общаться модуль TCP на машине пользователя и модуль на машине сервера. Эти два оконечных модуля TCP поддерживают информацию о состоянии соединения - виртуального канала. Этот виртуальный канал потребляет ресурсы обоих оконечных модулей TCP. Канал этот, как уже указывалось, является дуплексным. Один прикладной процесс пишет данные в TCP-порт, откуда они модулями соответствующих уровней по цепочке передаются по сети и выдаются в TCP-порт на другом конце канала, и другой прикладной процесс читает их отсюда - из своего TCP-порта. эмулирует (создает видимость) выделенную линию связи двух пользователей. Гарантирует неизменность передаваемой информации. Что входит на одном конце, выйдет с другого. Хотя в действительности никакая прямая линия отправителю и получателю в безраздельное владение не выделяется (другие пользователи могут пользовать те же узлы и каналы связи в сети в промежутках между пакетами этих), но извне это, практически, именно так и выглядит.

Как бы хорошо это не звучало, но это не панацея. Как уже отмечалось, установка TCP-виртуального канала связи требует больших расходов на

инициирование и поддержание соединения и приводит к задержкам передачи. Если вся эта суета - излишество, лучше обойтись без нее. Если все данные, предназначенные для пересылки, умещаются в одном пакете, и если вас не особенно заботит надежность доставки, то можно обойтись без TCP.

Имеется другой стандартный протокол транспортного уровня, который не отягощен такими накладными расходами. Этот протокол называется UDP - User Datagram Protocol - протокол пользовательских дейтаграмм. Он используется вместо TCP. Здесь данные помещаются не в TCP, а в UDP-конверт, который также помещается в IP-конверт. Этот протокол реализует дейтаграммный способ передачи данных.

Дейтаграмма - это пакет, передаваемый через сеть независимо от других пакетов без установления логического соединения и подтверждения приема. Дейтаграмма - совершенно самостоятельный пакет, поскольку сама содержит всю необходимую для ее передачи информацию. Ее передача происходит безо всякого предварения и подготовки. Дейтаграммы, сами по себе, не содержат средств обнаружения и исправления ошибок передачи, поэтому при передаче данных с их помощью следует принимать меры по обеспечению надежности пересылки информации. Методы организации надежности могут быть самыми разными, обычно же используется метод подтверждения приема посылкой эхо-отклика при получении каждого пакета с дейтаграммой.

UDP проще TCP, поскольку он не заботится о возможной пропаже данных, пакетов, о сохранении правильного порядка данных и так далее. UDP используется для клиентов, которые посылают только короткие сообщения и могут просто заново послать сообщение, если отклик подтверждения не придет достаточно быстро. Предположим, что вы пишите программу, которая просматривает базу данных с телефонными номерами где-нибудь в другом месте сети. Совершенно незачем устанавливать TCP связь, чтобы передать 33

или около того символов в каждом направлении. Вы можете просто уложить имя в UDP-пакет, запаковать это в IP-пакет и послать. На другом конце прикладная программа получит пакет, прочитает имя, посмотрит телефонный номер, положит его в другой UDP-пакет и отправит обратно. Что произойдет, если пакет по пути потеряется? Ваша программа тогда должна действовать так: если она ждет ответа слишком долго и становится ясно, что пакет затерялся, она просто повторяет запрос, то есть посылает еще раз то же послание. Так обеспечивается надежность передачи при использовании протокола UDP.

В отличие от TCP, данные, отправляемые прикладным процессом через модуль UDP, достигают места назначения как единое целое. Например, если процесс-отправитель производит 3 записи в UDP-порт, то процесс-получатель должен будет сделать 3 чтения. Размер каждого записанного сообщения будет совпадать с размером соответствующего прочитанного. Протокол UDP сохраняет границы сообщений, определяемые прикладным процессом. Он никогда не объединяет несколько сообщений в одно целое и не делит одно сообщение на части.

Альтернатива TCP-UDP позволяет программисту гибко и рационально использовать предоставленные ресурсы, исходя из своих возможностей и потребностей. Если нужна надежная доставка, то лучше может быть TCP. Если нужна доставка дейтаграмм, то - UDP. Если нужна эффективная доставка по длинному и ненадежному каналу передачи данных, то лучше использовать TCP. Если нужна эффективность на быстрых сетях с короткими соединениями, лучше всего будет UDP. Если потребности не попадают ни в одну из этих категорий, то выбор транспортного протокола не ясен. Прикладные программы, конечно, могут устранять некоторые недостатки выбранного протокола. Например, если вы выбрали UDP, а вам необходима надежность, то прикладная программа должна обеспечить надежность сама, как описано выше: требовать подтверждения, пересылки утерянных или увечных пакетов и так далее. Если

вы выбрали TCP, а вам нужно передавать записи, то прикладная программа должна вставлять метки в поток байтов так, чтобы можно было различить записи.

Некоторые стандартные протоколы семейства TCP/IP

FTP - протокол пересылки файлов. С помощью этого протокола организовывается пересылка файлов по сети.

TELNET - протокол эмуляции терминала удаленной машины. С помощью этого протокола организовываются сеансы работы на удаленных машинах сети - эмулируется терминал далекого компьютера.

SMTP - протокол пересылки простой почты - протокол, обеспечивающий работу e-mail.

TFTP - протокол простейшей (тривиальной) пересылки файлов. Осуществляет дейтаграммную пересылку файлов. Получается медленнее и дороже, чем по FTP.

DNS - протокол доменной (региональной) системы имен - протокол запросов на преобразование имен из доменной формы в машинную - числовую.

Служба времени - служба синхронизации разнесенных в сети часов. Это протокол сетевого времени - NTP.

Эхо - протокол для обратной передачи передаваемых по сети символов, служит, например, для подтверждения.

GGP - протокол «шлюз - шлюз» - протокол сквозных сетевых проводок.

HMP - протокол мониторинга хоста - протокол непрерывного слежения (контроля) за хостом (сетевым рабочим компьютером).

EGP - протокол внешнего шлюза.

ICMP - протокол межсетевых управляющих сообщений - сетевой протокол, предоставляющий процессам в сети возможность предпочтительного выбора в своем множестве маршрутов пересылки и других подобных вещей.

Создание сети с человеческим лицом. Прикладное обеспечение

И вот мы имеем возможность передавать информацию между различными точками в сети. Вот теперь мы можем начать работать над созданием дружественного интерфейса Internet, позаботиться об удобстве для пользователя. Для этого мы напишем программное обеспечение, которое будет понимать язык команд, выдавать сообщения об ошибках, подсказки, использовать для адресации сетевых компьютеров при общении с пользователем имена, а не числа и так далее. В модели ISO OSI на это работают уровни выше транспортного, то есть сеансовый, представления данных и прикладной. Вся эта деятельность направлена на повышение уровня удобства работы в сети, на создание систем, позволяющих пользоваться предоставляемыми возможностями обычному пользователю сети.

Ведь большинство пользователей совсем не волнует ни наличие надежного потока битов между машинами, ни пропускная способность этих линий или тонкости и особенности используемой технологии, ни даже экзотичность этой технологии. Они хотят использовать этот битовый поток для дела, как то: переслать файл, добраться до каких-то данных или просто поиграть в игру. Приложения - это части программного обеспечения. Их создают на основе сервиса TCP или UDP. Приложения позволяют пользователю достаточно просто справиться с возникшей задачей, не погружаясь в пучину технической информации о конкретной сети, о протоколах и так далее.

Прикладное обеспечение разнится очень сильно. Приложения могут быть от самодельной программы до патентованных продуктов, поставляемых различными фирмами (DEC, Microsoft и т.п.). Существует три стандартных Internet -приложения: удаленный доступ, передача файлов, электронная почта (e-mail); наряду с ними используются другие широко распространенные нестандартные приложения.

Предоставление услуг Internet построено по схеме «клиент - сервер».

Предоставление услуг осуществляется совместной работой двух процессов: на компьютере пользователя и на компьютере-сервере. Процесс на компьютере пользователя называется клиентом, а на компьютере-сервере - сервером. Клиент и сервер являются, по сути, частями одной программы, взаимодействующие по виртуальной связи в сети. Сервер по указаниям клиента выполняет соответствующие действия, например, пересылает клиенту файл. Для предоставления услуги совершенно необходимо наличие двух этих модулей - клиента и сервера, и их одновременная согласованная работа. Взаимодействие клиента и сервера описывается соответствующими стандартными протоколами, поэтому клиент и сервер могут быть выпущены совершенно разными производителями и работать на разнородных компьютерах. Поэтому же существует небольшая проблема нестандартности интерфейса клиента непосредственно уже с пользователем. Это взаимодействие может иметь совершенно различную форму: интерактивную, командную и так далее. Системы команд могут различаться. Но от этого сами возможности не изменяются, поскольку клиент и сервер всегда взаимодействуют одинаково - согласно протоколу.

Так как прикладным обеспечением снабжают по большей части через локальные сети, в разговоре о приложениях возникает вышеупомянутая проблема: команды, сообщения, справки, подсказки и тому подобное в разных локальных сетях могут в той или иной степени отличаться. Об этом не следует забывать при чтении руководств пользователя: сообщения могут отличаться, но смысл их будет такой же, то же касается и команд. Даже если они слегка отличаются, не стоит волноваться, большинство приложений имеет разумную систему подсказок и описание набора команд, где вы детально и конкретно сможете разузнать все, что вам понадобится.

Системы сетевых адресов

Региональная Система Имен

Числовые адреса хороши для связи машин, люди же предпочитают имена. Очень непросто разговаривать, используя машинную адресацию (как бы это звучало: «192.112.36.5 обещает вскоре...»?), еще труднее запомнить эти адреса. Поэтому компьютерам в Internet для удобства пользователей были присвоены собственные имена. Тогда описанный разговор принимает вид: «NIC обещает вскоре...». Все приложения Internet позволяют пользоваться системными именами вместо числовых адресов.

Как мы уже упоминали, для понимания полезно использовать почтовую аналогию. Сетевые численные адреса вполне аналогичны почтовой индексации. Машины, сортирующие корреспонденцию на почтовых узлах, ориентируются именно по индексам, и только если с индексами выходит какая-то несуразность, передают почту на рассмотрение людям, которые по адресу могут определить правильный индекс почтового отделения места назначения. Людям же приятнее и удобнее иметь дело с географическими названиями - это аналоги доменных имен.

Конечно, такое именование имеет свои собственные проблемы. Прежде всего, следует убедиться, что никакие два компьютера, включенные в сеть, не имеют одинаковых имен. Должно также обеспечить преобразование имен в числовые адреса, для того чтобы машины (и программы) могли понимать нас, пользующихся именами: техника по-прежнему общается на языке цифр.

В начале Internet размерами напоминала курилку, и иметь дело с именами было довольно просто. NIC создал регистратуру. Можно было послать запрос и в ответ высылали список имен и адресов. Этот файл, называется «host file» (файл рабочих ЭВМ), регулярно распространялся по всей сети - рассылался всем машинам. Имена были простыми словами, все были единственными. Если вы использовали имя, ваш компьютер просматривал этот файл и подставлял вместо имени реальный числовой адрес. Так же, как работает телефонный

аппарат со встроенным списком абонентов. Все было легко, просто и замечательно. Всем хватало простых имен, в курилке был один Джон, один Марк, один Анатолий.

Но по мере развития и расширения Internet возрастало количество пользователей, хостов, а потому увеличивался и упомянутый файл. Возникали значительные задержки при регистрации и получении имени новым компьютером, стало затруднительно изыскивать имена, которые еще никто не использовал, слишком много сетевого времени затрачивалось на рассылку этого огромного файла всем машинам, в нем упомянутым. Стало очевидно, - чтобы справиться с такими темпами изменений и роста сети, нужна распределенная оперативная система, опирающаяся на новый принцип. Таковая была создана, ее назвали «доменной системой имен» - DNS, а способ адресации - способом адресации по доменному принципу. DNS иногда еще называют региональной системой наименований.

Структура региональной системы имен

Доменная система имен - это метод назначения имен путем передачи сетевым группам ответственности за их подмножество имен. Каждый уровень этой системы называется доменом. Домены в именах отделяются друг от друга точками: ed.tusur.ru, db.ed.tomsk.ru, www.yandex.ru, www.microsoft.com. В имени может быть различное количество доменов, но практически их не больше пяти. По мере движения по доменам слева направо в имени, количество имен, входящих в соответствующую группу возрастает.

Первым в имени стоит название рабочей машины - реального компьютера с IP адресом. Это имя создано и поддерживается группой (например, компьютер www (это web-сервер на UNIX компьютере в учебной классе кафедры Электронных приборов) в группе ed (кафедра Электронных приборов), к которой он относится. Группа входит в более крупное подразделение tusur

(Томский государственный университет систем управления и радиоэлектроники, ТУСУР), которое в свою очередь, является частью национальной сети (Российской Федерации, домен ru). Для США наименование страны по традиции опускается, там самыми крупными объединениями являются сети образовательных (edu), коммерческих (com), государственных (gov), военных (mil) учреждений, а также сети других организаций (org) и сетевых ресурсов (net).

Группа может создавать или изменять любые ей подлежащие имена. Если ed решит поставить другой компьютер, например, Windows NT, и назвать его winnt, он ни у кого не должен спрашивать разрешения, все, что от него требуется, - это добавить новое имя в соответствующую часть соответствующей всемирной базы данных, и, рано или поздно, каждый, кому потребуется, узнает об этом имени. Аналогично, если в ТУСУРе решат создать новую группу, например, schools, они (домен tusur) могут это сделать также, ни у кого на то не спрашивая никакого соизволения. И тогда, если каждая группа придерживается таких простых правил и всегда убеждается, что имена, которые она присваивает, единственны во множестве ее непосредственных подчиненных, то никакие две системы, где бы те ни были в сети Internet, не смогут заиметь одинаковых имен.

Эта ситуация совершенно аналогична ситуации с присвоением географических названий - организацией почтовых адресов. Названия всех стран различаются. Различаются названия всех областей, республик в Федерации, и эти названия утверждаются в государственном масштабе из центра (конечно, обычно сами регионы заботятся об уникальности своих названий, поэтому здесь царит полная демократия: как республика хочет, так она и называется). В республиках - субъектах федерации - решают вопросы о названиях районов и округов, в пределах одной республики они различаются. Аналогично далее с городами и улицами городов. В разных городах могут быть

улицы с одинаковыми названиями: почему бы не быть во всех городах Союза по улице Ленина или Мира? Это улицы разных городов, и их не перепутать. В пределах же одного населенного пункта улицы всенепременно имеют разные названия, причем именование этих улиц целиком и полностью под ответственностью и началом соответствующего центрального органа данного населенного пункта (мэрии, сельсовета, горсовета). Таким образом, почтовый адрес на основе географических и административных названий однозначно определяет точку назначения.

Поскольку Internet - сеть мировая, требовался также способ передачи ответственности за имена внутри стран им самим. Сейчас принята двухбуквенная кодировка государств. Это оговорено в RFC 822. Так, например, домен Канада называется ca, Россия - ru, США - us и т.д. США также включили в эту систему структурирования для всеобщности и порядка. Всего же кодов стран почти 300, из которых около 100 имеет компьютерную сеть того или иного рода. Единый каталог Internet находится у SRI International (Менло-Парк, Калифорния, США) - государственной организации.

Поиск адреса по доменному имени

Теперь вы знаете, как соотносятся домены и создаются имена. Возможно, вы теперь озадачены: а как использовать эту замечательную систему? Автоматически. Вам надо лишь употребить имя на компьютере, который понимает, как обращаться с DNS. Вам никогда не придется самим разыскивать адрес, соответствующий этому имени, или подавать специальную команду для его поиска (в UNIX - команда nslookup). Вы, конечно, можете это проделать - для собственного удовольствия, но зачем, ведь этого совсем не требуется. Все компьютеры Internet способны пользоваться доменной системой. И работающий в сети компьютер всегда знает свой собственный сетевой адрес.

Когда вы пользуетесь именем, например, ms.tusur.ru, компьютер должен

преобразовать его в адрес. Для этого он начинает запрашивать помощь у DNS-серверов. Это узлы, рабочие машины, обладающие соответствующей базой данных, в число обязанностей которых входит обслуживание такого рода запросов. DNS-сервер начинает обработку имени с правого его конца и двигается по нему влево, то есть сначала производится поиск адреса в самой большой группе (домене), потом постепенно сужает поиск. Но для начала опрашивается на предмет наличия у него нужной информации местный узел. Здесь возможны три случая:

- Местный сервер знает адрес, потому, что этот адрес содержится в его части всемирной базы данных. Например, если вы подсоединены к сети Томского государственного университета систем управления и радиоэлектроники (tusur), то ваш местный сервер должен обладать информацией о всех компьютерах локальной сети этого института (ms, muma, ed и т.д.);
- Местный сервер знает адрес, потому, что кто-то недавно уже запрашивал тот же адрес. Когда запрашивается адрес, сервер DNS придерживает его у себя в памяти некоторое время, как раз на случай, если кто-нибудь еще захочет попозже того же адреса - это повышает эффективность системы;
- Местный сервер адрес не знает, но знает как его выяснить.

Как местный сервер может разузнать запрошенный адрес? В его прикладном или системном программном обеспечении имеется информация о том, как связаться с корневым сервером. Это сервер, который знает адреса серверов имен высшего уровня (самых правых в имени), здесь это уровень государств (ранга домена ru). У него запрашивается адрес компьютера, ответственного за зону ru. Местный DNS-сервер связывается с этим более общим сервером и запрашивает у него адрес сервера, ответственного за домен tusur.ru. Теперь уже запрашивается этот сервер и у него запрашивается адрес рабочей машины ms.

На самом деле, для повышения эффективности, поиск начинается не с самого верха, а с наименьшего домена, в который входите и вы, и компьютер, имя которого вы запросили. Например, если ваш компьютер имеет имя trailers.tomsk.ru, то опрос начнется (если имя не выяснится сразу) не со всемирного сервера, чтобы узнать адрес сервера группы ru, а сразу с группы ru, что сразу сокращает поиск и по объему, и по времени.

Этот поиск адреса совершенно аналогичен поиску пути письма без надписанного почтового индекса. Как определяется этот индекс? Все регионы пронумерованы - это первые цифры индекса. Письмо пересылается на центральный почтамт этого региона, где имеется справочник с нумерацией районов этого региона - это следующие цифры индекса. Теперь письмо идет на центральный почтамт соответствующего района, где уже знают все почтовые отделения в подопечном районе. Таким образом по географическому адресу определяется почтовый индекс, ему соответствующий. Также определяется и адрес компьютера в Internet, но путешествует не послание, а запрос вашего компьютера об этом адресе. И в отличие от случая с почтой, информация об адресе доходит до вас, как если бы районный почтамт места назначения отправлял вам письмо, любезно уведомляя вас на будущее об индексе, которого вы не изволили знать.

Замечания по региональной системе имен

Распространено несколько заблуждений, с которыми вы можете столкнуться, имея дело с именами. Приведем несколько верных утверждений в качестве опорных, чтобы вывести вас из заблуждений, или предостеречь от них:

- Части доменного имени говорят о том, кто ответственен за поддержку этого имени, то есть в чьем подчинении-ведении оно находится. Они могут вообще ничего не сообщать о владельце компьютера,

соответствующего этому IP-адресу, или даже (несмотря на коды стран), где же эта машина находится. Вполне можно иметь в Антарктиде машину с именем www.tusur.ru (главный web-сервер ТУСУРа, город Томск). Это совершенно ненормально, но никаким законам не противоречит.

- Части доменного имени даже не всегда указывают локальную сеть, в которой расположен компьютер. Часто доменные имена и сети перекрываются, и жестких связей между ними нет: две машины одного домена могут не принадлежать одной сети. Например, системы mx.ed.tusur.ru и ms.ed.tusur.ru могут находиться в совершенно разных сетях. И еще раз: доменные имена указывают на ответственного за домен.
- У машины может быть много имен. В частности, это верно для машин, предоставляющих какие-либо услуги, которые в будущем могут быть перемещены под опеку другой машины. Когда эти службы будут перемещены, то имя, под которым эта машина выступала в качестве такого сервера, будет передано новой машине-серверу вместе с услугами, - для внешних пользователей ничего не изменится. То есть они будут продолжать пользоваться этой службой, запрашивая ее по тому же имени, независимо от того, какой компьютер на самом деле занимается обслуживанием. Имена, по смыслу относящиеся к службе, называются «каноническими именами» или «именами» (cnames). В Internet они встречаются довольно часто.
- Для связи имена необязательны. Как-нибудь вам придет сообщение: «адресат неизвестен», что означает, что Internet не может преобразовать использованное вами имя в число, - имя более недееспособно в том виде, в котором его знает ваш компьютер. Однажды заполучив числовой эквивалент имени, ваша система перестает использовать для связи на машинном уровне доменную форму адреса.
- Запоминать лучше имена, а не числовые адреса. Некоторым кажется, что система имен это «еще одно звено в цепи, которое может выйти из строя». Но адреса привязаны к конкретным точкам сети. Если компьютер,

предоставляющий некие услуги, переносится из одного здания в другое, его сетевое расположение, а значит и адрес, скорее всего изменятся. Имя же менять не надо и не следует. Когда администратор присваивает новый адрес, ему нужно только обновить запись имени в базе данных так, чтобы имя указывало на новый адрес. Так как имя работает по-прежнему, вас совершенно не должно заботить то, что компьютер расположен уже в другом месте.

Региональная система имен, возможно, и выглядит сложно, но это одна из тех составляющих, делающих общение с сетью более простым и удобным. Несомненное преимущество доменной системы состоит в том, что она разбивает громадье Internet на набор вполне обозримых и управляемых частей. Хотя сеть включает миллионы компьютеров, все они поименованы, и именование это организовано в удобной рациональной форме, что упрощает работу.

Маршрутизация

Маршрутизация является необходимой составной частью функционирования сети. Она осуществляется на сетевом уровне, поэтому стоило бы рассказать о ней раньше, когда шел разговор о протоколе IP, но это не было сделано преднамеренно, только для того, чтобы сохранить генеральную линию изложения, не метаться из стороны в сторону, пытаясь объять необъятное.

Так как Интернет есть Сеть тетей, и к тому же она имеет совершенно гигантские размеры, то и маршрутизация в ней должна быть организована иерархически. Разные части Интернет проповедуют различные принципы такой иерархии, поэтому говорить здесь можно только о наиболее общих положениях.

Маршрутизация в Интернет происходит раздельно для различных

уровней группирования в Сети. Каждая локальная сеть использует свои методы маршрутизации. Определенные группы локальных сетей объединяются в RD - Routing Domains - домены (области) маршрутизации, в которых маршрутизация производится согласно единым принципам, алгоритмам и протоколам. Элементарным объектом маршрутизации в RD являются уже не отдельные компьютеры а сети Интернет.

Вообще говоря, возможна сквозная горизонтальная маршрутизация через границы RD, но это скорее исключение, сделанное для повышения эффективности использования сетевых линий связи в соседских отношениях, чем правило. Обычно же, маршрутизация на уровне адресации сетей, тем более - на уровне адресации конечных систем (ES - end system) происходит только внутри границ одного RD.

Сами RD полностью находятся внутри какого-либо одного AD - Administration Domain - административного домена. AD - это сеть или группа сетей, работающих под единым управляющим началом. В одном административном домене может быть много разных маршрутных доменов. AD обычно соответствует какой-либо организации или ассоциации. Сами AD могут объединяться в AD более высокого уровня и так далее. То есть, административный домен является основным иерархическим элементом в структуре Сети.

Маршрутизация сообщений между конечными системами (хостами) различных доменов маршрутизации происходит на уровне адресации таких доменов, то есть в RD имеется особый внешний маршрутизатор, знающий куда следует переправлять сообщения, адресованные в разные RD, он направляет пакеты по соответствующим путям в нужные RD, а доставка пакета внутри RD-получателя производится силами самого этого RD. При этом, если домены маршрутизации находятся в разных административных доменах, начинает

работать аналогичная схема с делегацией полномочий маршрутизации вверх.

При этом на каждом уровне маршрутизации возможно множество различных альтернативных маршрутов, по разному выходящих из домена данного уровня (трасса, ж/д вокзал, аэропорт и так далее)

Протоколы, которыми пользуются маршрутизаторы для прокладки путей по Сети, в общем основаны на нескольких базовых алгоритмах. Расскажем о двух разных протоколах: RIP и OSPF.

RIP

RIP - это Routeing Information Protocol - протокол маршрутной информации. RIP относится к широкоизвестному классу протоколов маршрутизации, основанных на алгоритме Беллмана-Форда, относящемуся к классу так называемых алгоритмов векторов расстояний. Этот протокол наиболее подходит для использования в качестве внутреннего (меж)шлюзового протокола (IGP), то есть для маршрутизации сообщений внутри автономной системы (AS), которая работает под неким единым началом и по единым принципам. RIP предназначен для работы в сетях (автономных системах) умеренных размеров, использующих достаточно однородную технологию. Он подходит для сетей многих компаний, научных и учебных заведений, использующих последовательные линии передачи данных, пропускные способности которых меняются в пределах сети незначительно.

RIP в качестве метрики может использовать только статические функции (постоянные во времени). Использование динамических метрик приводит к возникновению неустойчивостей, которые он обрабатывать не умеет.

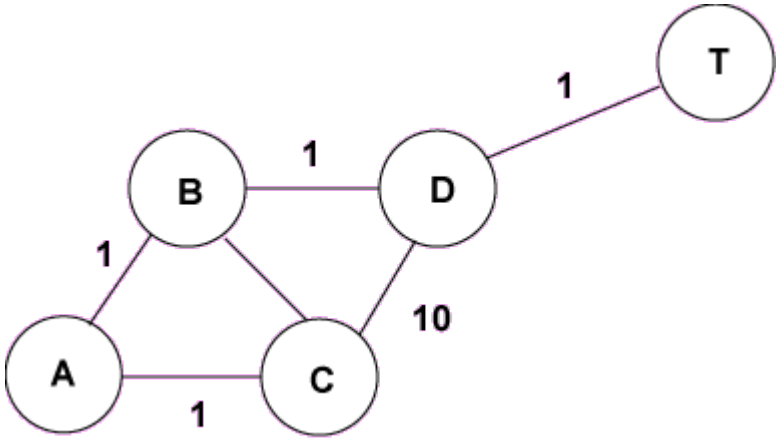
Алгоритмы векторов расстояний основаны на обмене между маршрутизаторами только информацией о «расстояниях» между ними, точнее о

соответствующих метриках. При этом считается, что сами сети являются точечными объектами, то есть расстояния между любыми двумя узлами одной сети равны нулю, поэтому расстояния между маршрутизаторами двух разных сетей равны расстоянию между этими сетями.

Каждый маршрутизатор имеет таблицу с перечислением всех других сетей данного домена маршрутизации с указанием тех своих соседей, через которых проходят пути к этим сетям, и соответствующих метрик - расстояний до этих сетей от сети данного маршрутизатора. Таким образом, эти таблицы расстояний индивидуальные у каждого маршрутизатора и отличаются от таблиц других маршрутизаторов.

Очевидно, что из этой таблицы расстояний составить собственно таблицу маршрутизации - пара пустяков - просто выбросить длины путей, и таблица готова. Вся проблема в том, как составить эти таблицы (вектора) расстояний так, чтобы они соответствовали действительности. Именно для этого и нужен алгоритм Беллмана-Форда.

Давайте придумаем простенькую систему сетей и будем вести рассуждения на ее примере.



Пусть узлы A, B, C, D и T являются маршрутизаторами сетей с

аналогичными названиями. Пусть, далее, все линии, кроме линии C-D имеют «длины» 1, а линия C-D - 10. Можно считать, что реально это не длины, а, например, стоимости пересылки по этим линиям связи.

Советуем вам самим проследить, что происходит в этой системе с таблицами расстояний. Пусть, например, начальное состояние векторов расстояний такое как показано в таблице:

Маршрутизатор A

Путь до	**лежит через**	**и имеет длину**
B	B	1
C	C	1
D	B	3
T	C	6

Маршрутизатор B

Путь до	**лежит через**	**и имеет длину**
C	C	1
D	D	1
A	A	1
T	D	2

Маршрутизатор C

Путь до	**лежит через**	**и имеет длину**
D	D	10
A	A	1
B	B	1
T	A	2

Маршрутизатор D

Путь до	**лежит через**	**и имеет длину**
A	B	2

B	B	1
C	C	10
T	T	1

Следует учесть, что в данных таблицах мы специально придумали наиболее идиотское их начальное состояние, чтобы была возможность убедиться, что алгоритм Беллмана-Форда приводит к нужному результату при любых начальных условиях. Естественно, расстояния между соседями должны всегда соответствовать действительности, идиотизировать их мы не имеем права, иначе, исчезнет источник достоверной информации о состоянии сети.

Алгоритм Беллмана-Форда велит действовать так.

Каждый маршрутизатор рассылает свою таблицу расстояний всем маршрутизаторам - его непосредственным соседям. Фактически это означает, что маршрутизатор сообщает всем своим соседям, что он думает, что от него добираться до разных сетей следует так-то и что эти пути имеют такие-то длины. Соседи воспринимают это всерьез и начинают корректировать с учетом этой информации собственные таблицы расстояний.

А именно, если маршрутизатор C видит, что его сосед A, знает более короткий путь до сети T, чем он сам, естественно маршрутизатор учитывает путь от него до соседа, просто складывая эти пути, то есть если оказывается, что через соседа A путь до сети T короче, то он изменяет запись в своей таблице расстояний, соответствующую сети T, вписывает туда, что до сети T путь лежит через этого самого соседа A и имеет длину:

```
(длина пути, сообщенная соседом A) + (длина пути до
соседа A)
```

Если же маршрутизатор C видит, что тот его сосед (A), через который проходит путь от него (C) к сети T, вдруг говорит, что путь от него (от A) до этой самой сети T, оказывается длиннее, чем он (A) думал раньше, то C безропотно меняет записи в своей таблице, соответствующей сети T, старое расстояние на новое, которое он вычисляет тем же очевидным образом:

```
(длина пути, сообщенная соседом) + (длина пути до соседа)
```

Очевидно, что расстояния между соседями должны быть известны заранее - это собственно и есть задаваемая метрика. И такие переговоры нужно повторять до полного удовлетворения. Вот, собственно и весь алгоритм Беллмана-Форда.

Существует математически строгое доказательство того, что этот алгоритм порождает сходящуюся последовательность векторов расстояний, причем, эта последовательность сходится к нужному вектору состояний для каждого маршрутизатора. Да, естественно, доказательство основано на условиях теоремы, рассматривающей только случай стационарной сети, то есть сети, в которой топология постоянна во времени: линии не рвутся, маршрутизаторы не ломаются и так далее. Предполагается, что участники этого процесса не перестают рассылать сообщения, и сами сообщения не вязнут навечно в недрах сети. При этом почти не делается никаких предположений о том, в каком порядке рассылаются сообщения с векторами расстояний, и в какие моменты времени может происходить изменение векторов расстояний. Как вы сами видите условия теоремы содержат очень слабые требования.

Отсутствие ограничений на моменты рассылки сообщений позволяет всем маршрутизаторам работать асинхронно, по своим собственным часам.

Отсутствие ограничений на начальные состояния векторов расстояний,

кроме лишь того, что все расстояния должны быть неотрицательными, означает, что если топология сети изменяется, то процесс все равно будет продолжать сходиться, но, скорее всего, уже к новому вектору расстояний. Действительно, при изменении топологии сети мы можем просто забыть, все что прежде было, и считать, что мы начинаем все сначала.

Итак, процесс сходится за конечное время при отсутствии изменений в топологии сети. Если изменения случаются, то процесс будет постепенно сходиться, но сама равновесная точка будет скакать под действием этих изменений топологии.

Если изменения топологии сети достаточно редки, то процесс будет успевать сходиться и хоть какое-то время будет осуществляться корректная маршрутизация, а если изменения топологии будут слишком часты, то алгоритм утратит дееспособность и маршрутизация совсем разладится.

Однако исходный алгоритм Беллмана-Форда не предусматривает случая, когда некая сеть или ее маршрутизатор становятся совсем недоступны, так как это фактически равносильно тому, что этот маршрутизатор перестает рассылать сообщения. В таком случае, теорема о сходимости неприменима, а процесс совершенно очевидно, никогда не сойдется.

Поэтому RIP развивает алгоритм Беллмана-Форда, обвешивая его разными хитростями.

Для случаев обрыва линий RIP должен иметь механизм диагностики обрыва. Диагностика выполняется просто: если сеть не отвечает соседу слишком долго, считается что линия связи оборвана. Этой линии присваивается метрика, называемая «бесконечностью». В действительности, это просто число, которое реальной метрикой быть не может, пока будем считать, что оно

является только признаком этой «бесконечности» и никаким иным целям не служит. «Бесконечность» при сложении с любым другим числом дает опять «бесконечность». И «бесконечность» больше любого другого числа.

Теперь самостоятельно (это очень просто) составьте таблицы расстояний для каждого маршрутизатора для приведенного примера сети, соответствующих стационарному состоянию системы сетей, убедитесь что они порождают корректные таблицы маршрутизации.

Все бы хорошо, да вот приехал дядя Вася на экскаваторе и перепахал линию B-D. Промоделируйте, что будет твориться в сети теперь. Особенно внимательно следите за тем, что будет происходить с маршрутизацией сообщений от A к T и от C к T. Видите? До A и C быстро дошло, что путь к T через B теперь закрыт. Но A продолжает считать, что путь к T по прежнему открыт через C, а C считает, что путь к T открыт через A. Так они кивают друг на друга и медленно увеличивают длину пути к T, проложенному друг через друга. Эта забава длится довольно долго, очевидно тем дольше, чем больше «длина» линии C-D.

Теперь представьте себе, что дядя Вася совсем разошелся и, пока шел этот процесс, перекопал линию C-D. A и C будут кивать друг на друга до бесконечности.

Для борьбы с такими зацикливаниями было решено использовать пресловутую «бесконечность». Ее сделали обычным конечным числом, но оставили правило, гласящее, что сумма «бесконечности» и любого другого числа есть «бесконечность». При такой трактовке «бесконечности» вышеописанное кивание A и C друг на друга, очевидно будет продолжаться до тех пор, пока они постепенно не доведут длину путей к T, проложенных друг через друга до «бесконечности» + 1. Тогда они поймут, что линия-то оборвана и

бросят это занятие.

Уловка сработала, но для ее практического использования следует придумать, каким числом должна быть «бесконечность». Очевидно, что с одной стороны, это число должно быть достаточно большим, чтобы никакой реальный путь не мог иметь такую большую длину. Однако, с другой стороны, это число должно быть как можно меньше, чтобы вышеописанный процесс «счета до бесконечности» продолжался как можно меньшее время. Сначала в качестве компромисса было выбрано 15. Позже, с ростом Интернет «бесконечность» довели до 30. Очевидно, что эта самая «бесконечность» ограничивает сверху возможную длину пути, что может быть нежелательно в больших сетях.

Существует множество других уловок, придуманных для избавления от «счета до бесконечности», но все они носят частный характер, и только «бесконечность» по-прежнему остается надежной гарантией от зацикливаний в общем случае.

OSPF

Начнем с указания недостатков протокола RIP.

Этот протокол использует фиксированную метрику - методику вычисления "длин" маршрутов для их сравнения друг с другом. Поэтому он неприменим в ситуациях, требующих динамической маршрутизации, в зависимости от текущего состояния сети (текущей загруженности линий, их надежности и т.п.). Попытки искусственного расширения возможностей RIP путем введения динамических метрик типа времен задержек при передаче по линии, пропускной способности и т.п. приводят к возникновению неустойчивостей, с которыми алгоритм Беллмана-Форда справиться не способен.

RIP можно использовать только в достаточно однородных сетях

умеренных размеров. Использование алгоритма Беллмана-Форда для маршрутизации в больших сетях очень неэффективно. При расширении сети увеличивается количество изменений топологии сети в единицу времени, что требует более частой передачи маршрутной информации. При этом также увеличивается и сама таблица маршрутизации, что в RIP означает увеличение объема передаваемой за один раз маршрутной информации.

К тому же, изменение топологии сети, например выход из строя какой-либо линии связи, приводит к многошаговому переходному процессу, приводящему общую маршрутную информацию (а значит и маршрутизацию) к новому равновесному состоянию, соответствующему правильной маршрутизации в новых условиях. Многошаговость этой процедуры перехода приводит к значительной длительности периода времени, в течение которого маршрутизация, вообще говоря, осуществляется некорректно. В такие периоды возможно временное зацикливание части трафика. Пакеты могут уходить в совершенно дурном направлении, превращаясь в марсиан (martians).

Еще одна проблема - ограничение, накладываемое RIP на количество проходимых пакетом узлов.

RIP также подразумевает выбор одного единственного маршрута, даже если существует множество равноценных альтернатив. Поэтому RIP может направлять весь трафик по одному единственному маршруту, сильно загружая соответствующие линии связи, игнорируя другие равноценные, которые могут оказаться совсем не задействованными в передаче данного трафика.

