# Java и WEB

# Java и WEB

**История**

Первоначально перед HTTP-серверами стояла простая задача: найти и отправить клиенту файл, указанный в полученном от клиента запросе. Запрос составлялся тоже очень просто по правилам протокола HTTP в специально придуманной форме URL.

Потом понадобилось сделать на сервере какую-либо небольшую предварительную обработку отправляемого файла. Появились включения на стороне сервера SSI (Server Side Include) и различные приемы динамической генерации страниц HTML. HTTP-сервер усложнился и стал называться Web-сервером.

# Java и WEB

Затем возникла необходимость выполнять на сервере процедуры. В запрос URL вставили возможность вызова процедур, а на сервере реализовали технологию CGI (Common Gateway Interface).

CGI действует следующим образом. При посылке сообщения мы указываем URL исполнимого файла некоторой программы, размещенной на машине-сервере. Получив сообщение, Web-сервер отыскивает и запускает эту программу и передает сообщение на ее стандартный ввод. Программа знает, что делать с полученным сообщением. Она обрабатывает сообщение и выводит результат обработки на свой стандартный вывод. Web-сервер подключается к стандартному выводу программы, принимает результат ее работы и отправляет его обратно клиенту.

CGI-программу можно написать на любом языке: C, C++, Pascal, Perl, ASP, PHP, лишь бы у нее был стандартный ввод и стандартный вывод. CGI-программы обычно лежат на сервере в каталоге cgi-bin.

# Java и WEB

Пример CGI-вызова:

```
http://some.firm.com/cgi-bin/mycgiprog.pl?name=Ivanov&age=27
```

Для составления таких запросов в язык HTML введен тег <form>.

Web-сервер, получив запрос, загружает и запускает CGI-процедуру (в предыдущем примере это процедура mycgiprog.pl), расположенную на сервере в каталоге cgi-bin, и передает ей значения "Ivanov" и "27" аргументов name и age. Процедура оформляет свой ответ в виде страницы HTML, которую Web-сервер отправляет клиенту.

# Java и WEB

Технология Java не могла пройти мимо такой насущной потребности и отозвалась на нее созданием **сервлетов** и языком **JSP** (JavaServer Pages).

**Сервлеты** (servlets) выполняются под управлением Web-сервера подобно тому, как апплеты выполняются под управлением браузера, откуда и произошло их название.

Для слежения за работой сервлетов и управления ими создается специальный программный модуль, называемый **контейнером сервлетов** (servlet container).

# Java и WEB

Контейнер сервлетов загружает сервлеты, инициализирует их, передает им запросы клиентов, принимает ответы. Сервлеты не могут работать без контейнера, как апплеты не могут работать без браузера.

Web-сервер, снабженный контейнером сервлетов и другими контейнерами, стал называться **сервером приложений** (application server, AS).

# Java и WEB

Чтобы сервлет мог работать, он должен быть зарегистрирован в контейнере, установлен (deploy) в него. Установка (deployment) сервлета в контейнер включает получение уникального имени и определение начальных параметров сервлета, запись их в конфигурационные файлы, создание каталогов для хранения всех файлов сервлета и другие операции.

Процесс установки сильно зависит от контейнера. Одному контейнеру достаточно скопировать сервлет в определенный каталог, например autodeploy/ или webapps/, другому надо после этого перезапустить контейнер, для третьего надо воспользоваться утилитой установки. В стандартном контейнере Java EE SDK такая утилита называется deploytool.

# Java и WEB

Один контейнер может управлять работой нескольких установленных в него сервлетов. При этом один контейнер способен в одно и то же время работать в нескольких виртуальных машинах Java, образуя распределенное Web-приложение. Сами же виртуальные машины Java могут работать на одном компьютере или на разных компьютерах.

Контейнеры сервлетов создаются как часть Web-сервера или как встраиваемый в него модуль. Большую популярность получили встраиваемые контейнеры Tomcat, разработанные сообществом Apache Software Foundition ( http://tomcat.apache.org/)

.

# Java и WEB

**Web-приложение**

Как правило, сервлет не выполняется один. Он работает в составе Web-приложения. **Web-приложение** (web application) составляют все ресурсы, написанные для обслуживания запросов клиента: сервлеты, JSP, страницы HTML, документы XML, другие документы, изображения и чертежи, музыкальные и видеофайлы. Спецификация "Java Servlet Specification" описывает структуру каталогов, содержащих все эти ресурсы.

Все Web-приложение целиком часто упаковывается в один файл по технологии JAR. Такой файл обычно получает расширение war (Web ARchive). Этот файл можно переносить с одного Web-сервера на другой, при этом многие контейнеры сервлетов могут запускать Web-приложение прямо из архива, не распаковывая его.

# Java и WEB

**Интерфейс `Servlet`**

Понятие "сервлет" описывается интерфейсом `Servlet`.

Контейнер инициализирует сервлет методом `init()` так же, как браузер инициализирует апплет, но, в отличие от апплета, у метода `init()`, описанного интерфейсом `Servlet`, есть аргумент типа `ServletConfig`:

```
public void init(ServletConfig conf);
```

Объект, описанный интерфейсом `ServletConfig`, создается Web-приложением и передается контейнеру для инициализации сервлета. Информация, необходимая для создания объекта, содержится в конфигурационном файле Web-приложения.

# Java и WEB

**Конфигурационный файл**

Конфигурационный файл (deployment descriptor) описывает ресурсы, составляющие Web-приложение: сервлеты, их фильтры и слушатели, страницы JSP, документы HTML и XML, изображения и документы других типов. Он формируется при создании Web-приложения и заполняется при установке сервлета и других ресурсов в контейнер.

