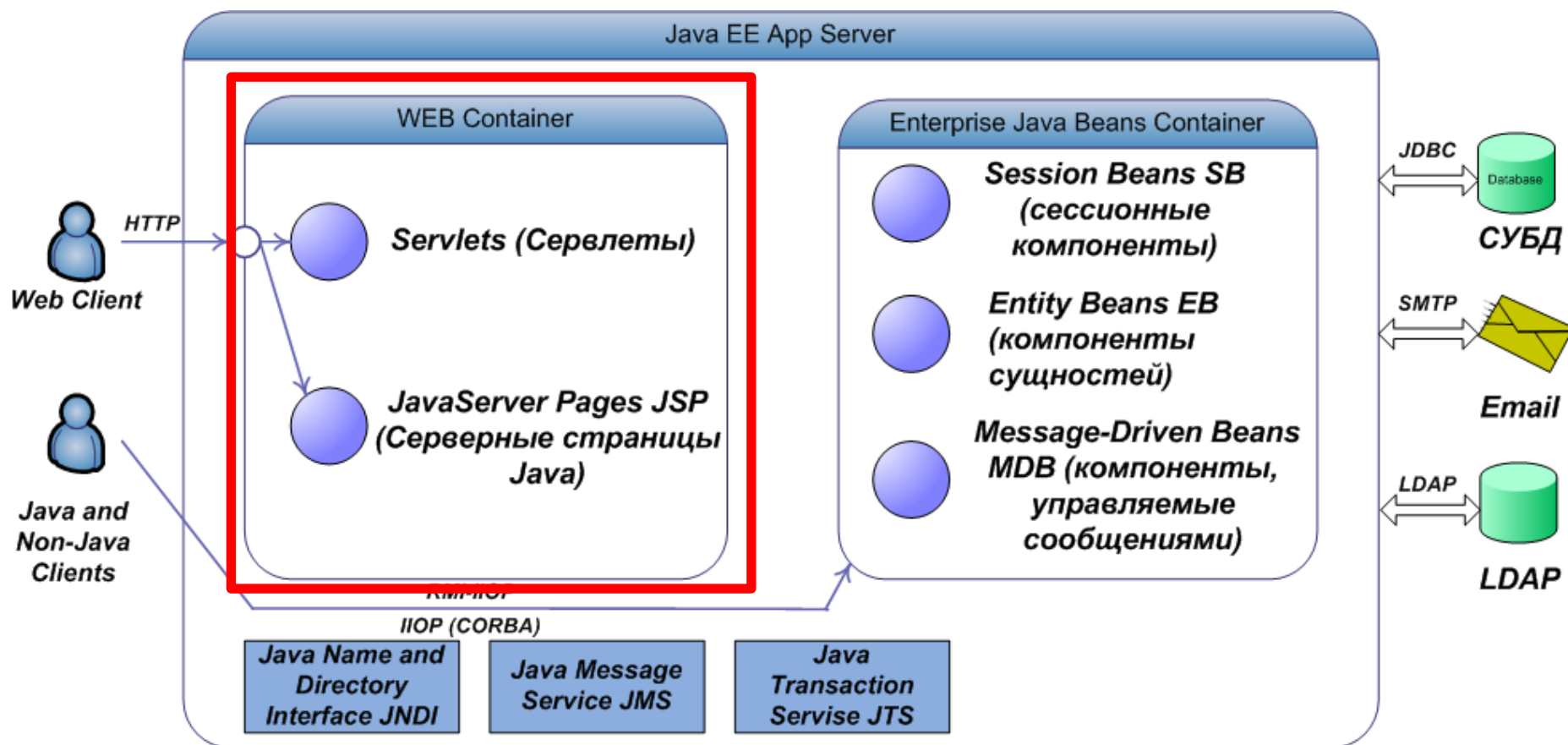


Веб-компоненты

Место и виды веб-компонентов



Сервлет

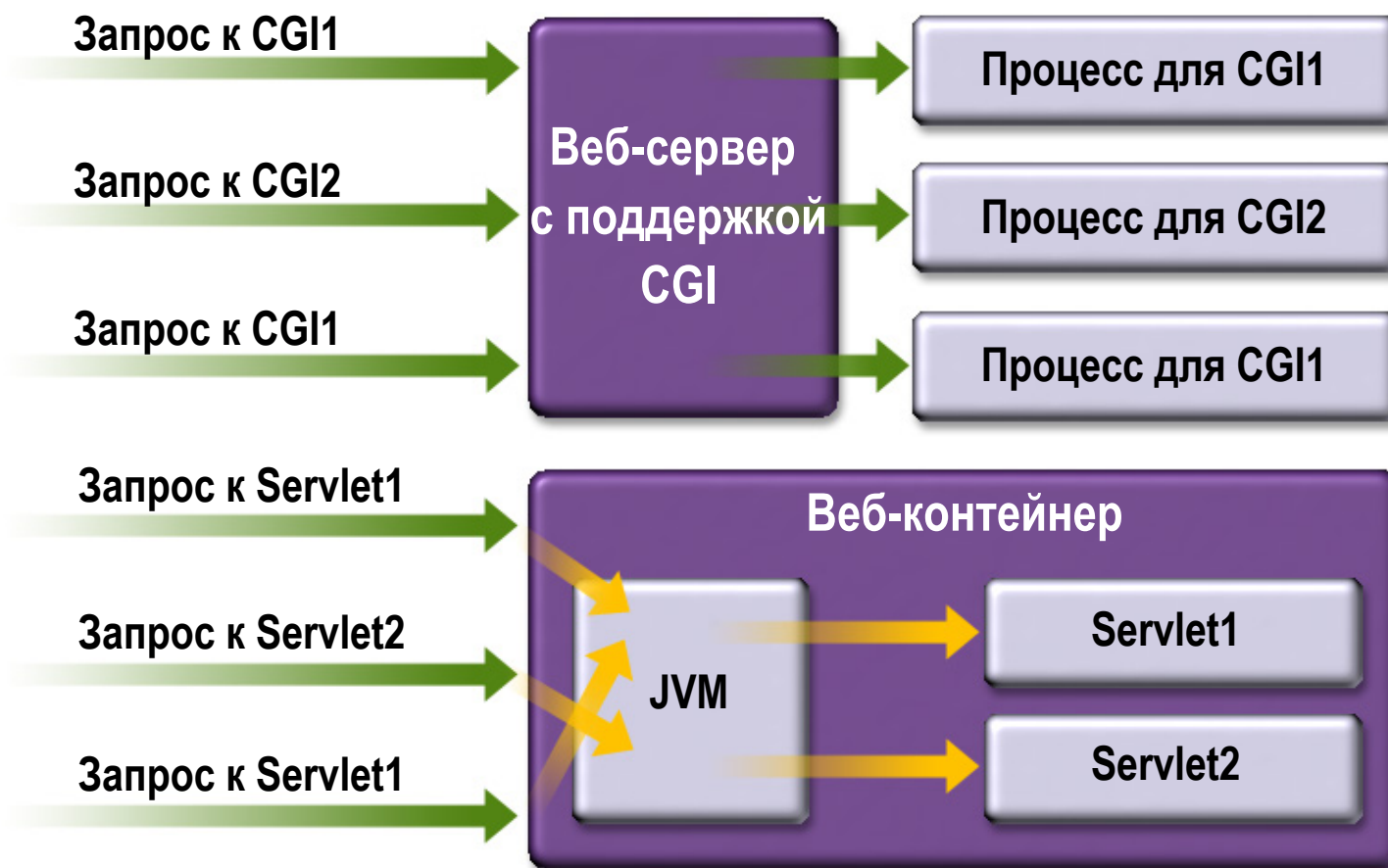
- Предназначен для реализации функций, доступных по веб-протоколам
 - > Расширяет возможности HTTP-сервера
 - > Связывается с определенными URL
- Java-класс, который реализует интерфейс `javax.servlet.Servlet`



