

Урок №6

Разработка Web-приложений с использованием технологии Java

Содержание

Введение	2
1. Основные архитектурные концепции современных приложений	4
2. Технология Java Servlets	9
2.1. Жизненный цикл сервлета	10
2.2. Web-сервер Apache Tomcat	12
2.3. Разработка простейшего сервлета	13
2.4. Обзор API сервлетов	16
2.4.1. Пакет javax.servlet	16
2.4.2. Пакет javax.servlet.http	17
2.5. Разработка сервлета с использованием среды NetBeans	18
2.6. Получение данных с запроса	27
2.7. Построение ответа	29
2.8. URL запроса	30
2.9. Включение ресурса в сервлет	32
2.10. Переадресация запроса другому Web-компоненту	33
2.11. Работа с сессиями	33
2.12. Работа с cookies	35
2.13. Пример разработки web-приложения с использованием сервлетов	35
2.13.1. Идея разработки	36
2.13.2. Постановка требований	36
2.13.3. Разработка доменной модели	37
2.13.4. Создание базовой инфраструктуры проекта	38
2.13.5. Разработка бизнес-логики и графического интерфейса приложения	39
Домашнее задание	53
Задача 1	53
Использованные информационные источники	54

Введение

Значительную долю современного рынка программного обеспечения занимают Web-приложения. Web-интерфейсом обладают как простейшие сайты, так и клиентские части больших корпоративных систем. Даже офисные приложения, которые традиционно были настольными, сейчас имеют Web-аналоги, не очень уступающие в функциональности своим прародителям (например, Google Docs). Широкому распространению Web-приложений способствует, в первую очередь, их мобильность и доступность: для работы с ними нужен только интернет-браузер (а не установка целого программного комплекса как в случае настольных приложений).

Студенту, как будущему специалисту из разработки программного обеспечения, вероятнее всего придется заниматься разработкой именно Web-ориентированных приложений, а для этого ему необходимы прочные знания того, как работают такие типы приложений в своей основе и какие подходы применяются для их разработки в разных технологиях программирования.

В этом уроке изложены основы разработки Web-приложений с использованием технологии Java. Так как мы уже знакомы с такими мощными технологиями Web-программирования, как PHP и ASP.NET, с одной стороны будет легко понять подход, применяемый в Java, а с другой – полезно расширить свой профессиональный кругозор за счет спектра Web-технологий, предлагаемых Java.

С точки зрения Java, выделяют два основных вида Web-приложений: презентационно-ориентированные (Web-приложения с графическим интерфейсом) и сервисно-ориентированные (Web-сервисы) [JEE_Tutorial]. В этом уроке мы сосредоточим свое внимание на основах разработки презентационно-ориентированных Web-приложений.

Базовые технологии Web-программирования Java являются составляющими Java Enterprise Edition. На рис. 0.1 проиллюстрирован подход к построению Web-платформы Java.

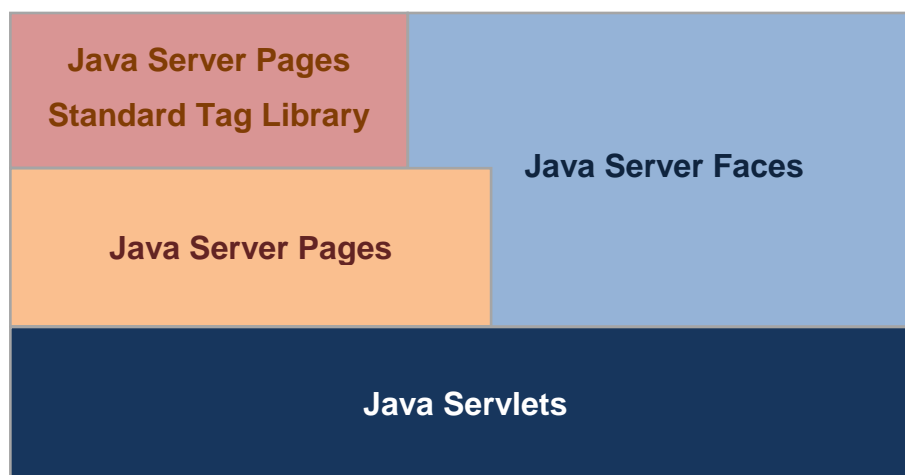


Рис. 0.1. Базовые технологии разработки Web-приложений на платформе Java

Фундамент этой платформы составляет технология Java Servlets (или просто «сервлеты»). Технология сервлетов предлагает простое и эффективное решение задачи серверной обработки клиентских запросов.

