БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ и ИНФОРМАТИКИ
Кафедра многопроцессорных сетей и систем

Рафеенко Е.Д.

# Web- программирование

## Введение в сервлеты

Содержание

# Введение

Сервлеты – это компоненты приложений Jakarta Enterprise Edition (Jakarta EE), выполняющиеся на стороне сервера, способные обрабатывать клиентские запросы и динамически генерировать ответы на них. Наибольшее распространение получили сервлеты, обрабатывающие клиентские запросы по протоколу HTTP.

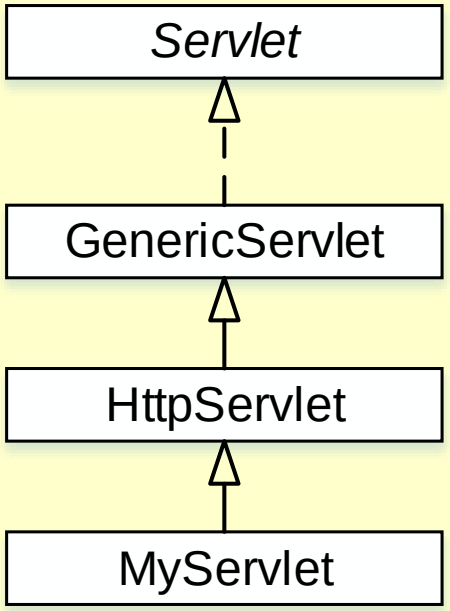
Сервлеты можно внедрять в различные сервера, так как API сервлета, который вы используете для его написания, ничего не «знает» ни о среде сервера, ни о его протоколе. Множество Web-серверов поддерживает технологию Java Servlet.

# Servlet API

Пакет jakarta.servlet (javax.servlet) обеспечивает интерфейсы и классы для написания сервлетов.

Центральной абстракцией API сервлета является интерфейс Servlet. Все сервлеты реализуют данный интерфейс напрямую, но более распространено расширение класса, реализующего его, как HttpServlet.

Интерфейс Servlet объявляет, но не реализует методы, которые управляют сервлетом и его взаимодействием с клиентами.



# Servlet API

Принимая запрос от клиента, сервлет получает два объекта:

1. ServletRequest, который инкапсулирует связь клиента с сервером.
2. ServletResponse, который инкапсулирует обратную связь сервлета с клиентом.

ServletRequest и ServletResponse – это интерфейсы, определенные пакетом jakarta.servlet.

# Интерфейс ServletRequest

Интерфейс ServletRequest дает сервлету доступ к:

1. Информации, такой как имена параметров, переданных клиентом, протоколы (схемы), используемые клиентом и имена удаленного хоста, создавшего запрос и сервера который их получает.
2. Входному потоку ServletInputStream. Сервлеты используют входной поток для получения данных от клиентов, которые используют протоколы приложений, такие как HTTP POST и PUT методы.

Интерфейсы, которые расширяют интерфейс ServletRequest, позволяют сервлету получать больше данных, характерных для протокола. К примеру, интерфейс HttpServletRequest содержит методы для доступа к главной информации по протоколу HTTP.

# Интерфейс ServletResponse

Интерфейс ServletResponse дает сервлету методы для ответа на запросы клиента. Он:

1. Позволяет сервлету устанавливать длину содержания и тип MIME ответа (метод setContentType(String type)).
2. Обеспечивает исходящий поток ServletOutputStream и Writer, через которые сервлет может отправлять ответные данные.

Интерфейсы, которые расширяют интерфейс ServletResponse, дают сервлету больше возможнойстей для работы с протоколами. Например, интерфейс HttpServletResponse содержит методы, которые позволяют сервлету манипулировать информацией заголовка HTTP.

# Дополнительные возможности HTTP сервлетов

В HTTP сервлетах есть возможность управления сессией (session-tracking). Автор сервлета может использовать этот API для поддержания состояния между сервлетом и клиентом, которое сохраняется на нескольких соединениях в течении определенного времени.

В HTTP сервлетах также есть объекты, которые обеспечивают создание файлов-cookie. Автор сервлета использует файлы-cookie API для сохранения данных, ассоциированных с клиентом и для получения этих данных.

# Жизненный цикл сервлета

*Контейнер*, например Tomcat, GlassFish, управляет средой исполнения для сервлетов. Вы можете настроить способ функционирования контейнера для отображения сервлетов внешнему миру.

Используя различные конфигурационные файлы контейнера, вы обеспечиваете мост от URL (введенного пользователем в браузере) к серверным компонентам, обрабатывающим запрос, в который транслируется URL. Во время работы приложения контейнер загружает и инициализирует сервлет (сервлеты) и управляет его *жизненным циклом*.