OSPF - это Open Shortest Path First - открой кратчайший путь первым - протокол следующего поколения. Помимо основной функции - маршрутизации - он предоставляет услуги, отсутствующие у RIP. Он оперативно распределяет трафик между равноценными маршрутами, оптимизируя таким образом

использование линий связи. Обеспечивает аутентификацию маршрутов и административный контроль, производя маршрутизацию областей. Также он предоставляет возможность маршрутизации по типу трафика.

OSPF использует иные механизмы сбора и использования маршрутной информации. Для составления таблиц маршрутизации он может использовать любые методы, в том числе алгоритм Беллмана-Форда.

Как уже было отмечено, в RIP не существует единой сводки информации о состояниях линий сети, но каждый маршрутизатор ведет свою собственную информационную базу о расстояниях от его подсети до всех остальных, а правильные таблицы маршрутизации получаются неявно - в результате многократных обменов информацией по специально подобранной схеме.

В OSPF сбор информации осуществляется в явном виде, причем, каждый маршрутизатор обладает полной информацией о состоянии сети, которую он получает в информационном обмене со своими коллегами, поэтому он сам может произвести все необходимые для составления таблицы маршрутизации вычисления в явном виде (по тому алгоритму, который ему подходит).

Как только случается измениться состоянию какой-либо линии связи в сети, ответственный за нее маршрутизатор рассылает всем остальным маршрутизаторам информацию о своих подопечных линиях связи и их текущих состояниях (о пропускной способности, о задержках, о загруженности и так далее). Рассылка информации производится методом лавинной маршрутизации, то есть каждый маршрутизатор сразу рассылает полученную им маршрутную информацию всем своим сотоварищам, которые еще не успели получить ее. При этом каждый маршрутизатор должен подтвердить получение этой информации. Маршрутизаторы, исходя из полученной информации, вычисляют соответствующие таблицы маршрутизации независимо и самостоятельно. Такое

вычисление происходит, очевидно, практически мгновенно.

При изменении состояния какой-либо линии сети, информация об этом изменении и о новом состоянии линии лавиной проходит через всю сеть, позволяя маршрутизаторам параллельно вычислить новые таблицы маршрутизации. Таким образом, реакция на изменения оказывается почти мгновенной.

OSPF обеспечивает маршрутизацию по оптимальным с требуемой точки зрения линиям за счет полностью перестраиваемой под нужды пользователя маршрутной метрики, для задания которой может использоваться произвольная (конечно неотрицательная) функция от задержки, пропускной способности, стоимости и любых других факторов. Например, трафик может направляться по самым дешевым маршрутам.

Очевидно, что OSPF не нуждается в ограничении маршрутизации одной единственной исходящей линией. Он может эффективно распределять трафик между несколькими равноценными альтернативными маршрутами, что позволяет более эффективно использовать линии связи.

Так как все вычисления производятся локально на каждом маршрутизаторе отдельно, отсутствует какое-либо ограничение на количество единичных (двухточечных) линий связи на одном маршруте, что позволяет работать в сетях значительно больших масштабов.

OSPF также делает простым осуществление маршрутизации по типу трафика. Такая маршрутизация позволяет разделять весь трафик на классы (до восьми классов) и предоставлять разные маршруты для разных классов. Например, передача файлов может осуществляться по спутниковой линии с большой пропускной способностью, обладающей, однако, значительными

задержками, и одновременно осуществляться передача трафика telnet по наземной арендованной линии с малой пропускной способностью, но малыми задержками.

4. Наиболее распространенные возможности Internet

Электронная почта (e-mail)

Оценки говорят, что в мире имеется более 100 миллионов пользователей электронной почты. В целом же в мире трафик электронной почты (протокол smtp) занимает только 3.7% всего сетевого. Популярность ее объясняется, как насущными требованиями, так и тем, что большинство подключений - подключения класса «доступ по вызову» (с модема), а у нас в России, вообще, в подавляющем большинстве случаев - доступ UUCP. E-mail доступна при любом виде доступа к Internet .

E-mail (Electronic mail) - электронная почта - электронный аналог обычной почты. С ее помощью вы можете посылать сообщения, получать их в свой электронный почтовый ящик, отвечать на письма ваших корреспондентов автоматически, используя их адреса, исходя из их писем, рассылать копии вашего письма сразу нескольким получателям, переправлять полученное письмо по другому адресу, использовать вместо адресов (числовых или доменных имен) логические имена, создавать несколько подразделов почтового ящика для разного рода корреспонденции, включать в письма текстовые файлы, пользоваться системой «отражателей почты» для ведения дискуссий с группой ваших корреспондентов и так далее. Из Internet вы можете посылать почту в сопредельные сети, если вы знаете адрес соответствующего шлюза, формат его обращений и адрес в той сети.

Используя e-mail, вы можете пользоваться ftp в асинхронном режиме. Существует множество серверов, поддерживающих такие услуги. Вы посылаете e-mail в адрес такой службы, содержащую команду этой системы, например, дать листинг какой-то директории, или переслать файл такой-то к вам, и вам приходит автоматически ответ по e-mail с этим листингом или нужным файлом. В таком режиме возможно использование почти всего набора

команд обычного ftp. Существуют серверы, позволяющие получать файлы по ftp не только с них самих, но с любого ftp-сервера, который вы укажете в своем послании e-mail.

E-mail дает возможность проводить телеконференции и дискуссии. Для этого используются, установленные на некоторых узловых рабочих машинах, mail reflector-ы. Вы посылаете туда сообщение с указанием подписать вас на такой-то рефлектор (дискуссию, конференцию, etc.), и вы начинаете получать копии сообщений, которые туда посылают участники обсуждения. Рефлектор почты просто по получении электронных писем рассылает их копии всем подписчикам.

E-mail дает возможность использования в асинхронном режиме не только ftp, но и других служб, имеющих подобные сервера, предоставляющие такие услуги. Например, сетевых новостей, Archie, Whois.

Пересылать по e-mail можно и двоичные файлы, не только текстовые. В UNIX, например, для этого используется программы UUENCODE и UUDECODE.

При пользовании e-mail, из-за ее оперативности, может сложиться ощущение телефонной связи, но всегда следует осознавать, что это все же почта. Все сообщения письменные, поэтому почти документированы. Придерживайтесь этикета, принятого в обычной корреспонденции. В дополнение к этому помните, что e-mail не обладает той степенью приватности, как обычная почта, никогда не пишите в посланиях e-mail ничего, чего вам бы не хотелось увидеть выставленным на всеобщее обозрение. Анонимность также исключена: источник прослеживается без труда. Не стоит пользоваться техническими особенностями вашего терминала.

Электронная почта - один из важнейших информационных ресурсов Internet. Она является самым массовым средством электронных коммуникаций. Любой из пользователей Internet имеет свой почтовый ящик в сети. Если учесть, что через Internet можно принять или послать сообщения еще в два десятка международных компьютерных сетей, некоторые из которых не имеют on-line сервиса вовсе, то становится понятным, что почта предоставляет возможности в некотором смысле даже более широкие, чем просто информационный сервис Internet. Через почту можно получить доступ к информационным ресурсам других сетей. Хорошим примером может служить доступ к архивам сети BITNET - документам и телеконференциям, которые ведутся на серверах списков (LISTSERVER) BITNET.

Принципы организации

Электронная почта во многом похожа на обычную почтовую службу. Корреспонденция подготавливается пользователем на своем рабочем месте либо программой подготовки почты, либо просто обычным текстовым редактором. Обычно программа подготовки почты вызывает текстовый редактор, который пользователь предпочитает всем остальным программам этого типа. Затем пользователь должен вызвать программу отправки почты (программа подготовки почты вызывает программу отправки автоматически). Стандартной программой отправки является программа sendmail. Sendmail работает как почтовый курьер, который доставляет обычную почту в отделение связи для дальнейшей рассылки. В Unix-системах программа sendmail сама является отделением связи. Она сортирует почту и рассылает ее адресатам. Для пользователей персональных компьютеров, имеющих почтовые ящики на своих машинах и работающих с почтовыми серверами через коммутируемые телефонные линии, могут потребоваться дополнительные действия.

Для работы электронной почты в Internet разработан специальный протокол Simple Mail Transfer Protocol (SMTP), который является протоколом

прикладного уровня и использует транспортный протокол TCP. Однако, совместно с этим протоколом ранее активно использовался и Unix-Unix-CoPy (UUCP) протокол. UUCP хорошо подходит для использования коммутируемых телефонных линий связи с малыми скоростями обмена и ненадежным соединением. Большинство пользователей электронной почты Relcom на заре развития Интернета в России реально пользовались для доставки почты на узел именно этим протоколом. Разница между SMTP и UUCP заключается в том, что при использовании первого протокола sendmail пытается найти машину-получателя почты и установить с ней взаимодействие в режиме on-line для того, чтобы передать почту в ее почтовый ящик. В случае использования SMTP почта достигает почтового ящика получателя за считанные минуты и время получения сообщения зависит только от того, как часто получатель просматривает свой почтовый ящик. При использовании UUCP почта передается по принципу «stop-go», то есть почтовое сообщение передается по цепочке почтовых серверов от одной машины к другой пока не достигнет машины-получателя или не будет отвергнуто по причине отсутствия абонента-получателя. С одной стороны, UUCP позволяет доставлять почту по плохим телефонным каналам, так как не требуется поддерживать линию все время доставки от отправителя к получателю, а с другой стороны, бывает обидно получить возврат сообщения через сутки после его отправки из-за того, что допущена ошибка в имени пользователя. В целом же общие рекомендации таковы: если имеется возможность надежно работать в режиме on-line и это является нормой, то следует настраивать почту для работы по протоколу SMTP, если линии связи плохие или on-line используется чрезвычайно редко, то лучше использовать UUCP.

Основой любой почтовой службы является система адресов. Без точного адреса невозможно доставить почту адресату. В Internet принята система адресов, которая базируется на доменном адресе машины, подключенной к сети. Например, для пользователя shandarov машины с адресом mail.ed.tusur.ru

почтовый адрес будет выглядеть как:

shandarov@mail.ed.tusur.ru

Таким образом, адрес состоит из двух частей: идентификатора пользователя, который записывается перед знаком «коммерческого эй» - «@», и доменного адреса машины, который записывается после знака «@».

Программа рассылки почты Sendmail сама преобразует адреса формата Internet в адреса формата UUCP, если доставка сообщения осуществляется по этому протоколу.

Протокол SMTP

Simple Mail Transfer Protocol был разработан для обмена почтовыми сообщениями в сети Internet. SMTP не зависит от транспортной среды и может использоваться для доставки почты в сетях с протоколами, отличными от TCP/IP и X.25. Достигается это за счет концепции IPCE (InterProcess Communication Environment). IPCE позволяет взаимодействовать процессам, поддерживающим SMTP в интерактивном режиме, а не в режиме «STOP-GO».

Модель протокола. Взаимодействие в рамках SMTP строится по принципу двусторонней связи, которая устанавливается между отправителем и получателем почтового сообщения. При этом отправитель инициирует соединение и посылает запросы на обслуживание, а получатель на эти запросы отвечает. Фактически, отправитель выступает в роли клиента, а получатель - сервера.

Канал связи устанавливается непосредственно между отправителем и получателем сообщения. При таком взаимодействии почта достигает абонента в течение нескольких секунд после отправки.

Дисциплины работы и команды протокола. Обмен сообщениями и инструкциями в SMTP ведется в ASCII-кодах. В протоколе определено несколько видов взаимодействия между отправителем почтового сообщения и его получателем, которые здесь называются дисциплинами.

Наиболее распространенной дисциплиной является отправка почтового сообщения, которая начинается по команде MAIL, идентифицирующей отправителя:

```
MAIL FROM: shandarov@ed.tusur.ru
```

Следующей командой определяется адрес получателя:

```
RCPT TO: shand@ms.tusur.ru
```

После того, как определен отправитель и получатель почтового сообщения, можно отправлять последнее:

```
DATA
```

Команда DATA вводится без параметров и идентифицирует начало ввода почтового сообщения. Сообщение вводится до тех пор, пока не будет введена строка с точкой в первой позиции. Согласно стандарту почтового сообщения RFC822 отправитель передает заголовок и тело сообщения, которые разделены пустой строкой. Сам протокол SMTP не накладывает каких-либо ограничений на информацию, которая заключена между командой DATA и «.» в первой позиции последней строки. Приведем пример обмена сообщениями при дисциплине отправки почты:

```
S: MAIL FROM:
R: 250 Ok
S: RCPT TO:
R: 250 Ok
S: DATA
R: 354 Start mail input; end with .
S: Это текст почтового сообщения
S: .
R: 250
```

Другой дисциплиной, определенной в протоколе SMTP является перенаправление почтового сообщения (forwarding). Если получатель не найден, но известно его местоположение, то сервер может выдать сообщение:

```
R: 251 User not local;
will forward to
```

Если сервер может сделать только предположение о дальнейшей рассылке, то ответ будет несколько иным:

```
R: 551 User not local;
please try
```

В список дисциплин, разрешенных протоколом SMTP входит кроме отправки почты еще и прямая рассылка сообщений. В этом случае сообщение будет отправляться не в почтовый ящик, а непосредственно на терминал пользователя, если пользователь в данный момент находится за своим терминалом. Прямая рассылка осуществляется по команде SEND, которая имеет такой же синтаксис, как и команда MAIL. Кроме SEND прямую рассылку осуществляют SOML (Send or Mail) и SAML (Send and Mail). Назначение этих

команд легко понять из их названия.

Для инициализации канала обмена почтой и его закрытия используются команды HELO и QUIT соответственно. Первой командой сеанса должна быть команда HELO.

Протокол допускает рассылку почтовых сообщений в режиме оповещения. Для этой цели отправитель в адресе получателя может указать несколько пользователей или групповой адрес. Обычно, программное обеспечение SMTP выбирает эту информацию из заголовка почтового сообщения и на ее основе формирует параметры команд протокола.

Если сообщение по какой-либо причине не может быть разослано, то получатель формирует сообщение о неразосланном сообщении:

```
S: MAIL FROM:<>
R:  250 Ok
S: RCPT TO: <@host.domain:JOE@host.domain>
R: 250 Ok
S: DATA
R: 354 send the mail data, end with .
S: Date 23 Oct 95 11:23:30
S: From: SMTP@remote.domain
S: To:
S:
S: Undelivered message.  Your message lost. 550 No such user.
S: .
```

При использовании доменных имен следует использовать канонические

имена, так как некоторые системы не могут определить синоним по базе данных named.

Кроме выше перечисленных дисциплин протокол позволяет отправителю и получателю меняться ролями друг с другом. Происходит это по команде TURN.

Для отладки или проверки соединения по SMTP можно использовать telnet. Для этого вслед за адресом машины следует ввести номер порта:

```
/users/local>telnet
mail.ed.tusur.ru 25
```

25 порт используется в Internet для обмена сообщениями по протоколу SMTP. В интерактивном режиме пользователь сам изображает клиента SMTP и может посмотреть реакцию удаленной машины на его действия.

Протокол POP3 (Post Office Protocol)

Протокол обмена почтовой информацией POP3 предназначен для разбора почты из почтовых ящиков пользователей на их рабочие места при помощи программ-клиентов. Если по протоколу SMTP пользователи отправляют корреспонденцию через Internet, то по протоколу POP3 пользователи получают корреспонденцию из своих почтовых ящиков на почтовом сервере в локальные файлы.

Формат почтового сообщения (RFC-822)

При обсуждении примеров отправки и получения почтовых сообщений уже упоминался формат почтового сообщения. Разберем его подробнее. Формат почтового сообщения Internet определен в документе RFC-822 (Standard for ARPA Internet Text Message). Это довольно большой документ, поэтому

рассмотрим формат сообщения на примерах. Почтовое сообщение состоит из трех частей: конверта, заголовка и тела сообщения. Пользователь видит только заголовок и тело сообщения. Конверт используется только программами доставки. Заголовок всегда находится перед телом сообщения и отделен от него пустой строкой. RFC-822 регламентирует содержание заголовка сообщения. Заголовок состоит из полей. Поля состоят из имени поля и содержания поля. Имя поля отделено от содержания символом «:». Минимально необходимыми являются поля Date, From, cc или To, например:

```
Date:     26 Aug 76 1429 EDT
From:     Jones@Registry.org
cc:
```

или

```
Date:    26 Aug 76 1429 EDT
From:    Jones@Registry.org
To:  Smith@Registry.org
```

Поле Date определяет дату отправки сообщения, поле From - отправителя, а поля cc и To - получателя(ей). Чаще заголовок содержит дополнительные поля:

```
Date:    26 Aug 76 1429 EDT
From:    George Jones
Sender:  Secy@SHOST
To:      Smith@Registry.org
Message-ID: <4231.629.XYzi-What@Registry.org>
```

В данном случае поле Sender указывает, что George Jones не является автором сообщения. Он только переслал сообщение, которое получил из

Secy@SHOST. Поле Message-ID содержит уникальный идентификатор сообщения и используется программами доставки почты. Следующее сообщение демонстрирует все возможные поля заголовка:

```
    Date:        27 Aug 76 0932
    From:        Ken Davis
    Subject:     Re: The Syntax in the RFC
    Sender:      KSecy@Other-host
    Reply-To:        Sam.Irving@Reg.Organization
    To:          George Jones
    cc:          Important folks:
                 Tom Softwood ,
                 "Sam Irving"@Other-Host;,
                 Standard Distribution:
                 /main/davis/people/standard@Other-Host
    Comment:     Sam is away on bisiness.
    In-Reply-To: , George`s message
    X-Special-action: This is a sample of user-defined
field-names.
    Message-ID:  <4331.629.XYzi-What@Other-Host
```

Поле Subject определяет тему сообщения, Reply-To - пользователя, которому отвечают, Comment - комментарий, In-Reply-To - показывает, что сообщение относится к типу «В ответ на Ваше сообщение, отвечающее на сообщение, отвечающее ...», X-Special-action - поле, определенное пользователем, которое не определено в стандарте.

Следует сказать, что формат сообщения постоянно дополняется и совершенствуется. В RFC-1327 введены дополнительные поля для совместимости с почтой X.400. Кроме этого, следует обратить внимание на

поля некоторых довольно часто встречающихся заголовков, которые не регламентированы в RFC-822. Так первое предложение заголовка, которое начинается со слова From, содержит UUCP-путь сообщения, по которому можно определить, через какие машины сообщение «пробиралось». Поле Received: содержит транзитные адреса почтовых серверов с датой и временем прохождения сообщения. Вся эта информация полезна при разборе трудностей с доставкой почты.

В заключение хотелось бы отметить, что возможности почты не ограничиваются только пересылкой корреспонденции. По почте можно получить доступ ко многим ресурсам Internet, которые имеют почтовых роботов, отвечающих на запросы страждущих. Поэтому имеет смысл более детально изучить программное обеспечение, поддерживающее e-mail. Время, затраченное на чтение документации и опыты, окупятся возможностью получения информации из информационных архивов сети.

Программное обеспечение почтового обмена

Согласно схеме почтового обмена взаимодействие между участниками этого обмена строится по классической схеме «клиент-сервер». При этом схему можно подразделить на несколько этапов. Первый - взаимодействие по протоколу SMTP между почтовым клиентом (Internet Mail, Thunderbird, Outlook и т.п.) и почтовым транспортным агентом (sendmail, smail, ntmail и т.п.), второй - взаимодействие между транспортными агентами в процессе доставки почты получателю, результатом которого является доставка почтового сообщения в почтовый ящик пользователя и третий - выборка сообщения из почтового ящика пользователя почтовым клиентом в почтовый ящик пользователя на машине пользователя по протоколу POP3 или IMAP.

Программа Sendmail

Основным средством рассылки почты в Internet является программа

sendmail. Она обеспечивает работу модульной системы рассылки, которая предназначена для получения и отправки корреспонденции, а также управления программами подготовки и просмотра почтовых сообщений. Sendmail позволяет организовать почтовую службу локальной сети и обмениваться почтой с другими серверами почтовых служб через специальные шлюзы. Sendmail может быть сконфигурирована для работы с различными почтовыми протоколами. Обычно это протоколы UUCP (Unix-Unix-CoPy) и SMTP (Simple Mail Transfer Protocol).

Sendmail работает как «отделение связи» обычной почтовой службы, которое принимает и пересылает почтовые сообщения. Sendmail может интерпретировать два типа почтовых адресов:

- почтовые адреса SMTP;
- почтовые адреса UUCP.

Первые являются стандартными адресами Internet и, фактически, являются стандартом де-факто. Именно этот адрес обычно указан на визитных карточках.

Sendmail можно настроить для поддержки:

- списка адресов-синонимов;
- списка адресов рассылки пользователя;
- автоматической рассылки почты через шлюзы;
- очередей сообщений для повторной рассылки почты в случае отказов при рассылке;
- работы в качестве SMTP-сервера;
- доступа к адресам машин через сервер доменных имен BIND;
- доступа к внешним серверам имен.

Принцип работы программы sendmail

Sendmail отправляет почту в два приема: сначала почтовые сообщения собираются в очереди, а затем отсылаются.

Каждое сообщение состоит из трех частей: конверта, заголовка и тела сообщения.

Конверт. Конверт состоит из адреса отправителя, адреса получателя и информации рассылки, которая используется программами подготовки, рассылки и получения почты. Конверт остается невидимым для отправителя и получателя почтового сообщения.

Заголовок. Заголовок состоит из стандартных текстовых строк, которые содержат адреса, информацию о рассылке и данные. Заголовок может быть частью подготовленного пользователем текстового файла, а может быть подготовлен и добавлен к телу сообщения программой подготовки почты. Данные из заголовка могут быть использованы для оформления конверта сообщения.

Тело сообщения. Первая пустая строка в файле почтового сообщения отделяет заголовок от тела сообщения. Все, что следует после этой строки, называется телом сообщения и передается по почте без изменений.

Sendmail может быть вызвана:

- программой подготовки сообщений для отправки уже подготовленных сообщений;
- программой получения почты для пересылки полученной из сети почты;
- непосредственно пользователем для отправки по почте файла или короткого сообщения;
- почтовым демоном, которым обычно является сама sendmail.

После того, как почта собрана, начинается ее рассылка. При этом

выполняются следующие действия:

- адреса отправителя и получателя преобразуются в формат сети-получателя почты;
- если необходимо, то в заголовок сообщения добавляются строки, позволяющие получателю отвечать на принятое сообщение (например: FROM: <адрес>);
- почта передается одной из программ рассылки почты.

Когда программа приема почты получает сообщение, она передает его программе sendmail для последующей рассылки. Если сообщение достигло машины адресата, то оно отправляется программой местной рассылки в почтовый ящик пользователя.

Первый этап рассылки - сбор сообщений. Sendmail получает почтовые сообщения из трех источников:

- командной строки или стандартного ввода;
- через SMTP-протокол (из сети);
- из очереди сообщений.

При получении сообщений из командной строки или стандартного ввода, sendmail вызывается пользователем с указанием адреса доставки сообщения. При этом выполняются следующие действия: определяется адрес отправителя, выбирается из командной строки адрес получателя и оба адреса преобразуются в соответствии с описанием файла конфигурации, определяется способ доставки сообщения, размещается заголовок в оперативной памяти для последующих преобразований, а тело сообщения размещается во временном файле для отправки без изменений.

При получении сообщений по протоколу SMTP, sendmail используется как программа клиента и сервера протокола. Протокол определен в RFC-821 и является основным для рассылки почты в Internet. В этом случае sendmail запускается как демон, который «слушает» порт TCP и в случае получения сообщения устанавливает соединение с удаленным клиентом SMTP. Как правило, таким клиентом является другая программа sendmail.

Программа подготовки почты на локальной машине также может использовать SMTP. Для этого sendmail открывает канал (pipe) межпроцессного обмена.

При получении сообщений из очереди используются временные файлы очередей. Эти очереди используются для хранения неразосланных сообщений. Сообщение хранится в двух файлах. В одном файле хранится тело сообщения, а в другом конверт и заголовок сообщения. Обычно sendmail опрашивает очереди через определенные администратором почтового сервера промежутки времени, на предмет наличия в них неразосланных сообщений.

Вторая стадия рассылки почты - рассылка сообщений.

Как только одним из описанных выше способов sendmail получила сообщение, делается попытка его отправить по адресу. Для этого sendmail определяет три параметра: программу рассылки, узел сети и получателя. Эта процедура производится по правилам, которые содержатся в файле конфигурации. Sendmail сохраняет одну копию тела сообщения во временном файле, а заголовок загружает в оперативную память. Для каждого сообщения программа доставки (рассылки) сообщений вызывается отдельно. Если сообщение должно быть доставлено на разные машины, то для каждой из машин также вызывается своя программа доставки. Некоторые программы могут обслуживать сразу несколько абонентов одной машины, если это невозможно, то для каждого абонента вызывается также своя программа

доставки. Рассматривают два типа рассылки: на удаленную машину и местную рассылку.

Рассылка на удаленную машину. Для вызова программы рассылки sendmail открывает pipe и запускает программу рассылки, командная строка которой находится в файле конфигурации. Sendmail записывает заголовок и тело сообщения в pipe. Если программа рассылки не использует протокол SMTP, то адрес получателя передается через pipe. Если используется SMTP, то открывается двунаправленный канал для интерактивного взаимодействия с удаленным сервером SMTP. Если в качестве транспортного протокола используется TCP, то sendmail не запускает внешнюю программу рассылки, а сама инициирует TCP-соединение с удаленным сервером SMTP.

Доставка местной почты. Если sendmail определяет, что адреса доставки местные, то происходит обращение к файлу адресных синонимов и производится преобразование адресов (расширение). Файл адресных синонимов можно использовать для перенаправления почты в файлы или для обработки местными программами. Пользователь может иметь и свой собственный файл адресных синонимов для управления рассылкой персональной почты. После преобразования адресов почта отправляется программе местной рассылки (например rmail).

Важным моментом при работе sendmail является алгоритм определения типа адресов. При использовании стандартного файла конфигурации применяются следующие правила: почта рассылается в соответствии с форматом адреса получателя, адреса при этом бывают местные, UUCP и SMTP.

Местные адреса имеют вид:

```
user
```

```
user@localhost
user@localhost.localdomain
user@alias
user@alias.localdomain
user@[local.host.internet.address]
localhost!user
localhost!localhost!user
user@localhost.uucp
```

Местный адрес - это адрес, который распознается как адрес машины, с которой осуществляется отправка почты.

Адреса UUCP имеют вид:

```
host!user
host!host!user
user@host.uucp
```

Если машина, с которой отправляется почта, имеет прямую линию связи по протоколу UUCP со следующей машиной (в адресе), то почта передается на эту машину, если такого соединения нет, то почта не рассылается и выдается сообщение об ошибке. Файл конфигурации должен содержать детальное описание маршрутов для пересылки сообщений на машины по протоколу UUCP.

Адреса SMTP - это адреса, описанные в стандарте RFC-822 или стандартные адреса Internet. Эти адреса имеют вид:

```
usr@host
usr@host.domain
```

```
<@host1,@host2,@host3:user@host4>
user@[remote.host`s.internet.address]
```

Почта с адресами SMTP рассылается по протоколу SMTP.

Если в системе для адресации используется Berkeley Internet Name Domain (BIND) сервер, то sendmail может определять адреса получателей, используя сервис BIND. Если BIND не используется, то sendmail сама определяет адреса.

При рассылке почты можно использовать и смешанную адресацию:

```
user%hostA@hostB - почта отправляется с машины hostB на машину hostA
user!hostA@hostB - почта отправляется с машины hostB на машину hostA
hostA!user%hostB - почта отправляется с hostA на hostB
```

Подводя итог обсуждению принципов работы sendmail, следует специально подчеркнуть тот факт, что почта реально рассылается двумя принципиально разными способами. При использовании протокола UUCP почта рассылается по принципу «stop-go», то есть сообщения передаются от машины к машине по указанному в адресе пути. Следует ясно представлять, если почта ушла с машины отправителя, то это не означает, что она поступит получателю. Промежуточная машина может вернуть почту назад, если не сможет разослать. Электронная почта действительно работает как система обычной почты, физически перемещая и храня сообщения на промежуточных почтовых станциях. При работе по протоколу SMTP почта реально отправляется только тогда, когда установлено интерактивное соединение с

программой-сервером на машине-получателе почты. При этом происходит обмен командами между клиентом и сервером протокола SMTP в режиме on-line. При смешанной адресации доставка почты происходит по смешанному сценарию. Как шла доставка и как маршрутизировалось сообщение можно определить из заголовка сообщения, которое вы получили.

Анализ типа адресов в программе sendmail - это самый главный процесс, так как по типу адреса получателя sendmail определяет каким способом сообщение будет разослано. Вызов программы доставки вмонтирован в правила преобразования адресов отправителя и получателя. При этом как только система решит, что дальнейшее преобразование адреса нецелесообразно, так сразу вызывается программа доставки. Наибольшее число сообщений об ошибках при рассылке сообщений связано как раз с определением адреса получателя. В этом процессе принимают участие, по крайней мере, два сервиса Internet: система рассылки почтовых сообщений и служба доменных имен. Sendmail постоянно обращается к службе доменных имен на предмет канонизации имен электронной почты сверяет эти имена с теми, которые закреплены за компьютером, на котором данная система установлена. Если имена совпадают, то осуществляется местная рассылка по адресам местной почты.

Протокол IMAP

Другим протоколом разбора почты является протокол IMAP (Interactive Mail Access Protocol), который по своим возможностям очень похож на POP3, но был разработан как более надежная альтернатива последнего и к тому же обладает более широкими возможностями по управлению процессом обмена с сервером.

Работа протокола осуществляется по 143 потру TCP. Главным отличием от POP является возможность поиска нужного сообщения и разбор заголовков сообщения.

Ниже приведен пример взаимодействия по протоколу IMAP

```
OK IMAP2 Server Ready
A001 LOGIN Fred Secret
A001 OK User Fred logged in
A002 SELECT INBOX
* FLAGS (Meeting Notice \Answered \Flagged \Deleted \Seen)
* 19 Exists
* 2 Recent
* A002 OK Select compete
A003 FETCH 1:19 ALL
* 1 Fetch ( .....
* 19 Fetch (....
A003 OK Fetch complete
A004 LOGOUT
* Bye IMAP2 server quitting
A004 OK Logout complete
```

Для поиска информации используются команды FIND с различными аргументами.

Спецификация MIME (Multipurpose Internet Mail Extension)

Стандарт MIME, или в нотации Internet RFC-1341, предназначен для описания тела почтового сообщения Internet. Предшественником MIME является стандарт почтового сообщения ARPA (RFC822). Стандарт RFC822 был разработан для обмена текстовыми сообщениями. С момента опубликования стандарта возможности аппаратных средств и телекоммуникаций ушли далеко вперед и стало ясно, что многие типы информации, которые широко используются в сети невозможно передать по почте без специальных ухищрений. Так в тело сообщения нельзя включить графику, аудио, видео и другие типы информации. RFC822 не дает возможностей для передачи даже текстовой информации, которую нельзя реализовать в 7-битовой кодировке

ASCII. Естественно, что при использовании RFC822 не может быть и речи о передаче размеченного текста для отображения его различными стилями. Ограничения RFC822 становятся еще более очевидными, когда речь заходит об обмене сообщениями в разных почтовых системах. Например, для приема/передачи сообщений из/в X.400, который позволяет иметь двоичные данные в теле сообщения, ограничения старого стандарта могут стать фатальными, так как не спасает старый испытанный способ кодировки информации процедурой uuencode, так как эти данные могут быть по-различному проинтерпретированы в X.400 и программе рассылки почты в Internet (mail-agent).

В некотором смысле стандарт MIME ортогонален стандарту RFC822. Если последний подробно описывает в заголовке почтового сообщения текстовое тело письма и механизм его рассылки, то MIME, главным образом, сориентирован на описание в заголовке письма структуры тела почтового сообщения и возможности составления письма из информационных единиц различных типов.

В стандарте зарезервировано несколько способов представления разнородной информации. Для этой цели используются специальные поля заголовка почтового сообщения:

- поле версии MIME, которое используется для идентификации сообщения, подготовленного в новом стандарте;
- поле описания типа информации в теле сообщения, которое позволяет обеспечить правильную интерпретации данных;
- поле типа кодировки информации в теле сообщения, указывающее на тип процедуры декодирования;
- два дополнительных поля, зарезервированных для более детального описания тела сообщения.

Стандарт MIME разработан как расширяемая спецификация, в которой подразумевается, что число типов данных будет расти по мере развития форм представления данных. При этом следует учитывать, что анархия типов (безграничное их увеличение) тоже не допустима. Каждый новый тип в обязательном порядке должен быть зарегистрирован в IANA (Internet Assigned Numbers Authority). Остановимся подробнее на форме и назначении полей, определяемых стандартом.

Поле версии MIME (MIME-Version)

Поле версии указывается в заголовке почтового сообщения и позволяет определить программе рассылки почты, что сообщение подготовлено в стандарте MIME. Формат поля выглядит как:

```
MIME-Version: 1.0
```

Поле версии указывается в общем заголовке почтового сообщения и относится ко всему сообщению целиком. Здесь уместно отметить, что в отличие от стандарта RFC822, стандарт MIME позволяет перемешивать поля заголовка сообщения с телом сообщения. Поэтому все поля делятся на два класса: общие поля заголовка, которые записываются в начале почтового сообщения и частные поля заголовка, которые относятся только к отдельным частям составного сообщения и записываются перед ними.

Поле типа содержания тела почтового сообщения (Content-Type)

Поле типа используется для описания типа данных, которые содержатся в теле почтового сообщения. Это поле сообщает программе чтения почты какого сорта преобразования необходимы для того, чтобы сообщение правильно проинтерпретировать. Эта же информация используется и программой рассылки при кодировании/декодировании почты. Стандарт MIME определяет

семь типов данных, которые можно передавать в теле письма: текст (text); смешанный тип (multipart); почтовое сообщение (message); графический образ (image); аудио информация (audio); фильм или видео (video); приложение (application). Общая форма записи поля выглядит как:

```
Content-Type:= type "/" subtype *[";" parameter]
type := "application" | "audio" | "image" | "message" |
"multipart" | "text" | "video"           | x-token
x-token := <Два символа "X-", за которыми без пробела
следует последовательность любых символов>
subtype := token
parameter:= attribute "=" value
attribute:= token
value := token / quoted-string
token := 1*<любой символ кроме пробела и управляющего
символа, или tspecials>
tspecials:=  "(" /")" / "<" / ">" / "@"  ; Обязательно
         /  "," / ";" / ":" / "\" / <">  ; должны быть,
         /  "/" / "[" / "]" / "?" / "."  ; заключены в
         /  "="                          ; кавычки.
```

Остановимся подробнее на каждом из типов, разрешенных стандартом MIME.

Text. Этот тип указывает на то, что в теле сообщения содержится текст. Основным подтипом типа «text» является «plain», что обозначает так называемый планарный текст. Понятие планарного текста появилось в связи с тем, что существует еще размеченный текст, то есть текст со встроенными в него символами управления отображением, и гипертекст, то есть текст, который можно просматривать не последовательно, а произвольно, следуя гипертекстовым ссылкам. Для обозначения размеченного текста используют

подтип «richtext», а для обозначения гипертекста подтип «html». Вообще говоря, «html» - это специальный вид размеченного текста, который используется для представления гипертекстовой информации в системе World Wide Web, которая получила в последнее время широкое распространение в Internet. Понятие размеченного текста требует более подробного обсуждения, так как его передача и интерпретация являются одной из причин появления стандарта MIME.

«Richtext» определяет текст со встроенными в него специальными управляющими последовательностями, которые в соответствии со стандартом языка разметки документов SGML называются тагами. Таги представляют из себя последовательность символов типа <строка-символов>. «Строка-символов» определяет управляющее действие. Таги делятся на таги начала элемента текста (<.....>) и таги конца элемента текста (""). В качестве примера такой разметки можно привести следующий фрагмент текста:

```
<bold>Now</bold> is the time for
<italic>all</italic> good men
 <smaller>(and <lt>women>)</smaller> to
<ignoreme></ignoreme> come
to the aid of their
<nl>
```

В этом фрагменте <bold> означает выделение "жирным" шрифтом, <italic> - курсив, <smaller> - мелкий шрифт, <lt> - знак "<", игнорирование обозначено как <ignoreme>, новая строка как <nl>.

Специальный тип разметки задается подтипом **«html»**. Это так называемый гипертекст. Разметка гипертекста строится по тому же принципу как и в тексте типа «richtext». Однако применяются таги, позволяющие описать гипертекстовые ссылки. К таким тагам относятся ".....", , . Таг " ..... определяет

следующий фрагмент текста, который будет просматриваться. При этом текст между тагом начала и тагом конца будет выделен в программе просмотра цветом или другим способом и используется как контекстная гипертекстовая ссылка. Таг задет встроенный в текст документа графический образ. В некотором смысле этот таг аналогичен "multipart", который разрешает комбинировать сообщение из нескольких фрагментов разного типа. Таг определяет "якорь", т.е. место внутри документа, на которое можно сослаться как на метку. В качестве примера такой разметки текста можно привести следующий фрагмент:

```
Это пример разметки документа в формате HTML.
        <H1>Это заголовок документа</H1>
        <P> - Это параграф.
        <A HREF="test.html#mark1">
Это пример гипертекстовой ссылки.</A>
        <IMG SRC="test.gif" ALIGN=Bottom>
Это встроенный image.
        <A NAME="mark1"></A>
Это "якорь" внутри текста документа.
```

"Multipart". Этот тип содержания тела почтового сообщения определяет смешанный документ. Смешанный документ может состоять из фрагментов данных разного типа. Данный тип имеет ряд подтипов.