Первый сервлет - Hello World!

```
public class HelloServlet extends HttpServlet {  
  
    public void doGet(HttpServletRequest request,  
                      HttpServletResponse response) {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println("<title>Hello World!</title>");  
    }  
    ...  
}
```

Сервлеты против CGI



Преимущества сервлетов

- Нет ограничений, присущих CGI
- Большое количество веб-серверов и сторонних утилит, поддерживающих сервлеты
- Доступ к практически любым Java API
- Лучшая надежность, производительность и масштабируемость
- Не зависят от платформы и сервера
- Безопасность: JVM проверяет код перед выполнением



Технология JavaServer Pages

- Основана на технологии сервлетов
- Поддерживает **разделение** бизнес-логики и представления
 - > Представление в форме **HTML** или **XML/XSLT**
 - > Бизнес-логика реализуется в
 - > **JavaBeans**
 - > **Custom Actions** (действиях, определяемых программистом)
 - > Упрощается поддержка и повторное использование

Первая JSP-страница - Hello World!

```
<html>
  Hello World!
  <br>
  <jsp:useBean id="clock"
               class="calendar.JspCalendar" />
  Today is
  <ul>
  <li>Day of month: <%= clock.getDayOfMonth() %>
  <li>Year: <%= clock.getYear() %>
  </ul>
</html>
```


Применение веб-компонентов

- **Front End Component** — координирует работу различных компонентов веб-приложения
 - > извлекает параметры запроса
 - > вызывает методы бизнес-логики
 - > перенаправляет запрос на presentation component для отображения данных, полученных от бизнес-методов
 - > **Сервлеты** — рекомендуется, **JSP** — нет
- **Presentation Component** — генерирует окончательный ответ
 - > **JSP** — рекомендуется
 - > **Сервлеты** — только для генерации ответа в двоичном формате (изображение, аудиофайл, ...)