Сервлеты имеют жизненный цикл, это означает, что при активизации сервлета все работает предсказуемо. Другими словами, для любого создаваемого вами сервлета всегда будут вызываться определенные методы в определенном порядке.

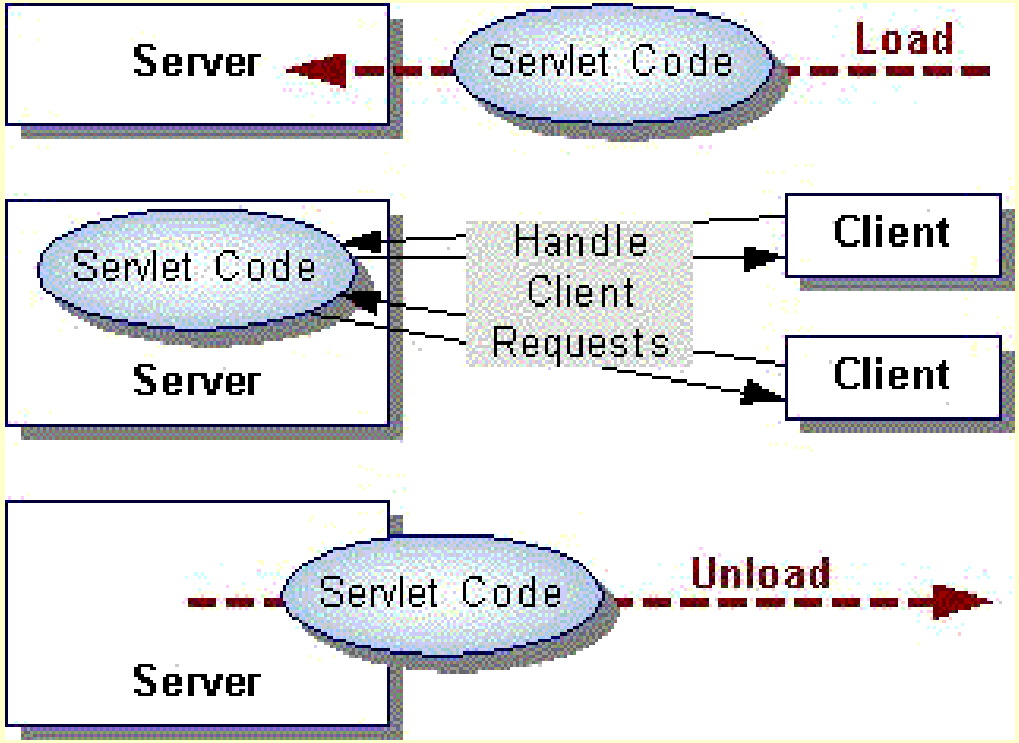
# Жизненный цикл сервлета

- Пользователь вводит URL в браузере. Конфигурационный файл Web-сервера указывает, что этот URL предназначен для сервлета, управляемого контейнером сервлетов на сервере.

  Если экземпляр сервлета еще не был создан (*существует только один экземпляр сервлета для приложения*), контейнер загружает класс и создает экземпляр объекта.
- Контейнер вызывает метод init() сервлета. Он дает сервлету возможность инициализировать данные и подготовиться для обработки запросов.
- Контейнер вызывает метод service() сервлета и передает HttpServletRequest и HttpServletResponse.
  Для каждого нового клиента при обращении к сервлету создается **независимый поток,** в котором производится вызов метода service(). Метод service() предназначен для одновременной обработки множества запросов.
- При необходимости, когда сервлет выполнил полезную работу, контейнер вызывает метод destroy() сервлета, в теле которого следует помещать код освобождения занятых сервлетом ресурсов.

# Жизненный цикл сервлета

Метод service() класса HttpServlet служит диспетчером для других методов, каждый из которых обрабатывает методы доступа к ресурсам.
В спецификации HTTP определены следующие методы: GET, HEAD, POST, PUT, DELETE, OPTIONS и TRACE. Наиболее часто употребляются методы GET и POST, с помощью которых на сервер передаются запросы, а также параметры для их выполнения.



# HttpServlet - Запросы и ответы

Методы класса HttpServlet, которые управляют клиентскими запросами принимают два аргумента:

1. Объект HttpServletRequest, который инкапсулирует данные от клиента
2. Объект HttpServletResponse, который инкапуслирует ответ к клиенту

# Объект HttpServletRequest

Объекты HttpServletRequest предоставляют доступ к данным HTTP заголовка и позволяют получить аргументы, которые клиент направил вместе с запросом.