Подтип "mixed" - задает сообщение, состоящее из нескольких фрагментов, которые разделены между собой границей, задаваемой в качестве параметра подтипа. Приведем простой пример:

```
From: Nathaniel Borenstein
    To: Ned Freed
```

```
        Subject: Sample message
        MIME-Version: 1.0
        Content-type: multipart/mixed; boundary="simple
boundary"

        This is the preamble.  It is to be ignored,
though it is a
    handy place for mail composers to include an
explanatory
    note to non-MIME compliant readers.
        --simple boundary

        This is implicitly typed plain ASCII text.
        It does NOT end with a linebreak.
        --simple boundary

        Content-type: text/plain; charset=us-ascii

        This is explicitly typed plain ASCII text.
        It DOES end with a linebreak.
        --simple boundary--

        This is the epilogue.  It is also to be ignored.
```

В данном примере поле "Content-Type" определяет подтип "mixed" и границу между фрагментами, как строку "--simple boundary--". В начале каждого фрагмента может быть задана своя строка с полем "Content-Type". Как видно из примера, существует два фрагмента, которые не отображаются: преамбула и эпилог, в которые можно поместить комментарии.

Другим подтипом может быть подтип "alternative". Данный подтип позволяет организовать вариабельный просмотр почтового сообщения в зависимости от типа программы просмотра. Приведем пример:

```
    From:  Nathaniel Borenstein
    To: Ned Freed
    Subject: Formatted text mail
    MIME-Version: 1.0
    Content-Type: multipart/alternative;
boundary=boundary42
    --boundary42

    1 фрагмент
    Content-Type: text/plain; charset=us-ascii

    plain text version of message goes here....
    --boundary42

    2 фрагмент
    Content-Type: text/richtext

.... richtext version of same message goes here ...
    --boundary42

    3 фрагмент
    Content-Type: text/x-whatever

.... fanciest formatted version of same  message  goes
```

```
here ...
        --boundary42--
```

В этом примере для работы с планарным текстом при использовании алфавитно-цифровых программ просмотра предназначен первый фрагмент текста. Для просмотра размеченного текста используется второй фрагмент, для специальной программы просмотра может быть подготовлен специальный вариант (фрагмент 3).

Подтип "digest" предназначен для многоцелевого почтового сообщения, когда различным частям хотят приписать более детальную информацию, чем просто тип:

```
From: Moderator-Address
MIME-Version: 1.0
Subject: Internet Digest, volume 42
Content-Type: multipart/digest;
    boundary="---- next message ----"

    ------ next message ----

From: someone-else
Subject: my opinion

...body goes here ...

    ------ next message ----

From: someone-else-again
Subject: my different opinion
```

```
... another body goes here...

    ------ next message ------
```

Приведенный пример показывает как можно воспользоваться подтипом "digest" для рассылки почты разным пользователям и по-разному поводу, используя поля "From:" и "Subject" в качестве частных заголовков.

Подтип "parallel" предназначен для составления такого почтового сообщения, части которого должны отображаться одновременно, что предполагает запуск сразу нескольких программ просмотра. Синтаксис такого сообщения аналогичен рассмотренным выше.

Тип "message". Данный тип предназначен для работы с обычными почтовыми сообщениями, которые однако не могут быть переданы по почте по разного рода причинам. Эти причины объясняются подтипами данного типа.

Подтип "partial" предназначен для передачи одного большого сообщения по частям для последующей автоматической сборки у получателя. Приведем пример передачи аудио сообщения разбитого на части:

```
X-Weird-Header-1: Foo
From: Bill@host.com
To: joe@otherhost.com
Subject: Audio mail
Message-ID: id1@host.com
MIME-Version: 1.0
Content-type: message/partial;
    id="ABC@host.com";
```

```
            number=1; total=2

    X-Weird-Header-1: Bar
    X-Weird-Header-2: Hello
    Message-ID: anotherid@foo.com
    Content-type: audio/basic
    Content-transfer-encoding: base64

    ... first half of encoded audio data goes here...
        and the second half might look something like
this:

    From: Bill@host.com
    To: joe@otherhost.com
    Subject: Audio mail
    MIME-Version: 1.0
    Message-ID: id2@host.com
    Content-type: message/partial;
        id="ABC@host.com"; number=2; total=2

    ... second half of encoded audio data goes here...
```

Атрибуты подтипа определяют идентификатор сообщения (id), номер порции (number) и общее число порций (total). Следует обратить внимание на то, что каждая часть имеет свое поле "Content-Type". Это означает, что все сообщение может состоять из частей разных типов.

Другим подтипом является "External-Body", который позволяет ссылаться на внешние, относительно сообщения, информационные источники. Этот подтип похож на гипертекстовую ссылку из типа "text". Приведем конкретный

пример:

```
From: Whomever
Subject: whatever
MIME-Version: 1.0
Message-ID: id1@host.com
Content-Type: multipart/alternative; boundary=42
--42
    Content-Type: message/external-body;
     name="BodyFormats.ps";
     site="thumper.bellcore.com";
     access-type=ANON-FTP;
     directory="pub";
     mode="image";
     expiration="Fri, 14 Jun 1991 19:13:14 -0400
(EDT)"
Content-type: application/postscript
--42
Content-Type: message/external-body;
     name="/u/nsb/writing/rfcs/RFC-XXXX.ps";
     site="thumper.bellcore.com";
     access-type=AFS
     expiration="Fri, 14 Jun 1991 19:13:14 -0400
(EDT)"
Content-type: application/postscript
--42
Content-Type: message/external-body;
     access-type=mail-server
     server="listserv@bogus.bitnet";
```

```
        expiration="Fri, 14 Jun 1991 19:13:14 -0400
(EDT)"
    Content-type: application/postscript
    get rfc-xxxx doc
    --42--
```

В данном примере приведено использование "External-Body" и "multipart/alternative". Все сообщение разбито на несколько фрагментов. В каждом из фрагментов находится ссылка на внешний файл. Реально тела почтового сообщения нет (границы программами просмотра не отображаются). Однако если программа просмотра способна работать с внешними протоколами, то можно ссылки разрешить автоматически, запуская соответствующий сервис.

Стандартным подтипом типа "message" является "rfc822". Данный подтип определяет сообщения стандарта RFC822.

Типы описания нетекстовой информации. Таких типов имеется четыре:

- «image» для описания графических образов. Наиболее часто используются файлы форматов GIF и JPEG.
- «audio» для описания аудио информации. Для воспроизведения сообщения данного типа требуется специальное оборудование.
- «video» для передачи фильмов. Наиболее популярным является формат MPEG.
- «application» для передачи данных любого другого формата, обычно используется для передачи двоичных данных для последующего промежуточного преобразования. Так если на машине стоит видео-карта с 512Kb памяти, а графика подготовлена в 256 цветах, то сначала ее следует преобразовать и здесь может помочь тип "application". Основной подтип

данного типа - "octet-stream", но существуют "ODA" и "Postscript".

Назначение данных типов ясно из названия - обозначение данных для последующей обработки как данных в форматах, определяемых подтипом.

Поле типа кодирования почтового сообщения (Content-Transfer-Encoding). Многие данные передаются по почте в их исходном виде. Это могут быть 7bit символы, 8bit символы, 64base символы и т.п. Однако при работе в разнородных почтовых средах необходимо определить механизм их представления в стандартном виде - US-ASCII. Для этого существуют процедуры кодирования такого сорта данных. Наиболее широко применяемая - uuencode. Для того, чтобы при получении данные были бы правильно распакованы и введено в стандарт поле "Content-Transfer-Encoding". Синтаксис этого поля следующий:

```
    Content-Transfer-Encoding:= "BASE64" / "QUOTED-
PRINTABLE" /
                                "8BIT"   / "7BIT" /
                                "BINARY" / x-token
```

Каждая из альтернатив применяется в своем подходящем случае. Альтернативы «8bit», «7bit», «BINARY» реально никакого преобразования не требуют, так как почта передается байтами и SMTP не делает различия между ними. Однако они введены для строгости описания типов. "BASE64" обычно используется в связке с типом «text/ISO-8859-1», «x-token» позволяет пользователю описать свою процедуру преобразования.

Дополнительные необязательные поля. Как уже говорилось ранее, стандарт определяет еще два дополнительных поля: "Content-ID" и "Content-Description". Первое поле определяет уникальный идентификатор содержания, а второе служит для комментария содержания. Ни то, ни другое программами

просмотра обычно не отображаются.

В заключении обсуждения стандарта MIME комплексный пример без комментариев:

```
MIME-Version: 1.0
From: Nathaniel Borenstein
Subject: A multipart example
Content-Type: multipart/mixed;
    boundary=unique-boundary-1

This is the preamble area of a multipart message.
Mail readers that understand multipart format
should ignore this preamble.
If you are reading this text, you might want to
consider changing to a mail reader that understands
how to properly display multipart messages.
--unique-boundary-1

...Some text appears here...
[Note that the preceding blank line means
no header fields were given and this is text,
with charset US ASCII.  It could have been
done with explicit typing as in the next part.]

--unique-boundary-1
Content-type: text/plain; charset=US-ASCII

This could have been part of the previous part,
```

```
but illustrates explicit versus implicit
typing of body parts.

--unique-boundary-1
Content-Type: multipart/parallel;
    boundary=unique-boundary-2

--unique-boundary-2
Content-Type: audio/basic
Content-Transfer-Encoding: base64

... base64-encoded 8000 Hz single-channel
u-law-format audio data goes here....

--unique-boundary-2

Content-Type: image/gif
Content-Transfer-Encoding: Base64

... base64-encoded image data goes here....

--unique-boundary-2--

--unique-boundary-1
Content-type: text/richtext

This is richtext.
Isn't it cool?
```

```
--unique-boundary-1
Content-Type: message/rfc822

From: (name in US-ASCII)
Subject: (subject in US-ASCII)
Content-Type: Text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: Quoted-printable

... Additional text in ISO-8859-1 goes here ...

--unique-boundary-1--
```

Подводя итоги обсуждения, еще раз следует отметить, что стандарт MIME позволяет расширить область применения электронной почты, обеспечить доступ к другим информационным ресурсам сети в стандартных форматах.

Файловые архивы Internet

В настоящее время, когда популярность World Wide Web достаточно велика, объем трафика передаваемого по сети Internet по протоколу FTP занимает тем не менее первое место, несколько опережая объем трафика по протоколу HTTP. В этом свете организация файловых архивов в рамках технологии TCP/IP является крайне актуальной задачей.

Архивы используют для решения разных задач, однако наиболее популярными в сети являются свободно доступные архивы или такие архивы, доступ к которым разрешен по анонимному идентификатору пользователя. Таким образом эти архивы можно использовать в качестве:

- коллекции свободно распространяемого программного обеспечения;
- коллекции программ для бета-тестирования;
- коллекции нормативных и регламентных документов;

- и т.п.

FTP-архив можно использовать и в качестве архива коммерческого программного обеспечения, которое используется в компании, только в этом случае такой архив не должен разрешать анонимного доступа к хранящимся в нем ресурсам.

Часто возможность авторизированного FTP-доступа используют и для обмена сообщениями, то есть в качестве средства коммуникации. Это происходит обычно в том случае, когда система электронной почты по тем или иным причинам не работает.

Доступ к архиву можно осуществлять не только из специализированной программы-клиента, но и из универсального броузера, например Netscape Communicator или Microsoft Internet Explorer, для поиска информации в FTP-архивах можно воспользоваться программой Archie.

При этом следует четко понимать, что Archie и FTP - это совершенно разные технологии. В большинстве случаев доступ к Archie-серверу пользователи осуществляют из Archie-клиента, который находится на той же машине, что и сервер, т.е. сначала пользователь по Telnet заходит как пользователь Archie, а потом использует программу-клиент (обычно она запускается в качестве оболочки) для доступа к Archie серверу.

Протокол FTP (File Transfer Protocol)

FTP (File Transfer Protocol или «Протокол Передачи Файлов») - один из старейших протоколов в Internet и входит в его стандарты. Обмен данными в FTP проходит по TCP-каналу. Построен обмен по технологии «клиент-сервер».

В FTP соединение инициируется интерпретатором протокола

пользователя. Управление обменом осуществляется по каналу управления в стандарте протокола TELNET. Команды FTP генерируются интерпретатором протокола пользователя и передаются на сервер. Ответы сервера отправляются пользователю также по каналу управления. В общем случае пользователь имеет возможность установить контакт с интерпретатором протокола сервера и отличными от интерпретатора пользователя средствами.

Команды FTP определяют параметры канала передачи данных и самого процесса передачи. Они также определяют и характер работы с удаленной и локальной файловыми системами.

Сессия управления инициализирует канал передачи данных. При организации канала передачи данных последовательность действий другая, отличная от организации канала управления. В этом случае сервер инициирует обмен данными в соответствии с параметрами, согласованными в сессии управления.

Канал данных устанавливается для того же host'a, что и канал управления, через который ведется настройка канала данных. Канал данных может быть использован как для приема, так и для передачи данных.

Возможна ситуация, когда данные могут передаваться на третью машину. В этом случае пользователь организует канал управления с двумя серверами и организует прямой канал данных между ними. Команды управления идут через пользователя, а данные напрямую между серверами.

Канал управления должен быть открыт при передаче данных между машинами. В случае его закрытия передача данных прекращается.

Режимы обмена данными

В протоколе большое внимание уделяется различным способам обмена данными между машинами различных архитектур. Действительно, чего только нет в Internet, от персоналок и Mac'ов до суперкомпьютеров. Все они имеют различную длину слова и многие различный порядок битов в слове. Кроме этого, различные файловые системы работают с разной организацией данных, которая выражается в понятии метода доступа.

В общем случае, с точки зрения FTP, обмен может быть поточный или блоковый, с кодировкой в промежуточные форматы или без нее, текстовый или двоичный. При текстовом обмене все данные преобразуются в ASCII и в этом виде передаются по сети. Исключение составляют только данные IBM mainframe, которые по умолчанию передаются в EBCDIC, если обе взаимодействующие машины IBM. Двоичные данные передаются последовательностью битов или подвергаются определенным преобразованиям в процессе сеанса управления. Обычно, при поточной передаче данных за одну сессию передается один файл данных, а при блоковом способе за одну сессию можно передать несколько файлов.

Описав в общих чертах протокол обмена, можно перейти к описанию средств обмена по протоколу FTP. Практически для любой платформы и операционной среды существуют как серверы, так и клиенты. Ниже описываются стандартные сервер и клиент Unix-подобных систем.

Программное обеспечение доступа к FTP-архивам

Для работы с Ftp-архивами необходимо следующее программное обеспечение: сервер, клиент и поисковая программа. Сервер обеспечивает доступ к ресурсам архива из любой точки сети, клиент обеспечивает доступ пользователя к любому архиву в сети, а поисковая система обеспечивает навигацию во всем множестве архивов сети.

В разных операционных системах эти компоненты Ftp-обмена изменяются как по форме, так и по возможностям, но некоторые общие принципы остаются, кроме этого, программы, ориентированные на интерфейс командной строки, по большей части остаются неизменными в разных операционных средах.

Сервер протокола - программа ftpd

Команда ftpd предназначена для обслуживания запросов на обмен информацией по протоколу FTP. Сервер обычно стартует в момент загрузки компьютера. Синтаксис запуска сервера следующий:

```
ftpd [-d] [-1] [-t timeout]
```

- d - опция отладки;
- 1 - опция автоматической идентификации пользователя;
- t - время пассивного ожидания команд пользователя.

Каждый сервер имеет свое описание команд, которое можно получить по команде help. Автоматическая идентификация пользователей осуществляется при помощи файла /etc/passwd. Пароль пользователя не должен быть пустым.

Существует специальный файл, в котором содержатся запрещенные пользователи, то есть те, кому обслуживание по протоколу FTP запрещено. Возможен вход в архив по идентификатору пользователя anonymous или ftp. В этом случае сервер принимает меры по ограничению доступа к ресурсам компьютера для данного пользователя. Обычно для таких пользователей создается специальная директория ftp, в которой размещают каталоги bin, etc и pub. В каталоге bin размещаются команды, разрешенные для использования, а в каталоге pub собственно сами файлы. Каталог etc закрыт для просмотра пользователем и в нем размещены файлы идентификации пользователей.

Программа обмена файлами - ftp

FTP - это интерфейс пользователя при обмене файлами по одноименному протоколу. Программа устанавливает канал управления с удаленным сервером и ожидает команд пользователя. Идентификатор удаленного сервера указывается либо аргументом программы, либо в команде интерфейса open.

Если команда ftp работает с пользователем и ожидает его команд, то на экране отображается приглашение «ftp>».

Синтаксис команды:

```
ftp [-v][-d][-i][-n][host]
```

- v - подавляет ответы сервера и статистику передачи данных;
- n - управляет режимом идентификации пользователя. Если указан этот ключ, то сначала проверяется файл .netrc;
- i - выключает подтверждения передачи файла при массовом копировании файлов;
- d - включает режим отладки;
- g - отключает прозрачность передачи имен.

В рамках данного курса нет возможности перечислить все команды ftp, поэтому остановимся только на самых необходимых.

Первой такой командой является команда open. По этой команде открывается сеанс работы с удаленным сервером:

```
ftp>open ed.tusur.ru
```

После выдачи такой команды последуют запросы идентификации пользователя. Зарегистрировать пользователя можно и по команде user:

```
ftp> user anonymous
```

В данном примере пользователь не имеет особых прав доступа на удаленном сервере и поэтому регистрируется как аноним. В ответ на запрос идентификации следует в этом случае ввести свой почтовый адрес. Обычно достаточно ввести что-то похожее на почтовый адрес для допуска к ресурсам архива, но бывают и дотошные серверы, которые проверяют наличие такого адреса, поэтому лучше никого не обманывать и честно регистрироваться.

Следующими по важности командами являются команды cd и ls (dir). Назначение этих команд достаточно прозрачно и понятно всем пользователям - навигация по дереву файловой системы и просмотр содержания каталогов. Здесь следует посоветовать пользоваться при просмотре каталогов командой ls с дополнительными параметрами:

```
ftp>ls -FC
```

В этом случае пользователь может получить многоколоночный отчет с указанием типов файлов. Однако не все серверы отрабатывают эту комбинацию.

Так как в процессе приема-передачи участвуют две машины, то кроме навигации в удаленной файловой системе нужна еще навигация в локальной файловой системе. Для этой цели служит команда lcd (локальная cd). Кроме этого пользователь может выдать и любую команду локальной оболочки, если предварит ее символом "!":

```
ftp> !pwd
```

По этой команде будет выдано имя текущей директории на локальной машине.

И, наконец, самыми важными являются команды приема/передачи данных get, put, mget, mput и bin. По командам get и put можно принять или передать один файл:

```
ftp> get README.TXT
```

Команды mget, mput предназначены для приема/передачи набора файлов:

```
ftp> mget *.gz
```

Из примера видно, что в последнем случае применяется маска "*". Обычно при передаче групп файлов для каждого файла запрашивается подтверждение. Для того, чтобы избежать этого перед приемом/передачей, следует выдать команду prompt. Последняя переключает режим запроса подтверждения и при повторном использовании этой команды состояние запроса подтверждения восстанавливается. Другой полезной командой является команда hash:

```
ftp> hash #
```

Символ "#" можно заменить на любой другой. При работе по медленным линиям или при передаче больших файлов после включения режима hash пользователь имеет возможность видеть процесс передачи данных (знак "#" выдается после передачи каждого блока). И последнее, на чем следует остановить внимание - это команда bin. После выдачи этой команды по умолчанию данные будут передаваться в режиме передачи двоичных данных.

Последнее чрезвычайно важно, так как при передаче в ASCII нельзя передать программы и архивированные данные. Часто бывает полезно включить режим bin и для символьных данных с произвольной длиной строки, например файлов postscript (*.ps), т.к. в ASCII режиме есть ограничение на длину строки (обычно 254 символа).

Для выхода из ftp следует выполнить команду quit.

Сервера World Wide Web

Распределенная информационная гипертекстовая система World Wide Web является одним из самых популярных, если не самым популярным, ресурсом Internet. Простота поддержки баз данных Web и проста использования программ доступа к ресурсам Web привели к тому, что скорость установки Web-серверов такова, что их количество удваивается каждые 62 дня. Интегрированные мультипротокольные интерфейсы World Wide Web объединяют в себе не только средство просмотра Web, но и доступ к FTP-архивам и средство работы с электронной почтой.

Практически, мультипротокольные программы типа Netscape Navigator, Internet Explorer, Opera и другие стали стандартным интерфейсом доступа в Сеть. Кроме этого, для разработки самих страниц Web не требуется какого-то изощренного программного обеспечения. Достаточно иметь обычный текстовый редактор и уже можно разрабатывать не только информационные страницы, но и стандартные формы ввода информации. Все это подвигло разработчиков программного обеспечения заговорить о технологии World Wide Web, как о технологии, способной удовлетворить множеству требований разнообразных задач, которые встречаются в организации информационной системы корпорации. После того, как Sun объявила о возможности использования в World Wide Web мобильных кодов Java, то последние сомнения о возможности организации корпоративного информационного сервиса на

основе технологии World Wide Web отпали, и вся концепция получила название Intranet.

Разберем историю проекта, архитектуру программного обеспечения и протоколы World Wide Web более подробно.

История развития, отцы-основатели, современное состояние

Что же предлагал Тим Бернерс-Ли в 1989 году и что из этого получилось? В документах «World Wide Web: Proposal for HyperText Project», направленных руководству CERN, он считал, что информационная система, построенная на принципах гипертекста, должна объединить все множество информационных ресурсов CERN, которое состояло из базы данных отчетов, компьютерной документации, списков почтовых адресов, информационной реферативной системы, наборов данных результатов экспериментов и т.п. Гипертекстовая технология должна была позволить легко «перепрыгивать» из одного документа в другой.

Проект делился на две фазы, или, как у нас принято говорить, очереди. Первая очередь (продолжительностью в три месяца) должна была показать жизнеспособность идеи проекта. В течении этого этапа работ предполагалось разработать программы-интерфейсы для работы в алфавитно-цифровом режиме и программу-интерфейс для Macintosh и NeXT, работающую в графическом режиме, сервер для доступа к ресурсам Usenet, сервер для доступа к информационно-поисковой системе CERN, гипертекстовый сервер и программу-шлюз между Internet и DECnet.

В последующие три месяца (вторая очередь) предполагалось разработать средства подготовки гипертекстовых документов, полноэкранную программу просмотра для VM/XA, X-Window-интерфейс и систему автоматической нотификации просматриваемых материалов.

Кроме программного обеспечения предполагалось разработать общий протокол обмена информацией в сети, метод отображения текста на экране компьютера, создать набор базовых документов, иллюстрирующих работу системы, который мог бы пополняться за счет документов пользователей, обеспечить поиск по ключевым словам в этом наборе документов.

Любопытно, что из проекта в обязательном порядке исключались всякие исследования, связанные с конвертированием информации из форматов каких-либо редакторов в форматы данных системы, возможностью работы с видео- и аудио-информацией, все работы, связанные с защитой информации от несанкционированного доступа.

На всю эту полугодовую работу автор просил 4-х разработчиков (software designers) и одного программиста, и для каждого из них отдельное рабочее место (компьютер того типа, для которого разработчик будет писать программное обеспечение). Кроме этого требовалось приобрести коммерческое программное обеспечение, которое было бы полезно при разработке системы (Guide, KMS, FrameMaker).

Как видно, запросы были невелики, и в октябре 1990 года проект стартовал. Уже в ноябре был реализован прототип системы для NeXT, к рождеству «задышал» line mode browser, разработке которого придавалось особое значение, так как он открывал доступ к системе через telnet, а в марте его можно было уже демонстрировать. Через год в Internet был установлен анонимный telnet для доступа в систему. Первое сообщение об WWW было послано в телеконференции: alt.hypertext, com.sys.next, comp.text.sgml и comp.mail. multimedia, в августе 1991 года.

По современным меркам результаты, которых достигли разработчики к

1991 году выглядят довольно скромно, если не вдаваться в суть работы и ограничиться только внешним ее проявлением. Сообщество Internet получило еще одну программу, работающую в режиме командной строки. Прошло еще целых полтора года до того момента, когда программа Mosaic, разработанная Марком Андресеном (Mark Andressen) из Национального Центра Суперкомпьютерных Приложений (NCSA), и построенная на принципах WWW, обеспечили бурный рост популярности «паутины» в Internet.

NCSA начала проект по разработке интерфейса в World Wide Web месяц спустя после объявления CERN. Одна из задач NCSA - это разработка доступных некоммерческих программ, с другой стороны NCSA изучает новые технологии на предмет их коммерческого применения в будущем. World Wide Web безусловно подходила под эти два параметра. Кроме того, спецификации WWW производили впечатление добротно выполненной академической работы с обзором литературы по данному вопросу, обилием ссылок и обоснованностью принятых решений. Мультипротокольный переносимый интерфейс в WWW, создание которого начала Группа Разработки Программного Обеспечения NCSA, был назван Mosaic. Пробная версия программы была закончена в первой половине 1993 года, а в августе 1993 была анонсирована альфа-версия для Internet.

Следует отметить, что сам проект Mosaic внес огромный вклад в развитие спецификаций World Wide Web, существенно обогатив различные компоненты системы. Разработчики Mosaic ввели в стандарты WWW большое количество новшеств. Агрессивная политика команды NCSA привела к тому, что многие программы-интерфейсы, разработанные в рамках ранних стандартов, постепенно стали отмирать, не выдержав конкуренции. Для самого NCSA это закончилось тем, что лидер команды, Марк Андресен, покинул в марте 1994 года NCSA и организовал коммерческую корпорацию Netscape. С этого момента начался новый этап борьбы, но теперь между старыми коллегами. Netscape

активно навязывает свои стандарты, что приводит к тому, что документы, подготовленные с расширениями Netscape неправильно отображаются Mosaic, а документы с расширенными возможностями NCSA могут вообще не отображаться Netscape.

Следует отметить, что проект NCSA преследовал большие цели, нежели просто программу-интерфейс в WWW. С самого начала Mosaic разрабатывалась как программа с возможностями доступа к ресурсам Internet посредством различных протоколов, в число которых входили FTP, telnet, NNTP, SMTP. Однако вначале предполагалось, что делаться это будет за счет вызова внешних, относительно Mosaic, программ. В настоящее время Netscape сам поддерживает, кроме перечисленных, протоколы доступа в Gopher и Wais. Последнее позволяет использовать Netscape, впрочем как и Mosaic, для работы вне рамок World Wide Web.

Mosaic на некоторое время затмила разработки CERN. Однако эта группа имела хорошо продуманную стратегию развития системы, которая включала в себя следующие основные моменты: разработка и поддержка стандартов спецификаций системы, разработка библиотеки свободно распространяемых мобильных кодов системы, полного комплекта средств, обеспечивающих разработку и реализацию компонентов системы на любом типе компьютера в сети, подготовка набора справочных и демонстрационных документов о состоянии сети и направлениях ее развития. Данная стратегия позволила распространять программное обеспечение, разработанное в рамках проекта в Internet, а наличие line mode broser'a позволила открыть возможности WWW для огромной аудитории пользователей алфавитно-цифровых устройств, подключенных в сеть. Некоторое время NCSA лидировала и по числу установок серверов, однако в настоящее время CERN обеспечил себе паритет и в этой области. Правда, и здесь не обошлось без "накладок". Так, форматы файлов конфигурации программы imagemap, обеспечивающей работу с графическими

гипертекстовыми ссылками, у этих двух серверов различны.

Другим показателем успешного развития работ является образование W3-консорциума. Консорциум образован после подписания соглашения между Масачусетским Технологическим Институтом (MIT, USA) и Национальным Институтом Информатики и Автоматики (INRA, France) с согласия CERN. Если не вдаваться в подробности, то смысл этого соглашения заключается в том, что все программное обеспечение аккумулируется в MIT, участники имеют право copyright на все разработанное программное обеспечение и спецификации. Программное обеспечение распространяется свободно. За представителем MIT закрепляется должность директора, а за представителем INRA - должность зам. директора. Взносы полноправных участников W3C составляют $50.000 в год, а ассоциированных членов - $5.000 в год, соглашение заключено на три года начиная с 1 октября 1994 года. Любопытно, что организации с годовым оборотом, превышающим $50 миллионов, обязаны регистрироваться как полноправные члены, и что консорциум надеется получать прибыль, превышающую $1,5 миллиона, так как предусмотрен порядок использования средств сверх этой суммы. Средства до этого предела используются на развитие системы и исследования.

Образование Netscape Corporation и W3C легко объяснимы с точки зрения роста популярности WWW. В марте 1993 года трафик World Wide Web составлял 0,1% от общего трафика сети NSF, сентябре 1993 года он уже составил 1,0% от общего трафика сети NSF. В октябре 1993 года количество зарегистрированных серверов WWW равнялось 500, а к июню 1994 года оно достигло 1500 и продолжало стремительно расти. В настоящий момент число web-серверов не поддается исчислению.

Следует отметить, что появление технологии WWW и ее бурный прогресс не одинок. Приблизительно в это же время появились и другие распределенные

информационные технологии в Internet. Это, в первую очередь, Gopher и Wais. Столь бурный рост этого сектора компьютерных технологий привел к появлению на свет очень интересного документа, подготовленного по заказу Комиссии Европейского Союза к ежегодной встрече руководителей Союза 24-25 июня 1994 года на Корфу. Документ прямо обращает внимание руководителей стран Союза на тот факт, что происходит бурный рост рынка информационных технологий, и если Союз не хочет в очередной раз оказаться на вторых ролях, то должен предпринять энергичные усилия по поддержке работ в этой области. Авторы доклада утверждают, что происходит очередная техническая революция, вызванная возможностями современных телекоммуникационных систем и компьютерных сетей. Авторы выделяют десять основных сфер применения новых технологий:

- работа посредством сети, то есть создание новых рабочих мест;
- обучение по сети;
- научные коммуникации;
- обычные услуги по сети;
- управление дорожным движением;
- управление воздушным движением;
- быстрое медицинское обслуживание;
- создание единой системы защиты прав потребителей и производителей информационных услуг;
- создание единой европейской административной сети;
- создание информационной сети общего пользования для всех граждан Союза.

В каком-то смысле учреждение W3C является ответом профессионалов на медлительность бюрократов из Комиссии ЕвроСоюза. Среди учредителей W3C один из авторов документа - Мартин Банжеманн (Martin Bangemann).

Следующим важным этапом развития технологии World Wide Web стало появление весной 1995 года языка программирования Java, анонсированного компанией Sun Microsystems. Если быть более точным, то прямое отношение к World Wide Web имеет не сам язык, а мобильные коды и возможность их интерпретации программами просмотра Web. Создав свой броузер (программу просмотра) HotJava, Sun смогла продемонстрировать, что идеология интерпретации языка разметки документов может быть расширена. В страницы теперь можно стало встраивать фрагменты программ, которые после передачи по сети активировались на компьютере пользователя, расширяя тем самым концепцию распределенных вычислений.

К этому времени кроме Java появились еще и языки управления сценариями просмотра документов, самым известным из которых стал JavaScript. Тем самым, к середине 1996 года технология World Wide Web превратилась в полноценную гипертекстовую технологию, которая стала позволять решать большинство из тех задач, до которых доросли локальные гипертекстовые системы.

Учитывая все сказанное выше, попытаемся подробно остановиться на особенностях World Wide Web и отдельных ее компонентах, спецификациях и способах наращивания системы за счет внешнего программного обеспечения, существующем программном обеспечении и особенностях его функционирования на различных компьютерных платформах. Этим вопросам и будут посвящены следующие несколько разделов.

Понятие гипертекста

В предыдущем разделе речь шла об истории и основных вехах развития World Wide Web. В последнее время часто приходится слышать, что WWW - это очень просто. Однако, за этой кажущейся простотой скрывается хорошо продуманная сложная система. При этом следует заметить, что система бурно

развивается. Для того, чтобы более точно описать это развитие, наши англоязычные коллеги используют эпитет "dramatic". Познакомимся более подробно с WWW.

В 1989 году, когда Т.Бернерс-Ли предложил свою систему, в мире информационных технологий наблюдался повышенный интерес к новому и модному в то время направлению - гипертекстовым системам. Сама идея, но не термин, была введена В.Бушем (Vannevar Bush) в 1945 году в предложениях по созданию электромеханической информационной системы Memex. Несмотря на то, что Буш был советником по науке президента Рузвельта, идея не была реализована. В 1965 году Т.Нельсон (Ted Nelson) ввел в обращение сам термин "гипертекст", развил и даже реализовал некоторые идеи, связанные с работой с "нелинейными" текстами. В 1968 году изобретатель манипулятора "мышь" Д.Енжильбард (Doug Engelbart) продемонстрировал работу с системой, имеющей типичный гипертекстовый интерфейс, и, что интересно, проведена эта демонстрация была с использованием системы телекоммуникаций. Однако внятно описать свою систему он не смог. В 1975 году идея гипертекста нашла воплощение в информационной системе внутреннего распорядка атомного авианосца "Карл Винстон", которая получила название ZOG. В коммерческом варианте система известна как KMS. Работы в этом направлении продолжались и, время от времени, появлялись реализации типа HyperCard фирмы Apple или HyperNode фирмы Xerox. В 1987 была проведена первая специализированная конференция Hypertext'87, материалам которой был посвящен специальный выпуск журнала "Communication ACM".

Идея гипертекстовой информационной системы состоит в том, что пользователь имеет возможность просматривать документы (страницы текста) в том порядке, в котором ему это больше нравится, а не последовательно, как это принято при чтении книг. Поэтому Т.Нельсон и определил гипертекст как нелинейный текст. Достигается это путем создания специального механизма

связи различных страниц текста при помощи гипертекстовых ссылок, то есть у обычного текста есть ссылки типа "следующий-предыдущий", а у гипертекста можно построить еще сколь угодно много других ссылок. Любимыми примерами специалистов по гипертексту являются энциклопедии, Библия, системы типа "Help".

Простой, на первый взгляд, механизм построения ссылок оказывается довольно сложной задачей, так как можно построить статические ссылки, динамические ссылки, ассоциированные с документом в целом или только с отдельными его частями, то есть контекстные ссылки. Дальнейшее развитие этого подхода приводит к расширению понятия гипертекста за счет других информационных ресурсов, включая графику, аудио- и видео-информацию, до понятия гипермедиа. Тем, кто интересуется более подробно различными схемами и способами разработки гипертекстовых систем, стоит обратиться к специальной литературе.

Основные компоненты технологии World Wide Web

К 1989 году гипертекст представлял новую, многообещающую технологию, которая имела относительно большое число реализаций с одной стороны, а с другой стороны делались попытки построить формальные модели гипертекстовых систем, которые носили скорее описательный характер и были навеяны успехом реляционного подхода описания данных. Идея Т.Бернерс-Ли заключалась в том, чтобы применить гипертекстовую модель к информационным ресурсам, распределенным в сети, и сделать это максимально простым способом. Он заложил три краеугольных камня системы из четырех существующих ныне, разработав:

- язык гипертекстовой разметки документов HTML (HyperText Markup Language);
- универсальный способ адресации ресурсов в сети URL (Universal

Resource Locator);

- протокол обмена гипертекстовой информацией HTTP (HyperText Transfer Protocol).

Позже команда NCSA добавила к этим трем компонентам четвертый:

- универсальный интерфейс шлюзов CGI (Common Gateway Interface).

Java не включается в этот список намеренно, т.к. область применения этого языка гораздо шире чем простое "оживление" World Wide Web.

Идея HTML - пример чрезвычайно удачного решения проблемы построения гипертекстовой системы при помощи специального средства управления отображением. На разработку языка гипертекстовой разметки существенное влияние оказали два фактора: исследования в области интерфейсов гипертекстовых систем и желание обеспечить простой и быстрый способ создания гипертекстовой базы данных, распределенной на сети.

В 1989 году активно обсуждалась проблема интерфейса гипертекстовых систем, т.е. способов отображения гипертекстовой информации и навигации в гипертекстовой сети. Значение гипертекстовой технологии сравнивали со значением книгопечатания. Утверждалось, что лист бумаги и компьютерные средства отображения/воспроизведения серьезно отличаются друг от друга, и поэтому форма представления информации тоже должна отличаться. Наиболее эффективной формой организации гипертекста были признаны контекстные гипертекстовые ссылки, а кроме того, было признано деление на ссылки, ассоциированные со всем документом в целом и отдельными его частями.

Самым простым способом создания любого документа является его набивка в текстовом редакторе. Опыт создания хорошо размеченных для

последующего отображения документов в CERN'e был - трудно найти физика, который не пользовался бы системой TeX или LaTeX. Кроме того к тому времени существовал стандарт языка разметки - Standard Generalised Markup Language (SGML).

Следует также принять во внимание, что согласно своим предложениям Т.Бернерс-Ли предполагал объединить в единую систему имеющиеся информационные ресурсы CERN, и первыми демонстрационными системами должны были стать системы для NeXT и VAX/VMS.

