**10 неделя**

**Лекция 19-20** Технологии исполняющиеся на стороне клиента и сервера. Механизм работы Web-сервера.

#### Средства доступа к базам данных на стороне сервера

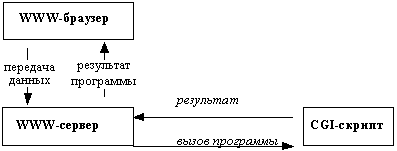
* CGI
* API
* FastCGI

**CGI**

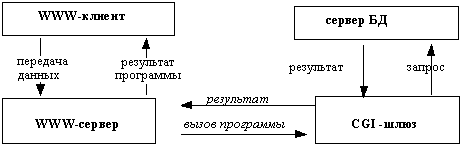
Common Gateway Interface - это спецификация интерфейса взаимодействия Web-сервера с внешними прикладными программами. Главное назначение CGI - обеспечение единообразного потока данных между сервером и работающим на нем приложением. CGI определяет:

1. порядок взаимодействия сервера с прикладной программой, в котором сервер выступает инициирующей стороной;
2. механизм реального обмена данными и управляющими командами в этом взаимодействии, что не определено в протоколе HTTP. Такие понятия, как метод доступа, переменные заголовка, MIME, типы данных, заимствованы из HTTP и делают спецификацию прозрачной для тех, кто знаком с самим протоколом.

Обычно гипертекстовые документы, возвращаемые по запросу клиента WWW сервером, содержат статические данные. CGI обеспечивает средства создания динамических Web-страниц на основе данных, полученных от пользователя. Программы, написанные в соответствии со спецификацией CGI, называются CGI-скриптами или шлюзами. Шлюз - это CGI-скрипт, который используется для обмена данными с другими информационными ресурсами Internet или приложениями-демонами такими, как, например, система управления базами данных. Обычная CGI-программа запускается Web-сервером для выполнения некоторой работы, возвращает результаты серверу и завершает свое выполнение (рис. 1).

  
*Рис. 1. Схема взаимодействия CGI-скрипта.*

Шлюз выполняется точно также, только, фактически, он инициирует взаимодействие в качестве клиента с третьей программой (рис. 2). Если эта третья программа является сервером БД, то шлюз становится клиентом СУБД, который посылает запрос по определенному порту соединения с системой управления базами данных, а после получения ответа пересылает его WWW-серверу.

  
*Рис.2. Схема взаимодействия CGI-шлюза.*

Обмен данными по спецификации CGI реализуется обычно через переменные окружения и стандартный ввод/вывод. Выбор механизма передачи параметров определяется методом доступа, который указывается в форме в атрибуте METHOD. Если используется метод GET, то передача параметров происходит с помощью переменных окружения, которые сервер создает при запуске внешней программы. Через них передается приложению как служебная информация (версия программного обеспечения, доменное имя сервера и др.), так сами данные (в переменной QUERY\_STRING). При методе POST для передачи используется стандартный ввод. А в переменных окружения фиксируется тип и длина передаваемой информации (CONTENT\_TYPE и CONTENT\_LENGTH).

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 модифицирует заголовок в соответствии со спецификацией HTTP. Обязательным для скриптов при генерировании документов "на лету", когда реального документа в файловой системе сервера не остается является только HTTP-заголовок *Content-type*, в котором указывается тип возвращаемого документа для правильной интерпретации браузером. Обычно в Content-type указывают текстовые типы text/plain и text/html. При использовании такого вида 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

При применение спецификаци CGI для обмена данными с внешними прикладными программами можно выделить следующие преимущества:

* Прозрачность использования;
* "Языковая" независимость - CGI-программы могут быть написаны на любом языке программирования или командном языке, имеющим средства работы со строками;
* Процессная изолированность - при запуске CGI-програмы на сервере порождается отдельный процесс и ошибочный CGI-скрипт не может сломать Web-сервер или получить доступ к закрытой информации;
* Открытость стандарта - CGI интерфейс применим на каждом Web-сервере;
* Архитектурная независимость - CGI не зависит от особенностей реализации архитектуры сервера (однопоточности, многопоточности и т.д.);