Непосредственно в интерфейсе HttpServletRequest объявлен ряд методов, позволяющих манипулировать содержимым запросов:

- String getParameter(String name) - возвращает значение именованного параметра.
- String[] getParameterValues(String name) - возвращает массив величин именованного параметра. Используется, если параметр может иметь более чем одну величину.
- Enumeration getParameterNames() - предоставляет имена параметров.
- String getQueryString() - возвращает строковую (String) величину необработанных данных клиента для HTTP запросов GET.
- BufferedReader getReader() - возвращает объект BufferedReader (текстовая информация) для считатывания необработанных данных (для HTTP запросов POST, PUT, и DELETE).
- ServletInputStream getInputStream() - возвращает объект ServletInputStream (двоичная информация) для считывания необработанных данных (для HTTP запросов для POST, PUT, и DELETE).

# Объект HttpServletResponse

Объект HttpServletResponse обеспечивает два способа возвращения данных пользователю:

Метод getWriter возвращает Writer

Метод getOutputStream возвращает ServletOutputStream

Используйте метод getWriter для возвращения пользователю *текстовых данных* и метод getOutputStream для *бинарных*.

Закрытие Writer или ServletOutputStream после отправления запроса позволит серверу определить, что ответ готов.

# Интерфейс HttpServletResponse

• void sendError(int sc, String msg) – сообщение о возникших ошибках, где sc – код ошибки, msg – текстовое сообщение;

• void setDateHeader(String name, long date) – добавление даты в заголовок ответа;

• void setHeader(String name, String value) – добавление параметров в заголовок ответа. Если параметр с таким именем уже существует, то он будет заменен.

# Данные HTTP заголовка

Необходимо установить данные HTTP заголовка, прежде чем вы получите доступ к объектам Writer или OutputStream. Класс HttpServletResponse предоставляет методы для доступа к данным заголовка. setContentType(String type) - устанавливает тип содержимого (Content-type).

```
response.setContentType("text/html");
```

# Первый сервлет

```
public class MyServlet
      extends jakarta.servlet.http.HttpServlet {
protected void doGet(HttpServletRequest request,
   HttpServletResponse response)
      throws ServletException, IOException {
      PrintWriter out = response.getWriter();
      String title = "Simple Servlet Output";
// сначала установите тип содержания и другие поля заголовков ответа
      response.setContentType("text/html");
      out.println("<HTML><HEAD><TITLE>");
      out.println(title);
      out.println("</TITLE></HEAD><BODY>");
      out.println("<H1>" + title + "</H1>");
      out.print("<P>This is ");
      out.print(this.getClass().getName());
      out.print(", using the GET method");
      out.println("</BODY></HTML>");
      out.close();
      }
}
```

# Первый сервлет

Сервлет расширяет класс HttpServlet и переопределяет метод doGet.

Чтобы ответить клиенту, метод doGet в примере использует Writer из объекта HttpServletResponse для возвращения клиенту текстовых данных. Прежде чем получить доступ к writer, пример устанавливает заголовок content-type. В конце метода doGet, после того как был отправлен запрос, Writer закрывается.

Для работы с конкретным HTTP-методом передачи данных достаточно расширить свой класс от класса HttpServlet и реализовать один из этих методов. Метод service () переопределять не надо, в классе HttpServlet он только определяет, каким HTTP-методом передан запрос клиента, и обращается к соответствующему методу doXxx().

# Развертывание проекта

При развертывании web-приложения, помимо самих класса сервлета, надо создать дескриптор развертывания, который оформляется в виде XML-файла web.xml.

Web-приложение поставляется в виде архива .war, содержащего все его файлы. На самом деле это zip-архив, расширение .war нужно для того, чтобы Web-контейнер узнавал архивы развертываемых на нем Web-приложений. Содержащаяся в этом архиве структура директорий должна включать директорию WEB-INF, вложенную непосредственно в корневую директорию приложения.

Директория WEB-INF содержит две поддиректории —

/classes для .class-файлов сервлетов, классов и интерфейсов EJB-компонент и других Java-классов, и

/lib для .jar и .zip файлов, содержащих используемые библиотеки.

Файл web.xml также должен находится непосредственно в директории WEB-INF.

War-файл может быть развернут в admin панели сервера Glassfish или Tomcat

# Структура папок Web приложения

- Application Root (Demo1)
- WEB-INF/web.xml
- WEB-INF/classes
  - Скомпилированные классы сервлетов и другие необходимые классы
- WEB-INF/lib
  - Необходимые библиотеки (jar файлы)

Любой JEE совместимый App Server требует такую структуру

# Описание сервлета

Для того, чтобы сервлет работал на сервере, необходимо
*либо* описать сервлет в файле web.xml - дескрипторе развертывания приложения,

*либо* проаннотировать класс-сервлет при помощи аннотации @WebServlet. Это аннотация уровня класса. Указывает, что аннотированный класс является сервлетом, и содержит некоторые метаданные о сервлете. Обязательный атрибут urlPatterns определяет шаблон URL запросов, вызывающих этот сервлет.

```
@WebServlet(name="MyServletname", urlPatterns = "/MyServlettest")
```

В этом случае описывать сервлет в файле web.xml не обязательно.
Нужно заметить, что дескриптор развертывания (web.xml) имеет приоритет над аннотациями. Т.е. он может переопределить конфигурацию, заданную механизмом аннотации.