Обычно гипертекстовые системы имеют специальные программные средства построения гипертекстовых связей. Сами гипертекстовые ссылки хранятся в специальных форматах или даже составляют специальные файлы. Такой подход хорош для локальной системы, но не для распределенной на множестве различных компьютерных платформ. В HTML гипертекстовые ссылки встроены в тело документа и хранятся как его часть. Часто в системах применяют специальные форматы хранения данных для повышения эффективности доступа. В WWW документы - это обычные ASCII- файлы, которые можно подготовить в любом текстовом редакторе. Таким образом, проблема создания гипертекстовой базы данных была решена чрезвычайно просто.

В качестве базы для разработки языка гипертекстовой разметки был выбран SGML (Standard Generalised Markup Language). Следуя академическим традициям, Бернерс-Ли описал HTML в терминах SGML (как описывают язык программирования в терминах формы Бекуса-Наура). Естественно, что в HTML были реализованы все разметки, связанные с выделением параграфов, шрифтов, стилей и прочего, так как реализация для NeXT подразумевала графический интерфейс. Важным компонентом языка стало описание встроенных и ассоциированных гипертекстовых ссылок, встроенной графики и обеспечение

возможности поиска по ключевым словам.

С момента разработки первой версии языка (HTML 1.0) прошло уже много лет. За это время произошло довольно серьезное развитие языка. Почти вдвое увеличилось число элементов разметки, оформление документов все больше приближается оформлению качественных печатных изданий, развиваются средства описания нетекстовых информационных ресурсов и способы взаимодействия с прикладным программным обеспечением. Совершенствуется механизм разработки типовых стилей. Фактически, в настоящее время HTML развивается в сторону создания стандартного языка разработки интерфейсов как локальных, так и распределенных систем.

Вторым краеугольным камнем WWW стала универсальная форма адресации информационных ресурсов. Universal Resource Identification (URI) представляет собой довольно стройную систему, учитывающую опыт адресации и идентификации e-mail, Gopher, WAIS, telnet, ftp и т.п. Но реально из всего, что описано в URI, для организации баз данных в WWW требуется только Universal Resource Locator (URL). Без наличия этой спецификации вся мощь HTML оказалась бы бесполезной. URL используется в гипертекстовых ссылках и обеспечивает доступ к распределенным ресурсам сети. В URL можно адресовать как другие гипертекстовые документы формата HTML, так и ресурсы e-mail, telnet, ftp, Gopher, WAIS, например. Различные интерфейсные программы по-разному осуществляют доступ к этим ресурсам. Одни, как например Netscape, сами способны поддерживать взаимодействие по протоколам, отличным от протокола HTTP, базового для WWW, другие, как например Chimera, вызывают для этой цели внешние программы. Однако, даже в первом случае, базовой формой представления отображаемой информации является HTML, а ссылки на другие ресурсы имеют форму URL. Следует отметить, что программы обработки электронной почты в формате MIME также имеют возможность отображать документы, представленные в формате HTML.

Для этой цели в MIME зарезервирован тип "text/html".

Третьим в нашем списке стоит протокол обмена данными в World Wide Web - HTTP (Hyper-Text Transfer Protocol). Данный протокол предназначен для обмена гипертекстовыми документами и учитывает специфику такого обмена. Так в процессе взаимодействия, клиент может получить новый адрес ресурса на сети (relocation), запросить встроенную графику, принять и передать параметры и т. п. Управление в HTTP реализовано в виде ASCII-команд. Реально, разработчик гипертекстовой базы данных сталкивается с элементами протокола только при использовании внешних расчетных программ или при доступе к внешним, относительно WWW, информационным ресурсам, например базам данных.

Последняя составляющая технологии WWW - это уже плод работы группы NCSA - спецификация CGI (Common Gateway Interface). CGI была специально разработана для расширения возможностей WWW за счет подключения всевозможного внешнего программного обеспечения. Такой подход логично продолжал принцип публичности и простоты разработки и наращивания возможностей WWW. Если команда CERN предложила простой и быстрый способ разработки баз данных, то NCSA развила этот принцип на разработку программных средств. Надо заметить, что в общедоступной библиотеке CERN были модули, позволяющие программистам подключать свои программы к серверу HTTP, но это требовало использования этой библиотеки. Предложенный и описанный в CGI способ подключения не требовал дополнительных библиотек и буквально ошеломлял своей простотой. Сервер взаимодействовал с программами через стандартные потоки ввода/вывода, что упрощает программирование до предела. При реализации CGI чрезвычайно важное место заняли методы доступа, описанные в HTTP. И хотя реально используются только два из них (GET и POST), опыт развития HTML показывает, что сообщество WWW ждет развития и CGI по мере усложнения

задач, в которых будет использоваться WWW-технология.

Архитектура построения системы

От описания основных компонентов перейдем к архитектуре взаимодействия программного обеспечения в системе World Wide Web. WWW построена по хорошо известной схеме "клиент-сервер".

Программа-клиент выполняет функции интерфейса пользователя и обеспечивает доступ практически ко всем информационным ресурсам Internet. В этом смысле она выходит за обычные рамки работы клиента только с сервером определенного протокола, как это происходит в telnet, например. Отчасти, довольно широко распространенное мнение, что Mosaic или Netscape, которые безусловно являются WWW-клиентами, это просто графический интерфейс в Internet, является верным. Однако, как уже было отмечено, базовые компоненты WWW-технологии (HTML и URL) играют при доступе к другим ресурсам Mosaic не последнюю роль, и поэтому мультипротокольные клиенты должны быть отнесены именно к World Wide Web, а не к другим информационным технологиям Internet. Фактически, клиент - это интерпретатор HTML. И как типичный интерпретатор, клиент в зависимости от команд (разметки) выполняет различные функции. В круг этих функций входит не только размещение текста на экране, но и обмен информацией с сервером по мере анализа полученного HTML-текста, что наиболее наглядно происходит при отображении встроенных в текст графических образов. При анализе URL-спецификации или по командам сервера клиент запускает дополнительные внешние программы для работы с документами в форматах, отличных от HTML, например GIF, JPEG, MPEG, Postscript и т.п. Вообще говоря, для запуска клиентом программ независимо от типа документа была разработана программа Luncher, но в последнее время гораздо большее распространение получил механизм согласования запускаемых программ через MIME-типы.

Другую часть программного комплекса WWW составляет сервер протокола HTTP, базы данных документов в формате HTML, управляемые сервером, и программное обеспечение, разработанное в стандарте спецификации CGI. До самого последнего времени (до образования Netscape) реально использовалось два HTTP-сервера: сервер CERN и сервер NCSA. Но в настоящее время число базовых серверов расширилось. Появился очень неплохой сервер для MS-Windows и Apachie-сервер для Unix- платформ. Существуют и другие, но два последних можно выделить из соображений доступности использования. Сервер для Windows - это shareware, но без встроенного самоликвидатора, как в Netscape. Учитывая распространенность персоналок в нашей стране, такое программное обеспечение дает возможность попробовать, что такое WWW. Второй сервер - это ответ на угрозу коммерциализации. Netscape уже не распространяет свой сервер Netsite свободно и прошел слух, что NCSA-сервер также будет распространяться на коммерческой основе. В результате был разработан Apache, который по словам его авторов будет freeware, и реализует новые дополнения к протоколу HTTP, связанные с защитой от несанкционированного доступа, которые предложены группой по разработке этого протокола и реализуются практически во всех коммерческих серверах.

База данных HTML-документов - это часть файловой системы, которая содержит текстовые файлы в формате HTML и связанные с ними графику и другие ресурсы. Особое внимание хотелось бы обратить на документы, содержащие элементы экранных форм. Эти документы реально обеспечивают доступ к внешнему программному обеспечению.

Прикладное программное обеспечение, работающее с сервером, можно разделить на программы-шлюзы и прочие. Шлюзы - это программы, обеспечивающие взаимодействие сервера с серверами других протоколов, например ftp, или с распределенными на сети серверами Oracle. Прочие

программы - это программы, принимающие данные от сервера и выполняющие какие-либо действия: получение текущей даты, реализацию графических ссылок, доступ к локальным базам данных или просто расчеты.

Все, что было сказано до этого момента, можно отнести к классической схеме World Wide Web. В настоящее время следует говорить об изменении общей архитектуры.

Произошел возврат к модульной структуре сервера World Wide Web. Этот возврат был реализован в виде спецификации API. API - это спецификация разработки прикладных модулей, которые встраиваются в сервер, точнее редактируются совместно с модулями сервера. Применение во всех серверах многопотоковой технологии выполнения подзадач делает такой способ расширения возможностей сервера более экономичным с точки зрения ресурсов вычислительной установки, чем разработка CGI-скриптов.

В дополнение к HTML активно стал применяться еще один язык разметки - VRML (Virtual Reality Modeling Language). В данном случае речь идет об описании трехмерных сен и возможности "бродить" по этим мирам. При этом в VRML также, как и в HTML предусмотрены гипертекстовые ссылки, что позволяет создавать смешанные базы данных, где информационный архив, например, можно представить в виде книг в библиотеке, среди которых может путешествовать автор, выбирая нужную ему тематику и источник, которые затем представляются в формате документа HTML.

Java-applet'ы - это мобильные коды Java, ссылки на которые вмонтированы в тело документа. При доступе к такому документу программа просмотра пользователя предварительно анализирует документ на предмет наличия в нем такого типа ссылок, и, если они существуют, то подкачивает мобильные коды в свою память. Коды могут сразу выполняться по мере

размещения их на компьютере пользователя, но могут активироваться и при помощи специальных команд.

Как видно из рисунка, изменения коснулись и клиентской части технологии. В настоящее время происходит постепенный переход от простой классической архитектуры клиент-сервер к архитектуре с сервером приложений, в роли которого выступает программа-клиент. В частности, NCSA опубликовала спецификацию CCI (Common Client Interface) для разработки приложений для работы с сервисами World Wide Web через программу Mosaic.

Завершая обсуждение архитектуры World Wide Web хотелось бы еще раз подчеркнуть, что ее компоненты существуют практически для всех типов компьютерных платформ и свободно доступны в сети. Любой, кто имеет доступ в Internet, может создать свой WWW-сервер, или, по крайней мере, посмотреть информацию с других серверов.

Язык гипертекстовой разметки HTML

Язык гипертекстовой разметки HTML (HyperText Markup Language) был предложен Тимом Бернерсом-Ли в 1989 году в качестве одного из компонентов технологии разработки распределенной гипертекстовой системы World Wide Web.

Разработчики HTML пытались решить две задачи:

- дать дизайнерам гипертекстовых баз данных простое средство создания документов;
- сделать это средство достаточно мощным, чтобы отразить имевшиеся на тот момент представления об интерфейсе пользователя гипертекстовых баз данных.

Первая задача была решена за счет выбора таговой модели описания документа. Такая модель широко применяется в системах подготовки документов для печати. Примером такой системы является хорошо известный язык разметки научных документов TeX, предложенный Американским Математическим Обществом, и программы его интерпретации.

К моменту создания HTML существовал стандарт языка разметки печатных документов - SGML (Standard Generalized Markup Language), который и был взят в качестве основы HTML. Предполагалось, что такое решение поможет использовать существующее программное обеспечение для интерпретации нового языка. Однако, будучи доступным широкому кругу пользователей Internet, HTML зажил своей собственной жизнью. Вероятно, многие администраторы баз данных WWW и разработчики программного обеспечения для этой системы имеют довольно смутное представление о стандартном языке разметки SGML.

Вторым важным моментом, повлиявшим на судьбу HTML, стал выбор в качестве элемента гипертекстовой базы данных обычного текстового файла, который хранится средствами файловой системы операционной среды компьютера. Такой выбор был сделан под влиянием следующих факторов:

- такой файл можно создать в любом текстовом редакторе на любой аппаратной платформе в среде любой операционной системы.
- к моменту разработки HTML существовал американский стандарт для разработки сетевых информационных систем - Z39.50, в котором в качестве единицы хранения указывался простой текстовый файл в кодировке LATIN1, что соответствует US ASCII.

Таким образом, гипертекстовая база данных в концепции WWW - это набор текстовых файлов, написанных на языке HTML, который определяет

форму представления информации (разметка) и структуру связей этих файлов (гипертекстовые ссылки).

Такой подход предполагает наличие еще одной компоненты технологии - интерпретатора языка. В World Wide Web функции интерпретатора разделены между сервером гипертекстовой базы данных и интерфейсом пользователя.

Сервер, кроме доступа к документам и обработки гипертекстовых ссылок, осуществляет также препроцессорную обработку документов, в то время как интерфейс пользователя осуществляет интерпретацию конструкций языка, связанных с представлением информации.

К настоящему времени наиболее распространена третья версия языка - HTML 3.0. Если первая версия языка (HTML 1.0) была направлена на представление языка как такового, где описание его возможностей носило скорее рекомендательный характер, вторая версия языка (HTML 2.0) фиксировала практику использования конструкций языка, версия ++ (HTML++) представляла новые возможности, расширяя набор элементов HTML в сторону отображения научной информации и таблиц, а также улучшения стиля компоновки изображений и текста, то версия 3.0 призвана упорядочить все нововведения и согласовать их с существующей практикой. Кроме этого, в версии 3.0 снова делается попытка формализации интерфейса пользователя гипертекстовой распределенной системы.

Принципы построения и интерпретации HTML

Таговая модель описывает документ как совокупность элементов, каждый из которых окружен тагами. По своему значению таги близки к понятию скобок "begin/end" в универсальных языках программирования, которые задают области действия имен локальных переменных и т. п. Таги определяют область действия правил интерпретации текстовых элементов документа. Типичным

примером такого рода является таг стиля Italic, который определяет область отображения курсива.

Текст на языке HTML:

```
Текст следующий за словом "Italic" <I>отображается как
курсив</I>.
```

Текст отображаемый программой интерпретации:

Текст следующий за словом "Italic" *отображается как курсив*.

В приведенном выше примере элемент текста, который должен быть выделен курсивом, заключен между тагом начала стиля "Italic" - и тагом конца стиля - . Общая схема построения элемента текста в формате HTML может быть записана в следующем виде:

```
"элемент" := <"имя элемента" "список атрибутов">
содержание элемента
```

Конструкция перед содержанием элемента называется тагом начала элемента, а конструкция, расположенная после содержания элемента, - тагом конца элемента.

Структура гипертекстовой сети задается гипертекстовыми ссылками. Гипертекстовая ссылка - это адрес другого HTML документа, который тематически, логически или каким-либо другим способом связан с документом, в котором ссылка определена.

Для записи гипертекстовых ссылок в системе WWW была разработана

специальная форма, которая называется Universe Resource Locator. Типичным примером использования этой записи можно считать следующий пример:

```
Этот текст содержит <A
HREF="http://ed.tusur.ru/altai/index.html">гипертекстовую
ссылку</A>.
```

В приведенном выше примере элемент "A", который в HTML называют якорем (anchor), использует атрибут "HREF", который обозначает гипертекстовую ссылку (HyperText REFerence), для записи этой ссылки в форме URL. Данная ссылка указывает на документ с именем "index.html" в директории "altai" на сервере "ed.tusur.ru", доступ к которому осуществляется по протоколу "http".

Гипертекстовые ссылки в HTML делятся на два класса: контекстные гипертекстовые ссылки и общие. Контекстные ссылки вмонтированы в тело документа, как это было продемонстрировано в предыдущем примере, в то время как общие ссылки связаны со всем документом в целом и могут быть использованы при просмотре любого фрагмента документа. Оба класса ссылок присутствуют в стандарте языка с самого его рождения, однако, первоначально наибольшей популярностью пользовались контекстные ссылки. Эта популярность привела к тому, что механизм использования общих ссылок практически полностью "атрофировался". Однако по мере стандартизации интерфейса пользователя и стилей представления информации разработчики языка снова вернулись к общим ссылкам и стремятся приспособить их к задачам управления этим интерфейсом.

Структура HTML-документа позволяет использовать вложенные друг в друга элементы. Собственно, сам документ - это один большой элемент с именем "HTML":

```
<HTML> Содержание документа </HTML>
```

Сам элемент HTML или гипертекстовый документ состоит из двух частей: заголовка документа (HEAD) и тела документа (BODY):

```
<HTML>
<HEAD>
Содержание заголовка
</HEAD>
<BODY>
Содержание тела документа
</BODY>
</HTML>
```

Приведенная выше форма записи определяет классический HTML-документ. Введение в язык HTML фреймов определило еще один шаблон документа:

```
<HTML>
<!--
Author: HTMLed User
Date:   January 21, 1996
-->
<HEAD>
</HEAD>
<FRAMESET COLS="40%,*">
<NOFRAMES>
<BODY>
Sorry there are not a frame support in your browser.
```

```
</BODY>
</NOFRAMES>
<FRAMESET ROWS="120,*,60">
<FRAME SRC=banner.htm NAME=banner>
<FRAME SRC="www.htm"  NAME=content>
<FRAME SRC="bottom.htm" NAME=bottom>
</FRAMESET>
<FRAMESET ROWS="100%">
<FRAME SRC="www_hist.htm" NAME=info>
</FRAMESET>
</FRAMESET>
</HTML>
```

В данном примере представлен документ, который состоит из трех окон внутри рабочего окна программы просмотра, в каждое из которых загружается обычный документ.

Рассмотрим пример классического документа:

```
<HTML>
<!--
Author: Evgeny Shandarov
Date:   November 18, 2006
-->
<HEAD>
<TITLE>This is a Baner</TITLE>
</HEAD>
<BODY BACKGROUND=www_wall.jpg VLINK=0000FF LINK=FF0000>
<CENTER>
```

```
<TABLE>
<TR><TD><IMG SRC="interne0.jpg"></TD>
<TD CENTER>
<H3>Информационные системы</H3>
<I>Кафедра Электронных приборов ТУСУР</I>
</TD></TR>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

Все, что расположено между <HTML> и </HTML> - это документ. Содержание элемента HEAD определяет заголовок документа, который состоит из двух элементов: TITLE и BASE. Вслед за заголовком начинается тело документа, которое содержит в своих первых строках некоторую вводную информацию и содержание документа, оформленное в виде списка.

Каждый документ в системе World Wide Web имеет свое имя, которое указывается в элементе TITLE заголовка документа. Его мы видим в первой строке интерфейса.

Контейнер BODY открывает тело документа. В качестве фона в этом элементе определена картинка back.gif. Эта картинка - "back.gif" - задана частичной формой спецификации URL, которая не задает полного адреса ресурса в сети.

Затем мы определили таблицу, состоящую из двух ячеек. В одной ячейке картинка, в то время как в другой - текстовый фрагмент. Текст определен как заголовок третьего уровня, который должен отображаться стилем Italic.

Протокол обмена гипертекстовой информацией (HyperText Transfer Protocol, HTTP)

HTTP - это протокол прикладного уровня, разработанный для обмена гипертекстовой информацией в сети Internet. Протокол используется одной из популярнейших систем Сети - Word Wide Web - с 1990 года.

Реальная информационная система требует гораздо большего количества функций, чем просто поиск. HTTP позволяет реализовать в рамках обмена данными набор методов доступа, базирующихся на спецификации универсального идентификатора ресурсов (Universal Resource Identifier), применяемого в форме универсального локатора ресурсов (Universe Resource Locator) или универсального имени ресурса (Universal Resource Name). Сообщения по сети при использовании протокола HTTP передаются в формате, схожим с форматом почтового сообщения Internet (RFC-822) или с форматом сообщений MIME (Multiperposal Internet Mail Exchange). HTTP используется для взаимодействия программ-клиентов с программами-шлюзами, разрешающими доступ к ресурсам электронной почты Internet (SMTP), спискам новостей (NNTP), файловым архивам (FTP), системам Gopher и WAIS. Протокол разработан для доступа к этим ресурсам посредством промежуточных программ-серверов (proxy), которые позволяют передавать информацию между различными информационными службами без потерь. Протокол реализует принцип "запрос/ответ". Запрашивающая программа - клиент - инициирует взаимодействие с отвечающей программой - сервером, и посылает запрос, включающий в себя метод доступа, адрес URI, версию протокола, похожее по форме на MIME сообщение с модификаторами типа передаваемой информации, информацию клиента, и, возможно, тело сообщения клиента. Сервер отвечает строкой состояния, включающей версию протокола и код возврата, за которой следует сообщение в форме, похожей на MIME. Данное сообщение содержит информацию сервера, метаинформацию и тело сообщения. Понятно, что в принципе, одна и та же программа может выступать и в роли сервера и в роли клиента (так собственно и происходит при использовании proxy-серверов).

При работе в Internet для обслуживания HTTP-запросов используется 80 порт TCP/IP. Практика использования протокола такова, что клиент устанавливает соединение и ждет ответа сервера. После отправки ответа сервер инициирует разрыв соединения. Таким образом, при передаче сложных гипертекстовых страниц соединение может устанавливаться несколько раз. Остановимся более подробно на механизме взаимодействия и форме передаваемой информации.

Форма запроса клиента

Программа-клиент посылает после установления соединения запрос серверу. Этот запрос может быть в двух формах: в форме полного запроса и в форме простого запроса. Простой запрос содержит метод доступа и запрос ресурса. Например:

```
GET http://ed.tusur.ru/
```

В этой записи слово GET обозначает метод доступа GET, а http://ed.tusur.ru/ - это запрос ресурса. Клиенты, которые способны поддерживать версии протокола выше 0.9 должны пользоваться полной формой запроса. При использовании полной формы в запросе указываются строка запроса, несколько заголовков (заголовок запроса или общий заголовок) и, возможно, тело обозначения ресурса. В форме Бекуса-Наура общий вид полного запроса выглядит так:

```
<Полный запрос> := <Строка Запроса> (<Общий заголовок>|
<Заголовок запроса>|<Заголовок обозначения
ресурса>)<символ новой строки>[<тело ресурса>]
```

Квадратные скобки здесь обозначают необязательные элементы заголовка. Строка запроса - это, практически, простой запрос ресурса. Отличие состоит в

том, что в строке запроса можно указывать различные методы доступа и за запросом ресурса следует указывать версию протокола. Например, для вызова внешней программы можно использовать следующую строку запроса:

```
POST http://ed.tusur.ru/cgi-bin/test HTTP/1.0
```

В данном случае используется метод POST и протокол версии 1.0.

Методы доступа

В настоящее время в практике World Wide Web реально используются только три метода доступа: POST, GET, HEAD.

GET - метод, позволяющий получить данные, заданные в форме URI в запросе ресурса. Если ссылаются на программу, то возвращается результат выполнения этой программы, но не текст программы. Дополнительные данные, которые надо передать для обработки, кодируются в запрос ресурса. Имеется разновидность метода GET - условный GET. При использовании этого метода сервер ответит на запрос только в том случае, если будут выполнены условия передачи. Это позволяет разгрузить сеть, избавив ее от передачи ненужной информации. Условие указывается в поле "if-Modified-Since" заголовка запроса. При использовании метода GET в поле тела ресурса возвращается собственно затребованная информация (текст HTML-документа, например).

HEAD - метод, аналогичный GET, но не возвращает тела ресурса. Используется для получения информации о ресурсе. Условного HEAD не существует. Данный метод используется для тестирования гипертекстовых ссылок.

POST - этот метод разработан для передачи большого объема информации на сервер. Им пользуются для аннотирования существующих ресурсов,

посылки почтовых сообщений, работы с формами интерфейсов к внешним базам данных и внешним исполняемым программам. В отличии от GET и HEAD, в POST передается тело ресурса, которое и является информацией из поля форм или других источников ввода. В первых версиях протокола были определены и другие методы доступа (DELETE, например), но они не нашли должного применения. Многие функции, которые возлагали на эти методы, можно успешно выполнять через POST.

Изменение числа методов доступа отражает практику использования HTTP. Однако, с исторической и методической точек зрения, первые версии протокола представляют несомненный интерес, особенно раздел, описывающий методы доступа. В версии 1993 года насчитывалось 13 различных методов доступа. Среди этих методов были такие, например, как:

- CHECKOUT - защита от несанкционированного доступа;
- PUT - замена содержания информационного ресурса;
- DELETE - удаление ресурса;
- LINK - создание гипертекстовой ссылки;
- UNLINK - удаление гипертекстовой ссылки;
- SPACEJUMP - переход по координатам;
- SEARCH - поиск.

Из этого списка видно, что протокол был действительно максимально ориентирован на работу с гипертекстовыми распределенными системами, причем не только с точки зрения потребителя, но и с точки зрения разработчика подобных систем. Однако, во-первых, как показывал опыт, практически не использовались методы доступа, связанные с изменением информации. Это объясняется прежде всего соображениями безопасности. Ни один администратор не позволит неизвестно кому менять информацию на его сервере. Во-вторых, методы SPACEJUMP и SEARCH были с успехом заменены

на функционально аналогичные CGI-скрипты. Из этого не следует, что их возрождение невозможно, например, в язык гипертекстовой разметки вернулись ссылки, общие для всего документа, но пока их реализация в протоколе отложена. В-третьих, не нашли практической реализации методы установления/удаления ссылок LINK и UNLINK. Но необходимость в них растет и связана она с реорганизацией сети. Многие информационные ресурсы меняют свои адреса, что вносит беспорядок в структуру сети, где и без этого начинающему пользователю трудно что-либо найти. Одним словом, вопрос о новых методах доступа все еще открыт, поэтому, видимо, спецификация HTTP еще не вышла на стадию RFC и остается в виде Internet Draft.

В обеих формах запроса важное место занимает форма запроса ресурса, которая кодируется в соответствии со спецификацией URI. Применительно к World Wide Web эта спецификация получила название URL. При обращении к серверу можно использовать как полную форму URL, так и упрощенную.

Полная форма содержит тип протокола доступа, адрес сервера ресурса, и адрес ресурса на сервере

Однако такой адрес реально нужен для работы через промежуточный сервер, так как тот может пересылать запросы. При обращении к первичному серверу клиент может опускать протокол и адрес, устанавливая взаимодействие с сервером по адресу, указанному в URL (в исходном документе), и порту 80, передавая только путь от корня сервера.

Здесь пока оставим обсуждение элементов запроса и обратимся к структуре ответа сервера. Дело в том, что элементы заголовков запроса и ответа одинаковые, и поэтому их имеет смысл обсудить после определения структуры ответа сервера.

Ответ сервера

Ответ сервера может быть, как и запрос, упрощенным или полным. При упрощенном ответе сервер возвращает только тело ресурса (например, текст HTML-документа). При полном ответе клиенту возвращается строка состояния (Status-Line), общий заголовок, заголовок ответа, заголовок ресурса и тело ресурса. В форме Бекуса-Наура полный ответ представляется следующим образом:

```
<Полный ответ> := <Строка состояния> (<Общий заголовок>|
<Заголовок ответа>|<Заголовок ресурса>) <символ новой
строки>[<тело ресурса>]
```

Строка состояния состоит из версии протокола, кода возврата и краткого описания этого кода. Например, она может выглядеть так:

```
HTTP/1.0 200 Success
```

Заголовок ответа сервера может состоять из адреса URI запрашиваемого ресурса, и/или наименования программы сервера, и/или кода идентификации для работы в защищенном режиме. Состав полей заголовка ресурса является общим и для запроса клиента и для ответа сервера, и состоит из разрешения на метод доступа, типа кодировки тела ресурса (содержания ресурса), длины тела ресурса, типа ресурса, время действия данной копии ресурса, времени последнего изменения ресурса и расширения заголовка.

Рассмотрим более подробно поля заголовка, обращая внимание на реальное применение каждого из них и возможность проявления ошибок при обработке этих полей разными программами (серверами и клиентами).

Если в заголовке ответа сервера указано предложение Location, то это

значит, что требуемый ресурс находится по другому адресу и его надо запросить заново. Такая процедура называется перенаправлением. Перенаправление в заголовке будет выглядеть как:

```
Location: http://ed.tusur.ru/risk/riskform.html
```

Имя и версия сервера указываются в поле Server. При использовании сервера Церн данное поле будет выглядеть как:

```
Server: CERN/3.0 libwww/2.17
```

В заголовке может встречаться и поле контроля доступа WWW-Authenticate, которое определяет способ доступа к закрытым ресурсам. Например, это поле может выглядеть как:

```
WWW-Athenticate: Basic realm="WallyWorld"
```

Кроме рассмотренных выше полей, в заголовке могут встретиться и другие поля, которые определяют содержание тела передаваемого ресурса. Эти поля относятся к заголовку ресурса, но в ответе сервера они встречаются вперемешку с общими полями. Поле Allow определяет разрешенные методы доступа к ресурсу:

```
Allow: GET,HEAD
```

Поле Content-Encoding определяет тип кодирования передаваемого ресурса:

```
Content-Encoding: x-gzip
```

В данном случае указывается, что ресурс является заархивированным

файлом в формате zip. Обычно ресурсы хранятся в виде, указанном в данном поле, и при их получении клиент должен обеспечить их преобразование в приемлемый для отображения вид. Это своего рода предварительная обработка данных, которая базируется обычно на MIME типах. В машинах, где недостаточно памяти на видео-адаптерах, используют предобработку для преобразования изображений в приемлемый для отображения вид. Так, например, для персональных компьютеров с 512 КБ памяти на видеоадаптере используют предобработку для преобразования 256-цветных картинок в 16-цветные. Если этого не делать, то можно наблюдать интересный эффект: в Unix-системах при работе с программами Chimera и Mosaic 256-цветные картинки удваиваются, то есть вместо одной на экране отображаются последовательно две картинки. Это связано с тем, что для 256 цветов нужно ровно в два раза больше памяти, чем для 16. Для того, чтобы избежать двойного отображения встроенных в текст картинок, их следует преобразовать.

Другим полем, которое проявляет себя при работе в сети, порождая ошибки отображения ранними версиями программ просмотра или ранним версиями программ серверов, является поле Content-Length. Поле Content-Length указывает размер (количество байтов) передаваемого ресурса. Это поле указывается как клиентом при работе по методу POST, так и сервером при ответе на запросы клиентов. При этом в ранних версиях программ-серверов может порождаться ошибка, вызванная возможностью вставки сервером некоторой информации в текст ресурса. Например, сервер NCSA (1.3) позволяет вставлять в текст HTML-документов фрагменты текста из других файлов при помощи выражения типа:

```
<!--#includes virtual="/include/commonheader.html" -->
```

В данном случае речь идет об общей заставке для всех документов некоторого раздела. На сервере в директории документов хранится файл одного

размера, но после выполнения подстановки размер файла увеличивается, однако, сервер сообщает клиенту старый размер файла. Некоторые программы-клиенты, как правило устаревшие, неправильно обрабатывают такой ответ и информация не отображается.

Поле Content-Type определяет тип информации, передаваемой сервером. Наиболее часто используются типы text/plain - простой тест и text/html - документ в формате HTML. Для сокращения трафика по сети существует несколько полей, связанных со временем передачи информации и периодичностью ее изменения на серверах. Поле Date определяет время отправки сообщения. Информация из данного поля сохраняется в файле статистики сервера и может быть использована для анализа доступа к ресурсам сервера из сети.

Поле Expires определяет время годности ресурса для использования. Если время использования вышло, то ресурс не должен передаваться. Точнее, его не следует передавать и принимать. О поле if-Modified-Sience уже упоминалось. Оно предназначено для того, чтобы не передавать имеющиеся у клиента копии ресурса, если не были произведены его изменения. Поле Pragma используется при передаче сообщений с сервера на сервер. Реально известно только одно значение данного поля: no-cache, которое запрещает запоминать в буфере данные для последующего использования.

Поле Referer используется для того, чтобы указать, из какого ресурса была осуществлена ссылка на ресурс. Данное поле - это мечта любого администратора базы данных сети. При помощи информации из этого поля можно определить, в каких WWW-страницах прописан конкретный сервер. От этого зависит количество обращений к серверу, "качество" пользователей, время отклика на информацию, размещаемую на сервере. При необходимости можно связаться с администратором этого сервера и уведомить его об изменениях на

вашем сервере.

Защита сервера от несанкционированного доступа

Отдельное место при обсуждении протокола занимают вопросы, связанные с обеспечением защиты ресурсов сервера от несанкционированного доступа. Как было отмечено в предыдущих разделах, первоначально разработка защищенных способов обмена данными в системе World Wide Web не предполагалась. Однако быстрое развитие популярности системы привело к тому, что многие коммерческие организации стали устанавливать серверы HTTP на свои машины. Кроме этого, конфиденциальной информации много и в научно-исследовательских государственных организациях. Таким образом, возникла необходимость в разработке механизмов защиты информации для системы WWW.

Проблема защиты информации на Internet - это отдельная большая тема. В данном разделе мы рассмотрим только обеспечение безопасности при использовании серверов HTTP.

При обсуждении этой проблемы полезно вспомнить схему WWW-технологии. Из анализа этой схемы видно, что в этой технологии имеется как минимум две потенциальные "дыры" .

Первая связана с чтением защищенных текстовых файлов. Для решения этой задачи имеется достаточно много традиционных механизмов, встроенных в операционные системы. Проблема возникает, если администратор системы решит использовать для размещения WWW-файлов и FTP-архива одно и тоже дисковое пространство. В этом случае защищенные WWW-файлы окажутся доступными для "анонимного" FTP-доступа. Многие серверы разрешают создавать в дереве поддерживаемых ими документов "домашние" страницы пользователей с помощью методов POST и GET. Это значит, что пользователи

могут изменять информацию на компьютере сервера. Данные, вводимые пользователем, передаются как тело ресурса при методе POST через стандартный ввод, а методе GET через переменные окружения. Естественно, что разрешение создания файлов на сервере протокола HTTP создает потенциальную опасность доступа к защищенной информации лиц, не имеющих права доступа к ней. Решается эта проблема путем создания специальных файлов прав пользователей сервера WWW.

Вторая возможность проникновения в компьютерную систему через сервер WWW связана с CGI-скриптами. CGI-скрипт - это программа, которую сервер HTTP может запускать для реализации механизмов, не предусмотренных в протоколе. Многие достаточно мощные информационные механизмы WWW реализованы посредством CGI-скриптов. К ним относятся: программы поиска по ключевым словам, программы реализации графических гипертекстовых ссылок - imagemap, программы сопряжения с системами управления базами данных и другие. Естественно, что при этом появляется возможность получить доступ к системным ресурсам. Обычно внешняя программа запускается с идентификатором пользователя, отличным от идентификатора сервера. Данный идентификатор указывается при конфигурировании сервера. Наиболее безопасным здесь является идентификатор пользователя nobody (65534). Основная опасность скриптов заключена в том, что данные в скрипт посылаются программой-клиентом. Для того, чтобы в качестве параметров не передавали "подозрительных" данных, многие серверы производят проверку параметров на наличие допустимых символов. Особенно опасны скрипты для тех, кто использует сервер на персональном компьютере с MS-Windows. В этом случае файловая система практически не защищена. Одной из характерных для скриптов проблем является размер входных данных. Многие "умные" серверы обрезают слишком большие входные потоки и тем самым защищают скрипты от "поломки". Кроме перечисленных выше опасностей, порождаемых природой сети и системы WWW, существует еще одна, связанная с такой экзотикой, как

мобильные коды. Мобильный код - это программа, которая может передаваться по сети для выполнения ее клиентом. Код встраивается в WWW-страницу при помощи тага - application. Например, Sun выпустила WWW-клиента HotJava, который позволяет интерпретировать язык Java. Существуют клиенты и для других языков, Safe-Tcl для Tcl, например. Главное назначение таких средств - реализация мультимедийных страниц и реализация работы в real-time. Опасность применения такого сорта страниц очевидна, так как повторяет способ распространения различного сорта вирусов. Однако пока речи о защите от такого сорта "взломов" не идет, видимо в силу довольно ограниченного применения данной возможности в сети.

Практически любой сервер имеет механизм назначения паролей и прав доступа для различных пользователей, который базируется на схеме идентификации протокола HTTP 1.0. Данная схема предполагает, что программа-клиент посылает серверу идентификатор пользователя и пароль. Понятно, что такой механизм не обеспечивает защиты передаваемой по сети информации, и она может стать легкой добычей злоумышленников.

Для того, чтобы этого не происходило, в рамках WWW ведется разработка других схем защиты. Они строятся на двух широко известных принципах: контроль доступа по IP-адресам и шифрация. Первый принцип реализован в программе типа "стена" (Firewall). Сервер разрешает обращаться к себе только с определенных IP-адресов и выполнять только определенные операции. Слабое место такого подхода с точки зрения WWW заключается в том, что обратиться могут через сервер-посредник, которому разрешен доступ к ресурсам защищенного первичного сервера. Поэтому применяют шифрование паролей и идентификаторов по аналогии с системой "Керберос". На принципе шифрования построен новый протокол SHTTP, который реализован в сервере Apache и в новых серверах CERN и NCSA. Однако реально широкого применения это программное обеспечение еще не нашло и находится в стадии

развития, а потому содержит достаточно большое количество ошибок.

Завершая обсуждение протокола HTTP и способов его реализации, нужно отметить, что в качестве широко доступного информационного ресурса WWW уже состоялась, следующий шаг - серьезные коммерческие применения. Но для этого необходимо еще внедрить в практику защищенные протоколы обмена, базирующиеся на HTTP. В последнее время появилось много новых программ, реализующих протокол HTTP. Это серверы и клиенты, написанные как для новых компьютерных платформ, так и для возможностей SHTTP. Реальную возможность попробовать свои силы в разработке различных WWW-приложений имеют и отечественные программисты.

Universal Resource Identifier - универсальный идентификатор. Спецификация универсального адреса информационного ресурса в сети

Из всех спецификаций World Wide Web только спецификация URI доведена до состояния RFC. За этим стандартом закреплен номер 1630. Выпущен этот документ в 1994 году и отражает состояние информационных ресурсов Internet на это время.

URI определяет способ записи (кодирования) адресов различных информационных ресурсов при обращении к ним из страниц WWW. Однако в последнее время данная спецификация стала встречаться и в почтовых сообщениях. При этом видимо предполагается, что пользователи почты должны использовать клиентов, поддерживающих этот формат сообщения (HTML). Реально, речь может идти о клиентах MIME.

Необходимость в URI была понятна разработчикам WWW с момента зарождения системы, так как предполагалось объединение в единую информационную среду средств, использующих различные способы идентификации информационных ресурсов. Первоначально это были FTP-

архивы, информационно-поисковая система Alise, и справочная система ЦЕРН. Однако Бернерс-Ли подошел к делу основательно и разработал спецификацию, которая включала в себя обращения к FTP, Gopher, WAIS, Usenet, E-mail, Prospero, Telnet, Whois, X.500 и, конечно, HTTP (WWW). В итоге была разработана универсальная спецификация, которая позволяет расширять список адресуемых ресурсов за счет появления новых схем.

Место применения URI - гипертекстовые ссылки, которые записываются в тагах <A HREF=URI> и <LINK HREF=URI>. Встраиваемые графические объекты также адресуются по спецификации URI в тагах <IMG SRC=URI> и <FIG SRC=URI>. Реализация URI для WWW называется URL (Uniform Resource Locator). Точнее, URL - это реализация схемы URI, отображенная на алгоритм доступа к ресурсам по сетевым протоколам. Существует еще и URN (Uniform Resource Name), которое отображает URI в пространство имен на сети. Появление URN связано с желанием адресовать части почтового сообщения MIME. Но здесь есть момент, который находится в стадии дебатов. Сообщение "живет" не более 5 дней. Если оно сохранено, то его можно превратить в другой информационный ресурс, например, WWW-страницу. Поэтому судьба URN еще не решена.

Принципы построения адреса WWW

В основу URI были заложены следующие принципы:

- Расширяемость - новые адресные схемы должны были легко вписываться в существующий синтаксис URI.
- Полнота - по возможности, любая из существовавших схем должна была описываться посредством URI.
- Читаемость - адрес должен был быть легко читаем пользователем, что вообще характерно для технологии WWW - документы вместе с ссылками

могут разрабатываться в обычном текстовом редакторе.

Расширяемость была достигнута за счет выбора определенного порядка интерпретации адресов, который базируется на понятии "адресная схема". Идентификатор схемы стоит перед остатком адреса, отделен от него двоеточием и определяет порядок интерпретации остатка.

Полнота и читаемость порождали коллизию, связанную с тем, что в некоторых схемах используется двоичная информация. Эта проблема была решена за счет формы представления такой информации. Символы, которые несут служебные функции, и двоичные данные отображаются в URI в шестнадцатеричном коде и предваряются символом "%".

Прежде, чем рассмотреть различные схемы представления адресов приведен пример простого адреса URI:

```
http://ed.tusur.ru/shandarov/index.html
```

Перед двоеточием стоит идентификатор схемы адреса - "http". Это имя отделено двоеточием от остатка URI, который называется "путь". В данном случае путь состоит из доменного адреса машины, на которой установлен сервер HTTP и пути от корня дерева сервера к файлу "index.html".

Кроме представленной выше полной записи URI, существует упрощенная. Она предполагает, что к моменту ее использования многие параметры адреса ресурса уже определены (протокол, адрес машины в сети, некоторые элементы пути). При таких предположениях автор гипертекстовых страниц может указывать только относительный адрес ресурса, то есть адрес относительно определенных базовых ресурсов.

Схемы адресации ресурсов Internet

В RFC-1630 рассмотрено 8 схем адресации ресурсов Internet и указаны две, синтаксис которых находится в стадии обсуждения.

Схема HTTP. Это основная схема для WWW. В схеме указывается ее идентификатор, адрес машины, TCP-порт, путь в директории сервера, поисковый критерий и метка. Приведем несколько примеров URI для схемы HTTP:

```
http://ed.tusur.ru/shandarov/manifest.html
```

Это наиболее распространенный вид URI, применяемый в документах WWW. Вслед за именем схемы (http) следует путь, состоящий из доменного адреса машины и полного адреса HTML-документа в дереве сервера HTTP.

В качестве адреса машины допустимо использование и IP-адреса:

```
http://144.206.160.40/risk/risk.html
```

Если сервер протокола HTTP запущен на другой, отличный от 80 порт TCP, то это отражается в адресе:

```
http://144.206.130.137:8080/altai/index.html
```

При указании адреса ресурса возможна ссылка на точку внутри файла HTML. Для этого вслед за именем документа может быть указана метка внутри документа:

```
http://ed.tusur.ru/altai/volume4.html#first
```

Символ "#" отделяет имя документа от имени метки. Другая возможность схемы HTTP - передача параметров. Первоначально предполагалось, что в качестве параметров будут передаваться ключевые слова, но, по мере развития механизма CGI-скриптов, в качестве параметров стала передаваться и другая информация.