Но CGI имеет также и существенные недостатки. Главная проблема заключается в затратах на выполнение CGI-приложений: поскольку на сервере для каждого очередного запроса порождается новый процесс, который завершается после его выполнения, то это приводит к невысокому быстродействию CGI-скрипта и снижает эффективность работы сервера. При использовании CGI-программ для доступа к базам данных из-за неподдержки непрерывного соединения Web-сервера и соответствующей СУБД очень сложно произвести процесс "ведения" пользователя базой данных, так как каждый раз при генерации очередного запроса требуется новое подключение. Но в то же время закрытие соединения после обработки каждого запроса сильно осложняет деятельность хакеров, так как при отсутствии постоянного подключения к БД проникнуть в нее гораздо сложнее. Другое достоинство этого "недостатка" состоит в том, что связь с Web-сервером устанавливается только на короткий промежуток времени, в результате чего он не перегружается и может выполнять другие задачи.

CGI`также ограничен по способности функционирования - спецификация предусматривает только простую "ответную" роль скрипта при генерации результата на запрос пользователя. CGI-программы не имеют взаимосвязей с установлением аутентификации пользователя и проверки его входных данных.

**API**

В ответ на ограничения и недостатки спецификации CGI была разработана спецификация прикладных модулей API, встроенных в сервер. Данное расширение Web-сервера запускается как динамическая библиотека и выполняет обработку каждого вызова сервера по отдельной структуре памяти, что значительно проще, чем создание отдельного процесса для каждого клиентского запроса. Наиболее известны два API-интерфейса - NSAPI компании Netscape и ISAPI компании Microsoft. Свободно распространяемый популярный Unix-сервер Apache также имеет модуль PHP, реализующий данный интерфейс. Приложения, работающие через API, соединяются с сервером значительно быстрее, чем CGI-программы, так как API выполняется в основном процессе сервера и постоянно находится в состоянии ожидания запросов, поэтому время на запуск программы и порождения нового процесса не требуется. API-интерфейс предоставляет и большую функциональность, чем CGI - можно написать дополнительные процедуры, осуществляющие контроль доступа к файлам, получающие доступ к log-файлам сервера и связывающиеся с другими этапами обработки запроса сервером.

Тем не менее спецификация API не имеет преимуществ CGI-интерфейса и поставщики API-модулей тоже сталкиваются с целым рядом проблем:

* "Языковая" зависимость - прикладные программы могут быть написаны только на языках, поддерживаемых в данном API (обычно это С/C++); Perl, наиболее популярный язык для CGI-скриптов, как правило, не используется в существующих поставляемых API-модулях.
* Неизолированность процесса - так как приложения выполняются в адресном пространстве сервера, то ошибочные программы могут "уронить" сервер или какое-либо приложение. Таким образом вполне возможно (намеренно или нет) сломать систему безопасности сервера.
* Ограниченность применения - написанные программы в соответствии с данным API могут использоваться только на данном сервере.
* Архитектурная зависимость - API-приложения зависимы от архитектуры сервера: если сервер поддерживает однопоточность, то многопотоковые приложения не получают никакого преимущества в быстродействии при выполнении. Также при изменении производителем архитектуры сервера, модуль API обычно тоже подвергается изменениям, и прикладные программы соответственно тоже требуют переделки или даже могут быть написаны заново.

**FastCGI** Интерфейс FastCGI сочетает в себе наилучшие аспекты спецификаций CGI и API. Взаимодействие в соответствии с FastCGI происходит сходным образом с CGI. FastCGI-приложения запускаются отдельными изолированными процессами. Отличие состоит в том, что эти процессы являются постоянно работающими и после выполнения запроса не завершаются, а ожидают новых запросов. Вместо использования переменных окружения операционной системы и стандартных потоков ввода/вывода протокол FastCGI объединяет информацию среды, стандартный ввод, вывод и сообщения об ошибках в единственное дуплексное соединение. Это позволяет FastCGI-программам выполняться на удаленных машинах, используя TCP-соединения между Web-сервером и FasstCGI-модулем.