# Описание сервлета в web.xml

<servlet> - блок, описывающий сервлеты

<display-name> - название сервлета

<description> - текстовое описание сервлета

<servlet-name> - имя сервлета

<servlet-class> - класс сервлета

<init-param> - блок, описывающий параметры инициализации сервлета

<param-name> - название параметра

<param-value> - значение параметра

# Описание сервлета в web.xml

<servlet-mapping> - блок, описывающий соответсвие url и запускаемого сервлета

<servlet-name> - имя сервлета

<url-pattern> - описывает url-шаблон

<session-config> - блок, описывающий параметры сессии

<session-timeout> - максимальное время жизни сессии

<login-config> - блок, описывающий параметры, как пользователь будет логиниться к серверу

<auth-method> - метод авторизации (BASIC, FORM, DIGEST, CLIENT-CERT)

# Web.xml

<welcome-file-list> - блок, описывающий имена файлов, которые будут пытаться открыться при запросе только по имени директории (без названия файла). Сервер будет искать первый существующий файл из списка и загрузит именно его

<welcome-file> - имя файла

<error-page> - блок, описывающий соответсвие ошибки и загружаемой при этом страницы

<error-code> - код произошедшей ошибки

<exception-type> - тип произошедшей ошибки

<location> - загружаемый файл

<taglib> - блок, описывающий соответсвие JSP Tag library descriptor с URI-шаблоном

<taglib-uri> - название uri-шаблона

<taglib-location> - расположение шаблона

# Web.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<web-app version="5.0"
        xmlns="https://jakarta.ee/xml/ns/jakartaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
        https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd">
<display-name>FirstWebProject</display-name>
   <servlet>
     <display-name>MyServletdisplay</display-name>
     <servlet-name>MyServletname</servlet-name>
     <servlet-class>by.bsu.servlet.MyServlet</servlet-class>
   </servlet>
```

# Web.xml (продолжение)

```
<servlet-mapping>
  <servlet-name>MyServletname</servlet-name>
  <url-pattern>/MyServlettest</url-pattern>
</servlet-mapping>
<session-config>
  <session-timeout>30</session-timeout>
</session-config>
<welcome-file-list>
  <welcome-file>index.html</welcome-file>
</welcome-file-list>
</web-app>
```

# Действия по вызову сервлета

После развертывания war архива на сервере нужно запустить браузер и ввести адрес:

**http://localhost:8080/DemoThymeleaf/MyServlettest**

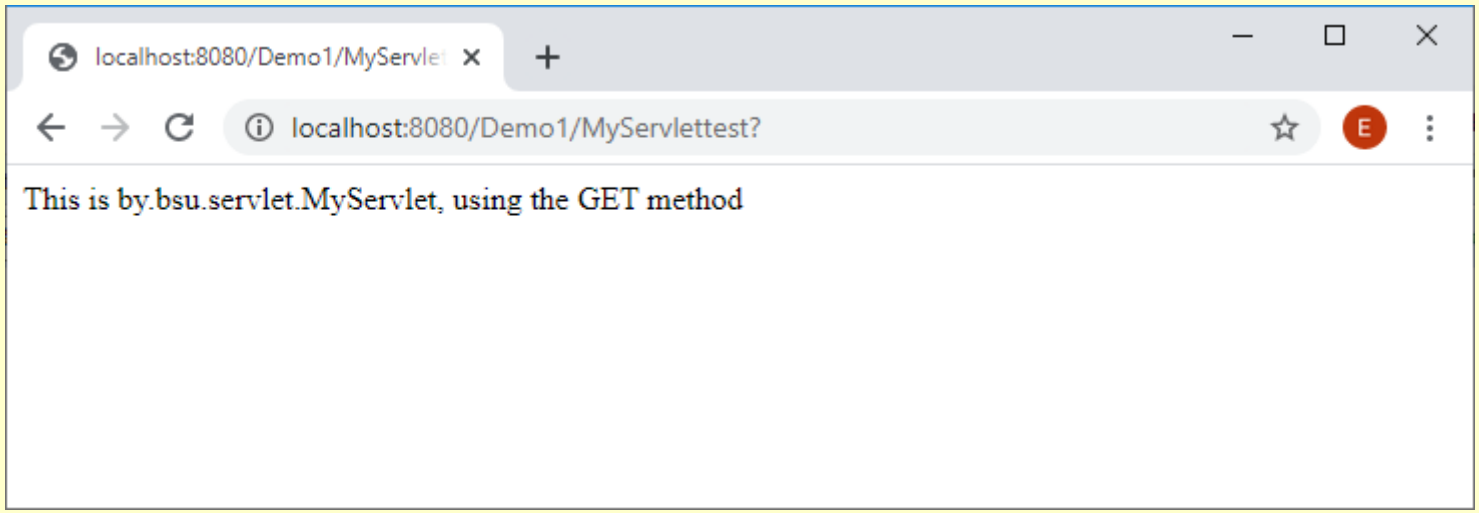
При обращении к сервлету из другого компьютера вместо **localhost** следует указать IP-адрес или имя компьютера.

Если вызывать сервлет из **index.html**, то тег **FORM** должен выглядеть следующим образом:

```
<FORM action="MyServlettest">
  <INPUT type="submit" value="Execute">
</FORM>
```

# Действия по вызову сервлета

Файл **index.html** помещается в папку **/webapps/DemoThymeleaf** и в браузере набирается строка:
**http://localhost:8080/Demo1/index.html**




Вывод сервлета после вызова метода doGet()

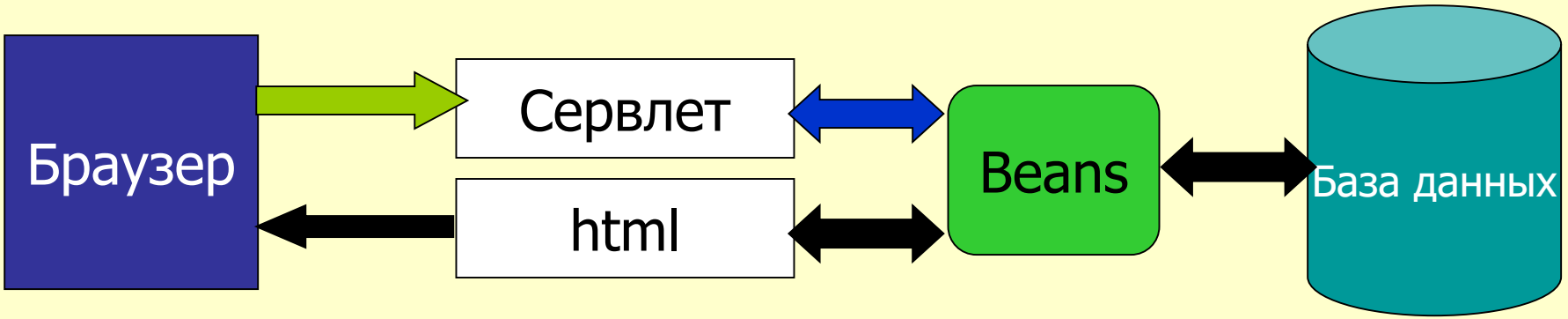
# Введение в ***Thymeleaf***

**Thymeleaf** (https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#introducing-thymeleaf) — это современный серверный механизм Java шаблонов для веб-приложений, способный обрабатывать HTML, XML, JavaScript, CSS и даже простой текст.

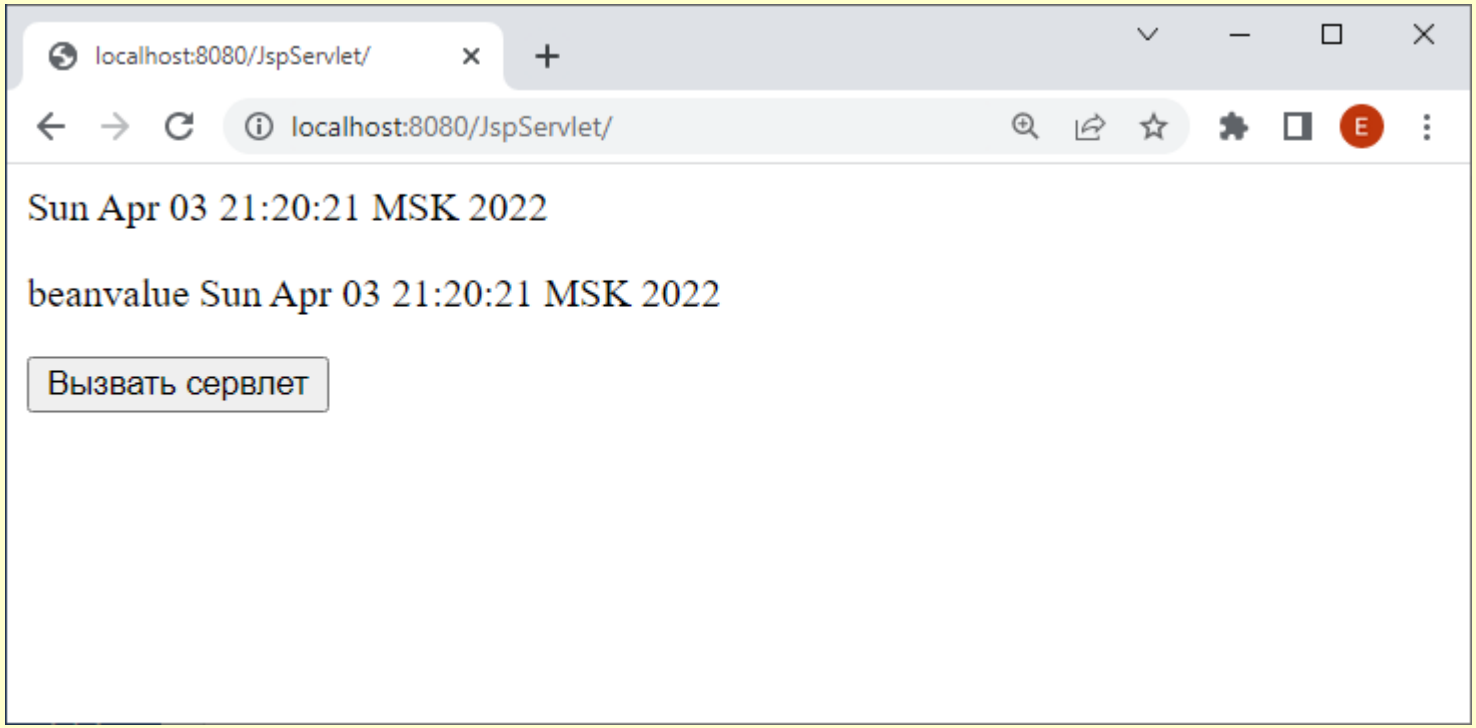
— обеспечивает разделение динамической и статической частей страницы, результатом чего является возможность изменения дизайна страницы, не затрагивая динамическое содержание.

# Взаимодействие сервлета и страницы html

Динамические страницы html ответственны за формирование пользовательского интерфейса и отображение информации, переданной с сервера. Сервлет выполняет роль контроллера запросов и ответов, то есть принимает http запросы от всех связанных с ним страниц, вызывает соответствующую бизнес-логику для их (запросов) обработки и в зависимости от результата выполнения решает, какую страницу поставить этому результату в соответствие.



# Динамическая страница с вызовом сервлета




Результат отображения index.html

# Передача информации между страницами Thymeleaf и сервлетом

Передачу информации между страницами Thymeleaf и сервлетом можно осуществлять, с помощью добавления атрибутов к объектам HttpServletRequest, HttpSession, ServletContext.

Интерфейсы HttpServletRequest, HttpSession, ServletContext имеют методы для работы с атрибутами:
setAttribute(String name, Object object);
getAttribute(String name);
removeAttribute(String name).

# Отслеживание сессии

Для поддержки статуса между сериями запросов от одного и того же пользователя используется механизм отслеживания сессии.
Сессия используется разными сервлетами для доступа к одному клиенту. Это удобно для приложений построенных на нескольких сервлетах.

Чтобы использовать отслеживание сессии:

- Создайте для пользователя сессию (объект HttpSession).
- Сохраняйте или читайте данные из объекта HttpSession.
- Уничтожьте сессию (необязательно).

# Получение сессии

HttpSession getSession(bool) объекта HttpServletRequest возвращает сессию пользователя. Когда метод вызывается со значением true, реализация при необходимости создает сессию. Значение false возвратит null, если сессия не обнаружена .

Чтобы правильно организовать сессию, Вам надо вызвать метод getSession прежде чем будет запущен выходной поток ответа.