```
http://ed.tusur.ru/isindex.html?keyword1+keyword2
```

В данном примере предполагается, что документ "isindex.html" - документ с возможностью поиска по ключевым словам. При этом в зависимости от поисковой машины (программы, реализующей поиск) знак "+" будет интерпретироваться либо как "AND", либо как "OR". Вообще говоря, "+" заменяет " " (пробел) и относится к классу неотображаемых символов. Если необходимо передать такой символ в строке параметров, то следует передавать в шестнадцатеричном виде его ASCII-код.

```
http://ed.tusur.ru/isindex.html?keyword1%20keyword2
```

В данном случае имеется один параметр, в котором два слова разделены пробелом. Символ "%" обозначает начало ASCII-кода, который продолжается до первого символа, отличного от цифры.

При использовании HTML Forms параметры передаются как поименованные поля:

```
http://ed.tusur.ru/isindex.html?field1=value1+field2=valu
e
```

Значения "field1" и "field2" - это имена полей, а "value1" и "value" - их значения. При этом приведенному выше URI может соответствовать следующая

HTML-форма:

```
<FORM ACTION=http://ed.tusur.ru/cgi-bin/test>
Введите значения полей:<BR>
Поле "field1":<INPUT NAME="filed1" VALUE="value1"><BR<
Поле "field2":<INPUT NAME="field2" VALUE="value2"><BR>
<HR>
</FORM>
```

Схема FTP. Данная схема позволяет адресовать файловые архивы FTP из программ-клиентов World Wide Web. При этом программа должна поддерживать протокол FTP. В данной схеме возможно указание не только имени схемы, адреса FTP-архива, но и идентификатора пользователя и даже его пароля. Наиболее часто данная схема используется для доступа к публичным архивам FTP:

```
ftp://ed.tusur.ru/pub/0index.txt
```

В данном случае записана ссылка на архив "ed.tusur.ru" с идентификатором "anonymous" или "ftp" (анонимный доступ). Если есть необходимость указать идентификатор пользователя и его пароль, то можно это сделать перед адресом машины:

```
ftp://nobody:password@ed.tusur.ru/users/local/pub
```

В данном случае эти параметры отделены от адреса машины символом "@", а друг от друга двоеточием. В некоторых системах можно указать и тип передаваемой информации, но данная возможность не стандартизирована. Стандарт рекомендует определять тип по характеру данных (текстовая информация - ASCII, двоичная - IMAGE). Следует также учитывать, что

употребление идентификатора пользователя и его пароля не рекомендовано, так как данные передаются незашифрованными и могут быть перехвачены. Реальная защита в WWW осуществляется другими средствами и построена на других принципах.

Схема MAILTO. Данная схема предназначена для отправки почты по стандарту RFC-822 (стандарт почтового сообщения). Общий вид схемы выглядит как:

```
mailto:shandarov@ed.tusur.ru
```

Схема TELNET. По этой схеме осуществляется доступ к ресурсу в режиме удаленного терминала. Обычно клиент вызывает дополнительную программу для работы по протоколу telnet. При использовании этой схемы необходимо указывать идентификатор пользователя, допускается использование пароля. Реально, доступ осуществляется к публичным ресурсам, и идентификатор и пароль являются общеизвестными, например, их можно узнать в базах данных Hytelnet.

```
telnet://guest:password@ed.tusur.ru
```

Схема FILE. WWW-технология используется как в сетевом, так и в локальном режимах. Для локального режима используют схему FILE.

```
file:///C|/text/html/index.htm
```

В данном примере приведено обращение к локальному документу на персональном компьютере с MS-DOS или MS-Windows. Следует заметить, что данная схема не может быть применена к CGI-скриптам. Очень часто, однако, пользователи пытаются применить file к скрипту, что является ошибкой. Любой

скрипт может быть запущен только сервером HTTP, так как ему надо передавать параметры и данные. Клиент запускает только программы просмотра на основе MIME-типов из заголовка сообщений сервера или по расширению файла.

Существует еще несколько схем. Эти схемы практически не используются или находятся в стадии разработки, поэтому останавливаться на них мы не будем.

Из приведенных выше примеров видно, что спецификация адресов ресурсов URI является довольно общей и позволяет проидентифицировать практически любой ресурс Internet. При этом число ресурсов может расширяться за счет создания новых схем. Они могут быть похожими на существующие, а могут и отличаться от них. Реальный механизм интерпретации идентификатора ресурса, опирающийся на URI, называется URL и пользователи WWW имеют дело именно с ним.

Common Gateway Interface - средство расширения возможностей технологии World Wide Web

Спецификация CGI была разработана в Центре Суперкомпьютерных Приложений Университета штата Иллинойс (NCSA). Работы над ней велись параллельно с Mosaic. С точки зрения общей архитектуры программного обеспечения World Wide Web, CGI определила все дальнейшее развитие системных средств. До появления этой спецификации все новые возможности реализовывались в виде модулей, включенных в библиотеку общих кодов ЦЕРН. Разработчики серверов должны были использовать эти коды для реализации программ или заменять их своими собственными аналогами. Это означало, что после компиляции сервера добавить в него новые возможности будет невозможно. CGI в корне изменила эту практику.

Главное назначение Common Gateway Interface - обеспечение

единообразного потока данных между сервером и прикладной программой, которая запускается из-под сервера. CGI определяет протокол обмена данными между сервером и программой. Для тех, кто знаком с протоколом HTTP, может показаться, что CGI - это просто подмножество этого протокола. Однако это не так. Во-первых, CGI определяет порядок взаимодействия сервера с прикладной программой, в котором сервер выступает инициирующей стороной, во-вторых, CGI определяет механизм реального обмена данными и управляющими командами в этом взаимодействии, что не определено в HTTP. Естественно, что такие понятия, как метод доступа, переменные заголовка, MIME, типы данных, заимствованы из HTTP и делают спецификацию прозрачной для тех, кто знаком с самим протоколом.

При описании различных программ, которые вызываются сервером HTTP и реализованы в стандарте CGI, используют следующую терминологию:

CGI-скрипт - программа, написанная в соответствии со спецификацией Common Gateway Interface. CGI-скрипты могут быть написаны на любом языке программирования (C, C++, PASCAL, FORTRAN и т.п.) или командном языке (shell, cshell, командный язык MS-DOS, Perl и т.п.).

Шлюз - это CGI-скрипт, который используется для обмена данными с другими информационными ресурсами Internet или приложениями-демонами. Обычная CGI-программа запускается сервером HTTP для выполнения некоторой работы, возвращает результаты серверу и завершает свое выполнение. Шлюз выполняется точно также, только, фактически, он инициирует взаимодействие в качестве клиента с третьей программой. Если эта третья программа является сервисом Internet, например, сервер Gopher, то шлюз становится клиентом Gopher, который посылает запрос по порту Gopher, а после получения ответа пересылает его серверу HTTP.

Аналогично происходит взаимодействие с серверами распределенных баз данных, например, Oracle.

Механизмы обмена данными

Собственно спецификация CGI описывает четыре набора механизмов обмена данными:

- через переменные окружения;
- через командную строку;
- через стандартный ввод;
- через стандартный вывод.

Переменные окружения. При запуске внешней программы сервер создает специфические переменные окружения, через которые передает приложению как служебную информацию, так и данные. Все переменные можно разделить на общие переменные окружения, которые генерируются при любой форме запроса, и запрос-ориентированные переменные.

К общим переменным окружения относятся:

- SERVER_SOFTWARE - определяет имя и версию сервера.
- SERVER_NAME - определяет доменное имя сервера.
- GATEWAY_INTERFACE - определяет версию интерфейса.

К запрос-ориентированным относятся:

- SERVER_PROTOCOL - протокол сервера. Вообще говоря, CGI разрабатывалась не только для применения в World Wide Web с протоколом HTTP, но и для других протоколов также, но широкое применение получила только в WWW.

- SERVER_PORT - определяет порт TCP, по которому осуществляется взаимодействие. По умолчанию для работы по HTTP используется 80 порт, но он может быть и переназначен при конфигурировании сервера.
- REQUEST_METHOD - определяет метод доступа к информационному ресурсу. Это важнейшая переменная в CGI. Разные методы доступа используют различные механизмы передачи данных. Данная переменная может принимать значения GET, POST, HEAD и т. п.
- PATH_INFO - передает программе путь, часть спецификации URL, в том виде, в котором она указана в клиентом. Реально это означает, что передается путь (адрес скрипта) в виде, указанном в HTML-документе.
- PATH_TRANSLATED - то же самое, что и PATH_INFO, но только после подстановки сервером определенных в его конфигурации вставок. Дело в том, что при конфигурировании сервера некоторым элементам (ветвям) дерева файловой системы можно назначить синонимы. Типичным примером такого сорта является назначение типа:

```
cgi-bin ------------> /usr/local/etc/httpd/cgi-bin
```

  В данном случае справа указано стандартное место CGI скриптов для сервера NCSA, а слева - его синоним. При получении скриптом test управления, в переменной окружения PATH_INFO будет значение:

  "/cgi-bin/test", а в PATH_TRANSLATED - "/usr/local/etc/httpd/cgi-bin/test".

- SCRIPT_NAME - определяет адрес скрипта так, как он указан клиентом. Если не указаны параметры, то значение этой переменной будут совпадать с PATH_INFO, но если переменные указаны, то все, что следует за знаком "?" будет отброшено.

PATH_INFO ----------> "/cgi-bin/search?nuclear+isotop"
SCRIPT+NAME --------> "/cgi-bin/search"

- QUERY_STRING - переменная определяет содержание запроса к скрипту. Чрезвычайно важна при использовании метода доступа GET. Возвращаясь к примеру с адресами скрипта укажем, что в QUERY_STRING помещается все, что записано после символа "?".

QUERY_STRING -------> "nuclear+isotop"

При этом никакого преобразования строки запроса сервером не производится. Все манипулирования с содержанием QUERY_STRING возложены на скрипт.

Следующий набор переменных связан с идентификацией пользователя и его машины:

- REMOTE_HOST - доменный адрес машины, с которой осуществляется запрос.
- REMOTE_ADDR - IP-адрес запрашивающей машины.
- AUTH_TYPE - тип идентификации пользователя. Используется в случае если скрипт защищен от несанкционированного использования.
- REMOTE_USER - используется для идентификации пользователя.
- REMOTE_IDENT - данная переменная порождается сервером, если он поддерживает идентификацию пользователя по протоколу RFC-931. Рекомендовано использование этой переменной для первоначального использования скрипта.

Следующие две переменные определяют тип и длину передаваемой информации от клиента к серверу.

- CONTENT_TYPE - определяет MIME-тип данных, передаваемых скрипту. Используя эту переменную можно одним скриптом обрабатывать различные форматы данных.
- CONTENT_LENGTH - определяет размер данных в байтах, которые передаются скрипту. Данная переменная чрезвычайно важна при обмене данными по методу POST, т.к. нет другого способа определить размер данных, которые надо прочитать со стандартного ввода.

Возможна передача и других переменных окружения. В этом случае перед именем указывается префикс "HTTP_". Отдельный случай представляют переменные, порожденные в заголовке HTML-документа в тагах META. Они передаются в заголовке сообщения и некоторые серверы могут порождать переменные окружения из этих полей заголовка.

Опции командной строки. Командная строка используется только при запросах типа ISIN-DEX. При HTML FORMS или любых других запросах неопределенного типа командная строка не используется. Если сервер определил, что к скрипту обращаются через ISINDEX-документ, то поисковый критерий выделяется из URL и преобразуется в параметры командной строки. При этом знаком разделения параметров является символ "+". Тип запроса определяется по наличию или отсутствию символа "=" в запросе. Если этот символ есть, то запрос не является запросом ISINDEX, если символа нет, то запрос принадлежит к типу ISIN-DEX. Параметры, выделенные из запроса, помещаются в массив параметров командной строки argv. При этом после из выделения происходит преобразование всех шестнадцатеричных символов в их ASCII-коды. Если число параметров превышает ограничения, установленные в командном языке, например в shell, то формирования командной строки не происходит и данные передаются только через QUERY_STRING. Вообще говоря, следует заранее подумать об объеме данных, передаваемом скрипту и

выбрать соответствующий метод доступа. Размер переменных окружения тоже ограничен, и если необходимо передавать много данных, то лучше сразу выбрать метод POST, т.е. передачу данных через стандартный ввод.

Формат стандартного ввода. Стандартный ввод используется при передаче данных в скрипт по методу POST. Объем передаваемых данных задается переменной окружения CONTENT_LENGTH, а тип данных - переменной CONTENT_TYPE. Если из HTML-формы надо передать запрос типа: a=b&b=c, то CONTENT_LENGTH=7, CONTENT_TYPE=application/x-www-form-urlencoded, а первым символом в стандартном вводе будет символ "a". Следует всегда помнить, что конец файла сервером в скрипт не передается, а поэтому завершать чтение следует по числу прочитанных символов. Позже мы разберем примеры скриптов и обсудим особенности их реализации в разных операционных системах.

Формат стандартного вывода. Стандартный вывод используется скриптом для возврата данных серверу. При этом вывод состоит из заголовка и собственно данных. Результат работы скрипта может передаваться клиенту без каких-либо преобразований со стороны сервера, если скрипт обеспечивает построение полного HTTP-заголовка, в противном случае сервер заголовок модифицирует в соответствии со спецификацией HTTP. Заголовок сообщения должен отделяться от тела сообщения пустой строкой. Обычно в скриптах указывают только три поля HTTP-заголовка:

```
Content-type, Location, Status.
```

Content-type указывается в том случае, когда скрипт сам генерирует документ "на лету" и возвращает его клиенту. В этом случае реального документа в файловой системе сервера не остается. При использовании такого сорта скриптов следует учитывать, что не все серверы и клиенты отрабатывают

так, как представляется разработчику скрипта. Так, при указании Content-type: text/html, некоторые клиенты не реализуют сканирования полученного текста на предмет наличия в нем встроенной графики. Обычно в Content-type указывают текстовые типы text/plain и text/html.

Location используется для переадресации. Иногда переадресация помогает преодолеть ограничения сервера или клиента на обработку встроенной графики или серверной предобработки. В этом случае скрипт создает файл на диске и указывает его адрес в Location. Сервер, таким образом, передает реально существующий файл. В последнее время серверы стали буферизовать возвращаемые клиентам данные, что приводит к решению вопросов, связанных с повторным запуском скриптов для встраивания графики и разгрузки компьютера с сервером HTTP.

Практика применения скриптов CGI

Применение скриптов широко практикуется в WWW. При их помощи, например, реализованы стеки графических гипертекстовых ссылок, встраивание даты в текст документов, встраивание ответов службы finger, доступ к базам данных и многое другое. Мы рассмотрим простейшие скрипты для распечатки параметров, передаваемых сервером, скрипты по обращению к shell, C скрипты, скрипты доступа к системе управления базами данных ingres и скрипт imagemap.

Простейшие скрипты и преобразование информации. Обсуждение начнем со скриптов, написанных на командном языке SHELL. Самый простой из них будет выглядеть как:

```
#!/bin/sh
echo Content-type: text/plain
echo
```

```
echo This is the result of script execution.
#The end of script
```

Первая строка определяет, что в качестве интерпретатора скрипта будет использован shell, вторая строка открывает заголовок сообщения, передаваемого скриптом серверу, и определяет тип передаваемой информации как обычный текст. Третья строка отделяет тело сообщения от его заголовка. В теле сообщения передается фраза из четвертой строки. Именно она и будет отображаться программой-интерфейсом пользователя.

В качестве следующего примера приведем скрипт, который отображает значения переменных окружения:

```
#!/bin/sh
echo Content-type: text/plain
echo
echo $REQUEST_METHOD
echo $QUERY_STRING
echo $CONTENT_TYPE
echo $CONTENT_LENGTH
#The end of script.
```

В данном скрипте пользователю будут возвращены значения указанных в строках команды echo переменных окружения.

Пользователю можно вернуть не только значения переменных окружения, но и результаты выполнения команд.

```
#!/bin/sh
echo Content-type: text/plain
echo
finger shandarov@ed.tusur.ru
```

```
#The end of script.
```

В результате выполнения этого скрипта пользователь получит информацию о пользователе shandarov с машины ed.tusur.ru.

При написании скриптов следует учитывать то, что сервер обычно стартует в момент, когда не все пути могут быть определены, поэтому при обращении к ресурсам следует указывать полные пути для этих ресурсов.

Общей проблемой, связанной с использованием скриптов, является проблема безопасности. Во-первых, скрипты обычно не пишут сами, как, например, скрипт imagemap, а заимствуют. Понятно, что чужая программа может содержать ошибки. Поэтому лучше пользоваться библиотеками проверенных скриптов, которые рекомендованы, например, World Wide Web Consortium. Во-вторых, если пользователю разрешено иметь свои страницы, то он может получить возможность выполнять свои скрипты на сервере, что тоже приводит к брешам в системе безопасности, особенно если это shell-скрипты. Существуют такие скрипты, которые требуют прав доступа к ресурсам машины. Эти права шире, чем права пользователя "nobody", например, при доступе к базам данных. Все эти моменты следует учитывать, как при написании скриптов, так и при разрешении использования различным группам пользователей.

Как правило, новые возможности WWW тестируются на скриптах, а затем, если эти возможности широко используются в практике, они включаются в стандарты различных компонентов системы и могут быть реализованы в новых возможностях серверов, как например imagemap и системы безопасности. Скрипты позволяют новым энтузиастам Сети приобщиться к сообществу и внести свою лепту в развитие системы.

5. Компоненты современных информационных систем

HyperText Transfer Protocol - протокол обмена WWW – серверов

HyperText Transfer Protocol (HTTP) - это протокол высокого уровня (а именно, уровня приложений), обеспечивающий необходимую скорость передачи данных, требующуюся для распределенных информационных систем гипермедиа. HTTP используется проектом World Wide Web с 1990 года.

Практические информационные системы требуют большего, чем примитивный поиск, модификация и аннотация данных. HTTP/1.0 предоставляет открытое множество методов, которые могут быть использованы для указания целей запроса. Они построены на дисциплине ссылок, где для указания ресурса, к которому должен быть применен данный метод, используется Универсальный Идентификатор Ресурсов (Universal Resource Identifier - URI), в виде местонахождения (URL) или имени (URN). Формат сообщений сходен с форматом Internet Mail или Multipurpose Internet Mail Extensions (MIME-Многоцелевое Расширение Почты Internet).

HTTP/1.0 используется также для коммуникаций между различными пользовательскими просмотрщиками и шлюзами, дающими гипермедиа доступ к существующим Internet протоколам, таким как SMTP, NNTP, FTP, Gopher и WAIS. HTTP/1.0 разработан, чтобы позволять таким шлюзам через proxy серверы, без какой-либо потери передавать данные с помощью упомянутых протоколов более ранних версий.

Общая Структура

HTTP основывается на парадигме запросов/ответов. Запрашивающая программа (обычно она называется клиент) устанавливает связь с обслуживающей программой-получателем (обычно называется сервер) и посылает запрос серверу в следующей форме: метод запроса, URI, версия

протокола, за которой следует MIME-подобное сообщение, содержащее управляющую информацию запроса, информацию о клиенте и, может быть, тело сообщения. Сервер отвечает сообщением, содержащим строку статуса (включая версию протокола и код статуса - успех или ошибка), за которой следует MIME-подобное сообщение, включающее в себя информацию о сервере, метаинформацию о содержании ответа, и, вероятно, само тело ответа. Следует отметить, что одна программа может быть одновременно и клиентом и сервером. Использование этих терминов в данном тексте относится только к роли, выполняемой программой в течение данного конкретного сеанса связи, а не к общим функциям программы.

В Internet коммуникации обычно основываются на TCP/IP протоколах. Для WWW номер порта по умолчанию - TCP 80, но также могут быть использованы и другие номера портов - это не исключает возможности использовать HTTP в качестве протокола верхнего уровня.

Для большинства приложений сеанс связи открывается клиентом для каждого запроса и закрывается сервером после окончания ответа на запрос. Тем не менее, это не является особенностью протокола. И клиент, и сервер должны иметь возможность закрывать сеанс связи, например, в результате какого-нибудь действия пользователя. В любом случае, разрыв связи, инициированный любой стороной, прерывает текущий запрос, независимо от его статуса.

HTTP запрос

Общие понятия

Запрос - это сообщение, посылаемое клиентом серверу.

Первая строка этого сообщения включает в себя метод, который должен быть применен к запрашиваемому ресурсу, идентификатор ресурса и используемую версию протокола. Для совместимости с протоколом HTTP/0.9,

существует два формата HTTP запроса:

```
 Запрос = Простой-Запрос | Полный-Запрос
 Простой-Запрос = "GET" SP Запрашиваемый-URI CRLF
 Полный-Запрос = Строка-Статус
         *(Общий-Заголовок    |    Заголовок-Запроса    |
Заголовок-Содержания ) CRLF
         [ Содержание-Запроса ]
```

Если HTTP/1.0 сервер получает Простой-Запрос, он должен отвечать Простым-Ответом HTTP/0.9. HTTP/1.0 клиент, способный обрабатывать Полный-Ответ, никогда не должен посылать Простой-Запрос.

Строка Статус начинается со строки с названием метода, за которым следует URI-Запроса и использующаяся версия протокола. Строка Статус заканчивается символами CRLF. Элементы строки разделяются пробелами (SP). В Строке Статус не допускаются символы LF и CR, за исключением заключающей последовательности CRLF.

```
Строка-Статус = Метод SP URI-Запроса SP Версия-HTTP CRLF
```

Следует отметить, что отличие Строки Статус Полного-Запроса от Строки Статус Простого- Запроса заключается в присутствии поля Версия-HTTP.

В поле Метод указывается метод, который должен быть применен к ресурсу, идентифицируемому URI-Запроса. Названия методов чувствительны к регистру. Существующий список методов может быть расширен.

```
Метод = "GET" | "HEAD" | "PUT" | "POST" | "DELETE" |
"LINK" | "UNLINK" | дополнительный-метод
```

Список методов, допускаемых отдельным ресурсом, может быть указан в поле Заголовок-Содержание "Баллов". Тем не менее, клиент всегда оповещается сервером через код статуса ответа, допускается ли применение данного метода для указанного ресурса, так как допустимость применения различных методов может динамически изменяться. Если данный метод известен серверу, но не допускается для указанного ресурса, сервер должен вернуть код статуса "405 Method Not Allowed", и код статуса "501 Not Implemented", если метод не известен или не поддерживается данным сервером. Общие методы HTTP/1.0 описываются ниже.

Метод GET служит для получения любой информации, идентифицированной URI-Запроса. Если URI- Запроса ссылается на процесс, выдающий данные, в качестве ответа будут выступать данные, сгенерированные данным процессом, а не код самого процесса (если только это не является выходными данными процесса).

Метод GET изменяется на "условный GET", если сообщение запроса включает в себя поле заголовка "If-Modified-Since". В ответ на условный GET, тело запрашиваемого ресурса передается только, если он изменялся после даты, указанной в заголовке "If-Modified-Since". Алгоритм определения этого включает в себя следующие случаи:

- Если код статуса ответа на запрос будет отличаться от "200 OK", или дата, указанная в поле заголовка "If-Modified-Since" некорректна, ответ будет идентичен ответу на обычный запрос GET.
- Если после указанной даты ресурс изменялся, ответ будет также идентичен ответу на обычный запрос GET.
- Если ресурс не изменялся после указанной даты, сервер вернет код статуса "304 Not Modified".

Использование метода условный GET направлено на разгрузку сети, так как он позволяет не передавать по сети избыточную информацию.

Метод HEAD аналогичен методу GET, за исключением того, что в ответе сервер не возвращает Тело- Ответа. Метаинформация, содержащаяся в HTTP заголовках ответа на запрос HEAD, должна быть идентична информации HTTP заголовков ответа на запрос GET. Данный метод может использоваться для получения метаинформации о ресурсе без передачи по сети самого ресурса. Метод "Условный HEAD", аналогичный условному GET, не определен.

Метод POST используется для запроса сервера, чтобы тот принял информацию, включенную в запрос, как субординантную для ресурса, указанного в Строке Статус в поле URI-Запроса. Метод POST был разработан, чтобы была возможность использовать один общий метод для следующих функций:

- Аннотация существующих ресурсов
- Добавление сообщений в группы новостей, почтовые списки или подобные группы статей
- Доставка блоков данных процессам, обрабатывающим данные
- Расширение баз данных через операцию добавления

Реальная функция, выполняемая методом POST, определяется сервером и обычно зависит от URI- Запроса. Добавляемая информация рассматривается как субординатная указанному URI в том же смысле, как файл субординатен каталогу, в котором он находится, новая статья субординатна группе новостей, в которую она добавляется, запись субординатна базе данных.

Клиент может предложить URI для идентификации нового ресурса,

включив в запрос заголовок "URI". Тем не менее, сервер должен рассматривать этот URI только как совет и может сохранить тело запроса под другим URI или вообще без него.

Если в результате обработки запроса POST был создан новый ресурс, ответ должен иметь код статуса, равный "201 Created", и содержать URI нового ресурса.

Метод PUT запрашивает сервер о сохранении Тело-Запроса под URI, равным URI-Запроса. Если URI-Запроса ссылается на уже существующий ресурс, Тело-Запроса должно рассматриваться как модифицированная версия данного ресурса. Если ресурс, на который ссылается URI-Запроса не существует, и данный URI может рассматриваться как описание для нового ресурса, сервер может создать ресурс с данным URI. Если был создан новый ресурс, сервер должен информировать направившего запрос клиента через ответ с кодом статуса "201 Created". Если существующий ресурс был модифицирован, должен быть послан ответ "200 OK", для информирования клиента об успешном завершении операции. Если ресурс с указанным URI не может быть создан или модифицирован, должно быть послано соответствующее сообщение об ошибке.

Фундаментальное различие между методами POST и PUT заключается в различном значении поля URI-Запроса. Для метода POST данный URI указывает ресурс, который будет управлять информацией, содержащейся в теле запроса, как неким придатком. Ресурс может быть обрабатывающим данные процессом, шлюзом в какой-нибудь другой протокол, или отдельным ресурсом, допускающим аннотации. В противоположность этому, URI для запроса PUT идентифицирует информацию, содержащуюся в Содержание-Запроса. Использующий запрос PUT точно знает какой URI он собирается использовать, и получатель запроса не должен пытаться применить этот запрос к какому-

нибудь другому ресурсу.

Метод DELETE используется для удаления ресурсов, идентифицированных с помощью URI-Запроса. Результаты работы данного метода на сервере могут быть изменены с помощью человеческого вмешательства (или каким-нибудь другим способом). В принципе, клиент никогда не может быть уверен, что операция удаления была выполнена, даже если код статуса, переданный сервером, информирует об успешном выполнении действия. Тем не менее, сервер не должен информировать об успехе до тех пор, пока на момент ответа он не будет собираться стереть данный ресурс или переместить его в некоторую недостижимую область.

Метод LINK устанавливает взаимосвязи между существующим ресурсом, указанным в URI-Запроса, и другими существующими ресурсами. Отличие метода LINK от остальных методов, допускающих установление ссылок между документами, заключается в том, что метод LINK не позволяет передавать в запросе Тело-Запроса, и в том, что в результате работы данного метода не создаются новые ресурсы.

Метод UNLINK удаляет одну или более ссылочных взаимосвязей для ресурса, указанного в URI- Запроса. Эти взаимосвязи могут быть установлены с помощью метода LINK или какого-нибудь другого метода, поддерживающего заголовок "Link". Удаление ссылки на ресурс не означает, что ресурс прекращает существование или становится недоступным для будущих ссылок.

Поля Заголовок-Запроса позволяют клиенту передавать серверу дополнительную информацию о запросе и о самом клиенте.

```
  Заголовок-Запроса = Accept | Accept-Charset | Accept-
Encoding | Accept-Language | Authorization | From | If-
```

```
Modified-Since | Pragma | Referer | User-Agent |
extension-header
```

Кроме того через механизм расширения могут быть определены дополнительные заголовки; приложения, которые их не распознают, должны трактовать эти заголовки, как Заголовок-Содержание.

В случае присутствия поля From, оно должно содержать полный E-mail адрес пользователя, который управляет программой-агентом, осуществляющей запросы. Этот адрес должен быть задан в формате, определенном в RFC 822. Формат данного поля следующий: From = "From" ":" спецификация адреса. Например:

```
From: webmaster@WWW.org
```

Данное поле может быть использовано для функций захода в систему, а также для идентификации источника некорректных или нежелательных запросов. Оно не должно использоваться, как несекретная форма разграничения прав доступа. Интерпретация этого поля состоит в том, что обрабатываемый запрос производится от имени данного пользователя, который принимает ответственность за применяемый метод. В частности, агенты-роботы должны использовать этот заголовок для того, чтобы можно было связаться с тем человеком, который отвечает за работу робота, в случае возникновения проблем. Почтовый Internet адрес, указывающийся в этом поле, не обязан соответствовать адресу того хоста, с которого был послан данный запрос. По возможности, адрес должен быть доступным Internet адресом вне зависимости от того, является ли он в действительности Internet E-mail адресом или Internet E-mail представлением адреса других почтовых систем.

Замечание: Клиент не должен использовать поле заголовка From без

позволения пользователя, так как это может войти в конфликт с его частными интересами или с местной, используемой им, системой безопасности. Настоятельно рекомендуется предоставление пользователю возможности запретить, разрешить или модифицировать это поле в любой момент перед запросом.

Поле заголовка If-Modified-Since используется с методом GET для того, чтобы сделать его условным: если запрашиваемый ресурс не изменялся во времени, указанного в этом поле, копия этого ресурса не будет возвращена сервером; вместо этого, будет возвращен ответ "304 Not Modified" без Тела-Ответа.

```
If-Modified-Since = "If-Modified-Since" ":" HTTP-дата
```

Пример использования заголовка:

```
If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

Целью этой особенности является предоставление возможности эффективного обновления информации локальных кэшей с минимумом передаваемой информации. Тот же результат может быть достигнут применением метода HEAD с последующим использованием GET, если сервер указал, что содержимое документа изменилось.

Поле заголовка User-Agent содержит информацию о пользовательском агенте, пославшем запрос. Данное поле используется для статистики, прослеживания ошибок протокола, и автоматического распознавания пользовательских агентов. Хотя это не обязательно, пользовательские агенты должны всегда включать это поле в свои запросы. Поле может содержать несколько строк, представляющих собой название программного продукта,

необязательную косую черту с указанием версии продукта, а также другие программные продукты, составляющие важную часть пользовательского агента. По соглашению, продукты указываются в списке в порядке убывания их значимости для идентификации приложения.

```
User-Agent = "User-Agent" ":" 1*( продукт )
продукт = строка ["/" версия-продукта]
версия-продукта = строка
```

Пример:

```
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
```

Строка, описывающая название продукта, должна быть короткой и давать информацию по существу - использование данного заголовка для рекламирования какой-либо другой, не относящейся к делу, информации не допускается и рассматривается, как не соответствующее протоколу. Хотя в поле версии продукта может присутствовать любая строка, данная строка должна использоваться только для указания версии продукта. Поле User-Agent может включать в себя дополнительную информацию в комментариях, которые не являются частью его значения.

HTTP ответ

После получения и интерпретации запроса, сервер посылает ответ в соответствии со следующей формой:

```
Ответ = Простой-Ответ | Полный-Ответ
Простой-Ответ = [ Содержание-Ответа ]
Полный-Ответ = Строка-Статус
        *(    Общий-Заголовок    |    Заголовок-Ответа    |
```

```
Заголовок-Содержания) CRLF
        [ Содержание-Ответа ]
```

Простой-Ответ должен посылаться только в ответ на HTTP/0.9 Простой-Запрос, или в том случае, если сервер поддерживает только ограниченный HTTP/0.9 протокол. Если клиент посылает HTTP/1.0 Полный-Запрос и получает ответ, который не начинается со Строки-Статус, он должен предполагать, что ответ сервера представляет собой Простой-Ответ, и обрабатывать его в соответствии с этим. Следует заметить, что Простой-Ответ состоит только из запрашиваемой информации (без заголовков) и поток данных прекращается в тот момент, когда сервер закрывает сеанс связи.