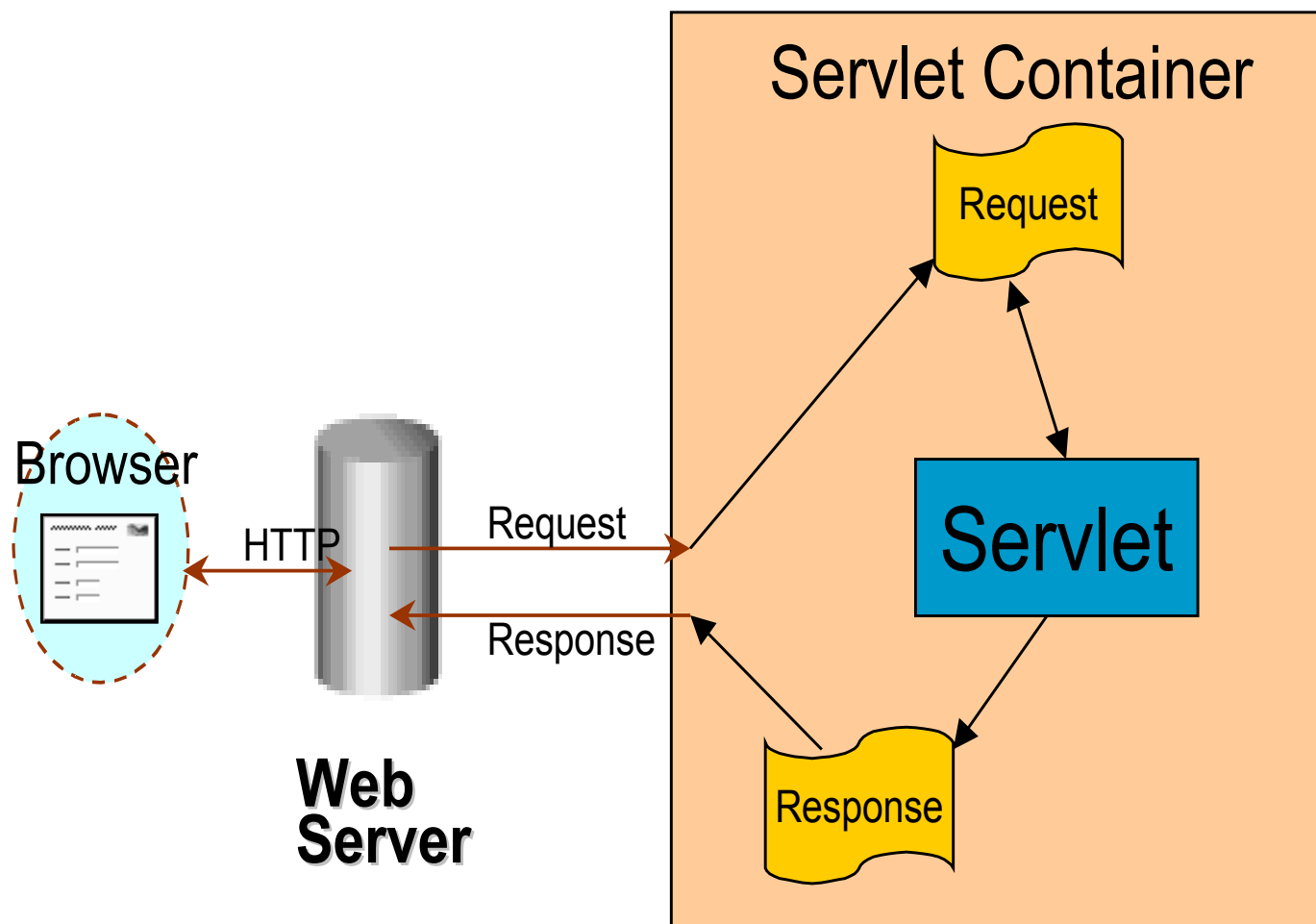
Сервлеты



Содержание

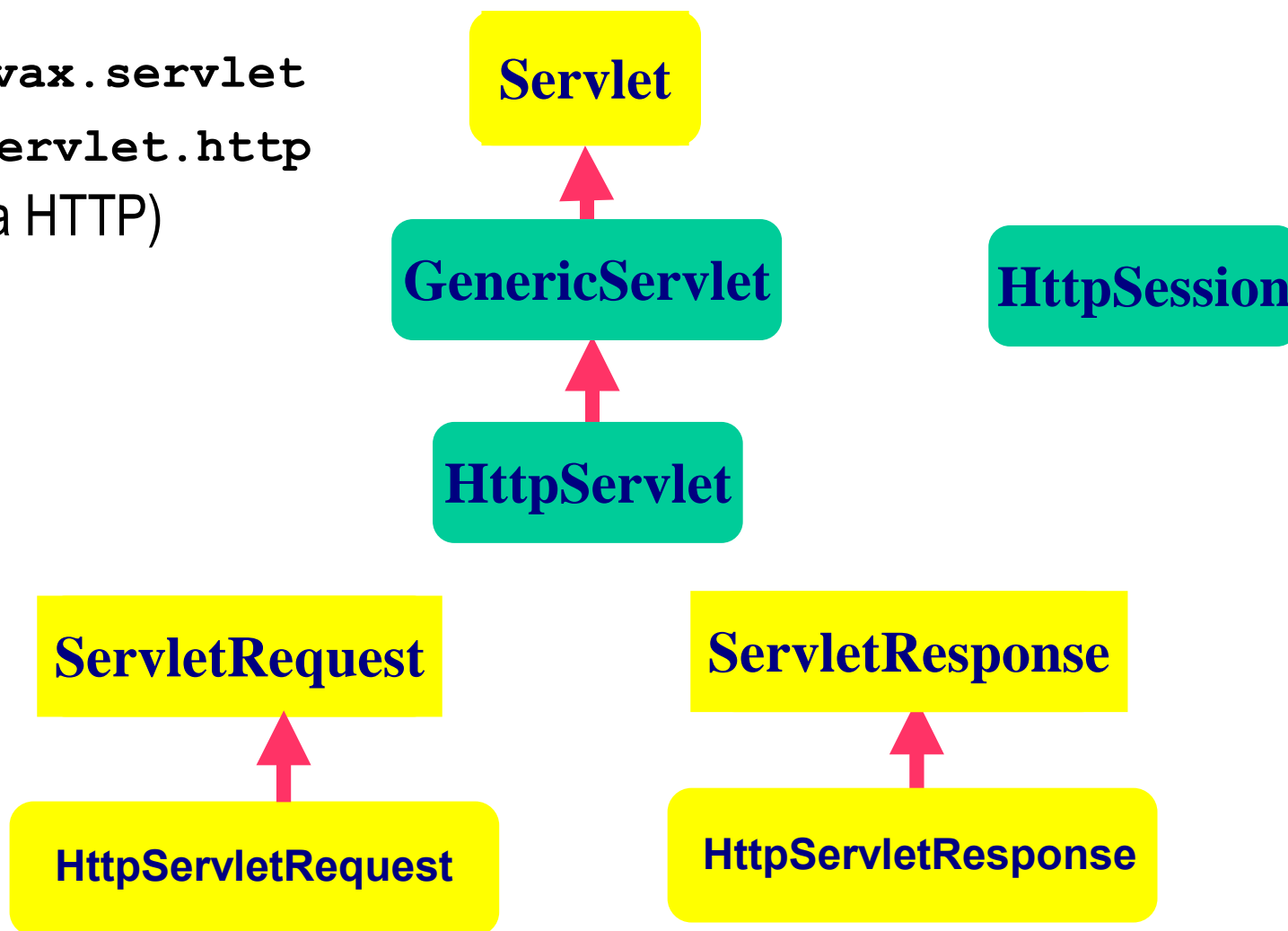
- Определение, ЖЦ, виды (Generic и HTTP)
- Конфигурация сервлета
- Контекст сервлета
- Запрос, ответ, специфика протокола HTTP
- Перенаправление запросов