```
public class SessionServlet extends HttpServlet {
   public void doGet (HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
      HttpSession session = request.getSession(true);
      out = response.getWriter();
      ...
    }
}
```

# Сохранение и получение данных сессии

Интерфейс HttpSession предоставляет методы, которые сохраняют и возвращают данные:

- Стандартные свойства сессии, такие как идентификатор сессии.
- Данные приложения, которые сохраняются в виде пары с именным ключом, когда имя это строка (String) и величина - объект Java.

Для доступа к атрибутам сессии используйте методы getAttribute(String attr) и setAttribute(String attr, Object value).

# Завершение сессии

Сессия пользователя может быть завершена вручную или, в зависимости от того, где запущен сервлет, автоматически. (Например, Java Web Server автоматически завершает сессию, когда в течение определенного времени не происходит запросов, по умолчанию 30 минут.) Завершение сессию означает удаление объекта HttpSession и его величин из системы.

Чтобы вручную завершить сессию, используйте метод сессии invalidate().

# Управление всеми браузерами

По умолчанию, прослеживание сессии использует закладки (cookie), чтобы ассоциировать идентификатор сессии с пользователем. Чтобы также поддерживать пользователей, у которых браузер не работает с закладками, или включен в режим игнорирования их, зачастую необходимо использовать перезапись URL. При данном механизме к ссылке добавляется многобуквенный параметр, который однозначно идентифицирует текущую сессию.