Для разработки графического интерфейса Web-приложений предназначена технология Java Server Pages (JSP). Она предполагает встраивание в статическую разметку HTML-страницы Java-кода, который выполняется Web-сервером. Этим JSP чем-то напоминает PHP, хотя её возможности значительно шире, так как она функционирует в рамках Java-платформы. Стандартная библиотека тегов (Java Server Pages Standard Tag Library) значительно расширяет базовые возможности JSP.

Технология Java Server Faces (JSF) представляет компонентно-ориентированный подход к Web-программированию. Она включает набор API для представления компонент пользовательского графического интерфейса и управления их состоянием, обработки событий, проверки вводимых пользователем данных. За принципом построения Web-интерфейсов JSF напоминает ASP.NET Web Forms.

В этом уроке мы, главным образом, сосредоточимся на основах разработки Web-приложений на платформе Java и детально рассмотрим фундаментальную технологию – Java Servlets.

1. Основные архитектурные концепции современных приложений

Перед тем, как приступить к изучению Web-программирования с использованием Java сделаем краткий обзор фундаментальных архитектурных подходов, которые применяются в построении современных приложений, а также представим классическую модель Web-приложения.

Простейшими, с точки зрения архитектуры, являются настольные приложения (desktop application). Такие приложения, как правило, состоят из одного или нескольких исполняемых модулей, которые размещаются на одной рабочей машине (рис.1.1). Типичными примерами таких приложений служат редактор MS Word, среда разработки NetBeans, FTP-клиент FileZilla и др. Главные преимущества настольных приложений:

- 1) высокое быстродействие и удобность пользовательского графического интерфейса;
- 2) простота установки, так как для подавляющего большинства таких приложений разработаны инсталляторы.

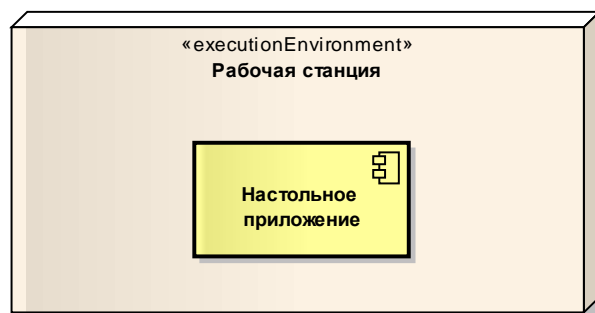


Рис. 1.1. Настольное приложение

Недостатки настольных приложений:

- 1) необходимость установки на клиентской машине, что несколько усложняет получение обновлений и часто делает программу зависимой от платформы;
- 2) затрудненные и плохо масштабируемые возможности построения систем, состоящих из нескольких взаимодействующих настольных приложе-

ний (на платформе Windows такие системы строятся на основе технологии COM/DCOM (рис. 1.2), но через технические сложности её использования имеют сравнительно ограниченные возможности).