Принцип работы сервлета



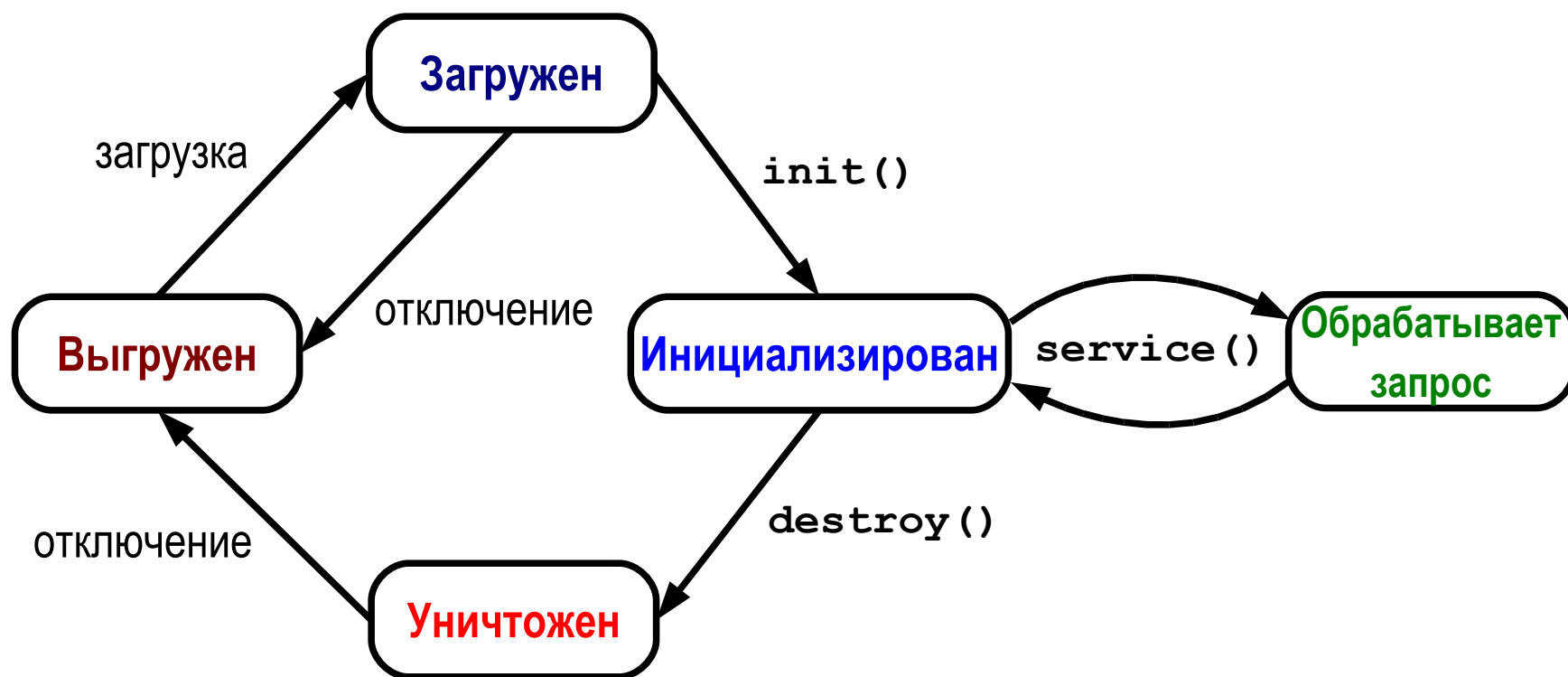
Классы и интерфейсы Servlet API

Пакеты `javax.servlet`
и `javax.servlet.http`
(специфика HTTP)



Жизненный цикл сервлета

- Жизненным циклом управляет веб-контейнер



Методы управления жизненным циклом сервлета

- `init()`
 - > Вызывается **1 раз** при инициализации экземпляра сервлета
 - > В нем выполняется вся настройка, в т.ч. выделение ресурсов
- `destroy()`
 - > Вызывается перед удалением экземпляра сервлета
 - > В нем выполняется освобождение ресурсов

Настройка сервлета

- Интерфейс **ServletConfig** обеспечивает доступ к **строковым** параметрам инициализации, размещающимся в **установочном дескрипторе**

```
public void init(ServletConfig config) throws
    ServletException {
    super.init(config);
    String driver = getInitParameter("driver");
    String fURL = getInitParameter("url");
    try {
        openDBConnection(driver, fURL);
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```




Пример настройки сервлета: установочный дескриптор (web.xml)

```
<web-app>
  <servlet>
    <servlet-name>chart</servlet-name>
    <servlet-class>ChartServlet</servlet-class>
    <init-param>
      <param-name>driver</param-name>
      <param-value>
        COM.cloudscape.core.RmiJdbcDriver
      </param-value>
    </init-param>