Первая строка Полного-Запроса является Строкой-Статус, состоящей из версии протокола, за которой следует цифровой код статуса и ассоциированное с ним текстовое предложение. Все элементы Строки-Статус разделены пробелами. Не разрешены символы CR и LF, за исключением завершающей последовательности CRLF.

```
Строка-Статус=Версия-HTTP SP Статус-Код SP Фраза-Об'яснение.
```

Так как Строка-Статус всегда начинается с версии протокола "HTTP/" 1*ЦИФРА "." 1*ЦИФРА (например HTTP/1.0), существование этого выражения рассматривается как основное для определения того, является ли ответ Простым-Ответом, или Полным-Ответом. Хотя формат Простого-Ответа не исключает появления подобной строки (что привело бы к неправильной интерпретации сообщения ответа и принятию его за Полный-Ответ), вероятность такого появления близка к нулю.

Элемент Статус-Код представляет собой 3-х цифровой целый код,

идентифицирующий результат попытки интерпретации и удовлетворения запроса. Фраза-Об'яснение, следующая за ним, предназначена для краткого текстового описания Статус-Кода. Статус-Код нацелен на то, чтобы его использовала машина, а Фраза-Об'яснение предназначена для человека. Клиент не обязан исследовать и выводить на экран Фразу-Об'яснение.

Первая цифра Статус-Кода предназначена для определения класса ответа. Последние две цифры не выполняют никакой категоризирующей роли. Существует 5 значений для первой цифры:

- 1xx: Информационный - Не используется, но зарезервирован для использования в будущем
- 2xx: Успех - Запрос был полностью получен, понят, и принят к обработке.
- 3xx: Перенаправление - Клиенту следует предпринять дальнейшие действия для успешного выполнения запроса. Необходимое дополнительное действие иногда может быть выполнено клиентом без взаимодействия с пользователем, но настоятельно рекомендуется, чтобы это имело место только в тех случаях, когда метод, использующийся в запросе безразличен (GET или HEAD).
- 4xx: Ошибка клиента - Запрос, содержащий неправильные синтаксические конструкции, не может быть успешно выполнен. Класс 4xx предназначен для описания тех случаев, когда ошибка была допущена со стороны клиента. Если клиент еще не завершил запрос, когда он получил ответ с Статус-Кодом- 4xx, он должен немедленно прекратить передачу данных серверу. Данный тип Статус-Кодов применим для любых методов, употребляющихся в запросе.
- 5xx: Ошибка Сервера - Сервер не смог дать ответ на корректно поставленный запрос. В этих случаях
- сервер либо знает, что он допустил ошибку, либо не способен обработать запрос. За исключением ответов на запросы HEAD, сервер посылает

описание ошибочной ситуации и то, является ли это состояние временным или постоянным, в Содержание-Ответа. Данный тип Статус-Кодов применим для любых методов, употребляющихся в запросе.

Отдельные значения Статус-Кодов и соответствующие им Фразы-Об'яснения приведены ниже. Данные Фразы-Об'яснения только рекомендуются - они могут быть замещены любыми другими фразами, сохраняющими смысл и допускающимися протоколом.

```
Статус-Код = "200" ; OK |
             "201" ; Created |
             "202" ; Accepted |
             "203" ; Provisional Information |
             "204" ; No Content |

             "300" ; Multiple Choices |
             "301" ; Moved Permanently |
             "302" ; Moved Temporarily |
             "303" ; Method |
             "304" ; Not Modified |

             "400" ; Bad Request |
             "401" ; Unauthorized |
             "402" ; Payment Required |
             "403" ; Forbidden |
             "404" ; Not Found |
             "405" ; Method Not Allowed |
             "406" ; None Acceptable |
             "407" ; Proxy Authentication Required |
             "408" ; Request Timeout |
```

```
"409" ; Conflict |
"410" ; Gone |

"500" ; Internal Server Error |
"501" ; Not Implemented |
"502" ; Bad Gateway |
"503" ; Service Unavailable |
"504" ; Gateway Timeout |
Код-Рассширения
```

Код-Расширения = 3ЦИФРА
Фраза-Об'яснение = строка *(SP строка)

От HTTP приложений не требуется понимание всех Статус-Кодов, хотя такое понимание, очевидно, желательно. Тем не менее, от приложений требуется способность распознавания классов Статус-Кодов (идентифицирующихся первой цифрой) и отношение ко всем Статус-Кодам статуса ответа, как если бы они были эквивалентны Статус-Коду x00.

Поля заголовка ответа позволяют серверу передать дополнительную информацию об ответе, которая не может быть внесена в Строку-Статус. Эти поля заголовков не предназначены для передачи информации о содержании ответа, передаваемого в ответ на запрос, но там может быть информация собственно о сервере.

```
Заголовок-Ответа= Public | Retry-After | Server | WWW-
Authenticate | extension-header
```

Хотя дополнительные поля заголовка ответа могут быть реализованы через механизм расширения, приложения, которые не распознают эти поля,

должны обрабатывать их как поля Заголовок-Содержание. Полное описание этих полей можно получить в спецификации протокола HTTP в CERN.

Что такое cookie?

Cookie является решением одной из наследственных проблем HTTP спецификации. Эта проблема заключается в непостоянстве соединения между клиентом и сервером, как при FTP или Telnet сессии, т.е. для каждого документа (или файла) при передаче по HTTP протоколу посылается отдельный запрос. Включение cookie в HTTP протокол дало частичное решение этой проблемы.

Cookie это небольшая порция информации, которую сервер передает клиенту. Клиент (броузер) будет хранить эту информацию и передавать ее серверу с каждым запросом как часть HTTP заголовка. Некоторые cookie хранятся только в течение одной сессии, они удаляются после закрытия броузера. Другие, установленные на некоторый период времени, записываются в файл. Обычно этот файл называется 'cookie.txt'.

Что можно делать с помощью cookie?

Сами по себе cookies не могут делать ничего, это только лишь некоторая информация. Однако, сервер может реагировать на содержащуюся в cookies информацию. Например, в случае авторизованного доступа к чему либо через WWW, в cookies сохраняется login и password в течение сессии, что позволяет не вводить их при запросе каждого запаролированного документа. Другой пример: cookies могут использоваться для построения персонализированных страниц. Чаще всего встречается такое - на некотором сервере Вас просят ввести свое имя, и каждый раз, когда Вы заходите на первую страницу этого сервера, Вам пишут что-то типа "Hello, your_name!". На использовании cookies также часто строят функцию оформления заказа в онлайновых магазинах, в частности, в Амазоне, такая своеобразная виртуальная корзина покупателя, как в обычном реальном супермаркете.

Формат и синтаксис Cookie

Полное описание поля Set-Cookie HTTP заголовка:

```
Set-Cookie: NAME=VALUE; expires=DATE; path=PATH;
domain=DOMAIN_NAME; secure
```

Минимальное описание поля Set-Cookie HTTP заголовка:

```
Set-Cookie: NAME=VALUE;
```

NAME=VALUE - строка символов, исключая перевод строки, запятые и пробелы. NAME-имя cookie, VALUE - значение.

expires=DATE - время хранения cookie, т.е. вместо DATE должна стоять дата в формате Wdy, DD-Mon-YYYY HH:MM:SS GMT, после которой истекает время хранения cookie. Если этот атрибут не указан, то cookie хранится в течение одного сеанса, до закрытия броузера.

domain=DOMAIN_NAME - домен, для которого значение cookie действительно. Например, domain=cit-forum.com. В этом случае значение cookie будет действительно и для сервера cit-forum.com, и для www.cit-forum.com. Но не радуйтесь, указания двух последних периодов доменных имен хватает только для доменов иерархии "COM", "EDU", "NET", "ORG", "GOV", "MIL", и "INT". Для доменов иерархии "RU" придется указывать три периода.

Если этот атрибут опущен, то по умолчанию используется доменное имя сервера, с которого было выставлено значение cookie.

path=PATH - этот атрибут устанавливает подмножество документов, для

которых действительно значание cookie. Например, указание path=/win приведет к тому, что значение cookie будет действительно для множества документов в директории /win/, в директории /wings/ и файлов в текущей директории с именами типа wind.html и windows.shtml

Если этот атрибут не указан, то значение cookie распространяется только на документы в той же директории, что и документ, в котором было установлено cookie.

secure - если стоит такой маркер, то информация cookie пересылается только через HTTPS (HTTP с использованием SSL). Если этот маркер не указан, то информация пересылается обычным способом.

Синтаксис HTTP заголовка для поля Cookie

Когда запрашивается документ с HTTP сервера, броузер проверяет свои cookie на предмет соответствия домену сервера и прочей информации. В случае, если найдены удовлетворяющие всем условиям значения cookie броузер посылает их в серверу в виде пары имя/значение:

```
Cookie: NAME1=OPAQUE_STRING1; NAME2=OPAQUE_STRING2 ...
```

В случае, если cookie принимает новое значение при имеющемся уже в броузере cookie с совпадающими NAME, domain и path, старое значение затирается новым. В остальных случаях новые cookies добавляются.

Использование expires не гарантирует сохранность cookie в течение заданного периода времени, поскольку клиент (броузер) может удалить запись вследствие нехватки выделенного места или каких-либо других лимитов.

Клиент (броузер) имеет следующие ограничения:

- всего может храниться до 300 значений cookies
- каждый cookie не может превышать 4Кбайт
- с одного сервера или домена может храниться до 20 значений cookie

Если ограничение 300 или 20 превышается, то удаляется первая по времени запись. При превышении 4К - корректность такого cookie страдает - отрезается кусок записи (с начала этой записи) равный превышению.

В случае кэширования документов, например, proxy-сервером, поле Set-cookie HTTP заголовка никогда не кэшируется.

Если proxy-сервер принимает ответ, содержащий поле Set-cookie в заголовке, предполагается, что поле таки доходит до клиента вне зависимости от статуса 304 (Not Modified) или 200 (OK).

Соответственно, если клиентский запрос содержит в заголовке Cookie, то он должен дойти до сервера, даже если установлен If-modified-since.

Примеры

Ниже приведено несколько примеров, иллюстрирующих использование cookies

Первый пример:

Клиент запрашивает документ и принимает ответ:

```
  Set-Cookie: CUSTOMER=WILE_E_COYOTE; path=/;
expires=Wednesday, 09-Nov-99 23:12:40 GMT
```

Когда клиент запрашивает URL с путем "/" на этом сервере, он посылает:

```
 Cookie: CUSTOMER=WILE_E_COYOTE
```

Клиент запрашивает документ и принимает ответ:

```
  Set-Cookie: PART_NUMBER=ROCKET_LAUNCHER_0001; path=/
```

Когда клиент запрашивает URL с путем "/" на этом сервере, он посылает:

```
Cookie: CUSTOMER=WILE_E_COYOTE;
PART_NUMBER=ROCKET_LAUNCHER_0001
```

Клиент получает:

```
  Set-Cookie: SHIPPING=FEDEX; path=/foo
```

Когда клиент запрашивает URL с путем "/" на этом сервере, он посылает:

```
  Cookie: CUSTOMER=WILE_E_COYOTE;
PART_NUMBER=ROCKET_LAUNCHER_0001
```

Когда клиент запрашивает URL с путем "/foo" на этом сервере, он посылает:

```
  Cookie: CUSTOMER=WILE_E_COYOTE;
PART_NUMBER=ROCKET_LAUNCHER_0001; SHIPPING=FEDEX
```

Второй пример:

Клиент принимает:

```
  Set-Cookie: PART_NUMBER=ROCKET_LAUNCHER_0001; path=/
```

Когда клиент запрашивает URL с путем "/" на этом сервере, он посылает:

```
Cookie: PART_NUMBER=ROCKET_LAUNCHER_0001
```

Клиент принимает:

```
Set-Cookie: PART_NUMBER=RIDING_ROCKET_0023; path=/ammo
```

Когда клиент запрашивает URL с путем "/ammo" на этом сервере, он посылает:

```
Cookie: PART_NUMBER=RIDING_ROCKET_0023;
PART_NUMBER=ROCKET_LAUNCHER_0001
```

Комментарий: здесь мы имеем две пары имя/значение с именем "PART_NUMBER".

Это наследие из предыдущего примера, где значение для пути "/" прибавилось к значению для "/ammo".

HTTP-сервер Apache

Самый распространенный Web-сервер в мире - это Apache. По данным компании Netcraft (http://www.netcraft.com/Survey/) общее число Web-узлов, работающих под его управлением, к концу 1998 г. достигло 2 млн. (55% общего числа узлов) и постоянно растет. Для сравнения: на долю серверов Microsoft приходится 25%, Netscape -7%. Будучи бесплатной открытой программой, предназначенной для бесплатных же Unix-систем (FreeBSD, Linux и др.), Apache по функциональным возможностям и надежности не уступает коммерческим серверам, а широкие возможности конфигурирования позволяют настроить его для работы практически с любой конкретной системой. Существуют локализации сервера для различных языков, в том числе и для

русского.

Для тонкой настройки сервера Apache используется набор конфигурационных файлов. Рассмотрим здесь одну из наиболее важных процедур конфигурирования сервера Apache.

Файл httpd.conf

Конфигурационный файл httpd.conf является основным и содержит настройки, связанные с работой Web-сервера, виртуальных серверов, а также всех его программных модулей. Кроме того, именно в нем настраивается перекодирование русских букв при передаче от сервера к клиенту и обратно.

Директива Port, помещенная в самом начале файла, определяет номер порта для http-сервера; по умолчанию это 80. При необходимости можно приписать серверу другой порт или несколько портов, для чего служит директива Listen.

Директива HostnameLookups с параметром on или off включает или, соответственно, отключает преобразование численных IP-адресов клиентов, получивших документы с сервера, в доменные имена. Такое преобразование несколько замедляет работу сервера, но при числе посещений менее 10 000 в сутки это, как правило, практически не заметно.

Директивы User и Group задают пользователя, который будет администрировать сервер. С точки зрения безопасности нежелательно указывать здесь существующего пользователя, имеющего доступ к каким-либо другим ресурсам или файлам. Лучше создать отдельного пользователя и группу специально для http-сервера, например:

```
User www
Group www
```

Директивы ServerRoot, ErrorLog, CustomLog определяют соответственно корневой каталог http-сервера, путь к журналу регистрации ошибок (error_log) и

путь к общему журналу обращений к серверу (access_log).

Директива CacheNegotiatedDocs разрешает кэширование документов, полученных с сервера. По умолчанию этот режим отключен, но, поскольку пропускная способность отечественных Internet-каналов еще долго будет оставлять желать лучшего, хорошо бы его включить: тогда пользователю не придется ждать загрузки картинок при каждом обращении к вашей странице.

Настройка виртуальных серверов в файле httpd.conf

В большинстве случаев один http-сервер способен обрабатывать запросы, поступающие на различные, так называемые виртуальные, Web-серверы. Виртуальные серверы могут иметь как один и тот же IP-адрес, но разные доменные имена, так и разные IP-адреса. С точки зрения пользователя второй вариант чуть более предпочтителен, поскольку запрос к серверу, отличающемуся от основного только доменным именем, должен содержать его имя, а некоторые старые браузеры, не поддерживающие протокол HTTP/1.1 (например, Microsoft Internet Explorer 2.0), не включают в запрос эту информацию. Однако такие браузеры выходят из употребления (сейчас их уже менее 0,5% общего числа); с другой стороны, выделение собственного IP-адреса каждому виртуальному серверу может быть неоправданной растратой адресного пространства компании.

Для описания адресов и доменных имен виртуальных серверов служат директивы ServerName, ServerAlias, NameVirtualHost и VirtualHost. Они необходимы, только если вам нужно установить более одного виртуального сервера.

Описание виртуальных серверов с различными IP-адресами

```
ServerName ed.tusur.ru

<VirtualHost 192.168.1.1>
```

```
DocumentRoot /www/ed.tusur.ru
ServerName ed.tusur.ru
ErrorLog /var/log/error_log.ed.tusur.ru
CustomLog /var/log/access_log.ed.tusur.ru combined
...
</VirtualHost>

<VirtualHost 192.168.1.3>
DocumentRoot /www/rzi.tusur.ru
ServerName rzi.tusur.ru
ErrorLog /var/log/error_log.rzi.tusur.ru
CustomLog /var/log/access_log.rzi.tusur.ru combined
...
</VirtualHost>
```

В листинге приведен фрагмент конфигурационного файла для случая виртуальных серверов с различными IP-адресами, в следующем листинге - аналогичный фрагмент для случая, когда серверы различаются только доменным именем.

Описание виртуальных серверов, различающихся только доменным именем

```
ServerName ed.tusur.ru
NameVirtualHost 192.168.1.1

<VirtualHost 192.168.1.1>
DocumentRoot /www/ed.tusur.ru
ServerName ed.tusur.ru
ErrorLog /var/log/error_log.ed.tusur.ru
```

```
CustomLog /var/log/access_log.ed.tusur.ru combined
...
</VirtualHost>

<VirtualHost 192.168.1.1>
DocumentRoot /www/forum.ed.tusur.ru
ServerName forum.ed.tusur.ru
ServerAlias *.forum.ed.tusur.ru
ErrorLog /var/log/error_log.forum.ed.tusur.ru
CustomLog /var/log/access_log.forum.ed.tusur.ru combined
...
</VirtualHost>
```

Директива ServerName, находящаяся вне секций VirtualHost, определяет имя основного сервера, т. е. сервера, корневой каталог которого задан директивой DocumentRoot в файле srm.conf. Виртуальные серверы наследуют настройки основного; при необходимости специальной настройки соответствующие директивы помещаются в секции VirtualHost, относящейся к данному серверу. Допустимы любые директивы, которые могут встретиться в файлах httpd.conf и srm.conf, например DocumentRoot, ErrorLog, CustomLog, Location, ServerAdmin.

Из последнего листинга видно, как используется директива ServerAlias, если необходимо создать несколько виртуальных серверов с одинаковым содержанием.

Сервер баз данных MySQL

Как только мы задумываемся об упорядоченном хранении информации, неизбежно приходим к мысли о создании базы данных. Создав ее, например, в любом настольной СУБД, мы облегченно вздыхаем и на какое-то время успокаиваемся. Но это "какое-то время" длится недолго - на горизонте уже

маячит следующая, продиктованная жизненной необходимостью задача: нужно обеспечить доступ к данным через интернет с рабочих станций, на которых, вполне возможно, установлены различные ОС. И вот тут на помощь приходит быстрая и надежная СУБД MySQL.

Эта СУБД ведет свое начало от mSQL. Разработчики нуждались в более быстрой, надежной и гибкой СУБД. В результате был создан новый SQL-интерфейс при почти не измененном API-интерфейс mSQL. Этот API был выбран, чтобы облегчить перенос программ других разработчиков.

SQL - это язык структурированных запросов (Structured Query Language), международный стандарт языка для доступа к базам данных.

MySQL - это, если смотреть в общем, SQL-сервер; иначе говоря - программа, которая принимает запросы, написанные на SQL, и отсылает обратно определенные ответы. Ответами могут быть: данные, количество строк, задействованных в запросе, или просто строка.

СУБД MySQL предоставляет в ваше распоряжение подмножество языка SQL, соответствующее спецификации ANSI SQL 92.

Для обеспечения доступа к MySQL имеется несколько путей. Это можно сделать или через клиент (как текстовый, так и графический), или через какой-то язык программирования.

Ядро, на котором сформирован MySQL, представляет собой набор подпрограмм, которые уже много лет использовались в высокотребовательном окружении. Это богатый и полезный функциональный набор, в то время как MySQL все еще находится в разработке.

Основные цели MySQL: быстродействие и ошибкоустойчивость. Эта СУБД прекрасно справляется с обработкой очень значительных массивов данных (большое количество пользователей сообщают о работе с тысячами и даже миллионами записей). В листах рассылок нередки сообщения об базах данных в несколько гигабайт. Кроме того, MySQL в запросах к большим таблицам превосходит многие другие системы. Она очень стабильна и превосходно работает даже в случаях, когда несколько сотен пользователей

нуждаются в доступе к одним и тем же данным. Исходя из вышесказанного легко понять, почему MySQL пользуется такой популярностью у разработчиков веб-приложений. MySQL также очень масштабируема, работает на всем - от персональных компьютеров до больших многопроцессорных систем.

Одно из замечательных свойств MySQL заключается в том, что пользователь имеет доступ к MySQL-серверу независимо от операционной системы, языка программирования или клиента, которым пользуется. Находясь на UNIX-системе, пользователь может связываться с MySQL-сервером, работающим на MacOS и Windows-платформах.

Для взаимодействия с MySQL-сервером можно использовать множество языков программирования. Можно написать программы на C, C++, Java, PHP, Perl, TCL, Python - и они будут работать.

В MySQL все организовано по схеме "ничего лишнего", поэтому и отсутствуют некоторые функции, которыми оснащены другие базы данных. MySQL не поддерживает вложенных запросов, тригерров; отсутствует также встроенная поддержка XML, OLAP и конструктов (constraits). Но зато есть новации (например, возможность кешировать запросы). Отсутствующие функции были принесены в жертву быстродействию, которое обеспечивает эта СУБД (хотя, если какие-то из этих функций все же понадобятся, можно использовать разработками сторонних производителей).

Mysql - быстрая и надежная система управления базами данных, хотя ее редко используют, набирая все команды вручную. Обычно она играет свою роль за кулисами, обслуживая разнообразнейшие веб-сайты. В этом случае все управление этой СУБД обеспечивается, например, через язык написания скриптов PHP.

6. Средства реализации современных информационных систем

Язык разметки гипертекстовых страниц HTML

HTML (HyperText Markup Language - язык маркировки гипертекстов) - язык разметки гипертекстового документа. Говоря проще, это набор средств для описания визуальных свойств (позиция, размер, цвет и т.д.) различных элементов, в частности текста или графики. Под гипертекстовым документом подразумеваются документы с гипертекстовыми ссылками - указателями на другие гипертекстовые документы.

HTML документ в самом простом случае представляет собой один текстовый файл с разширением .htm или .html - никакой разницы между ними нет, можете использовать то, которое Вам больше нравится. Создавать HTML возможно при помощи обычного текстового редактора. Только не стоит использовать программы, которые добавляют форматирование (например, Microsoft Word). Это такие программы в которых можно, например выделить часть текста жирным фоном или вставить картинку - все это испортит HTML файл, для корректного создания которого необходим чисто текстовый редактор.

Язык разметки гипертекста (HyperText Markup Language - HTML) можно использовать для представления:

- гипертекстовых новостей, почты, сопровождающей информации и сопутствующей гиперсреды,
- меню с опциями
- результатов запросов к базам данных
- простых структурированных документов со встроенной графикой
- гипертекстовых обзоров имеющейся информации

Программа World Wide Web (W3) инициирует каналы передачи связной информации по всему земному шару. Язык HTML предоставляет простой формат для предоставления этой информации. Требуется, чтобы все

программы, совместимые с W3, могли поддерживать язык HTML. Программа W3 использует протокол Internet (протокол передачи гипертекста - HTTP), который позволяет передавать кодированную информацию между клиентом и сервером, при этом результат возвращается через расширенное MIME сообщение. Поэтому язык HTML является лишь одним, но довольно важным, из описаний, используемых в программе W3.

Данные в формате HTML похожи на текстовой файл, за исключением того, что некоторые из символов интерпретируются как разметка. Разметка придает документу некую структуру.

Данные представляют собой иерархию элементов. Каждый элемент имеет имя, атрибуты и несет некую информацию. Большинство элементов представлены в документе в виде начальной метки, указывающей имя и атрибуты. Далее следует собственно содержание элемента. И наконец, заканчивает все это конечная метка. Например,

```
<HTML>
    <TITLE>
        Простой блок данных
     </TITLE>
    <H1>
        Пример структуры
    </H1>
 Обычный параграф

      <P>

      <UL>

         <LI>Первая запись, включающая
```

```
            <A NAME="URI">

               текст

            </A>

         <LI>Вторая запись

      </UL>

   </HTML>
```

Некоторые элементы языка (такие как P, LI) являются пустыми. Они не имеют поля данных, и ограничиваются лишь начальной меткой.

В остальных элементах поле данных представляет собой набор символов и вложенных элементов. Заметим, что описание HTML DTD фактически накладывает некие ограничения на количество допустимых вложений - большинство элементов не могут быть вложены в другие элементы. Ни один из элементов не может быть вложенным сам в себя рекурсивным образом. Анкеры и выделенные символы могут быть помещены в другие конструкции.

Каждый элемент начинается с метки, меткой же и заканчивается каждый непустой элемент. Начальные метки выделяются символами < и >, а конечные - символами </ и >.

Имя элемента следует в метке сразу за символом открытия <. Имя начинается с буквы, за которой могут следовать еще 33 буквы, цифры, пробела или дефиса. В именах игнорируется разница между прописными и строчными буквами.

Начальная метка позволяет вставить между именем и символом > пробелы и атрибуты. Атрибут состоит из имени, символа равенства и значения. Слева и справа от символа равенства можно оставлять пробелы.

Значение атрибута указывается в виде строки, заключенной в одинарные или двойные кавычки.

Чтобы определить значение атрибута, осуществляется анализ данной строки в формате RCDATA. Например, такой подход позволяет представлять символы кавычек в значении атрибута как обращения к символам по числовому значению. Длина строки со значением атрибута после такого анализа не должна превышать 1024 символов.

Если мы хотим включить в HTML документ комментарий таким образом, чтобы он игнорировался анализатором, необходимо поставить перед ним и после него ограничители <!-- и --> соответственно. Весь текст, расположенный между начальным ограничителем и символами --, будет игнорироваться. Следовательно, комментарии не могут быть вложенными. В заключительном ограничителе между -- и символом > можно вставлять пробелы (но в начальном ограничителе между <! и -- вставки не допускаются). Например,

```
<HEAD>
<TITLE>HTML Guide: Recommended Usage</TITLE>
<!-- $Id: recommended.html,v 1.3 93/01/06 18:38:11
connolly Exp $ -->
</HEAD>
```

Элементы языка HTML

Здесь приведен список элементов, используемых в языке HTML. Документы должны (но не обязательно) содержать элемент HEAD, за которым

следует элемент BODY.

Документы старого типа могут содержать лишь данные обычных элементов HEAD и BODY, причем в любом порядке. Это осуждается, но тем не менее, анализаторы должны воспринимать такое построение документа. Обратите внимание также на статус элементов.

Общие свойства документа

Элемент **HEAD** содержит всю информацию о документе в целом. Однако он не содержит какого-либо текста. Последний является лишь частью документа и должен находиться в элементе BODY. В элементе заголовка HEAD можно использовать лишь строго заданный набор элементов.

Нижеприведенные элементы определяют общие свойства документа. Они должны появляться в элементе HEAD. Порядок элементов значения не имеет.

- TITLE - Название элемента.
- ISINDEX - Элемент, посылаемый серверу вместе с документом, предназначенным для информации к поиску. .
- NEXTID - Параметр, используемый текстовыми редакторами для создания , уникальных идентификаторов. , (Устарел и не рекомендуется использовать.).
- LINK - Элемент, определяющий связь этого документа с другими. В , документе может присутствовать несколько элементов LINK.
- BASE - Запись, сделанная на языке URL при фиксировании данного , документа.

Форматирование текста

В элементе BODY документа встречаются элементы из приведенного ниже списка. Они выстроены в том порядке, в каком должны подаваться на устройство вывода.

Заголовки

(Headings) Язык поддерживает заголовки разделов различных уровней.

Анкеры

(Anchors) Части текста, которые формируют начало и/или конец связей в гипертексте, называются, анкерами и формируются меткой A.

Метки параграфов

(Paragraph marks) Элемент P указывает на границу между параграфами.

Стиль адреса

(Address style) Этот элемент указывает, в каком стиле предстает перед клиентом элемент ADDRESS.

Выделенный блок текста.

(Blockquote style) .

Списки списки, словари и т.д.

Преформатированный текст

(Preformatted text) Части текста, предварительно отформатированные с использованием шрифта фиксированной ширины.

Выделение символов

(Character highlighting) Элементы форматирования, не вызывающие разбиения на параграфы.

Графика

IMG - Метка IMG может использоваться для включения в текст графических изображений.

В противоположность элементу HEAD элемент BODY содержит всю ту информацию, из которой собственно и состоит рассматриваемый документ. Порядок следования элементов здесь именно тот, в каком они предстают перед читателем.

Анкер (элемент A)

Анкер - это некий текст, который указывает на начало и/или конец связи в гипертексте. Текст между открывающей и закрывающей метками определяет начало связи или указываемое ею место (или и то, и другое вместе). Метка анкера может иметь следующие атрибуты:

- HREF Необязательный. (Адрес гипертекстовой ссылки) Если атрибут HREF установлен, то анкер является точно выверенный текстом - началом соединения. Если читатель выбрал этот текст, то ему (ей) будет представлен другой элемент, чей сетевой адрес определяется значением HREF атрибута. Формат сетевого адреса определяется в другом месте. Такой подход позволяет с помощью формы HREF="#индикатор" ссылаться на другой анкер в том же самом документе. Если же анкер относится к другому документу, атрибут является относительным именем, именем относительно данного документа (либо он указывает базовый адрес, если таковой имеется).
- NAME Необязательный. Если этот атрибут указан, то он позволяет данному анкеру быть местом в документе, на которое ссылается какой-либо анкер. Значение атрибута является идентификатором анкера. Идентификатор анкера - это произвольная строка текста, которая тем не менее уникальна в пределах рассматриваемого HTML документа. Другие документы тоже могут создавать ссылки именно на этот анкер, помещая его идентификатор в поле адреса документа после символа #.
- REL Необязательный. Атрибут REL может дать взаимоотношение(ия) в описанной ранее связи гипертекста. Значение атрибута - это список значений для взаимоотношений, написанный через запятую. Значения атрибута и их семантика будут регистрироваться комитетом по языку HTML. Если ничего не указано, то по умолчанию предполагается, что взаимоотношениях не несут каких-либо значений. Атрибут REL нельзя применять, если нет атрибута HREF.
- REV Необязательный. Полностью аналогичен атрибуту REL за исключением того, что тип соединения имеет обратную семантику. Связь

из анкера A в анкер B с атрибутом REL="X" полностью аналогична связи из B в A с атрибутом REV="X". Анкер может иметь оба атрибута REL и REV.

- URN Необязательный. Если этот атрибут указан, то это определяет универсальный номер ресурса для данного документа.
- TITLE Необязательный. Данный атрибут является чисто информационным. Если атрибут присутствует в анкере, его значение должно (может прим.ред.) совпадать со значением элемента TITLE в документе, чей адрес указан в атрибуте HREF.
- METHODS Необязательный. Значение этого атрибута - строка. Она должна представлять собой список через запятую методов HTTP, которые программа общего пользования в состоянии поддерживать.

Все приведенные выше атрибуты являются необязательными, хотя для того, чтобы анкер действовал, нужны NAME и HREF. См. также описание LINK.

Пример использования анкеров

```
See <A HREF="http://info.cern.ch/">CERN</A>'s information
for more details.

A <A NAME=serious>serious</A> crime is one which is
associated with imprisonment.

The Organization may refuse employment to anyone
convicted of a <a href="#serious">serious</A> crime.
```

Замечание 1. Универсальные номера для ресурсов (Universal resource numbers - URN) должны обеспечивать распознавание документа в случае

обнаружения его дубликатов. Должно существовать программное обеспечение клиента, осуществляющее отсев копий для уже имеющейся информации.

Формат номеров URN обсуждается различными рабочими группами из инженерного подразделения сети Internet (1993). (На сегодняшний день спецификация URN не определена. Прим.ред.)

Замечание 2. Атрибут названия для связей (TITLE).

Соединение может содержать атрибут TITLE. Если этот атрибут имеется, он должен давать название документа, чей адрес определен в атрибуте HREF. Есть по крайней мере две причины для использования такого атрибута.

- Программа просмотра может запросить показ названия документа в качестве предварительного условия для его выборки. Например, в виде метки с записью, или маленького ящика, возникающего, когда мышь попадает на анкер или же при вызове документа.
- Некоторые документы не имеют названия, так что использование атрибута названия для связи является для них единственным способом получить название. В основном это документы, не являющиеся размеченным текстом, графикой, текстом и меню для программы Gopher. Именно так работает упомянутая программа Gopher. Очевидно, что это приводит к дублированию данных и было бы рискованно безоглядно надеяться на то, что атрибут названия у соединения будет корректным и уникальным для соответствующего документа.

Замечание. 3 Атрибут метода для связи (METHODS).

Анкеры и связи используют атрибут метода для указания действий, которые клиент может применять к объектам. Эти действия более точно формулируются в HTTP протоколе, если таковой применяется. Однако этот атрибут, как и атрибут TITLE, в силу некоторых причин может использоваться для повышения информативности соединения. Например, программа чтения может вызывать различные способы визуализации информации в зависимости

от разрешенного в атрибуте метода (например, клиент, осуществляющий поиск, может пользоваться различными иконами).

Элемент выделения блока (BLOCKQUOTE)

Элемент BLOCKQUOTE допускает обработку специальным образом текста, выделенного в каком-либо источнике.

Типовая обработка элемента. Типовая обработка может заключаться в дополнительном смещении текста влево или вправо и/или в использовании наклонного шрифта. Элемент BLOCKQUOTE приводит к разбиению текста на параграфы, а также обычно к появлению пустой строки или пробелов между выделенным блоком и предшествующим/ последующим текстом.

Обработка с единым шрифтом может, к примеру, привести к появлению в начале строки символа ">", что соответствует стилю выделения в системе Internet почты.

Пример:

```
I think it ends
<BLOCKQUOTE>Soft you now, the fair Ophelia, Nymph, in thy orisons,
be all my sins remembered.
</BLOCKQUOTE>
but I am not sure.
```

Заголовки (Headings)

Обрабатывается до шести уровней заголовков (Заметим, что узел в гипертексте, как правило, нуждается в меньшем количестве уровней, чем сочинение, чья структура целиком определяется применением заголовков).

Элемент заголовка несет в себе все изменения шрифтов, разбиение на параграфы до и после, пробелы, необходимые, например, для обработки заголовка. Язык HTML не требует применения иных средств для выделения символов или разбивки текста на параграфы.

Заголовок H1 относится к самому верхнему уровню и рекомендуется в качестве начального для узла в гипертексте. Предполагается, что текст первого заголовка будет соответствовать запросам клиента, уже производящего анализ связанной с этим узлом информации. Это отличает заголовок (heading) от названия (title), которое должно характеризовать данный узел в более широком плане.

Элементы заголовка: <H1>, <H2>, <H3>, <H4>, <H5>, <H6>.

Было бы отклонением от правил при переходе от заголовка к заголовку пропускать какой-либо уровень, например, ставить после элемента H1 сразу элемент H3. Хотя такая практика и не запрещена, но нежелательна и может привести к странным результатам при написании других реализаций языка HTML.

Пример

```
<H1>This is a heading</H1>
Here is some text
<H2>Second level heading</H2>
Here is some more text
```

- H1 Толстый, очень крупный шрифт, текст центрирован. Между заголовком и последующим текстом вставляется одна или две пустых строки. При выводе на принтер заголовок печатается на новой странице.
- H2 Толстый крупный шрифт. Без отступа. До и после заголовка

помещаются одна или две пустых строки.

- H3 Наклонный большой шрифт. До и после заголовка помещаются одна или две пустые строки. С небольшим отступом.
- H4 Толстый нормальный шрифт. Отступ больше, чем в H3. До и после заголовка помещается пустая строка.
- H5 Наклонный нормальный шрифт. Отступ как у заголовка H4. Пустая строка ставится перед заголовком, но не после.
- H6 Толстый шрифт. Отступ такой же, как у обычного текста и больше, чем у H5. Перед заголовком ставится пустая строка.

IMG: Встроенные изображения

Статус: дополнительный

Элемент IMG позволяет вставлять информацию из другого документа. Последний обычно является иконкой, маленькой картинкой и т.д. Элемент IMG не предназначен для вставки дополнительного HTML текста.

Те анализаторы гипертекста, которые не могут показывать встроенные изображения, элементы IMG игнорируют. Авторам документов следует взять на заметку, что некоторые анализаторы могут показывать (или печатать на принтере) связанные с данным документом изображения, но не встроенные. Если изображение имеет большое значение, может оказаться более разумным создать с ним связь, нежели делать это изображение встроенным в гипертекст. Если же изображение является в значительной степени декоративным, более удобным будет применение элемента IMG.

Элемент IMG является пустым (не имеет заключительной метки) и имеет два атрибута:

- SRC Значением этого атрибута является URL документа, который должен быть вставлен в гипертекст. Синтаксис этого атрибута такой же,

как и у атрибута HREF для метки A. Атрибут SRC является обязательным.

- ALIGN Это атрибут, принимая значения TOP, MIDDLE или BOTTOM, определяет, верхняя, средняя или нижняя часть изображения должна быть поставлена вровень с текстом.

В тексте анкеров допускается применение элементов IMG.