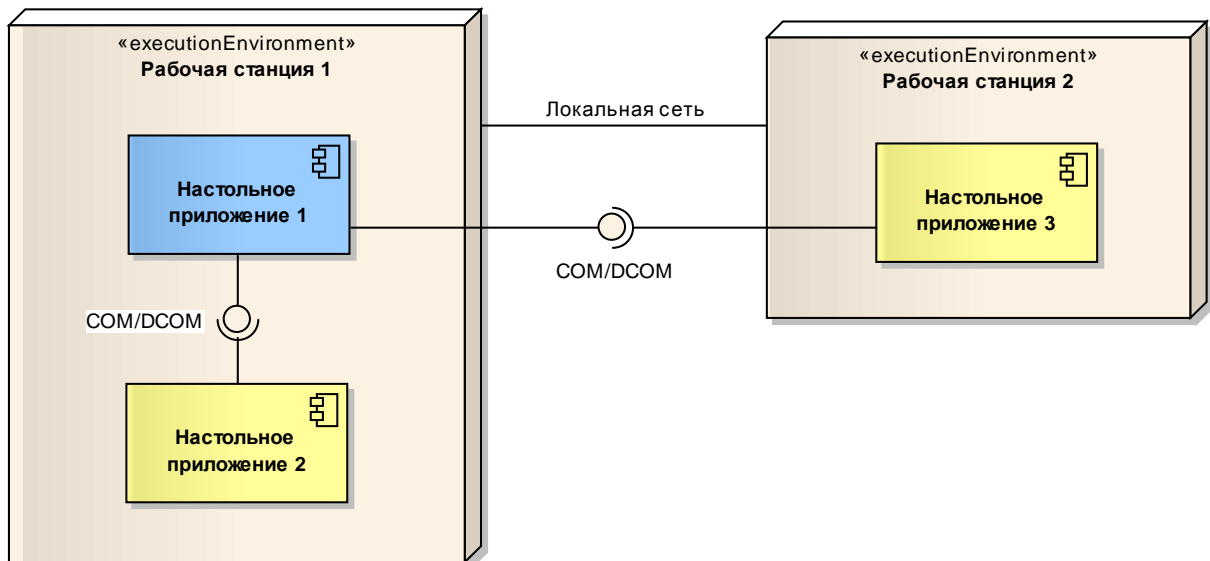


Рис. 1.2. Взаимодействие настольных приложений на основе технологии COM/DCOM

Но основным недостатком настольных приложений является невозможность построения на их основе сложных распределенных систем (например, системы автоматизации работы большого предприятия).

Следующим шагом в развитии программной архитектуры стала архитектура «клиент-сервер». Такой подход предполагает наличие двух звеньев: клиента, который посылает запросы серверу, и сервера, который обрабатывает запросы клиента (рис. 1.3).

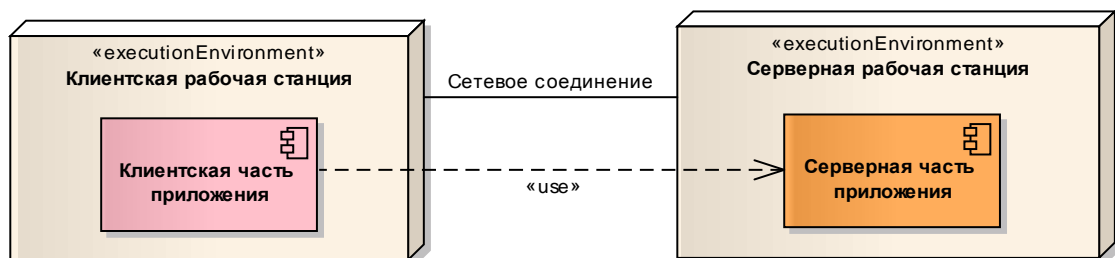


Рис. 1.3. Архитектура клиент-сервер

Часто в качестве сервера выступает сервер баз данных, а в качестве клиента – настольное приложение, которое общается с сервером через сеть посредством SQL-запросов (рис. 1.4).