    <init-param>
      <param-name>url</param-name>
      <param-value>
        jdbc:cloudscape:rmi:CloudscapeDB
      </param-value>
    </init-param>
  </servlet>
</web-app>
```



Методы управления жизненным циклом сервлета

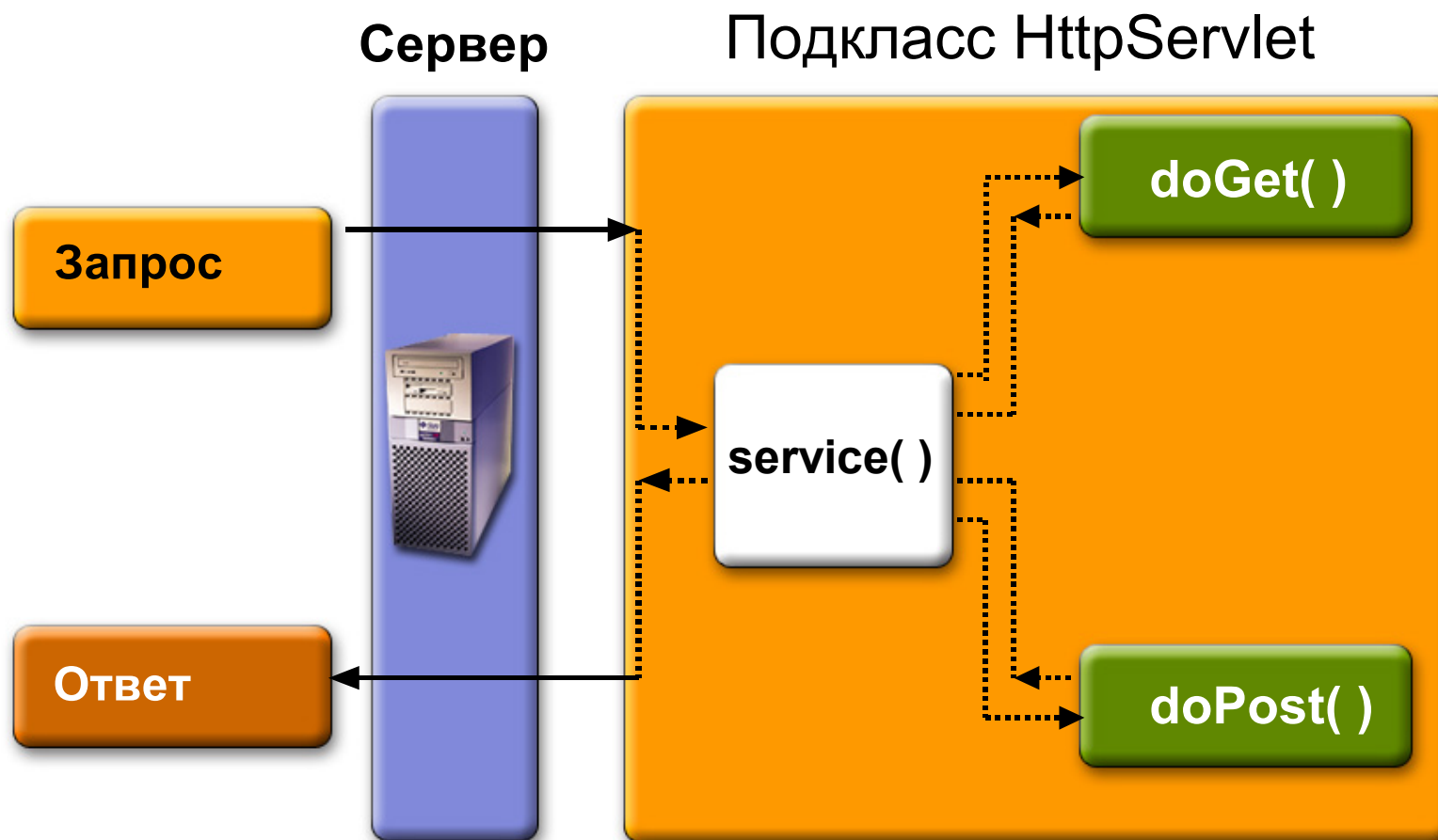
- **service ()** В классе `javax.servlet.GenericServlet`
 - > абстрактный метод
- **service ()** В классе `javax.servlet.http.HttpServlet`
 - > конкретный метод (реализация)
 - > делегирует обработку в методы `doGet()`, `doPost()`, ...
 - > **не переопределяйте** этот метод
- **doGet ()**, **doPost ()**, **doXxx ()** В классе `javax.servlet.http.HttpServlet`
 - > обработка HTTP-запросов с методами GET, POST, ...
 - > **переопределяйте** именно эти методы



Различие между методами `service()` и `doGet()/doPost()`

- `service()` работает с обобщенными запросами и ОТВЕТАМИ:
 - > `service(ServletRequest request, ServletResponse response)`
- `doGet()` и `doPost()` работают с HTTP-запросами и HTTP-ответами:
 - > `doGet(HttpServletRequest request, HttpServletResponse response)`
 - > `doPost(HttpServletRequest request, HttpServletResponse response)`

Методы doGet() и doPost()



Легенда:  Реализуется в подклассе

Что делать в методах doGet() и doPost()?

- Получить информацию (HTTP-параметры) из HTTP-запроса
- Установить (set) и получить (get) атрибуты объектов из различных контекстов
- Выполнить бизнес-логику (обратиться к БД)
- Перенаправить запрос другому веб-компоненту (необязательно)
- Составить HTTP-ответ и отправить его клиенту

Пример реализации doGet()

```
public void doGet (HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {

    // Получить объект "messages" из контекста сессии
    HttpSession session = request.getSession(true);
    ResourceBundle messages = (ResourceBundle)session.getAttribute("messages");

    // Установить заголовки и размер буфера перед получением объекта-"писателя"
    response.setContentType("text/html");
    response.setBufferSize(8192);
    PrintWriter out = response.getWriter();

    // Записать в ответ заголовочную часть HTML-документа
    out.println("<html>" +
               "<head><title>" + messages.getString("TitleBookDescription") +
               "</title></head>");

    // Получить объект диспетчера для баннера и включить баннер в ответ
    RequestDispatcher dispatcher =
        getServletContext().getRequestDispatcher("/banner");

    if (dispatcher != null)
        dispatcher.include(request, response);
}
```

Пример реализации doGet()

```
// Получить ID книги для отображения (получить HTTP-параметр)
String bookId = request.getParameter("bookId");
if (bookId != null) {

    // и сведения о книге (выполнить бизнес-логику)
    try {
        BookDetails bd = bookDB.getBookDetails(bookId);
        Currency c = (Currency) session.getAttribute("currency");
        if (c == null) {
            c = new Currency();
            c.setLocale(request.getLocale());
            session.setAttribute("currency", c);
        }
        c.setAmount(bd.getPrice());

        // Вывести полученные сведения
        out.println("...");
    } catch (BookNotFoundException ex) {
        response.resetBuffer();
        throw new ServletException(ex);
    }
}
out.println("</body></html>");
out.close();
}
```

Контексты

- Поддерживают **обмен информацией** между веб-компонентами в виде **атрибутов**
 - > Атрибут — это пара объектов «имя/значение»
- Методы для доступа к атрибутам в контексте
 - > `getAttribute()` и `setAttribute()`
- Определено 4 контекста
 - > Приложение, сессия, запрос, страница

Контексты

- **Приложение** - `javax.servlet.ServletContext`
 - > Доступен всем веб-компонентам в рамках одного веб-приложения
- **Сессия** - `javax.servlet.http.HttpSession`
 - > Доступен всем веб-компонентам, обрабатывающим запрос, сделанный в рамках сессии
- **Запрос** - подтип `javax.servlet.ServletRequest`
 - > Доступен всем веб-компонентам, обрабатывающим конкретный запрос
- **Страница** - `javax.servlet.jsp.PageContext`
 - > Доступен на конкретной **JSP**-странице

Контекст сервлета - ServletContext

- **Один** объект ServletContext для конкретного веб-приложения в конкретной JVM
- **Функции:**
 - 1) доступ к общим параметрам конфигурации для всего веб-приложения
 - 2) обмен информацией между веб-компонентами с помощью атрибутов контекста
 - 3) перенаправление запросов
 - 4) запись сообщений в системный журнал
 - 5) получение разнообразной информации
 - веб-контейнер, версия Servlet API, веб-приложение, ресурсы веб-приложения



Примеры использования контекста сервлета

```
public class CatalogServlet extends HttpServlet {
    private BookDB bookDB;
    public void init() throws ServletException {
        // Получить значение атрибута из контекста приложения
        bookDB = (BookDB)getServletContext().getAttribute("bookDB");
    }
    public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException {
        ...
        // Получить объект диспетчера для баннера и включить баннер в ответ
        RequestDispatcher dispatcher =
            session.getServletContext().getRequestDispatcher("/banner");

        if (dispatcher != null)
            dispatcher.include(request, response);