```
String encodeURL = response.encodeURL("index.html" );
```

# Использование Cookie

- Закладки (cookies) используются для хранения части информации на машине клиента.
- Закладки передаются клиенту в коде ответа в HTTP.
- Клиенты автоматически возвращают закладки, добавляя код в запросы в HTTP заголовках.
- Каждый заголовок HTTP запроса и ответа именован и имеет единственное значение.
- Множественные закладки могут иметь одно и тоже имя.

# Использование Cookie

Чтобы отправить закладку нужно:

1. создать объект Cookie,
2. установить любые атрибуты:

```
Cookie getBook = new Cookie("Buy", bookId);
```

3. отправить закладку:

```
response.addCookie(getBook);
```

Чтобы извлечь информацию из закладки:

1. запросить все закладки из пользовательского запроса,
2. найти закладку или закладки с интересующим именем, используя стандартные программные операции,
3. получить значения закладок, которые были найдены

# Создание Cookie

Конструктор класса jakarta.servlet.http.Cookie создает закладу с начальным именем и значением. Вы можете изменить значение закладки позже, вызвав метод setValue.

Если сервлет возвращает ответ пользователю, используя Writer, создавайте закладку, прежде чем обратитесь к Writer.

```
public void doGet (HttpServletRequest request,
HttpServletResponse  response)
 throws ServletException, IOException {
    response.setContentType("text/html");
    String bookId = request.getParameter("Buy");
   if (bookId != null) {
      Cookie getBook = new Cookie("Buy", bookId);
      response.addCookie(getBook);
   }
   PrintWriter out = response.getWriter();
   out.println("<html>" + "<head><title> Book Catalog
    </title></head>" + ...);
   ...
}
```

# Установка атрибутов закладки

Можно установить максимальный возраст закладки. Этот атрибут полезен, например, для удаления закладки.

```
public void doGet (HttpServletRequest request,
HttpServletResponse  response)
 throws ServletException, IOException {
   String bookId = request.getParameter("Remove");
   if (bookId != null) {
      Cookie getBook = new Cookie("Buy", bookId);
      getBook.setMaxAge(0);
      response.addCookie(getBook);

   }
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   out.println("<html> <head>" + "<title>Your Shopping
    Cart</title>" + ...);
   ...
}
```

# Отправка закладки

Закладки отправляются как заголовки ответа клиенту; они добавляются с помощью метода addCookie класса HttpServletResponse.

```
public void doGet (HttpServletRequest request,
HttpServletResponse
 response)
 throws ServletException, IOException {
   if (values != null) {
      bookId = values[0];
      Cookie getBook = new Cookie("Buy", bookId);
      getBook.setComment("User has indicated a desire " +
       "to buy this book from the bookstore.");
      response.addCookie(getBook);
   }
   ...
}
```

# Запрашивание закладок

Клиенты возвращают закладки как поля, добавленные в HTTP заголовок запроса.

Cookie[] getCookies() из класса HttpServletRequest – возвращает все закладки ассоциированные к данному хосту.

```
public void doGet (HttpServletRequest request,
HttpServletResponse
 response)
 throws ServletException, IOException {
   String bookId = request.getParameter("Remove");
   if (bookId != null) {
      Cookie[] cookies = request.getCookies();

   }
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   out.println("<html> <head>" + "<title>Your Shopping
Cart</title>" + ...);
   ...
}
```

# Получение значения закладки

String getValue() - возвращает значение закладки.

```
public void doGet (HttpServletRequest request,
HttpServletResponse
 response)
 throws ServletException, IOException {
   String bookId = request.getParameter("Remove");
   if (bookId != null) {
      Cookie[] cookies = request.getCookies();