Пример

```
Warning: < IMG SRC ="triangle.gif">
Thus must be done by a qualified
technician.
< A HREF="Go">< IMG SRC ="Button"> Press to start</A>
```

Списки

Список - это последовательность параграфов, каждому из которых может предшествовать специальная метка или очередной номер. Синтаксис списка:

```
<UL>
<LI> list element
<LI> another list element ...
</UL>
```

Открывающими метками для списка могут быть UL, OL, MENU или DIR. Сразу за открывающей меткой должен следовать первый элемент списка.

Список элементов, имеющих типовые алгоритмы обработки:

- UL Список многострочных параграфов, обычно разделенных несколькими пробелами и/или размеченный кружками или крупными черными точками.
- OL Этот элемент похож на элемент UL, за исключением того, что параграфы нумеруются.

- MENU Список параграфов меньшего размера. Обычно на одну запись приходится лишь одна строка, а ее стиль более компактен, чем в случае элемента UL.
- DIR Список элементов, чей размер, как правило, не превышает 20 символов. Элементы могут размещаться в несколько колонок на странице, причем ширина такой колонки обычно 24 символа. Намного лучше, если программа обработки в состоянии оптимизировать ширину колонки в зависимости от ширины составляющих ее элементов.

Пример использования

```
<OL>
<LI> When you get to the station, leave
by the southern exit, on platform one.
<LI>Turn left to face toward the mountain
<LI>Walk for a mile or so until you reach the
"Asquith Arms" then
<LI>Wait and see ...
</OL>

< MENU >
<LI>The oranges should be pressed fresh
<LI>The nuts may come from a packet
<LI>The gin must be good quality
</MENU>

< DIR >
<LI>A-H<LI>I-M
<LI>M-R<LI>S-Z
</DIR>
```

P: Метка параграфа

Пустой элемент P служит разделителем параграфов. Конкретная процедура обработки (отступы, инструкции и т.д.) здесь не оговаривается и может зависеть от наличия иных меток, стилей и т.д.

Метка <P> ставится между двумя частями текста для их разделения.

Нет нужды применять <P> для создания пустого места вокруг заголовка, списка, адреса или выделенных элементов, которые уже сами по себе предполагают наличие разделителей параграфов. Создание пустых мест вокруг перечисленных элементов - обязанность программы обработки. Соседство метки разделителя параграфов и такого элемента, который сам автоматически создает разделители параграфов, может привести к непредсказуемым последствиям. Следует избегать того, чтобы метке разделителя параграфов предшествовал или следовал за ней такой элемент.

Обычно метка <P> создает небольшой вертикальный пропуск между параграфами (одна строка или полстроки). Этого не происходит (как правило) в тексте элементов ADDRESS и (даже) PRE. В некоторых версиях в обычном тексте метка <P> может также создавать небольшой отступ слева в первой строке открываемого ею параграфа.

Примеры использования

```
<h1>What to do</h1>
This is a one paragraph.<p>This is a second.
< P >
This is a third.
```

Выделение символов

Элементы выделения позволяют форматировать отдельные части текста особым образом, производить выделение и т.д. Метки выделения не приводят к разбиению на параграфы и могут применяться к отдельным кускам текста

внутри параграфов. Как и все метки, не поддерживаемые имеющимися версиями анализаторов языка HTML, эти метки будут игнорироваться, однако размеченный ими текст будет обрабатываться как и любой другой.

Все метки выделения имеют соответствующие им завершающие метки, как в этом примере

```
This is <EM>emphasized</EM> text.
```

Практическая реализация одних стилей выделения более очевидна. Для других - менее. Логические стили можно применять в любом месте, если, к примеру, нет нужды ссылаться в тексте на процедуру форматирования (например, "обязательно использование наклонного шрифта для отдельных частей текста").

Замечание

Анализаторы, не способные изобразить какой-либо стиль выделения символов, могут представить его с некоторой потерей качества изображения с применением альтернативного стиля или стиля по умолчанию. Некоторые версии анализаторов могут игнорировать все метки, так что поставщикам информации желательно не придавать меткам выделения большую смысловую нагрузку.

Физические стили

- TT Шрифт фиксированной ширины.
- B Толстый или еще каким-либо образом выделенный шрифт.
- I Наклонный шрифт (или искаженный каким-либо образом, если просто наклон невозможен).
- U Подчеркивание.

Логические стили

- EM Выделение символов (обычно наклон шрифта). (смысловое усиление определенного слова или фразы)
- STRONG Более четкое выделение (обычно применение более жирного шрифта). (выделение, привлечение внимания)
- CODE Пример кода. Обычно фиксированный шрифт (не путать с элементом PRE). (формулы, выражения.)
- SAMP Последовательность символов. (названия команд, примеры)
- KBD Текст, набираемый пользователем. Этот стиль применяется в описаниях.
- VAR Имя переменной. (имена переменных в примерах, формулах)
- DFN Пример определения к какому-либо термину. Обычно жирный наклонный шрифт или просто жирный. (Официально считается расширением в версии HTML 2.0)
- CITE Цитата. Обычно наклонный шрифт. (названия документов, выдержки из документов, цитируемые фразы и т.д)

Пример использования

```
This text contains an <em>emphasized</em> word.
<strong>Don't assume</strong> that it will be italic!
It was made using the <CODE>EM</CODE> element. A citation
is
typically italic and was no formal necessary structure:
<cite>Moby Dick></cite> is a book title.
```

FORM тэг в HTML документах

Синтаксис

FORM тэг определяет форму для заполнения в HTML документе. В одном документе может быть определено несколько форм для заполнения, но

вложенные FORM операторы не разрешены. Формат оператора FORM выглядит следующим образом:

```
<FORM ACTION="url" METHOD="POST">...</FORM>
```

Его атрибуты следующие:

- ACTION - URL сервера запросов, куда будет отослано содержание формы после подтверждения. Если это поле отсутствует, будет использован URL текущего документа.
- METHOD - HTTP/1.0 метод используемый для посылки содержания заполненной формы на сервер. Этот метод зависит от того, как работает конкретный сервер запросов. Настоятельно рекомендуется использование метода POST. Возможные варианты следующие:

    - GET - это метод по умолчанию, который приводит к добавлению содержимого заполненной формы к URL, как и в нормальном запросе.
    - POST при использовании этого метода содержимое заполненной формы пересылается не как часть URL, а как содержимое тела запроса.

- ENCTYPE - задает тип кодирования содержимого заполненной формы. Этот атрибут действует только когда используется метод POST и даже в этом случае имеет только одно возможное значение (которое является значением по умолчанию)- application/x-www-form-urlencoded.

Внутри FORM оператора может находиться все, что угодно, кроме другого оператора FORM. Согласно спецификации, для задания интерфейсных элементов внутри оператора FORM используются тэги INPUT, SELECT, и

TEXTAREA.

INPUT - Тэг INPUT используется для задания простого элемента ввода внутри FORM. Это одиночный тэг, его ничего не окружает и он не имеет закрывающего тэга - т.е. он используется подобно тэгу IMG.

Атрибуты для тэга INPUT следующие:

TYPE - должен быть один из:

- "hidden" : пользователю не предлагается поля для ввода, но содержимое тэга передается при подтверждении и посылке формы. Это значение может быть использовано для передачи информации состояния при взаимодействии клиента и сервера.
- "image" : картинка, по которой вы можете сделать щелчок мышью или другим указывающим устройством, что приводит к немедленному подтверждению и отсылке формы. Координаты выбранной точки измеряются в точках от верхнего левого угла и возвращаются (наряду с другими компонентами формы) точно так же, как для элемента Image.
- "text" поле ввода текста, значение по умолчанию
- "password" (поле ввода пароля; вводимые символы представляются как звездочки)
- "checkbox" (кнопка, принимающая положения on (включено) и off (выключено))
- "radio" (кнопка, принимающая положения on и off; причем остальные кнопки с тем же параметром NAME ведут себя по принципу "одна из многих")
- "submit" (кнопка, действие которой сводится к отсылке содержимого заполненной формы на сервер запросов)
- "reset" (кнопка, которая устанавливает во всех интерфейсных элементах значения по умолчанию)

NAME - символическое имя (оно не показывается) для этого поля ввода. Это поле должно присутствовать для всех полей ввода кроме "submit" и "reset", т.к. оно используется в строке запроса при идентификации поля ввода при посылке ее на сервер после подтверждения формы.

VALUE - для полей ввода текста или пароля, может быть использовано для задания начального содержания поля. Для checkbox или radio button, VALUE задает значение кнопки, когда она находится в отмеченном состоянии (неотмеченные кнопки опускаются при посылке запроса); значение по умолчанию для checkbox или radio button - "on". Для типов "submit" и "reset", VALUE может быть использовано для задания надписи на этих кнопках.

CHECKED (значение не требуется) - указывает, что данная кнопка типа checkbox или radio button отмечена по умолчанию; это поле работает только для кнопок типа checkbox и radio button.

SIZE - физический размер поля ввода в символах; это поле действует только для элементов ввода текста или пароля. Если не присутствует явно, выставляется 20. Многострочные поля ввода текста могут быть заданы с помощью SIZE=ширина,высота; например SIZE=60,12. Заметим, что SIZE атрибут не должен использоваться для задания многострочных полей ввода, т.к. можно воспользоваться тэгом TEXTAREA.

MAXLENGTH - максимальное количество введенных символов, которые будут приниматься для ввода, верно только для полей ввода текста и пароля (и только в однострочных элементах). По умолчанию - неограниченно. Подразумевается, что поля ввода должны прокручиваться.

Тэг SELECT

Внутри <FORM> ... </FORM>, может присутствовать любое количество тэгов SELECT, свободно перемешанных с другими HTML элементами (включая INPUT и TEXTAREA элементы) и текстом (но не дополнительных элементов FORM). Тэг SELECT во многих графических клиентах представляется как

меню или список.

В отличие от INPUT, SELECT имеет открывающий и закрывающий тэги. Внутри оператора SELECT разрешена только последовательность тэгов OPTION, за каждым из которых следует некоторое количество простого текста (без HTML выражений), например:

```
<SELECT NAME="a-menu">
<OPTION> First option.
<OPTION> Second option.
</SELECT>
```

Атрибуты оператора SELECT следующие:

- NAME - символическое имя для этого SELECT элемента. Это поле должно присутствовать, т.к. оно используется при посылке запроса (аналогично оператору INPUT).
- SIZE - если SIZE равен 1 или если этот атрибут опущен, по умолчанию SELECT будет представлен как меню опций Motif. Если SIZE = 2 или более, SELECT будет представлен как окно выбора; значение SIZE тогда будет определять, сколько элементов списка будут видны.
- MULTIPLE - если присутствует (нет значения), задает, что SELECT должен позволять множественный выбор из списка. Наличие MULTIPLE принуждает SELECT быть представленным как список выбора, вне зависимости от значения SIZE.

Атрибуты OPTION следующие:

- SELECTED задает, что эта опция выбрана по умолчанию. Если SELECT позволяет множественный выбор (с помощью атрибута MULTIPLE), как

SELECTED могут быть помечены несколько опций.

- VALUE – значение переменной, передаваемой из формы

Тэг TEXTAREA

Тэг TEXTAREA может быть использован для расположения многострокового поля ввода с необязательным содержимым по умолчанию в форме. Атрибуты тэга TEXTAREA следующие:

- NAME символическое имя поля ввода.
- ROWS число строк в поле ввода(высота).
- COLS число столбцов в поле ввода (ширина).

TEXTAREA имеет полосы прокрутки, так что может быть введено любое количество текста. Элемент TEXTAREA требует и открывающий и закрывающий тэги. TEXTAREA без содержания по умолчанию выглядит примерно так:

```
<TEXTAREA NAME="foo" ROWS=4 COLS=40></TEXTAREA>
```

TEXTAREA с содержанием по умолчанию выглядит так:

```
<TEXTAREA NAME="foo" ROWS=4 COLS=40>
Default contents go here.
</TEXTAREA>
```

Содержание по умолчанию должно быть строгим ASCII текстом. Символы перевода строки воспринимаются (так что в примере выше до и после текста "Default contents go here." будет пустая строка).

Подтверждение и посылка формы

Для метода GET

Когда нажимается кнопка submit, содержимое формы будет добавлено к URL в следующей форме:

```
action?name=value&name=value&name=value
```

(здесь "action" - URL, заданное атрибутом ACTION в тэге FORM, или URL текущего документа, если атрибут ACTION не был задан).

Нестандартные символы в примерах "name" или "value" будут опущены, при этом имеются в виду также "=" and "&". Это означает, что включения "=", которые разделяют имена и значения (names и values), и включения "&", которые разделяют пары name/value, не опускаются.

Для полей ввода текста и пароля, значением будет то, что введет пользователь. Если пользователь оставит это поле пустым, значение value также будет пустым , но в строке запроса будет присутствовать фрагмент "name=".

Для кнопок типа checkbox и radio button, значение value определяется атрибутом VALUE в том случае, когда кнопка отмечена. Неотмеченные кнопки при составлении строки запроса игнорируются целиком. Несколько кнопок типа checkbox могут иметь один атрибут NAME (и различные VALUE), если это необходимо. Кнопки типа radio button предназначены для того, чтобы вести себя по принципу "одна из всех" и должны иметь одинаковый атрибут NAME и различные атрибуты VALUE.

Для метода POST

Содержимое формы кодируется точно как для метода GET (см. выше), но вместо добавления содержимого формы к URL, заданной атрибутом формы ACTION, в качестве запроса , содержимое посылается блоком данных как часть

операции POST. Если присутствует атрибут ACTION, то значение URL, которое там находится, определяет, куда послать этот блок данных. Этот метод особенно рекомендуется при посылке больших блоков данных.

Язык программирования PHP

PHP был задуман примерно в конце 1994 года Расмусом Ледорфом (Rasmus Lerdorf). Ранние невыпущенные версии использовались на его домашней странице для того, чтобы следить за тем кто просматривал его интерактивное резюме. Первая используемая версия стала доступна где-то в начале 1995 и была известна как Personal Home Page Tools. Она состояла из очень упрощенного движка синтаксического анализатора, который понимал только несколько специальных макрокоманд и ряд утилит, которые затем были в общем использовании на домашних страницах. Гостевые книги, счетчики и некоторые другие дополнения.

Довольно трудно дать какую-либо жесткую статистику, но отмечено, что к 1996 г. PHP/FI был использован по крайней мере на 15,000 веб-сайтах во всем мире. В середине 1997г. эта цифра выросла до более чем 50,000. В середине 1997г. также наблюдалось изменение в разработке PHP. Из частного любимого проекта Расмуса, которому способствовала горстка людей , это превратилось в намного более организованную рабочую группу. Синтаксический анализатор был заново переписан Зевом Сураски(Zeev Suraski) и Анди Гутмансом(Andi Gutmans), и этот новый синтаксический анализатор стал основой для PHP Версии 3.

Возможности PHP

HTTP-аутентификация средствами PHP

HTTP аутентификация в PHP доступна только при использовании модуля Apache. В модуле Apache PHP-скрипт, может использовать функцию Header() для отправки сообщения "Authentication Required" браузеру клиента, вызвав тем

самым окно диалога Username/Password. Как только пользователь заполняет поля username и password, URL содержащий PHP-скрипт будет вызван заново с переменными $PHP_AUTH_USER, $PHP_AUTH_PW и $PHP_AUTH_TYPE содержащими введенную информацию. В данном случае обеспечивается только "Основная" аутификация.

Фрагмент примера сценария, который производит аутентификацию клиента на странице, должен быть следующим:

```
<?php
  if(!isset($PHP_AUTH_USER)) {
    Header("WWW-Authenticate: Basic realm=\"My Realm\"");
    Header("HTTP/1.0 401 Unauthorized");
    echo "Text to send if user hits Cancel button\n";
    exit;
  } else {
    echo "Hello $PHP_AUTH_USER.<P>";
    echo "You entered $PHP_AUTH_PW as your password.<P>";
  }
?>
```

Вместо просто распечатывания $PHP_AUTH_USER и $PHP_AUTH_PW, Вы вероятно хотели бы проверить имя_пользователя и пароль для проверки правильности. Возможно, посылая запрос к базе данных, или, ища пользователя в dbm или текстовом файле.

Будьте внимательны при использовании браузера Internet Explorer. Он весьма придирчив к порядку заголовков. Отправка заголовка WWW-Authenticate перед заголовком HTTP/1.0 401 возможно даст аутентификацию в любом случае.

Чтобы предотвратить от записи кем-то сценарий , который определяет пароль для страницы, которая была опознана через традиционный внешний механизм, PHP_AUTH переменные не будут установлены, если допускается внешнее установление подлинности для той специфической страницы. В этом случае может быть использована переменная $REMOTE_USER чтобы идентифицировать внешне-опознанного пользователя.

Обратите внимание, однако, что вышеупомянутое не защищает от кого - то, кто может управлять не-аутентифицированным URL используя перехваченный пароль из аутентифицированных URL на том же самом сервере.

И Netscape и Internet Explorer очистит локальный кэш окна аутентификации после получения ответа сервера 401. Это эффективно как мера отключения пользователей("logout"), вынуждающая их повторно ввести их username и пароль. Некоторые используют это для отключения пользователя по истечении интервала времени("time out"), или обеспечивают кнопку "Log Out".

Поддержка file upload

PHP может принимать файлы, загруженные из любого браузера, отвечающего стандартам RFC-1867 (которыми являются, например, Netscape Navigator 3 или старше, Microsoft Internet Explorer 3 с исправлениями от Microsoft, или старше). Эта возможность позволяет людям загружать файлы. С PHP-аутификацией и функциями манипулирования файлами, вы имеете полный контроль над тем, кому позволять загружать файлы и что должно быть выполнено с файлом, если он был загружен.

Экран загрузки файла может быть организован созданием специальной формы, которая выглядит примерно так:

```
<FORM ENCTYPE="multipart/form-data" ACTION="_URL_"
METHOD=POST>
<INPUT TYPE="hidden" name="MAX_FILE_SIZE" value="1000">
Send this file: <INPUT NAME="userfile" TYPE="file">
<INPUT TYPE="submit" VALUE="Send File">
</FORM>
```

URL должен указать на php html файл. Скрытое поле MAX_FILE_SIZE должно предшествовать полю ввода файла и означает максимально допустимый размер файла. Значение определяется в байтах. Для этого файла при успешной загрузке будут определены следующие переменные :

- $userfile - Временное имя файла под которым загруженный файл загружается в машину сервера.
- $userfile_name - Исходное имя файла в системе отправителя.
- $userfile_size - Размер загруженного файла в байтах.
- $userfile_type - Тип MIME файла, если броузер предоставил эту информацию. Например может быть "image/gif". Обратите внимание, что компонент вышеупомянутых переменных "$userfile" - это любое значение поля Name тега INPUT с TYPE=file обозначенное в форме загрузки. В приведенном выше примере формы загрузки мы назвали его "userfile".

По умолчанию файлы будут сохранены в заданном по умолчанию временном каталоге сервера. Его можно изменить, установкой переменной среды TMPDIR в среде, в которой PHP выполняется. Хотя, использование при ее установке обращения PutEnv () изнутри сценария PHP не будет работать.

Скрипт PHP, который получает загруженный файл, должен определить, что должно быть выполнено с загруженным файлом. Вы можете, например, использовать переменную $file_size, чтобы отбросить любые файлы, которые

являются или слишком маленькими или слишком большими. Вы могли бы использовать переменную $file_type, чтобы отбросить любые файлы, которые не соответствуют некоторым критериям типа. В любом случае, вы должны или удалить файл из временного каталога или переместить эго в другое место.

Файл будет удален из временного каталога в конце запроса, если он не перемещен или переименован.

Поддержка HTTP cookie

PHP поддерживает HTTP cookies. Cookies - механизм для сохранения данных в удаленном браузере и, таким образом, - трэкинг или идентификация пользователей. Вы можете устанавливать файлы cookie используя функцию setcookie(). Cookies - часть HTTP заголовка, так что функция SetCookie() должна вызваться прежде чем браузеру послан какая-нибудь информация для вывода. Это - то же самое ограничение, которое касается и функции Header().

Любой cookie, посланный Вам от клиента будет автоматически превращен в переменную PHP точно так же как данные методов GET и POST. Если вы желаете назначить множественные значения одиночному cookie - просто добавьте [] к имени cookie. Для более подробной информации см. функцию setcookie ().

Поддержка баз данных

PHP поддерживает ряд различных баз данных, и в режиме работы в собственной системе команд и через ODBC, включая:

- Adabas D
- MySQL
- dBase
- Oracle
- Empress
- PostgreSQL

- FilePro
- Solid
- Informix
- Sybase
- InterBase
- Velocis
- mSQL
- Unix dbm

Регулярные выражения

Регулярные выражения используются для сложного манипулирования строками в PHP. Функции, которые поддерживают регулярные выражения:

- ereg()
- ereg_replace()
- eregi()
- eregi_replace()
- split()

Все эти функции принимают строку регулярного выражения как их первый параметр. PHP использует расширенные регулярные выражения POSIX как определено в POSIX 1003.2. Для полного описания регулярных выражений POSIX см. соответствующие разделы руководства (regex), в каталоге regex дистрибутива PHP.

Пример регулярных выражений

```
ereg("abc",$string);
/* Возвращает 'истина', если "abc"
   найдено в строке $string. */
```

```
    ereg("^abc",$string);
    /* Возвращает 'истина', если "abc"
       найдено в начале строки $string. */

    ereg("abc$",$string);
    /*  Возвращает 'истина', если "abc"
       найдено в конце строки $string. */

    eregi("(ozilla.[23]|MSIE.3)",$HTTP_USER_AGENT);
    /*  Возвращает 'истина', если браузер клиента
       - Netscape 2, 3 или MSIE 3. */

    ereg("([[:alnum:]]+) ([[:alnum:]]+) ([[:alnum:]]+)",
         $string,$regs);
    /* Помещает три слова - $regs[1], $regs[2] и
$regs[3], разделенные пробелом. */

    ereg_replace("^","<BR>",$string);
    /* Устанавливает тег <BR> в начало строки $string. */

    ereg_replace("$","<BR>",$string);
    /* Устанавливает тег <BR> в конец строки $string. */

    ereg_replace("\n","",$string);
    /* Отсекает символ "возврат каретки" в строке
$string. */
```

Обработка ошибок

В PHP есть 4 типа ошибок и предупреждений. Это:

- 1 - Нормальные Ошибки Функции(Normal Function Errors)
- 2 - Нормальные Предупреждения(Normal Warnings)
- 4 - Ошибки Синтаксического Анализатора(Parser Errors)
- 8 - Уведомления(Notices) : предупреждения, которые Вы можете проигнорировать но, которые могут подразумевать ошибки в вашем коде

Эти 4 типа комбинируются при определении ошибки, сообщая уровень. Ошибка по умолчанию, возвращает уровень 7, который является комбинацией 1 + 2 + 4, или все ошибки за исключением примечаний. Этот уровень может быть изменен в файле php3. ini директивой error_reporting. Он также может быть установлен в вашем файле Apache httpd.conf директивой php3_error_reporting, или же это может быть произведено во времени выполнения сценария, с использованием функции error_reporting ().

Все выражения PHP могут также вызываться с префиксом "@", который выключает сообщение об ошибке, специфичное для этого выражения. Если ошибка произошла во время выполнения такого выражения, и допускается возможность track_errors, Вы можете найти сообщения об ошибках в глобальной переменной $php_errormsg.

Синтаксис и грамматика

Синтаксис PHP заимствован непосредственно из C, Java и Perl также повлияли на синтаксис данного языка.

Переход из HTML

Есть три способа выхода из HTML и перехода в "режим PHP кода":

```
1.  <? echo("простейший способ, инструкция обработки
```

```
SGML\n"); ?>

2.  <?php echo("при работе с XML документами делайте
так\n"); ?>

3.  <script language="php">
        echo ("некоторые редакторы (подобные FrontPage)
не любят обрабатывающие инструкции");
    </script>;

4.  <% echo("От PHP 3.0.4 можно факультативно применять
ASP-тэги"); %>
```

Разделение инструкций

Инструкции (утверждения) разделяются также как в C или Perl - точкой с запятой.

Закрывающий тэг (?>) тоже подразумевает конец утверждения, поэтому следующие записи эквивалентны:

```
<php
    echo "Это тест";
?>

<php echo "Это тест" ?>
```

HTML Формы (GET и POST)

Когда программой-обработчиком формы является PHP-скрипт, переменные этой формы автоматически доступны для данного скрипта PHP.

Например, рассмотрим следующую форму:

```
<form action="foo.php3" method="post">
    Name: <input type="text" name="name"><br>
    <input type="submit">
</form>
```

При активизации формы PHP создаст переменную $name, значением которой будет то содержимое, которое было введено в поле Name: данной формы.

PHP также воспринимает массивы в контексте переменных формы, но только одномерные. Вы можете, например, группировать взаимосвязанные переменные вместе или использовать это свойство для определения значений переменных при множественном выборе на входе:

Более сложные переменные формы:

```
<form action="array.html" method="post">
    Name: <input type="text" name="personal[name]"><br>
    Email: <input type="text" name="personal[email]"><br>
    Beer: <br>
    <select multiple name="beer[]">
        <option value="warthog">Warthog
        <option value="guinness">Guinness
        </select>
    <input type="submit">
</form>
```

Если PHP-атрибут track_vars включен, через установку конфигурации track_vars или директивой <?php_track_vars?>, тогда переменные, активизированные посредством методов POST или GET, будут также находиться в глобальных ассоциативных массивах $HTTP_POST_VARS и $HTTP_GET_VARS соответственно.

PHP очевидным образом поддерживает HTTP cookies, как это определено в Netscape's Spec. Cookies являются механизмом хранения данных в удаленном браузере, используемым для поддержки процедуры обмена или идентификации ответа пользователя. Cookies можно устанавливать используя функцию SetCookie(). Cookies являются частью заголовка HTTP, поэтому функция SetCookie() должна вызываться прежде чем какие-либо передаваемые данные посылаются браузеру. Это такое же ограничение, как и для функции Header(). Любые cookies, посылаемые вам клиентом, автоматически преобразовываются в переменные PHP, также как данные методов GET и POST.

Если необходимо назначить множественные значения одному cookie, просто добавьте квадратные скобки [ ] к имени cookie. Например:

```
SetCookie ("MyCookie[]", "Testing", time()+3600);
```

Учтите, что текущий cookie заменит предыдущий с тем же именем в вашем браузере, если только путь или домен не являются различными. Поэтому, при работе с программами обслуживания карт вы можете использовать для сохранения данных счетчик и посылать его значения дальше и т.п.

Пример функции SetCookie:

```
$Count++;
SetCookie ("Count", $Count, time()+3600);
```

```
SetCookie ("Cart[$Count]", $item, time()+3600);
```

Переменные окружения

PHP автоматически создает переменные окружения, как и обычные переменные.

```
echo $HOME;  /* Показывает переменную окружения HOME,
если она установлена. */
```

Хотя при поступлении информации механизмы GET, POST и Cookie также атоматически создают переменные PHP, иногда лучше явным образом прочитать переменную окружения, для того чтобы быть уверенным в получении ее правильной версии. Для этого может использоваться функция getenv(). Для установки значения переменной окружения пользуйтесь функцией putenv().

Кроме вышеперечисленных, PHP включает в себя массу различных функций, что делает его незаменимым инструментом при создании информационных систем.

Структурированный язык запросов SQL

Одним из основных преимуществ реляционного подхода к организации баз данных (БД) является то, что пользователи реляционных БД получают возможность эффективной работы в терминах простых и наглядных понятий таблиц, их строк и столбцов без потребности знания реальной организации данных во внешней памяти.

Реляционная модель данных, содержащая набор четких предписаний к

базовой организации любой реляционной системы управления базами данных (СУБД), позволяет пользователям работать в ненавигационной манере, т.е. для выборки информации из БД человек должен всего лишь указать список интересующих его таблиц и те условия, которым должны удовлетворять выбираемые данные. СУБД скрывает от пользователя выполняемые ей последовательные просмотры таблиц, выполняя их наиболее эффективным образом. Очень важная особенность реляционных систем состоит в том, что результатом выполнения любого запроса к таблицам БД является также таблица, которую можно сохранить в БД и/или по отношению к которой можно выполнять новые запросы.

Базовым требованием к реляционным СУБД является наличие мощного и в тоже время простого языка, позволяющего выполнять все необходимые пользователям операции. В последние годы таким повсеместно принятым языком стал язык реляционных БД SQL - Structured Query Language (теперь все чаще название языка понимается как Standard Query Language).

До появления SQL в СУБД (независимо от того, на какой модели они основывались) приходилось поддерживать по крайней мере три языка, которые обычно имели мало общего: язык определения данных (ЯОД), служащий для спецификации структур БД (обычно общую структуру БД называют схемой БД); язык манипулирования данными (ЯМД), позволяющий создавать прикладные программы, взаимодействующие с БД; и язык администрирования БД (ЯАДБ), с помощью которого можно было выполнять служебные действия (например, изменять структуру БД или производить ее настройку с целью повышения эффективности). Кроме того, если требовалось предоставить пользователям СУБД интерактивный доступ к БД, приходилось вводить еще один язык, операторы которого выполняются в диалоговом режиме. Язык SQL позволяет решать все эти задачи.

История языка баз данных SQL

Язык для взаимодействия с БД SQL появился в середине 70-х и был разработан в рамках проекта экспериментальной реляционной СУБД System R. Исходное название языка SEQUEL (Structured English Query Language) только частично отражает суть этого языка. Конечно, язык был ориентирован главным образом на удобную и понятную пользователям формулировку запросов к реляционной БД, но на самом деле являлся полным языком БД, содержащим помимо операторов формулирования запросов и манипулирования БД средства определения и манипулирования схемой БД; определения ограничений целостности и триггеров; представлений БД; структур физического уровня, поддерживающих эффективное выполнение запросов; авторизации доступа к отношениям и их полям; точек сохранения транзакции и откатов. В языке отсутствовали средства синхронизации доступа к объектам БД со стороны параллельно выполняемых транзакций: с самого начала предполагалось, что необходимую синхронизацию неявно выполняет СУБД.

В настоящее время SQL реализован практически во всех коммерческих реляционных СУБД, все фирмы провозглашают соответствие своей реализации стандарту SQL, и на самом деле реализованные диалекты SQL очень близки. Это произошло не сразу и не просто.

Наиболее близкими к System R являлись две системы фирмы IBM - SQL/DS и DB2. Как кажется (документальных подтверждений этому автор не имеет), обе эти системы прямо использовали реализацию System R. Отсюда предельная близость реализованных диалектов SQ к SQL System R. Из SQL System R были удалены только те части, которые были недостаточно проработаны (например, точки сохранения) или реализация которых вызывала слишком большие технические трудности (например, ограничения целостности и триггеры). Можно назвать этот путь к коммерческой реализации SQL движением сверху вниз.

Другой подход применялся в таких системах, как Oracle и Informix. Несмотря на различие в способе разработки этих систем, реализация SQL происходила "снизу вверх". В первых выпущенных на рынок реализациях SQL в этих системах использовалось минимальное и очень ограниченное подмножество SQL System R. В частности, в первой известной автору реализации SQL в СУБД Oracle в операторах выборки не допускалось использование вложенных подзапросов.

Тем не менее, несмотря на эти ограничения и на очень слабую на первых порах эффективность, ориентация фирм на поддержание разных аппаратных платформ и заинтересованность пользователей в переходе к реляционным системам позволили фирмам добиться коммерческого успеха и приступить к совершенствованию своих реализаций. В текущих версиях Oracle и Informix поддерживаются достаточно мощные диалекты SQL, хотя реализация иногда вызывает сомнения.

Особенностью большинства современных коммерческих СУБД, затрудняющей анализ существующих диалектов SQL, является отсутствие полного описания языка. Обычно описание разбросано по разным руководствам и перемешано с описанием специфических для данной системы языковых средств, не имеющих отношения к SQL. Тем не менее можно сказать, что базовый набор операторов SQL, включающий операторы определения схемы БД, выборки и манипулирования данными, авторизации доступа к данным, поддержки встраивания SQL в языки программирования и операторы динамического SQL, в коммерческих реализациях относительно устоялся и более или менее соответствует стандарту.

Стандартизация SQL

Деятельность по стандартизации языка SQL началась практически

одновременно с появлением первых его коммерческих реализаций. Первый из числа имеющихся у автора документ датирован октябрем 1985 г. и является уже очередным проектом стандарта ANSI/ISO.

Понятно, что в качестве стандарта нельзя было использовать SQL System R. Во-первых, этот вариант языка не был должным образом технически проработан. Во-вторых, его слишком сложно было бы реализовать (кто знает, как бы сложилась дальнейшая история SQL, если бы были полностью реализованы все идеи System R). С другой стороны, первые коммерческие реализации языка настолько различались, что ни один из реализованных диалектов не имел шансов быть принятым в качестве стандарта.

Анализ доступных документов показывает, что процесс происходил очень сложно с использованием далеко не только научных доводов. В результате, принятый в 1989 г. Международный стандарт SQL во многих частях имеет чрезвычайно общий характер и допускает очень широкое толкование. В этом стандарте полностью отсутствуют такие важные разделы, как манипулирование схемой БД и динамический SQL. Многие важные аспекты языка в соответствии со стандартом определяются в реализации.

Возможно, наиболее важными достижениями стандарта SQL являются четкая стандартизация синтаксиса и семантики операторов выборки и манипулирования данными и фиксация средств ограничения целостности БД, включающих возможности определения первичного и внешних ключей отношений и так называемых проверочных ограничений целостности. Средства определения внешних ключей позволяют легко формулировать требования так называемой целостности БД по ссылкам.

Современное состояние SQL

В этом разделе мы коротко рассмотрим основные особенности стандарта

языка SQL 1989 года. Как было видно из приведенного краткого обзора наиболее продвинутых серверов баз данных, во всех них полностью реализован именно стандарт SQL-89. Поэтому при разработке достаточно сложных информационных систем необходимо хотя бы в общих чертах представлять основные свойства языка.

Типы данных

В языке SQL/89 поддерживаются следующие типы данных: CHARACTER, NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, DOUBLE PRECISION. Эти типы данных классифицируются на типы строк символов, точных чисел и приблизительных чисел.

К первому классу относится CHARACTER. Спецификатор типа имеет вид CHARACTER (lenght), где lenght задает длину строк данного типа. Заметим, что в SQL/89 нет типа строк переменного размера, хотя во многих реализациях они допускаются. Литеральные строки символов изображаются в виде 'последовательность-символов' (например, 'example').

Представителями второго класса типов являются NUMERIC, DECIMAL (или DEC), INTEGER (или INT) и SMALLINT. Спецификатор типа NUMERIC имеет вид NUMERIC [(precision [, scale]). Специфицируются точные числа, представляемые с точностью precision и масштабом scale. Здесь и далее, если опущен масштаб, то он полагается равным 0, а если опущена точность, то ее значение по умолчанию определяется в реализации.

Спецификатор типа DECIMAL (или DEC) имеет вид NUMERIC [(precision [, scale]). Специфицируются точные числа, представленные с масштабом scale и точностью, равной или большей значения precision.

INTEGER специфицирует тип данных точных чисел с масштабом 0 и

определяемой в реализации точностью. SMALLINT специфицирует тип данных точных чисел с масштабом 0 и определяемой в реализации точностью, не большей, чем точность чисел типа INTEGER.

Литеральные значения точных чисел в общем случае представляются в форме [+ -] <целое-без-знака> [.<целое-без-знака>].

Наконец, к классу типов данных приблизительных чисел относятся типы FLOAT, REAL и DOUBLE PRECISION. Спецификатор типа FLOAT имеет вид FLOAT [(precision)]. Специфицируются приблизительные числа с двоичной точностью, равной или большей значения precision.

REAL специфицирует тип данных приблизительных чисел с точностью, определенной в реализации. DOUBLE PRECISION специфицирует тип данных приблизительных чисел с точностью, определенной в реализации, большей, чем точность типа REAL.

Литеральные значения приблизительных чисел в общем случае представляются в виде <литеральное-значение-точного-числа>E<целое-со-знаком>.

Заметим, что хотя с использованием языка SQL можно определить схему БД, содержащую данные любого из перечисленных типов, возможность использования этих данных в прикладных системах зависит от применяемого языка программирования. Весь набор типов данных можно использовать, только если программировать на ПЛ/1. Поэтому в некоторых реализациях SQL типы данных с масштабом и точностью вообще не поддерживаются.

Хотя правила встраивания SQL в программы на языке Си не определены в SQL/89, в большинстве реализаций, поддерживающих такое встраивание,

имеется следующее соответствие между типами данных SQL и типами данных Си: CHARACTER соответствует строкам Си; INTEGER соответствует long; SMALLINT соответствует short; REAL соответствует float; DOUBLE PRECISION соответствует double (именно такое соответствие утверждено в стандарте SQL/92).

Заметим еще, что в большинстве реализаций SQL поддерживаются некоторые дополнительные типы данных, например, DATE, TIME, INTERVAL, MONEY. Некоторые из этих типов специфицированы в стандарте SQL/92, но в текущих реализациях синтаксические и семантические свойства таких типов могут различаться.

Средства определения схемы

Средства определения схемы БД в стандарте SQL/89 относятся к наиболее слабым и допускающим различную интерпретацию частям стандарта. Более того, мне неизвестна ни одна реализация, в которой поддерживался бы в точности такой набор средств определения схемы.