        ...
        // Вывод сообщения в системный журнал
        getServletContext().log("Life is good!");
        ...
    }
}
```

Сессия - HTTPSession

- Механизм для поддержания **сеанса работы клиента** с веб-сервером
 - > Протокол HTTP — протокол **без состояния**
- Хранит на сервере информацию, специфичную **для конкретного клиента**, в промежутках между его обращениями к серверу
 - > В виде атрибутов
 - > Пример: «корзина» для покупок в Интернет-магазине



Пример использования сессии

```
public class CashierServlet extends HttpServlet {  
    public void doGet (HttpServletRequest request,  
                      HttpServletResponse response)  
        throws ServletException, IOException {  
  
        // Получить сессию и корзину с покупками  
        HttpSession session = request.getSession();  
        ShoppingCart cart =  
            (ShoppingCart) session.getAttribute("cart");  
  
        // Определить общую стоимость покупок  
        double total = cart.getTotal();  
    }  
}
```

Запрос к сервлету

- Содержит данные, переданные клиентом
- Все запросы к сервлету реализуют интерфейс **ServletRequest**, предоставляющий функции:
 - > Получение параметров, переданных клиентом
 - > Хранение атрибутов
 - > Работа с телом запроса
 - > Получение сведений о клиенте и сервере

Обработка запроса

данные,
клиент, сервер,
заголовки

Запрос



Сервлет 1

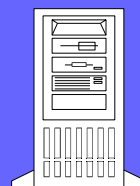


Сервлет 2



Сервлет 3

Ответ



Веб-сервер

Запрос — получение параметров

- В запросе может быть любое количество параметров
- Параметры отправляются из HTML-форм:
 - > GET: в URL
 - > POST: в теле запроса
- `getParameter("paraName")`
 - > Возвращает **строковое** значение параметра `paraName`
 - > Может потребоваться преобразование String в нужный тип данных
 - > Возвращает **null**, если параметр отсутствует
 - > Работает одинаково для GET- и POST-запросов

Пример получения параметров

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Collecting Three Parameters</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">Please Enter Your Information</H1>

<FORM ACTION="/sample/servlet/ThreeParams">
  First Name:  <INPUT TYPE="TEXT" NAME="param1"><BR>
  Last Name:   <INPUT TYPE="TEXT" NAME="param2"><BR>
  Class Name:  <INPUT TYPE="TEXT" NAME="param3"><BR>
  <CENTER>
    <INPUT TYPE="SUBMIT">
  </CENTER>
</FORM>

</BODY>
</HTML>
```

Collecting Three Parameters

Please Enter Your In

First Name:

Last Name:

NickName:

Пример получения параметров

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
/** Простой сервлет, считывающий три параметра из HTML-формы */
public class ThreeParams extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Your Information";
        out.println("<HTML>" +
                    "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                    "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
                    "<UL>\n" +
                    "  <LI><B>First Name in Response</B>: "
                    + request.getParameter("param1") + "\n" +
                    "  <LI><B>Last Name in Response</B>: "
                    + request.getParameter("param2") + "\n" +
                    "  <LI><B>NickName in Response</B>: "
                    + request.getParameter("param3") + "\n" +
                    "</UL>\n" +
                    "</BODY></HTML>");
    }
}
```



Запрос — установка атрибутов

- Веб-контейнер может установить атрибуты с дополнительными сведениями о запросе
 - > например: атрибут `javax.servlet.request.X509Certificate` для HTTPS
- Сервлет может установить атрибуты для передачи данных по цепочке обработки запроса
 - > `void setAttribute(java.lang.String name, java.lang.Object o)`
 - > Атрибуты устанавливаются **перед** вызовом диспетчера запросов **RequestDispatcher**



Запрос — сведения о клиенте и сервере

- Клиент
 - > `Locale request.getLocale()`
 - > Региональные настройки клиента (локаль)
 - > `String request.getRemoteAddr()`
 - > IP-адрес клиента
 - > `String request.getRemoteHost()`
 - > Имя хоста клиента
- Сервер
 - > `String request.getServerName()`
 - > например, "www.sun.com"
 - > `int request.getServerPort()`
 - > например, 8080



Запрос — получение прочих сведений

- Тело запроса
 - > `ServletInputStream getInputStream()` - двоичный поток
 - > `java.io.BufferedReader getReader()` - текстовый поток
- Используемый протокол
 - > `java.lang.String getProtocol()`
- Тип содержимого
 - > `java.lang.String getContentType()`
- Использование защищенного соединения (HTTPS)
 - > `boolean isSecure()`



HTTP-запрос - HttpServletRequest

- Содержит данные, переданные HTTP-клиентом в HTTP-сервлет
- Создается веб-контейнером и передается сервлету как параметр методов `doGet()` и `doPost()`
- `HttpServletRequest` расширяет `ServletRequest` и предоставляет дополнительные сведения:
 - > Метод HTTP-запроса, URL запроса
 - > Заголовки
 - > Куки (Cookies)
 - > HTTP-сессия
 - > Сведения о пользователе и безопасности

URL HTTP-запроса: компонент path

schema://[**user:password**@]**host**[:**port**]/[**path**][?**query**][#**hash**]

- [**path**] СОСТОИТ ИЗ
 - > /<корень веб-приложения>
 - > /<путь к сервлету>
 - > /<информация о пути>
- Методы в HttpServletRequest
 - > getContextPath(), getServletPath(), getPathInfo(),
getPathTranslated(), getQueryString()
- Примеры
 - http://localhost:8080/**hello1**/**greeting**
 - http://localhost:8080/**hello1**/**greeting.jsp**
 - http://daydreamer/**catalog**/**lawn**/index.html

Работа с заголовками HTTP-запроса

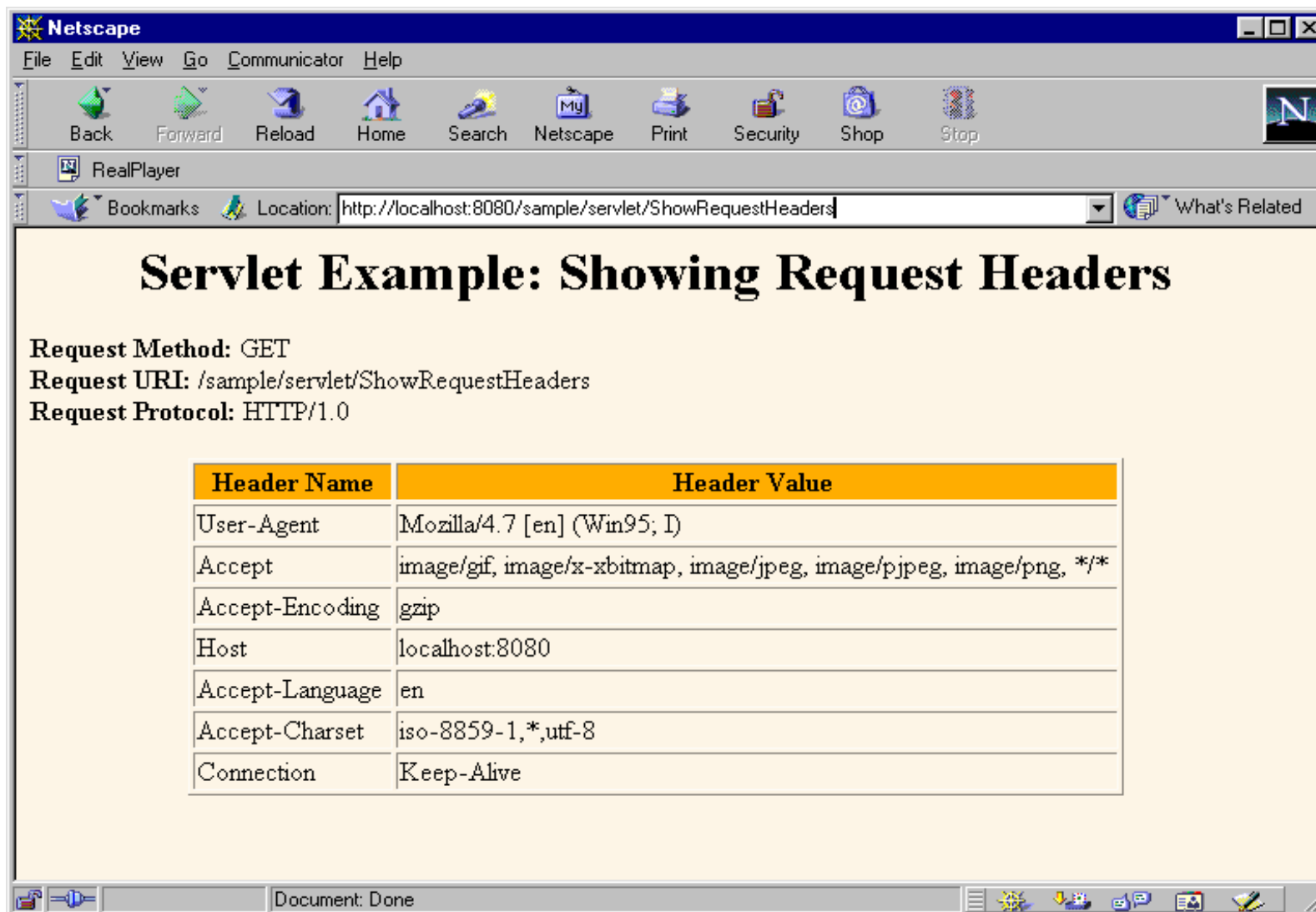
- Перечень заголовков запроса
 - > `java.util.Enumeration getHeaderNames()`
- Перечень строковых значений заголовка
 - > `java.util.Enumeration getHeaders(name)`
- Значение заголовка в виде строки, целого числа, момента времени
 - > `String getHeader(String name)`
 - > `int getIntHeader(String name)`
 - > `long getDateHeader(String name)`
- Куки (значения заголовка Cookie — см. RFC 2965)
 - > `Cookie[] getCookies()`



Отображение заголовков HTTP-запроса