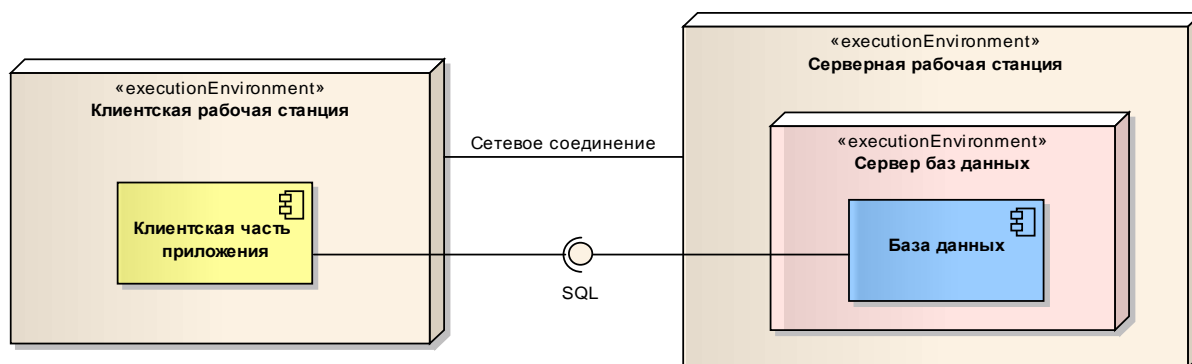


Рис. 1.4. Архитектура клиент-сервер с использованием сервера баз данных

Из такого построения системы следует, что значительная часть логики приложения реализуется на клиентской части (так называемый, «толстый» клиент) и на сервере баз данных (в виде хранимых процедур, функций, триггеров). Такая архитектура получила название двухзвенная (или двухуровневая). Двухзвенная архитектура предоставляет значительно больше гибкости, чем обычные настольные приложения. На её основе можно строить достаточно большие распределенные системы. Недостатками двухзвенной архитектуры, в частности, являются:

- 1) чрезмерно «тяжелый» клиент, так как он реализует значительную часть логики приложения;
- 2) зависимость клиентской части от платформы;
- 3) реализация на сервере баз данных несвойственной для него логики, например, разного рода финансовых вычислений;
- 4) необходимость прямого взаимодействия клиента и сервера баз данных, что может вызвать проблемы производительности при наличии большего числа клиентских подключений, а также создает большую опасность несанкционированного доступа к данным.

Следующим шагом в развитии архитектуры распределенных систем стала многозвенная (многоуровневая) архитектура. В простейшем случае приложение является трехзвенным, т. е. состоит из трех компонент, например: клиента, сервера приложений и сервера баз данных (рис. 1.5).

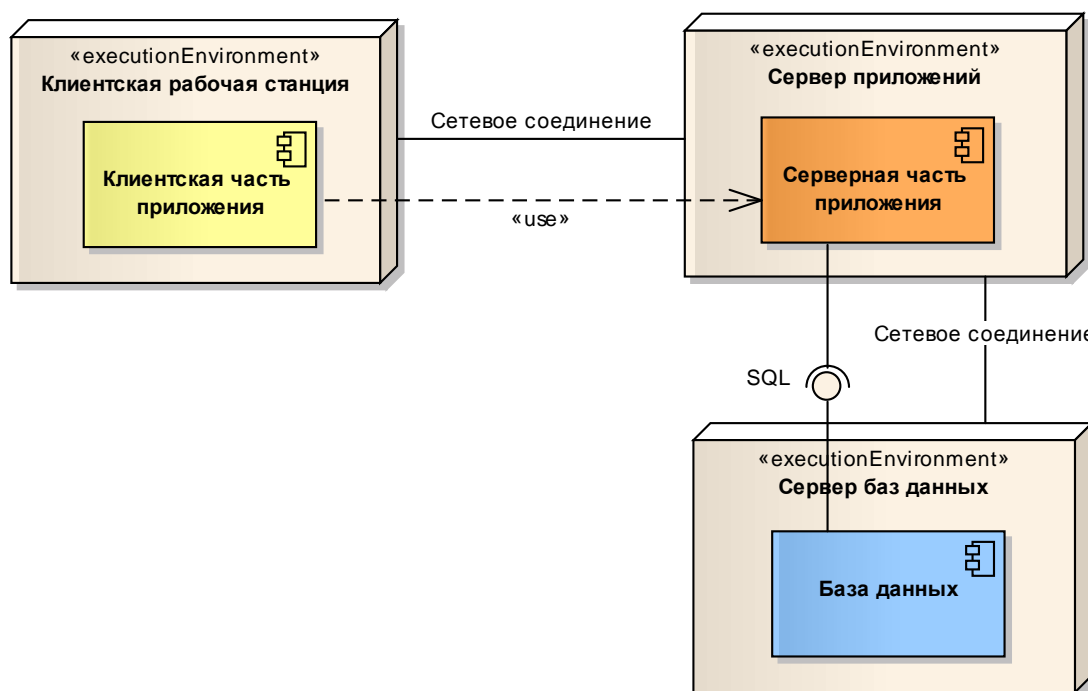


Рис. 1.5. Трехзвенная архитектура

Главное преимущество такого подхода заключается в четком распределении обязанностей. Основная логика приложения реализуется сервером приложений. Клиент, в таком случае, является тонким и взаимодействует только с сервером приложений. Последний, в свою очередь, обрабатывает запросы клиента и работает с сервером баз данных. На базу данных при этом возлагается хранение данных и, возможно, некоторая логика их обработки. Очевидно, что среднее звено (сервер приложений) может быть разбито на множество подуровней (отсюда и название «многоуровневая архитектура»).