Таким образом, преимущества FastCGI состоят в следующем:

* Быстродействие - благодаря постоянному функционированию FsatCGI-процессов обеспечивается обслуживание одним процессом многих запросов, что решает задачу и связанные с ней проблемы порождения нового процесса на отдельный клиентский запрос.
* Простота применения и легкость миграции из CGI.
* "Языковая" независимость - как и CGI, FastCGI-приложения могут быть написаны на любых языках программирования или командных языках.
* Изолированность процессов - "неисправные" FastCGI-программы не могут разрушить ядро сервера или какие-либо другие приложения, а также получить секретную служебную информацию.
* Совместимость - FastCGI поддерживается во всех открытых продуктах, включая коммерческие серверы Netscape и Microsoft, NCSA сервер и свободно распространяемый Apache.
* Архитектурная независимость - FastCGI интерфейс не зависит от особенностей реализации серверной архитектуры и прикладные программы могут быть как одно-, так и многопоточными.
* Распределенность - FastCGI обеспечивает возможность выполнять приложения удаленно, что используется для распределенной загрузки и управления внешними Web-сайтами.

#### Доступ к базам данных на стороне клиента. Java-технология

Для обеспечения доступа к базам данных на стороне Web-клиента применяется Java-технология. Java - это современный объектно-ориентированный язык программирования для разработки приложений, созданный специально для распределенных сред. Технология Java позволяет создавать полноценные приложения для работы с компьютерной графикой, файловыми системами и компьютерными сетями.

Одно из важных свойств Java-технологии - это мобильность, суть которой заключается в том, что написанный на Java код может исполняться на любой компьютерной платформе. Java-приложения компилируются в особый код (так называемый байт-код), исполняемый на виртуальной машине (Java Virtual Machine). Байт-код является универсальным форматом программы, единым для всех аппаратных платформ - и для рабочих станций, и для больших универсальных ЭВМ, и для персональных компьютеров. Java-технология обеспечивает быстрый цикл компиляции и отладки программ. Еще на стадии компиляции проводится выявление многих ошибок и частичная оптимизация программ. Средства разработки, содержащие виртуальную машину внутри себя, обеспечивают контроль приложений на стадии исполнения (переполнение стека, отслеживание границ массивов, поиск резервов для оптимизации и др.).

Пользователю готовых Java-приложений достаточно иметь клиентскую программу, имитирующую работу виртуальной машины. Виртуальная машина представляет собой довольно компактный интерпретатор байт-кода Java. Перед первым запуском нового приложения виртуальная машина проверяет его код на принадлежность к байт-коду (на правильность инструкций Java), безопасность команд для компьютера и локальной сети, соответствие разрешенным операциям, а также на целый ряд дополнительных условий. Это необходимо, поскольку приложения, распространяемые по сети, создаются разными людьми с различными намерениями, причем дурные намерения тоже не исключены. Непосредственно перед запуском виртуальная машина производит сборку модулей и устанавливает связи между именами, при этом поиск недостающих модулей производится не только в системе, но и на серверах Internet. Затем, собственно, и начинается работа приложений.

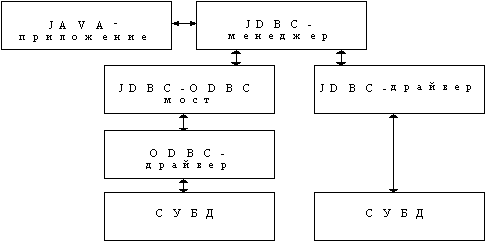
Технология разработки HTML-документа позволяет написать произвольное количество дополнительных Java-программ, откомпилировать их в мобильные коды и поставить ссылки на соответствующие коды в теле HTML-документа. Такие дополнительные Java-программы называются апплетами (Java-applets). Получив доступ к документу, содержащему ссылки на апплеты, клиентская программа просмотра запрашивает у Web-сервера все мобильные коды. Коды могут начать выполняться сразу после размещения в компьютере клиента или быть активизированы с помощью специальных команд.