```
// Выводит все заголовки принятого запроса
public class ShowRequestHeaders extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Servlet Example: Showing Request Headers";
        out.println("<HTML>" + ...
            "<B>Request Method: </B>" +
            request.getMethod() + "<BR>\n" +
            "<B>Request URI: </B>" +
            request.getRequestURI() + "<BR>\n" +
            "<B>Request Protocol: </B>" +
            request.getProtocol() + "<BR><BR>\n" +
            ...
            "<TH>Header Name<TH>Header Value");
        Enumeration headerNames = request.getHeaderNames();
        while(headerNames.hasMoreElements()) {
            String headerName = (String)headerNames.nextElement();
            out.println("<TR><TD>" + headerName);
            out.println("    <TD>" + request.getHeader(headerName));
        }
        ...
    }
}
```

Отображение заголовков HTTP-запроса



The screenshot shows the Netscape Communicator window. The title bar says "Netscape". The menu bar includes File, Edit, View, Go, Communicator, and Help. The toolbar contains icons for Back, Forward, Reload, Home, Search, Netscape, Print, Security, Shop, and Stop. The address bar shows the location: `http://localhost:8080/sample/servlet/ShowRequestHeaders`. The main content area displays the title "Servlet Example: Showing Request Headers" and the following information:

Request Method: GET
Request URI: /sample/servlet/ShowRequestHeaders
Request Protocol: HTTP/1.0

Header Name	Header Value
User-Agent	Mozilla/4.7 [en] (Win95; I)
Accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding	gzip
Host	localhost:8080
Accept-Language	en
Accept-Charset	iso-8859-1,*,utf-8
Connection	Keep-Alive

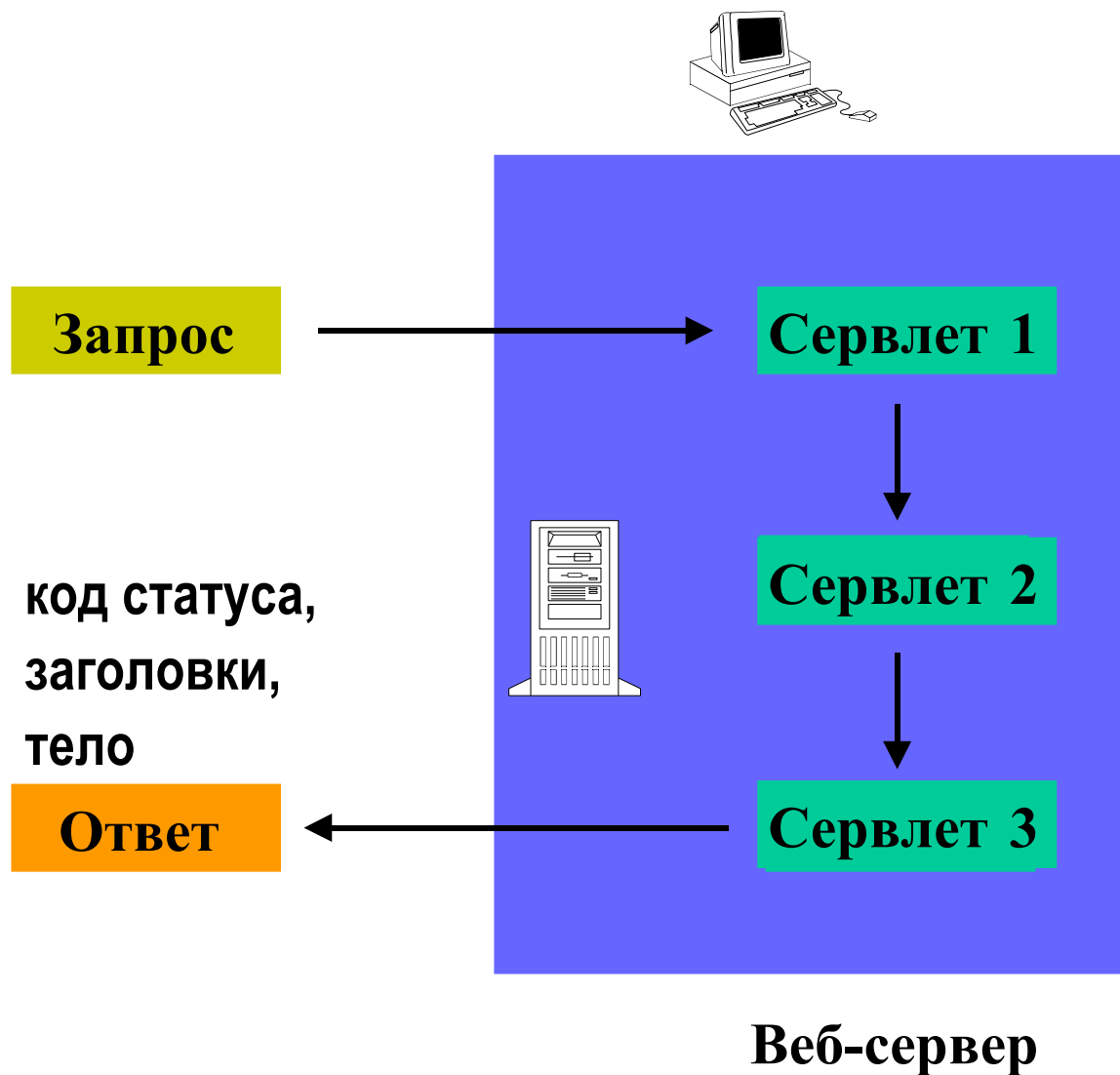
The status bar at the bottom shows "Document: Done".



Ответ сервлета

- Содержит данные, которые сервлет передает клиенту
- Все ответы сервлета реализуют интерфейс **ServletResponse**
 - > Формирование тела ответа
 - > Определение характеристик ответа (MIME-тип, кодировка и локаль)
 - > Управление буферизацией
- **HttpServletResponse** расширяет ServletResponse
 - > Код статуса HTTP-ответа
 - > Заголовки и куки
 - > Перенаправление запроса

Формирование ответа



Процедура формирования ответа

- 1) Установить заголовки ответа
- 2) Установить некоторые свойства ответа
 - Размер буфера
- 3) Получить поток вывода для ответа
- 4) Записать тело ответа в поток вывода



Пример простого формирования ответа

```
public class HelloServlet extends HttpServlet {  
    public void doGet(HttpServletRequest request,  
                        HttpServletResponse response)  
        throws ServletException, IOException {  
  
        // Установить заголовки ответа  
        response.setContentType("text/html");  
        // Установить размер буфера  
        response.setBufferSize(8192);  
        // Получить поток вывода  
        PrintWriter out = response.getWriter();  
        // Записать тело ответа в поток вывода  
        out.println("<title>First Servlet</title>");  
        out.println("<big>Hello J2EE Programmers! </big>");  
    }  
}
```

Установка кода статуса для HTTP-ответа

- `public void setStatus(int statusCode)`
 - > Константы для кодов статуса определены в `HttpServletResponse`
 - > Код статуса по умолчанию - **200 (OK)**
- Коды ошибок (**400-599**) можно использовать в методах `sendError`
 - > `public void sendError(int statusCode)`
 - > Сервер может обработать ошибку особым образом
 - > `public void sendError(int statusCode, String message)`
 - > Сообщение возвращается в небольшом HTML-документе

Примеры возврата ошибки

```
try {
    returnAFile(fileName, out)
}
catch (FileNotFoundException e) {
    response.setStatus(response.SC_NOT_FOUND);
    out.println("Response body");
}
```

действует так же, как

```
try {
    returnAFile(fileName, out)
}
catch (FileNotFoundException e)
{
    response.sendError(response.SC_NOT_FOUND);
}
```




Установка произвольных заголовков ответа

- Установка значения заголовка в виде строки, целого числа, момента времени
 - > `public void setHeader(String headerName, String headerValue)`
 - > `public void setDateHeader(String name, long millisecs)`
 - > `public void setIntHeader(String name, int headerValue)`
- Добавление значения заголовка
 - > `addHeader`, `addDateHeader`, `addIntHeader`

Установка типичных заголовков ответа

- **setContentType**
 - > Устанавливает заголовок Content-Type. Используется практически во всех сервлетах
- **setContentLength**
 - > Устанавливает заголовок Content-Length. Используется для постоянных HTTP-соединений
- **addCookie**
 - > Добавляет значение к заголовку Set-Cookie
- **sendRedirect**
 - > Устанавливает заголовок Location и изменяет код статуса



Формирование тела ответа

- Сервлет практически всегда возвращает тело ответа
- `java.io.PrintWriter`
 - > возвращается методом `getWriter()`
 - > для генерации текстовых данных (обычный текст, HTML-документ, XML-документ)
- `javax.servlet.ServletOutputStream`
 - > возвращается методом `getOutputStream()`
 - > для генерации двоичных данных (изображение, видео, звук)