По принципу многоуровневой архитектуры работают приложения и с Web-интерфейсом (рис. 1.6). Для случая трех звеньев в качестве клиентской части выступает приложение, которое загружается пользователем в Web-браузере, в качестве сервера приложений выступает Web-сервер, на котором работает серверная часть приложения.

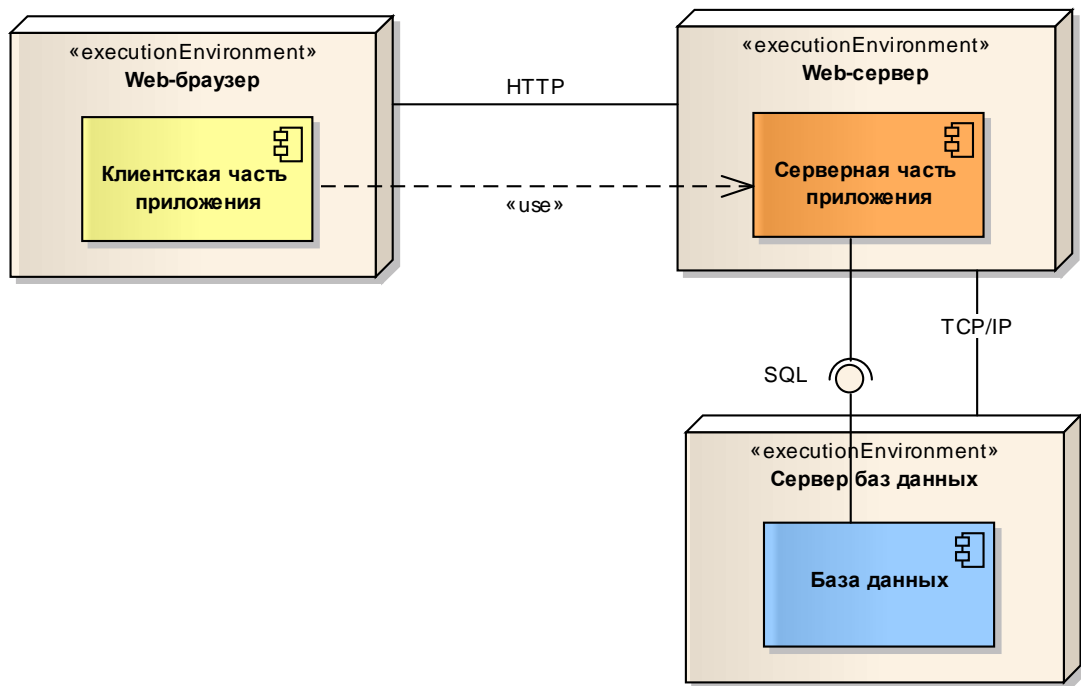


Рис. 1.6. Web-приложение с трехуровневой архитектурой

Нам раньше приходилось разрабатывать приложения такого типа при изучении дисциплин PHP и ASP.NET. В этом уроке мы рассмотрим основы разработки Web-приложений с использованием технологии Java.

2. Технология Java Servlets

Сервлет – специальный Java-класс, использующийся для расширения возможностей сервера, на котором разворачиваются приложения, работающие за принципом «запрос-ответ» [JSP_Tutorial]. Иными словами, сервлеты представляют собой серверные обработчики запросов клиента. Зачастую, сервлеты используются для обработки HTTP-запросов. В таком случае, сервлет представляет собой компонент, расширяющий возможности Web-сервера. Он работает под управлением так называемого сервлет-контейнера, или web-контейнера, который, в свою очередь, является составной частью Web-сервера или сервера приложений.

Классы и интерфейсы сервлетов содержатся в пакетах `javax.servlet` и `javax.servlet.http`. Все сервлеты должны реализовать интерфейс `javax.servlet.Servlet`, который представляет методы для управления жизненным циклом сервлета. Зачастую, сервлеты общего назначения наследуются от класса `javax.servlet.GenericServlet`, который представляет реализацию по умолчанию методов жизненного цикла сервлета. При разработке сервлетов, предназначенных для обработки HTTP-запросов, расширяют класс `javax.servlet.http.HttpServlet`, который предлагает методы `doGet()` и `doPost()`, реагирующие на одноименные HTTP-запросы.