Поскольку апплет представляет собой произвольную Java-программу, то, в частности, он может быть специализирован для работы с внешними базами данных. Опираясь на использование классов, апплет может получить от пользователя информацию, характеризующую его запрос к базе данных, в том же виде, как если бы использовался стандартный механизм форм языка HTML, а может применять какой-либо другой интерфейс.

Для взаимодействия Java-апплета с внешним сервером баз данных разработан специализированный протокол JDBC, который, фактически, сочетает функции шлюзования между интерпретатором мобильных Java-кодов и интерфейсом ODBC (Open Data Base Connectivity). JDBC - это разработанный JavaSoft прикладной программный SQL интерфейс API JDBC к базам данных. Этот API позволяет использовать стандартный набор процедур высокого уровня для доступа к различным БД.

JDBC базируется на интерфейсе уровня вызовов X/Open SQL CLI - основе ODBC. Прикладной программный интерфейс JDBC реализуется поверх других SQL-API, включая ODBC. То есть, все базы данных, допускающих работу с ODBC, будут взаимодействовать с JDBC без изменений. При использовании JDBC Internet-пользователи подключаются к Web-серверу и загружают HTML-документ с апплетом. Апплет выполняется на клиентской ЭВМ в среде браузера и устанавливает связь с сервером базы данных. Механизм связи с базами данных является классом Java.

Архитектура JDBC состоит из двух уровней: JDBC API, который обеспечивает связь между приложением и менеджером JDBC и драйвер JDBC API, который поддерживает связь между JDBC менеджером и драйвером (рис.3). Разработчики имеют возможность взаимодействовать напрямую с ODBC посредством моста JDBC-ODBC.

  
*Рис.3. Схема взаимодействия Java-приложений с сервером БД*

JDBC API представляет собой набор классов (пакет или package) для установки соединений с базами данных, создания SQL-выражений, обработки результатов. JDBC-драйвера могут быть написаны на Java и загружены как часть апплета или быть написаны используя "родной" код компьютера (как шлюз к существующим библиотекам СУБД API). Примером такого шлюза является JDBC-ODBC мост (JDBC-ODBC bridge). Он транслирует JDBC запросы в вызовы ODBC, что позволяет получить доступ к огромному множеству существующих ODBC драйверов.

Использование Java-апплетов в общем обеспечивает более гибкое решение.Так как апплет - это часть HTML-документа, то для включения нового апплета нужно всего-навсего перекомпоновать документ без задействия Web-cервера. Апплеты передаются по сети только в тот момент, когда их нужно выполнять. При этом они могут загружаться из разных мест и после загрузки взаимодействовать друг с другом.

С другой стороны, байт-код Java исполняется интерпретатором, а не является откомпилированной на данной платформе программой. Отсюда возникает первый очевидный недостаток - это скорость выполнения кода, так как интерпретатор работает гораздо медленнее откомпилированной программы. Собственно, и другие свойства технологии (объектная ориентируемость, использование многопоточности, отсутствие адресной арифметики и т.п.) в большинстве случаев при стандартной комплектации оборудования, только тормозят выполнение программы.

Основной протокол обмена апплетами - HTTP. Это значит, что они передаются по сети точно также, как и другие ресурсы Website, и приобретают свое особое значение только в момент получения их браузером. Учитывая неэффективность реализации HTTP поверх TCP, можно сказать, что и обмен апплетами тоже неэффективен, если для каждого обмена устанавливается свое TCP-соединение. В любом случае загрузка страниц с Java происходит гораздо медленнее, чем без Java.