Поэтому, чтобы добиться мобильности прикладной системы в достаточно широком классе реализаций SQL/89, необходимо тщательно локализовать компоненты определения схемы БД. Думаю, что лучше всего сосредоточить всю работу со схемой БД в одном модуле и иметь в виду, что при переходе к другой СУБД очень вероятно потребуется переделка этого модуля.

Особо отметим, что в SQL/89 вообще отсутствуют какие-либо средства изменения схемы БД: нет возможности удалить схему таблицы, добавить к схеме таблицы новый столбец и т.д. Во всех реализациях такие средства поддерживаются, но они могут различаться и синтаксисом, и семантикой.

Несмотря на отсутствие особых надежд на то, что удастся встретить

реализацию, поддерживающую язык определения схем SQL/89, мы коротко опишем этот язык (без синтаксических деталей), чтобы оценить на содержательном уровне возможности SQL/89 в этой части и получить хотя бы какие-то средства сравнения разных реализаций.

Оператор определения схемы

В соответствии с правилами SQL/89 каждая таблица данной БД имеет простое и квалифицированное имена. В качестве квалификатора имени выступает "идентификатор полномочий" таблицы, который обычно в реализациях совпадает с именем некоторого пользователя, квалифицированное имя таблицы имеет вид:

```
<идентификатор полномочий>.<простое имя>
```

Подход к определению схемы в SQL/89 состоит в том, что все таблицы с одним идентификатором полномочий создаются (определяются) путем выполнения одного оператора определения схемы. При этом в стандарте не определяется способ выполнения оператора определения схемы: должен ли он выполняться только в интерактивном режиме или может быть встроен в программу, написанную на традиционном языке программирования.

В операторе определения схемы содержится идентификатор полномочий и список элементов схемы, каждый из которых может быть определением таблицы, определением представления (view) или определением привилегий. Каждое из этих определений представляется отдельным оператором SQL/89, но все они, как уже говорилось, должны быть встроены в оператор определения схемы.

Для этих операторов мы приведем синтаксис, поскольку это позволит более четко описать их особенности.

Определение таблицы

Оператор определения таблицы имеет следующий синтаксис:

```
<table definition> ::=
 CREATE TABLE <table name> (<table element> [{,<table
 element>}...])
<table element> ::=
 <column definition>
 <table constraint definition>
```

Кроме имени таблицы, в операторе специфицируется список элементов таблицы, каждый из которых служит либо для определения столбца, либо для определения ограничения целостности определяемой таблицы. Требуется наличие хотя бы одного определения столбца. Оператор CREATE TABLE определяет так называемую базовую таблицу, то есть реальное хранилище данных.

Для определения столбцов таблицы и ограничений целостности используются специальные операторы, которые должны быть вложены в оператор определения таблицы.

Определение столбца

Оператор определения столбца описывается следующими синтаксическими правилами:

```
<column definition> ::=
 <column name> <data type>
 [<default clause>] [<column constraint>...]
<default clause> ::=
```

```
 DEFAULT { <literal> | USER | NULL }
<column constraint> ::=
 NOT | NULL [<unique specification>]
| <references specification>
| CHECK (<search condition>)
```

Как видно, кроме обязательной части, в которой определяется имя столбца и его тип данных, определение столбца может содержать два необязательных раздела: раздел значения столбца по умолчанию и раздел ограничений целостности столбца.

В разделе значения по умолчанию указывается значение, которое должно быть помещено в строку, заносимую в данную таблицу, если значение данного столбца явно не указано. Значение по умолчанию может быть указано в виде литеральной константы с типом, соответствующим типу столбца; путем задания ключевого слова USER, которому при выполнении оператора занесения строки соответствует символьная строка, содержащая имя текущего пользователя (в этом случае столбец должен иметь тип символьных строк); или путем задания ключевого слова NULL, означающего, что значением по умолчанию является неопределенное значение. Если значение столбца по умолчанию не специфицировано, и в разделе ограничений целостности столбца указано NOT NULL, то попытка занести в таблицу строку с неспецифицированным значением данного столбца приведет к ошибке.

Указание в разделе ограничений целостности NOT NULL приводит к неявному порождению проверочного ограничения целостности для всей таблицы (см. следующий подраздел) "CHECK (C IS NOT NULL)" (где C - имя данного столбца). Если ограничение NOT NULL не указано, и раздел умолчаний отсутствует, то неявно порождается раздел умолчаний DEFAULT NULL. Если указана спецификация уникальности, то порождается

соответствующая спецификация уникальности для таблицы.

Если в разделе ограничений целостности указано ограничение по ссылкам данного столбца (<reference specification>), то порождается соответствующее определение ограничения по ссылкам для таблицы:

```
FOREIGN KEY(C) <reference specification>.
```

Наконец, если указано проверочное ограничение столбца, то условие поиска этого ограничения должно ссылаться только на данный столбец, и неявно порождается соответствующее проверочное ограничение для всей таблицы.

Определение ограничений целостности таблицы

Ограничения целостности таблицы обладают следующим синтаксисом:

```
<table constraint definition> ::=
 <unique constraint definition>
 <referential constraint definition>
 <check constraint definition>
<unique constraint definition> ::=
 <unique specification> (<unique column list>)
<unique specification> ::= UNIQUE PRIMARY KEY
<unique column list> ::= <column name> [{,<column
name>}...]
<referential constraint definition> ::=
 FOREIGN KEY (<referencing columns>) <references
specification>
<references specification> ::=
 REFERENCES <referenced table and columns>
```

```
<referencing columns> ::= <reference column list>
<referenced table and columns> ::=
 <table name> [(<reference column list>)]
<reference column list> ::= <column name> [{,<column
name>}...]
<check constraint definition> ::= CHECK (<search
condition>)
```

Для одной таблицы может быть задано несколько ограничений целостности, в том числе те, которые неявно порождаются ограничениями целостности столбцов. Стандарт SQL/89 устанавливает, что ограничения таблицы фактически проверяются при выполнении каждого оператора SQL.

Замечание: Наличие правильно подобранного набора ограничений БД очень важно для надежного функционирования прикладной информационной системы. Вместе с тем, в некоторых СУБД ограничения целостности практически не поддерживаются. Поэтому при проектировании прикладной системы необходимо принять решение о том, что более существенно: рассчитывать на поддержку ограничений целостности, но ограничить набор возможных СУБД, или отказаться от их использования на уровне SQL, сохранив возможность использования не самых современных СУБД.

Далее T обозначает таблицу, для которой определяются ограничения целостности.

Ограничение уникальности

Каждое имя столбца в списке уникальности должно именовать столбец T и не должно входить в этот список более одного раза. При определении столбца, входящего в список уникальности, должно быть указано ограничение столбца NO NULL. Среди ограничений уникальности T не должно быть более одного

определения первичного ключа (ограничения уникальности с ключевым словом RIMARY KEY).

Действие ограничения уникальности состоит в том, что в таблице T не допускается появление двух или более строк, значения столбцов уникальности которых совпадают.

Ограничение по ссылкам

Ограничение по ссылкам от заданного набора столбцов CT таблицы T на заданный набор столбцов CT1 некоторой определенной к этому моменту таблицы T1 определяет условие на содержимое обеих этих таблиц, при котором ссылки можно считать корректными.

Если список столбцов CT1 явно специфицирован в определении ограничения по ссылкам, то требуется, чтобы этот список явно входил в какое-либо определение уникальности таблицы T1. Если же список CT1 не специфицирован явно в определении ограничения по ссылкам таблицы T, то требуется, чтобы в определении таблицы T1 присутствовало определение первичного ключа, и список CT1 неявно полагается совпадающим со списком имен столбцов из определения первичного ключа таблицы T1. Имена столбцов списков CT и CT1 должны именовать столбцы таблиц T и T1 соответственно и не должны появляться в списках более одного раза. Списки столбцов CT и CT1 должны содержать одинаковое число элементов, и столбец таблицы T, идентифицируемый i-ым элементом списка CT должен иметь тот же тип, что столбец таблицы T1, идентифицируемый i-ым элементом списка CT1.

По определению, таблицы T и T1 удовлетворяют заданному ограничению по ссылкам, если для каждой строки s таблицы T такой, что все значения столбцов s, идентифицируемых списком CT, не являются неопределенными, существует строка s1 таблицы T1 такая, что значения столбцов s1,

идентифицируемых списком CT1, позиционно равны значениям столбцов s, идентифицируемых списком CT. По человечески это можно сформулировать так: ограничение по ссылкам удовлетворяется, если для каждой корректной ссылки существует объект, на который она ссылается. В привычной программистам терминологии, ограничение по ссылкам не позволяет производить "висячие" ссылки, не ведущие ни к какому объекту.

Проверочное ограничение

Проверочное ограничение специфицирует условие, которому должна удовлетворять в отдельности каждая строка таблицы T. Это условие не должно содержать подзапросов, спецификаций агрегатных функций, а также ссылок на внешние переменные или параметры. В него могут входить только имена столбцов данной таблицы и литеральные константы.

Таблица удовлетворяет проверочному ограничению целостности в том и только в том случае, когда вычисление условия для каждой строки таблицы дает true.

Замечание: В некоторых реализациях допускаются расширенные механизмы ограничений по ссылкам и проверочных ограничений. Следует быть внимательным, если не желать выходить за пределы возможностей стандарта.

Определение представлений

Механизм представлений (view) является мощным средством языка SQL, позволяющим скрыть реальную структуру БД от некоторых пользователей за счет определения представления БД, которое реально является некоторым хранимым в БД запросом с именованными столбцами, а для пользователя ничем не отличается от базовой таблицы БД (с учетом технических ограничений). Любая реализация должна гарантировать, что состояние представляемой таблицы точно соответствует состоянию базовых таблиц, на которых

определено представление. Обычно вычисление представляемой таблицы (материализация соответствующего запроса) производится каждый раз при использовании представления.

В стандарте SQL/89 оператор определения представления имеет следующий синтаксис:

```
<view definition> ::=
 CREATE VIEW <table name> [(<view column list>)] AS
<query
 specification> [WITH CHECK OPTION]
<view column list> ::= <column name> [{,<column
name>}...]
```

Определяемая представляемая таблица V является изменяемой (то есть по отношению к V можно использовать операторы DELETE и UPDATE) в том и только в том случае, если выполняются следующие условия для спецификации запроса: в списке выборки не указано ключевое слово DISTINCT; каждое арифметическое выражение в списке выборки представляет собой одну спецификацию столбца, и спецификация одного столбца не появляется более одного раза; в разделе FROM указана только одна таблица, являющаяся либо базовой таблицей, либо изменяемой представляемой таблицей; в условии выборки раздела WHERE не используются подзапросы; в табличном выражении отсутствуют разделы GROUP BY и HAVING.

Замечание: Эти ограничения являются очень сильными. В реализациях они могут быть ослаблены. Но если стремиться к мобильности, не следует пользоваться расширенными возможностями.

Если в списке выборки спецификации запроса имеется хотя бы одно

арифметическое выражение, состоящее не из одной спецификации столбца, или если одно имя столбца участвует в списке выборки более одного раза, определение представления должно содержать список имен столбцов представляемой таблицы. Более просто, нужно явно именовать столбцы представляемой таблицы, если эти имена не наследуются от столбцов таблиц раздела FROM спецификации запроса.

Требование WITH CHECK OPTION в определении представления имеет смысл только в случае определения изменяемой представляемой таблицы, которая определяется спецификацией запроса, содержащей раздел WHERE. При наличии этого требования не допускаются изменения представляемой таблицы, которые приводят к появлению в базовых таблицах строк, не видимых в представляемой таблице (т.е. таких строк, которые не удовлетворяют условию поиска раздела WHERE спецификации запроса). Если WITH CHECK OPTION в определении представления отсутствует, такой контроль не производится.

Определение привилегий

В соответствии с идеологией языка SQL контроль прав доступа данного пользователя к таблицам БД производится на основе механизма привилегий. Фактически, этот механизм состоит в том, что для выполнения любого действия над таблицей пользователь должен обладать соответствующей привилегией (реально все возможные действия описываются фиксированным стандартным набором привилегий). Пользователь, создавший таблицу, автоматически становится владельцем всех возможных привилегий на выполнение операций над этой таблицей. В число этих привилегий входит привилегия на передачу всех или некоторых привилегий по отношению к данной таблице другому пользователю, включая привилегию на передачу привилегий. Иногда поддерживается и обратная операция изъятия привилегий от пользователя, ранее их получившего.

В SQL/89 определяется упрощенная схема механизма привилегий. Во-первых, "раздача" привилегий возможна только при определении таблицы. Во-вторых, пользователь, получивший некоторые привилегии от других пользователей, может передать их дальше только при определении схемы.

Определение привилегий производится в следующем синтаксисе:

```
<privilege definition> ::=
 GRANT <privileges> ON <table name> TO <grantee>
 [{,<grantee>}...] [WITH GRANT OPTION]
<privileges> ::=
 ALL PRIVILEGES
 <action> [{,<action>}...]
<action> ::= SELECT INSERT DELETE
 UPDATE [(<grant column list>)]
 REFERENCES [(<grant column list>]
<grant column list> ::= <column name> [{,<column
name>}...]
<grantee> ::= PUBLIC <authorization identifier>
```

Смысл механизма определения привилегий в SQL/89 достаточно понятен из этого синтаксиса. Заметим лишь, что привилегией REFERENCES по отношению к указанным столбцам таблицы T1 необходимо обладать, чтобы иметь возможность при определении таблицы T определить ограничение по ссылкам между этой таблицей и существующей к этому моменту таблицей T1.

Еще раз заметим, что хотя в общем смысле во всех SQL-ориентированных СУБД поддерживается механизм защиты доступа на основе привилегий, реализации могут различаться в деталях. Это опять то место, которое нужно локализовывать, если стремиться к созданию мобильной прикладной системы.

Средства манипулирования данными

Структура запросов

Для того, чтобы можно было более или менее точно рассказать про структуру запросов в стандарте SQL/89, необходимо начать со сводки синтаксических правил:

```
<cursor specification> ::=
 <query expression> [<order by clause>]
<query expression> ::=
 <query term>
 <query expression> UNION [ALL] <query term>
<query term> ::=
 <query specification>
 (<query expression>)
<query specification> ::=
 (SELECT [ALL DISTINCT] <select list> <table expression>)
<select statement> ::=
 SELECT [ALL DISTINCT] <select list> INTO <select target
 list> <table expression>
<subquery> ::=
 (SELECT [ALL DISTINCT] <result specification>
 <table expression>
<table expression> ::=
 <from clause>
 [<where clause>] [<group by clause>] [<having clause>]
```

Язык допускает три типа синтаксических конструкций, начинающихся с ключевого слова SELECT: спецификация курсора (cursor specification), оператор

выборки (select statement) и подзапрос (subquery). Основой всех них является синтаксическая конструкция "табличное выражение (table expression)". Семантика табличного выражения состоит в том, что на основе последовательного применения разделов from, where, group by и having из заданных в разделе from таблиц строится некоторая новая результирующая таблица, порядок следования строк которой неопределен и среди строк которой могут находиться дубликаты (т.е. в общем случае таблица-результат табличного выражения является мультимножеством строк). На самом деле именно структура табличного выражения наибольшим образом характеризует структуру запросов языка SQL/89. Мы рассмотрим структуру и смысл разделов табличного выражения ниже, но до этого немного подробнее обсудим три упомянутые конструкции, включающие табличные выражения.

Спецификация курсора

Наиболее общей является конструкция "спецификация курсора". Курсор - это понятие языка SQL, позволяющее с помощью набора специальных операторов получить построчный доступ к результату запроса к БД. К табличным выражениям, участвующим в спецификации курсора, не предъявляются какие-либо ограничения. Как видно из сводки синтаксических правил, при определении спецификации курсора используются три дополнительных конструкции: спецификация запроса, выражение запросов и раздел ORDER BY.

Спецификация запроса

В спецификации запроса задается список выборки (список арифметических выражений над значениями столбцов результата табличного выражения и констант). В результате применения списка выборки к результату табличного выражения производится построение новой таблицы, содержащей то же число строк, но вообще говоря другое число столбцов, включающих результаты вычисления соответствующих арифметических выражений из

списка выборки. Кроме того, в спецификации запроса могут содержаться ключевые слова ALL или DISTINCT. При наличии ключевого слова DISTINCT из таблицы, полученной применением списка выборки к результату табличного выражения, удаляются строки-дубликаты; при указании ALL (или просто при отсутствии DISTINCT) удаление строк-дубликатов не производится.

Выражение запросов

Выражение запросов - это выражение, строящееся по указанным синтаксическим правилам на основе спецификаций запросов. Единственной операцией, которую разрешается использовать в выражениях запросов, является операция UNION (объединение таблиц) с возможной разновидностью UNION ALL. К таблицам-операндам выражения запросов предъявляется то требование, что все они должны содержать одно и то же число столбцов, и соответствующие столбцы всех операндов должны быть одного и того же типа. Выражение запросов вычисляется слева направо с учетом скобок. При выполнении операции UNION производится обычное теоретико-множественное объединение операндов, т.е. из результирующей таблицы удаляются дубликаты. При выполнении операции UNION ALL образуется результирующая таблица, в которой могут содержаться строки-дубликаты.

Раздел ORDER BY

Наконец, раздел ORDER BY позволяет установить желаемый порядок просмотра результата выражения запросов. Синтаксис ORDER BY следующий:

```
<order by clause> ::=
 ORDER BY <sort specification> [{,<sort
specification>}...]
<sort specification> ::=
 {<unsigned integer> <column specification>} [ASC DESC]
```

Как видно из этих синтаксических правил, фактически задается список столбцов результата выражения запросов, и для каждого столбца указывается порядок просмотра строк результата в зависимости от значений этого столбца (ASC - по возрастанию (умолчание) DESC - по убыванию). Столбцы можно задавать их именами в том и только в том случае, когда (1) выражение запросов не содержит операций UNION или UNION ALL и (2) в списке выборки спецификации запроса этому столбцу соответствует арифметическое выражение, состоящее только из имени столбца. Во всех остальных случаях в разделе ORDER BY должен указываться порядковый номер столбца в таблице-результате выражения запросов.

Оператор выборки

Оператор выборки - это отдельный оператор языка SQL/89, позволяющий получить результат запроса в прикладной программе без привлечения курсора. Поэтому оператор выборки имеет синтаксис, отличающийся от синтаксиса спецификации курсора, и при его выполнении возникают ограничения на результат табличного выражения. Фактически, и то, и другое диктуется спецификой оператора выборки как одиночного оператора SQL: при его выполнении результат должен быть помещен в переменные прикладной программы. Поэтому в операторе появляется раздел INTO, содержащий список переменных прикладной программы, и возникает то ограничение, что результирующая таблица должна содержать не более одной строки. Соответственно, результат базового табличного выражения должен содержать не более одной строки, если оператор выборки не содержит спецификации DISTINCT, и таблица, полученная применением списка выборки к результату табличного выражения, не должна содержать более одной несовпадающих строк, если спецификация DISTINCT задана.

Замечание: В диалекте SQL СУБД Oracle поддерживается расширенный вариант оператора выборки, результатом которого не обязательно является

таблица из одной строки. Такое расширение не поддерживается ни в SQL/89, ни в SQL/92.

Подзапрос

Наконец, последняя конструкция SQL/89, которая может содержать табличные выражения, - это подзапрос, то есть запрос, который может входить в предикат условия выборки оператора SQL. В SQL/89 к подзапросам применяется то ограничение, что результирующая таблица должна содержать в точности один столбец. Поэтому в синтаксических правилах, определяющих подзапрос, вместо списка выборки указано "выражение, вычисляющее значение", т.е. арифметическое выражение. Заметим еще, что поскольку подзапрос всегда вложен в некоторый другой оператор SQL, то в качестве констант в арифметическом выражении выборки и логических выражениях разделов WHERE и HAVING можно использовать значения столбцов текущих строк таблиц, участвующих в (под)запросах более внешнего уровня. Более подробно об этом см. ниже, при описании семантики табличных выражений.

Табличное выражение

Стандарт SQL/89 рекомендует рассматривать вычисление табличного выражения как последовательное применение разделов FROM, WHERE, GROUP BY и HAVING к таблицам, заданным в списке FROM. Раздел FROM имеет следующий синтаксис:

```
<from clause> ::=
 FROM <table reference> ({,<table reference>}...]
<table reference> ::=
 <table name> [<correlation name>]
```

Раздел FROM

Результатом выполнения раздела FROM является расширенное декартово

произведение таблиц, заданных списком таблиц раздела FROM. Расширенное декартово произведение (расширенное, потому что в качестве операндов и результата допускаются мультимножества) в стандарте определяется следующим образом:

"Расширенное произведение R есть мультимножество всех строк r таких, что r является конкатенацией строк из всех идентифицированных таблиц в том порядке, в котором они идентифицированы. Мощность R есть произведение мощностей идентифицированных таблиц. Порядковый номер столбца в R есть n+s, где n - порядковый номер порождающего столбца в именованной таблице T, а s - сумма степеней всех таблиц, идентифицированных до T в разделе FROM."

Как видно из синтаксиса, рядом с именем таблицы можно указывать еще одно имя "correlation name". Фактически, это некоторый синоним имени таблицы, который можно использовать в других разделах табличного выражения для ссылки на строки именно этого вхождения таблицы.

Если табличное выражение содержит только раздел FROM (это единственный обязательный раздел табличного выражения), то результат табличного выражения совпадает с результатом раздела FROM.

Раздел WHERE

Если в табличном выражении присутствует раздел WHERE, то следующим вычисляется он. Синтаксис раздела WHERE следующий:

```
<where clause> ::= WHERE <search condition>
<search condition> ::=
 <boolean term>
 ( <search condition> OR <boolean term>
```

```
<Boolean term> ::=
 <boolean factor>
 ( <boolean term> AND <boolean factor>
<boolean factor> ::= [NOT] <boolean primary>
<boolean primary> ::= <predicate> (<search condition>)
```

Вычисление раздела WHERE производится по следующим правилам:

Пусть R - результат вычисления раздела FROM. Тогда условие поиска применяется ко всем строкам R, и результатом раздела WHERE является таблица, состоящая из тех строк R, для которого результатом вычисления условия поиска является true. Если условие выборки включает подзапросы, то каждый подзапрос вычисляется для каждого кортежа таблицы R (в стандарте используется термин "effectively" в том смысле, что результат должен быть таким, как если бы каждый подзапрос действительно вычислялся заново для каждого кортежа R).

Заметим, что поскольку SQL/89 допускает наличие в базе данных неопределенных значений, то вычисление условия поиска производится не в булевой, а в трехзначной логике со значениями true, false и unknown (неизвестно). Для любого предиката определено, в каких ситуациях он может порождать значение unknown. Булевские операции AND, OR и NOT работают в трехзначной логике следующим образом:

```
true AND unknown = unknown
unknown AND true = unknown
unknown AND unknown = unknown
true OR unknown = true
unknown OR true = true
unknown OR unknown = unknown
```

```
NOT unknown = unknown
```

Среди предикатов условия поиска в соответствии с SQL/89 могут находиться следующие предикаты: предикат сравнения, предикат between, предикат in, предикат like, предикат null, предикат с квантором и предикат exists. Сразу заметим, что во всех реализациях SQL на эффективность выполнения запроса существенно влияет наличие в условии поиска простых предикатов сравнения (предикатов, задающих сравнение столбца таблицы с константой). Наличие таких предикатов позволяет СУБД использовать индексы при выполнении запроса, т.е. избегать полного просмотра таблицы. Хотя в принципе язык SQL позволяет пользователям не заботиться о конкретном наборе предикатов в условии выборки (лишь бы они были синтаксически и семантически правильны), при реальном использовании SQL-ориентированных СУБД такие технические детали стоит иметь в виду.

Предикат сравнения

Синтаксис предиката сравнения определяется следующими правилами:

```
<comparison predicate> ::=
 <value expression> <comp op>
 {<value expression> <subquery>}
<comp op> ::=
 = <> < > <= >=
```

Через "<>" обозначается операция "неравенства". Арифметические выражения левой и правой частей предиката сравнения строятся по общим правилам построения арифметических выражений и могут включать в общем случае имена столбцов таблиц из раздела FROM и константы. Типы данных арифметических выражений должны быть сравнимыми (например, если тип столбца a таблицы A является типом символьных строк, то предикат "a = 5"

недопустим).

Если правый операнд операции сравнения задается подзапросом, то дополнительным ограничением является то, что мощность результата подзапроса должна быть не более единицы. Если хотя бы один из операндов операции сравнения имеет неопределенное значение, или если правый операнд является подзапросом с пустым результатом, то значение предиката сравнения равно unknown.

Заметим, что значение арифметического выражения не определено, если в его вычислении участвует хотя бы одно неопределенное значение. Еще одно важное замечание из стандарта SQL/89: в контексте GROUP BY, DISTINCT и ORDER BY неопределенное значение выступает как специальный вид определенного значения, т.е. возможно, например, образование группы строк, значение указанного столбца которых является неопределенным. Для обеспечения переносимости прикладных программ нужно внимательно оценивать специфику работы с неопределенными значениями в конкретной СУБД.

Предикат between

Предикат between имеет следующий синтаксис:

```
<between predicate> ::=
 <value expression>
 [NOT] BETWEEN <value expression> AND <value expression>
```

Результат "x BETWEEN y AND z" тот же самый, что результат "x >= y AND x <= z". Результат "x NOT BETWEEN y AND z" тот же самый, что результат "NOT (x BETWEEN y AND z)".

Предикат in

Предикат in определяется следующими синтаксическими правилами:

```
<in predicate> ::=
 <value expression> [NOT] IN
 {<subquery> (<in value list>)}
<in value list> ::=
 <value specification>
 {,<value specification>}...
```

Типы левого операнда и значений из списка правого операнда (напомним, что результирующая таблица подзапроса должна содержать ровно один столбец) должны быть сравнимыми.

Значение предиката равно true в том и только в том случае, когда значение левого операнда совпадает хотя бы с одним значением списка правого операнда. Если список правого операнда пуст (так может быть, если правый операнд задается подзапросом), или значение "подразумеваемого" предиката сравнения x = y (где x - значение арифметического выражения левого операнда) равно false для каждого элемента y списка правого операнда, то значение предиката in равно false. В противном случае значение предиката in равно unknown. По определению значение предиката "x NOT IN S" равно значению предиката "NOT (x IN S)".

Предикат like

Предикат like имеет следующий синтаксис:

```
<like predicate> ::=
 <column specification> [NOT] LIKE <pattern>
```

```
 [ESCAPE <escape character>]
<pattern> ::= <value specification>
 <escape character> ::= <value specification>
```

Типы данных столбца левого операнда и образца должны быть типами символьных строк. В разделе ESCAPE должен специфицироваться одиночный символ.

Значение предиката равно true, если pattern является подстрокой заданного столбца. При этом, если раздел ESCAPE отсутствует, то при сопоставлении шаблона со строкой производится специальная интерпретация двух символов шаблона: символ подчеркивания ("_") обозначает любой одиночный символ; символ процента ("%") обозначает последовательность произвольных символов произвольной длины (может быть, нулевой).

Если же раздел ESCAPE присутствует и специфицирует некоторый одиночный символ x, то пары символов "x_" и "x%" представляют одиночные символы "_" и "%" соответственно.

Значение предиката like есть unknown, если значение столбца, либо шаблона неопределено.

Значение предиката "x NOT LIKE y ESCAPE z" совпадает со значением "NOT x LIKE y ESCAPE z".

Предикат null

Предикат null описывается синтаксическим правилом:

```
<null predicate> ::=
 <column specification> IS [NOT] NULL
```

Этот предикат всегда принимает значения true или false. При этом значение "x IS NULL" равно true тогда и только тогда, когда значение x неопределено. Значение предиката "x NOT IS NULL" равно значению "NOT x IS NULL".

Предикат с квантором

Предикат с квантором имеет следующий синтаксис:

```
<quantified predicate> ::=
 <value expression> <comp op> <quantifier> <subquery>
<quantifier> ::=
 <all> <some>
<all> ::= ALL
<some> ::= SOME ANY
```

Обозначим через x результат вычисления арифметического выражения левой части предиката, а через S результат вычисления подзапроса.

Предикат "x <comp op> ALL S" имеет значение true, если S пусто или значение предиката "x <comp op> s" равно true для каждого s, входящего в S. Предикат "x <comp op> ALL S" имеет значение false, если значение предиката "x <comp op> s" равно false хотя бы для одного s, входящего в S. В остальных случаях значение предиката "x <comp op> ALL S" равно unknown.

Предикат "x <comp op> SOME S" имеет значение false, если S пусто или значение предиката "x <comp op> s" равно false для каждого s, входящего в S. Предикат "x <comp op> SOME S" имеет значение true, если значение предиката "x <comp op> s" равно true хотя бы для одного s, входящего в S. В остальных случаях значение предиката "x <comp op> SOME S" равно unknown.

Предикат exists

Предикат exists имеет следующий синтаксис:

```
<exists predicate> ::=
EXISTS <subquery>
```

Значением этого предиката всегда является true или false, и это значение равно true тогда и только тогда, когда результат вычисления подзапроса не пуст.

Раздел GROUP BY

Если в табличном выражении присутствует раздел GROUP BY, то следующим выполняется он. Синтаксис раздела GROUP BY следующий:

```
<group by clause> ::=
 GROUP BY <column specification> [{,<column
specification>}...]
```

Если обозначить через R таблицу, являющуюся результатом предыдущего раздела (FROM или WHERE), то результатом раздела GROUP BY является разбиение R на множество групп строк, состоящего из минимального числа групп таких, что для каждого столбца из списка столбцов раздела GROUP BY во всех строках каждой группы, включающей более одной строки, значения этого столбца равны. Для обозначения результата раздела GROUP BY в стандарте используется термин "сгруппированная таблица".

Раздел HAVING

Наконец, последним при вычислении табличного выражения используется раздел HAVING (если он присутствует). Синтаксис этого раздела следующий:

```
<having clause> ::=
 HAVING <search condition>
```

Раздел HAVING может осмысленно появиться в табличном выражении только в том случае, когда в нем присутствует раздел GROUP BY. Условие поиска этого раздела задает условие на группу строк сгруппированной таблицы. Формально раздел HAVING может присутствовать и в табличном выражении, не содержащем GROUP BY. В этом случае полагается, что результат вычисления предыдущих разделов представляет собой сгруппированную таблицу, состоящую из одной группы без выделенных столбцов группирования.

Условие поиска раздела HAVING строится по тем же синтаксическим правилам, что и условие поиска раздела WHERE, и может включать те же самые предикаты. Однако имеются специальные синтаксические ограничения по части использования в условии поиска спецификаций столбцов таблиц из раздела FROM данного табличного выражения. Эти ограничения следуют из того, что условие поиска раздела HAVING задает условие на целую группу, а не на индивидуальные строки.

Поэтому в арифметических выражениях предикатов, входящих в условие выборки раздела HAVING, прямо можно использовать только спецификации столбцов, указанных в качестве столбцов группирования в разделе GROUP BY. Остальные столбцы можно специфицировать только внутри спецификаций агрегатных функций COUNT, SUM, AVG, MIN и MAX, вычисляющих в данном случае некоторое агрегатное значение для всей группы строк. Аналогично обстоит дело с подзапросами, входящими в предикаты условия выборки раздела HAVING: если в подзапросе используется характеристика текущей группы, то она может задаваться только путем ссылки на столбцы группирования.

Результатом выполнения раздела HAVING является сгруппированная таблица, содержащая только те группы строк, для которых результат вычисления условия поиска есть true. В частности, если раздел HAVING присутствует в табличном выражении, не содержащем GROUP BY, то результатом его выполнения будет либо пустая таблица, либо результат выполнения предыдущих разделов табличного выражения, рассматриваемый как одна группа без столбцов группирования.

Агрегатные функции и результаты запросов

Агрегатные функции (в стандарте SQL/89 они называются функциями над множествами) определяются в SQL/89 следующими синтаксическими правилами:

```
<set function specification> ::=
 COUNT(*) <distinct set function>
 <all set function>
<distinct set function> ::=
 { AVG MAX MIN SUM COUNT } (DISTNICT <column
specification>)
<all set function> ::=
 { AVG MAX MIN SUM } ([ALL] <value expression>)
```

Как видно из этих правил, в стандарте SQL/89 определены пять стандартных агрегатных функций: COUNT - число строк или значений, MAX - максимальное значение, MIN - минимальное значение, SUM - суммарное значение и AVG - среднее значение.

Семантика агрегатных функций

Агрегатные функции предназначены для того, чтобы вычислять некоторое значение для заданного множества строк. Таким множеством строк может быть

группа строк, если агрегатная функция применяется к сгруппированной таблице, или вся таблица. Для всех агрегатных функций, кроме COUNT(*), фактический (то есть требуемый семантикой) порядок вычислений следующий: на основании параметров агрегатной функции из заданного множества строк производится список значений. Затем по этому списку значений производится вычисление функции. Если список оказался пустым, то значение функции COUNT для него есть 0, а значение всех остальных функций - null.

Пусть T обозначает тип значений из этого списка. Тогда результат вычисления функции COUNT - точное число с масштабом и точностью, определяемыми в реализации. Тип результата значений функций MAX и MIN совпадает с T. При вычислении функций SUM и AVG тип T не должен быть типом символьных строк, а тип результата функции - это тип точных чисел с определяемыми в реализации масштабом и точностью, если T - тип точных чисел, и тип приблизительных чисел с определяемой в реализации точностью, если T - тип приблизительных чисел.

Вычисление функции COUNT(*) производится путем подсчета числа строк в заданном множестве. Все строки считаются различными, даже если они состоят из одного столбца со значением null во всех строках.

Если агрегатная функция специфицирована с ключевым словом DISTINCT, то список значений строится из значений указанного столбца. (Подчеркнем, что в этом случае не допускается вычисление арифметических выражений!) Далее из этого списка удаляются неопределенные значения, и в нем устраняются значения-дубликаты. Затем вычисляется указанная функция.

Если агрегатная функция специфицирована без ключевого слова DISTINCT (или с ключевым словом ALL), то список значений формируется из значений арифметического выражения, вычисляемого для каждой строки

заданного множества. Далее из списка удаляются неопределенные значения и производится вычисление агрегатной функции. Обратите внимание, что в этом случае не допускается применение функции COUNT!

Замечание: оба ограничения, указанные в двух предыдущих абзацах, являются более техническими, чем принципиальными, и могут отсутствовать в конкретных реализациях. Тем не менее, это ограничения стандарта SQL/89, и их нужно придерживаться при мобильном программировании.

Результаты запросов

Агрегатные функции можно разумно использовать в спецификации курсора, операторе выборки и подзапросе после ключевого слова SELECT (будем называть в этом подразделе все такие конструкции списком выборки, не забывая о том, что в случае подзапроса этот список состоит только из одного элемента), и в условии выборки раздела HAVING. Стандарт допускает более экзотические случаи использования агрегатных функций в подзапросах (агрегатная функция на группе кортежей внешнего запроса), но на практике они встречаются очень редко.

Рассмотрим различные случаи применения агрегатных функций в списке выборки в зависимости от вида табличного выражения.

Если результат табличного выражения R не является сгруппированной таблицей, то появление хотя бы одной агрегатной функции от множества строк R в списке выборки приводит к тому, что R неявно рассматривается как сгруппированная таблица, состоящая из одной (или нуля) групп с отсутствующими столбцами группирования. Поэтому в этом случае в списке выборки не допускается прямое использование спецификаций строк R: все они должны находиться внутри спецификаций агрегатных функций. Результатом запроса является таблица, состоящая не более чем из одной строки, полученной

путем применения агрегатных функций к R.

Аналогично обстоит дело в том случае, когда R представляет собой сгруппированную таблицу, но табличное выражение не содержит раздела GROUP BY (и, следовательно, не содержит раздел HAVING). Если в случае предыдущего абзаца было два варианта формирования списка выборки: только с прямым указанием столбцов R или только с указанием их внутри спецификаций агрегатных функций, то в данном случае возможен только второй вариант. Результат табличного выражения явно объявлен сгруппированной таблицей, состоящей из одной группы, и результат запроса можно формировать только путем применения агрегатных функций к этой группе строк. Опять результатом запроса является таблица, состоящая не более чем из одной строки, полученной путем применения агрегатных функций к R.

Наконец, рассмотрим случай, когда R представляет собой "настоящую" сгруппированную таблицу, т.е. табличное выражение содержит раздел GROUP BY и, следовательно, определен, по крайней мере, один столбец группирования. В этом случае правила формирования списка выборки полностью соответствуют правилам формирования условия выборки раздела HAVING: допускается прямое использование спецификации столбцов группирования, а спецификации остальных столбцов R могут появляться только внутри спецификаций агрегатных функций. Результатом запроса является таблица, число строк в которой равно числу групп в R, и каждая строка формируется на основе значений столбцов группирования и агрегатных функций для данной группы.

ЛИТЕРАТУРА

1. Клименко С., Уразметов В. Internet. Среда обитания информационного общества. Москва. 1995.
2. Дейт К. Введение в системы баз данных. Наука. Москва. 1980.
3. Кузнецов С. Основы современных баз данных. Учебное пособие. Центр Информационных технологий.
4. Кузнецов С. Проектирование и разработка корпоративных информационных систем. Центр информационных технологий. 1998.