Первая спецификация стандарта сервлетов была выпущена компанией Sun Microsystems в июне 1997 года. После этого стандарт постоянно развивался, и на момент написания данного урока последней является версия 3.0, выпущенная в декабре 2009 года. Эта версия поддерживается платформами JSE 6 и JEE 6.

Одной из самых универсальных и старых технологий обработки запросов на стороне Web-сервера, есть технология CGI (Common Gateway Interface). Программы, вызываемые на стороне сервера, зачастую писались с использованием таких языков как Perl или C/C++. Но эта технология имела целый ряд недостатков, в частности: необходимость запуска отдельного про-

цесса на сервере для каждого клиентского запроса (что даже при относительно небольшом количестве клиентских подключений отрицательно сказывалось на производительности), платформенная зависимость, плохая масштабируемость приложений. Сервлеты, не обладая основными недостатками CGI, стали той технологией, которая пришла на смену CGI.

Сервлеты составляют фундамент Web-программирования на Java. Все клиентские запросы к Web-серверу явным или неявным для нас образом обрабатываются сервлетами: будь то запрос к JSP-странице или вызов удаленной процедуры в GWT-приложении. Поэтому понимание механизма работы сервлетов и знание Servlet API является важным для успешного изучения и последующего эффективного использования Web-программирования на платформе Java.

2.1. Жизненный цикл сервлета

Каждый сервлет реализует интерфейс `javax.servlet.Servlet`, который определяют три основных метода жизненного цикла: `init()`, `service()` и `destroy()`.

Как было сказано выше, сервлет работает под управлением сервлет-контейнера. Если сервлет вызывается некоторым запросом, сервлет-контейнер выполняет следующие действия:

1. Если экземпляр сервлета не создан, то
 - a. Загружает класс сервлета.
 - b. Создает экземпляр класса сервлета.
 - c. Инициализирует сервлет путем вызова метода `init()`.
2. Вызывает метод `service()`, передав ему объекты `request` (запрос) и `response` (ответ).

Жизненный цикл сервлета схематически изображен на рис. 2.1.



Рис. 2.1. Жизненный цикл сервлета

Сервлет-контейнер при создании экземпляра класса сервлета вызывает его метод `init()`, в котором осуществляется инициализация экземпляра. Если создание экземпляра сервлета с каких-либо причин невозможно, в методе `init()` должно генерироваться исключение `UnavailableException()`. Обратим особое внимание на то, что *создание экземпляра сервлета и, соответственно, вызов его метода `init()` происходит не при каждом запросе сервлета, а лишь однажды, после чего созданный объект сервлета может обслуживать множество запросов клиента.*

При клиентском вызове сервлета сервлет-контейнер вызывает его метод `service()`, который и отвечает за обработку запроса. Этот метод принимает два параметра: `ServletRequest`, содержащий информацию о запросе, и `ServletResponse`, представляющий сгенерированный сервлетом ответ.

При уничтожении объекта сервлета сервлет-контейнер вызывает его метод `destroy()`, в котором освобождаются связанные с сервлетом ресурсы.

При уничтожении сервлета все вызовы метода `service()` должны быть завершены. В общем случае указать точное время уничтожения экземпляра сервлета невозможно (это может произойти при необходимости освобождения памяти или при завершении работы Web-сервера).

2.2. Web-сервер Apache Tomcat

Одним из наиболее популярных Web-серверов, поддерживающих сервлет-контейнер, является Apache Tomcat (официальная страница - <http://tomcat.apache.org/index.html>). Это Open Source продукт компании Apache Software Foundation, предназначенный, главным образом, для разработки развертывания Java-приложений, реализованных на основе технологий сервлетов и Java Server Pages.

На момент написания этого урока последней версией Apache Tomcat была 7 (официальная страница <http://tomcat.apache.org/download-70.cgi>). Эта версия поддерживает Servlets 3.0 и JSP 2. Часть примеров этого урока мы будем демонстрировать с использованием Apache Tomcat 7. Поэтому рекомендуется получить этот продукт с официального сайта и установить.