      for(i=0; i<cookies[i].getValue().equals(bookId); i++)
{
            . . .
      }
   }
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   out.println("<html> <head>" + "<title>Your Shopping
Cart</title>" + ...);
   ...
}
```

# Совместное использование ресурсов сервлетами

Интерфейс ServletContext используется для взаимодействия с контейнером сервлетов. Сервлеты исполняемые на одном сервере могут совместно использовать ресурсы с помощью методов интерфейса ServletContext для манипулирования атрибутами:

• void setAttribute(String name, Object object) – добавляет атрибут и его значение в контекст; обычно это производится во время инициализации. Когда несколько сервлетов используют атрибут, каждый должен проинициализировать этот атрибут. А раз так, каждый сервлет должен проверить значение атрибута, и устанавливать его только в том случае если предыдущий сервлет не сделал этого.

• Object getAttribute(String name) – возвращает совместный ресурс.
• Enumeration getAttributeNames() – получает список имен атрибутов;

• void removeAttribute(String name) – удаляет совместный ресурс.

# Интерфейс ServletContext

- ServletContext getContext(String uripath) – позволяет получить доступ к контексту других ресурсов данного контейнера сервлетов;
- String getServletContextName() – возвращает имя сервлета, которому принадлежит данный объект интерфейса ServletContext.
- String getCharacterEncoding() – определение символьной кодировки запроса.

# Интерфейс ServletConfig

Представляет собой конфигурацию сервлета, используется в основном на этапе инициализации. Все параметры для инициализации устанавливаются в web.xml

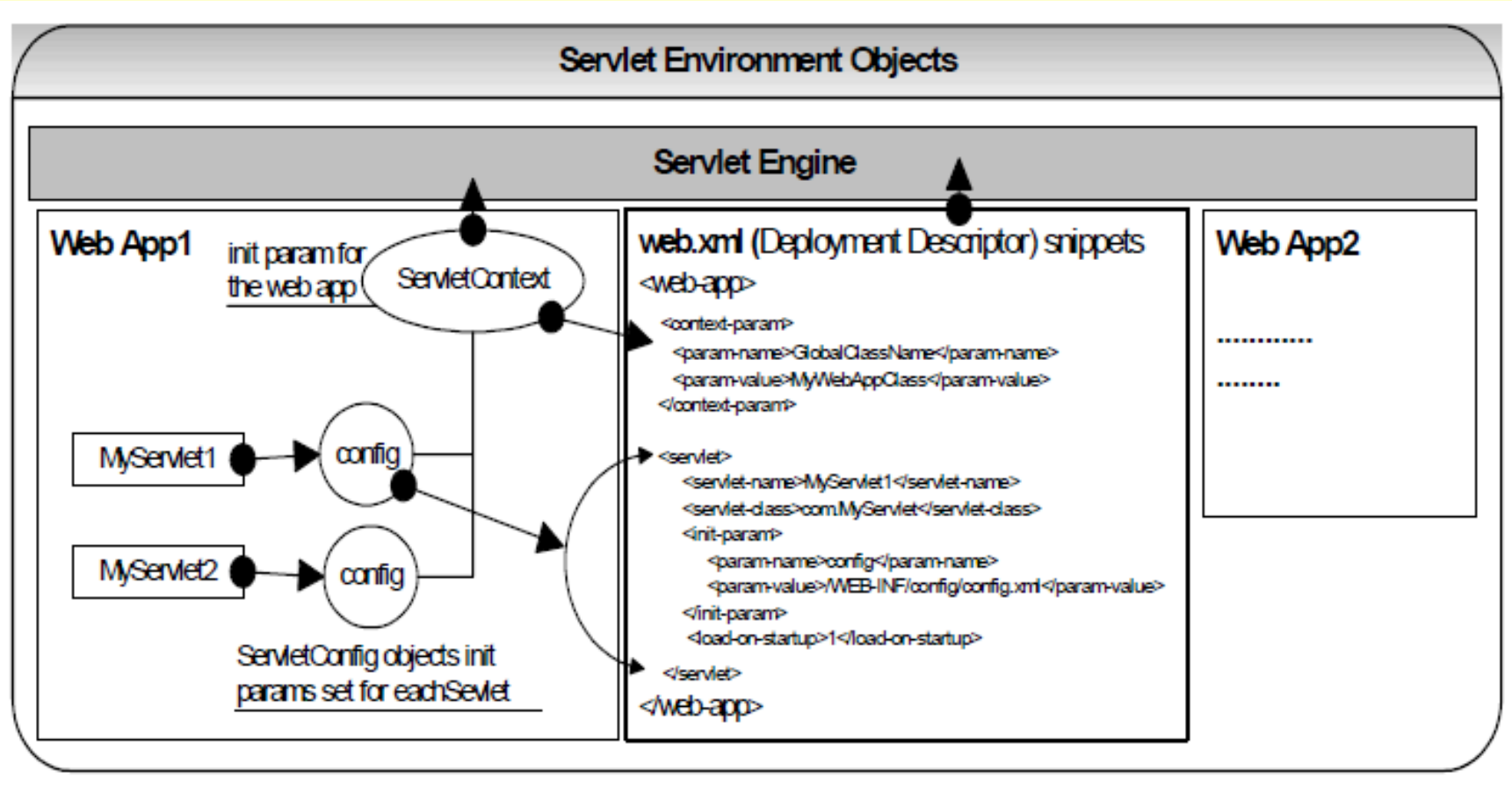
Некоторые методы класса:

- String getServletName() – определение имени сервлета;

- Enumeration getInitParameterNames() – определение имен параметров
•инициализации сервлета из дескрипторного файла web.xml;

- String getInitParameter(String name) – определение значения конкретного параметра по его имени.

# ServletConfig vs ServletContext



# Информационные ресурсы
# Организационные вопросы