Конфигурационный файл записывается на языке XML и называется `web.xml`. Он располагается в каталоге `WEB-INF`, одном из каталогов Web-приложения, и создается вручную, утилитой установки сервлета в контейнер или с помощью IDE, вроде NetBeans или Eclipse.

# Java и WEB

Каждая фирма-производитель контейнера сервлетов предоставляет свою утилиту установки или make-файл, содержащий команды установки. Надо заметить, что вместо построителя **make** в технологии Java часто используются другие построители, решающие туже задачу что и **make**, но написанные на языке Java: построитель **ant**, разработанный Apache Software Foundation в рамках проекта Jakarta, построитель **Maven** — еще одна разработка Apache Software Foundation, или **Ivy** — опять-таки разработка Apache.

Утилита установки контейнера Tomcat запускается из браузера, она расположена на его странице `/manager/html`. Для того чтобы запустить утилиту, в браузере надо набрать примерно такую строку: `http://localhost:8080/manager/`.

# Java и WEB

**Интерфейс ServletConfig**

Каждый объект типа `ServletConfig` содержит имя сервлета, извлеченное из элемента `<servlet-name>` конфигурационного файла, набор начальных параметров, взятых из элементов `<init-param>`, и контекст сервлета в виде объекта типа `ServletContext`. Эти конфигурационные параметры сервлет может получить методами описанными в интерфейсе `ServletConfig`:

```
public String getServletName();
public Enumeration getInitParameterNames();
public String getInitParameter(String name);
public ServletContext getServletContext();
```

Начальные параметры записываются в конфигурационный файл `web.xml` во время установки сервлета вручную или с помощью утилиты установки. Механизм задания и чтения начальных параметров сервлета очень похож на механизм определения параметров апплета, записываемых в теги <param> и читаемых методами getParameter() апплета.

# Java и WEB

**Контекст сервлета**

Для всех сервлетов, работающих в рамках одного Web-приложения, создается один контекст. **Контекст** (context) сервлетов составляют каталоги и файлы, описывающие Web-приложение. Они содержат, в частности, код сервлета, изображения, чертежи, конфигурационный файл `web.xml` и его фрагменты, - все относящееся к сервлету.

При инициализации сервлета некоторые сведения о его контексте заносятся в объект типа `ServletContext`. Методы этого интерфейса позволяют сервлету получить сведения, содержащиеся в контексте.

Метод `getServerInfo()` позволяет получить имя и версию Java EE SDK, методы `getMajorVersion()` и `getMinorVersion()` возвращают номер версии и модификации Servlet API.

# Java и WEB

В контексте можно определить начальные параметры, общие для всего Web-приложения. Они задаются в конфигурационном файле web.xml в элементах <context-param>. Их имена и значения можно получить методами:

```
public Enumeration getInitParameterNames();
public String getInitParameter(String name);
```

Кроме строковых параметров в контексте допустимо определить атрибуты, значениями которых могут служить объекты любых типов Java. Их имена и значения можно получить методами:

```
public Enumeration getAttributeNames();
public Object getAttribute(String name);
```

# Java и WEB

Установить и удалить атрибуты можно методами

```
public void setAttribute(String name, Object value);
public void removeAttribute(String name);
```

Атрибуты — это удобный способ сохранять объекты, общие для всего Web-приложения, разделяемые всеми сервлетами, входящими в Web-приложение, и независимые от отдельных запросов.

**Метод Service**

Основная работа сервлета заключена в методе:

```
public void service(ServletRequest req, ServletResponse resp);
```

# Java и WEB

К этому методу контейнер обращается автоматически после завершения метода `init()` и передает ему объект `req` типа `ServletRequest`, содержащий всю информацию, находящуюся в запросе клиента. Созданием и заполнением объекта `req` тоже занимается контейнер сервлетов. Кроме того, контейнер создает и передает методу `service()` ссылку на пустой объект `resp` типа `ServletResponse`.

Метод `service()` обрабатывает сведения, содержащиеся в объекте `req`, и заносит результаты обработки в объект `resp`. Заполненный объект `resp` передается контейнеру, который через Web-сервер отправляет ответ клиенту. Все эти действия выполняются методами, описанными в интерфейсах `ServletRequest` и `ServletResponse`.

# Java и WEB

**Цикл работы сервлета**

Сервлет загружается контейнером, как правило, при первом запросе к нему или во время запуска контейнера. После выполнения запроса сервлет может быть оставлен в спящем состоянии, ожидая следующего запроса, или выгружен, предварительно выполнив метод `destroy()`. Это зависит от реализации контейнера сервлетов.

Работа сервлета начинается с метода `init()`, затем выполняется метод `service()`, который может создавать объекты, обращаться к их методам, связываться с базами данных и удаленными объектами, выполняя обычную работу обычного класса Java. При этом надо учитывать, что сервлету может быть направлено сразу несколько запросов. Число одновременных запросов в промышленных системах достигает сотен и тысяч. Для обработки каждого запроса контейнер создает новый подпроцесс (`thread`), выполняющий метод `service()`.

# Java и WEB

**Класс GenericServlet**

Абстрактный класс `GenericServlet` реализует сразу интерфейсы `Servlet` и `ServletConfig`.

Кроме реализации методов обоих интерфейсов в него введен пустой метод `init()` без аргументов. Этот метод выполняется автоматически после метода `init(ServletConfig)`, поэтому удобно всю инициализацию записывать в метод `init()` без аргументов, не заботясь о вызове `super.init(config)`.

Класс `GenericServlet` оставляет нереализованным только метод `service()`. Удобно создавать сервлеты, расширяя этот класс и переопределяя только метод `service()`.