Для начала, отметим два важные момента, необходимые для использования Apache Tomcat:

1. Библиотека Servlets API не является составной частью JDK – она поставляется вместе с сервлет-контейнером. В случае Apache Tomcat эта библиотека носит название `servlet-api.jar` и лежит в папке `lib` его корневого каталога.

2. Для разворачивания Web-приложений на Apache Tomcat необходимо папку с Web-приложением или war-файл (web archive – jar-архив с web-приложением) разместить в папке `webapps` корневого каталога.

2.3. Разработка простейшего сервлета

Для начала, разработаем простейший сервлет (назовем его «Hello Servlet») и разместим его на Web-сервере. Для написания кода сервлета не будем пользоваться специализированными инструментами – ограничимся обычным текстовым редактором (например, Notepad++) и компилятором `javac` из стандартного состава JDK.

Для реализации поставленной задачи выполним следующие шаги:

1. Разработка исходного кода.
2. Компиляция.
3. Разработка дескриптора развертывания.
4. Размещение на Web-сервере.
5. Тестирование.

Шаг 1. Разработка исходного кода

Класс сервлета `HelloServlet` наследуем от класса `javax.servlet.GenericServlet`, который, в свою очередь, реализует интерфейс `javax.servlet.Servlet`. В классе `HelloServlet` переопределим метод `service()`, отвечающий за обработку запросов клиента. Обратим внимание, что метод `service()` принимает два параметра: `request` – запрос клиента, и `response` – ответ на запрос. Исходный код класса `HelloServlet`:

```
import java.io.*;
import javax.servlet.*;

public class HelloServlet extends GenericServlet {

    @Override
    public void service(ServletRequest request,
        ServletResponse response) throws ServletException, IOException {

        // определяем тип содержимого ответа
        response.setContentType("text/html");
        // создаем поток ответа
        PrintWriter writer = response.getWriter();
        try {
            // записываем текст в поток ответа
            writer.println("<b>Hello from servlet</b>");
        } finally {
            // закрываем поток ответа
            writer.close();
        }
    }
}
```

```
}  
}  
}
```

Шаг 2. Компиляция

Для компиляции сервлета воспользуемся стандартным Java-компилятором из JDK. В командной строке следует указать команду:

```
javac HelloServlet.java -classpath "C:\Program Files\Apache Software  
Foundation\Tomcat 7.0.4\common\lib\servlet-api.jar"
```

Обратим внимание на то, что для успешной компиляции сервлета необходимо указать в параметре `classpath` ссылку на библиотеку `servlet-api.jar` (которая, как мы помним, поставляется вместе с Apache Tomcat).

В случае успешной компиляции сервлета мы получим файл с байт-кодом `HelloServlet.class`.

Шаг 3. Разработка дескриптора развертывания

Для описания того, как компоненты web-приложения должны работать на web-сервере, предназначен, так называемый, дескриптор развертывания, который представляется файлом `web.xml`:

```
<?xml version="1.0" ?>  
  
<web-app>  
  <!-- определение сервлета -->  
  <servlet>  
    <!-- имя, или идентификатор, сервлета -->  
    <servlet-name>HelloServlet</servlet-name>  
    <!-- полное имя класса сервлета -->  
    <servlet-class>HelloServlet</servlet-class>  
  </servlet>  
  
  <!-- связывание сервлета с url запроса -->  
  <servlet-mapping>  
    <!-- имя сервлета -->  
    <servlet-name>HelloServlet</servlet-name>  
    <!-- url запроса, на который вызывается сервлет -->  
    <url-pattern>/Hello</url-pattern>  
  </servlet-mapping>  
  
</web-app>
```

В нашем случае в дескрипторе определяется URL запроса, посредством которого клиент сможет получать доступ к сервлету.

Шаг 4. Размещение на Web-сервере

Создадим следующую файловую структуру:



Папку `HelloServlet` копируем в папку `webapps` Web-сервера Apache Tomcat. Возможен также вариант развертывания приложения в виде `war`-файла (web archive). Для этого надо содержание папки `HelloServlet` заархивировать утилитой `jar` (стандартный архиватор из JDK). Для этого, например, можно воспользоваться командой (текущей должна быть папка `HelloServlet`):

```
jar cf HelloServlet.war *
```

Полученный таким образом файл `HelloServlet.war` следует разместить непосредственно в папке `webapps` Web-сервера Apache Tomcat.