Java-технология является еще технологией, находящейся в стадии разработки. Интерфейс взаимодействия прикладных программ CGI имеет уже достаточный опыт в применении для обработки данных и доступа к БД. По сравнению с Java, он является не только отлаженным механизмом, но и более простым и удобным средством для разработчиков CGI-программ, так как они могут быть созданы на любом языке, имеющим средства работы со строками.

Выбор средства доступа к базам данных во многом определяется не только эффективностью того или иного механизма, но также и наилучшим его сопряжением с соответствующей СУБД. От того, какие средства предоставляет сама СУБД для доступа к своим базам данных из внешних прикладных программ может зависеть выбор предпочтения. Сейчас разработано достаточно много коммерческих СУБД, но все же хочется обратить внимание на свободно распространяемые продукты, которые часто оказываются не менее эффективными, но из-за "неизвестности" не достаточно широко используются. Одним из таких некоммерческих продуктов является СУБД POSTGRES95, которая устанавливается на большинстве существующих платформ -DEC Alpha AXP on OSF/1 2.0, HP PA-RISC on HP-UX 9.0, i386 Solaris, SUN SPARC on Solaris 2.4, SUN SPARC on SunOS 4.1.3, DEC MIPS on Ultrix 4.4, Intel x86 on Linux 1.2 and Linux ELF, BSD/OS 2.0, 2.01, 2.1, IBM on AIX 3.2.5, SGI MIPS on IRIX 5.3, DG/UX 5.4R3.10 и др.

Рассмотрим пример.

Предположим, ваш товарищ создал свой сайт по адресу www.petrov.ru и расположил на нем свое резюме по адресу http://www.petrov.ru/resume.htm. Что происходит, когда вы просматриваете данное резюме со своего компьютера?

Как только вы набираете в своем браузере вышеуказанный URL, браузер получает информацию об используемом протоколе (http), имени сервера (www.petrov.ru) и запрашиваемом файле (resume.htm). Браузер обращается к DNS-серверу, чтобы перевести имя сервера www.petrov.ru в IP-адрес, который служит для нахождения нужного сервера, устанавливает связь с Web-сервером и, используя протокол http, запрашивает искомый ресурс. Сервер посылает браузеру HTML-страницу, чтобы он мог отобразить данный документ. Браузер считывает HTML-тэги, воссоздает страницу на экране вашего компьютера, и вы видите результат своего запроса.

Обычно даже простая Web-страница содержит не только текст, но и графику. Иными словами, страница состоит из нескольких файлов, то есть в общем случае передаваемый на ваш компьютер файл может быть не только текстовым, но и графическим или еще каким-либо. Как же браузер распознает эти файлы? Вначале сервер посылает текстовое сообщение, которое информирует клиента о том, какой файл он будет отправлять: что это будет — текст в формате HTML, или графика в формате jpg, или что-нибудь другое, и только после этого передает содержание данного файла. Несмотря на то что обычно Web-страница состоит из многих файлов, за время одного запроса Web-сервер посылает только один файл. То есть, получив HTML-текст и найдя в нем, например, ссылку на графический элемент, браузер посылает новый запрос на сервер, и при этом не обязательно на тот же самый, с которого пришел HTML-текст. Вероятно, большинству пользователей Интернета знакома ситуация, когда загрузка страницы происходит с задержкой и вначале отображается текст, а потом начинают появляться картинки. Данная ситуация как раз отражает последовательность загрузки элементов Web-страницы. Для того чтобы скачать каждый новый файл, браузер должен установить новую сессию. При этом, поскольку современные серверы и браузеры работают в многопоточном режиме, одновременно могут выполняться несколько сессий, но для скачивания каждого нового файла инициируется своя сессия.

Если запрашиваемого ресурса по данному адресу нет, то Web-сервер выдаст сообщение, которое вы, вероятно, не раз видели на своем экране: 404/File not found (файл не найден).