Шаг 5. Тестирование

Прежде всего, убедимся в том, что запущен Web-сервер Apache Tomcat. Для это воспользуемся URL <http://localhost:8080/> (предполагается, что сервер развернут на локальной рабочей станции и ним прослушивается порт 8080) – в случае успеха в окне браузера должна отобразиться страница приветствия Apache Tomcat.

Если Web-сервер работает, следует указать в адресной строке браузера URL разработанного выше сервлета – <http://localhost:8080/HelloServlet/Hello>. В случае успеха - получим следующий результат:



Следует также отметить что, имеет значение регистр символов: например, при указании URL <http://localhost:8080/HelloServlet/hello> Web-сервер выдаст ошибку 404 (что свидетельствует о недоступности указанного ресурса).

Студенту рекомендуется повторить описанный выше пример. Исходные коды примера представлены в папке `Samples/01_HelloServlet`.

2.4. Обзор API сервлетов

2.4.1. Пакет `javax.servlet`

В пакете `javax.servlet` представлены классы и интерфейсы, составляющие фундамент технологии сервлетов. Они применимы для разработки сервлетов общего назначения, т.е. не обязательно тех, которые обрабатывают только HTTP-запросы. Наиболее важными элементами этого пакета являются интерфейс `Servlet`, представляющий общий контракт для всех без исключения сервлетов, и класс `GenericServlet`, который реализует интерфейс `Servlet` и определяет все его методы, кроме метода `service()`, который непосредственно отвечает за обработку запроса клиента.

В табл. 2.1-2.3 представлено схематическое описание наиболее важных элементов этого пакета.

Табл. 2.1. Интерфейсы пакета `javax.servlet`

№	Название интерфейса	Описание
1.	<code>RequestDispatcher</code>	Определяет объект, который принимает запрос клиента и пересылает его другому web-элементу или сервлет-контейнеру.
2.	<code>Servlet</code>	Общий интерфейс сервлета: определяет методы, которые должен реализовывать каждый сервлет.
3.	<code>ServletConfig</code>	Представляет объект, использующийся сервлет-контейнером для передачи параметров при инициализации сервлета. Параметры инициализации сервлета могут быть указаны в дескрипторе развертывания.
4.	<code>ServletContext</code>	Определяет набор методов, посредством которых сервлет взаимодействует с сервлет-контейнером.
5.	<code>ServletRequest</code>	Определяет объект, представляющий информацию о клиентском запросе.
6.	<code>ServletResponse</code>	Определяет объект, представляющий ответ клиенту, который формируется сервлетом.

Табл. 2.2. Классы пакета javax.servlet

№	Название класса	Описание
1.	GenericServlet	Абстрактный базовый класс сервлета общего назначения. Реализует интерфейсы Servlet и ServletConfig. Классы сервлетов удобно наследовать от него, переопределив только его абстрактный метод service().
2.	ServletInputStream	Класс потока для считывания бинарных данных с запроса клиента.
3.	ServletOutputStream	Класс потока для записи бинарных данных в ответ клиенту.

Табл. 2.3. Исключения пакета javax.servlet

№	Название класса-исключения	Описание
1.	ServletException	Общее исключение сервлета.
2.	UnavailableException	Подкласс класса ServletException. Свидетельствует о том, что сервлет временно или постоянно недоступен. Генерируется методом init(), когда создание экземпляра сервлета невозможно, например, через неверные параметры инициализации.

Более детальную информацию о составе пакета javax.servlet можно получить из официальной документации

<http://download.oracle.com/javaee/6/api/javax/servlet/package-summary.html>.

2.4.2. Пакет javax.servlet.http

В пакете javax.servlet.http представлены средства разработки сервлетов, которые специализируются на обработке HTTP-запросов, т.е. используются для разработки Web-приложений. Одним из основных элементов этого пакета является класс javax.servlet.http.HttpServlet, который является подклассом класса javax.servlet.GenericServlet и представляет разработчику удобные методы для обработки Http-запросов (doGet(), doPost() и др.). Наиболее важные элементы этого пакета описаны в табл. 2.4 и 2.5.

Табл. 2.4. Интерфейсы пакета javax.servlet.http

№	Название интерфейса	Описание
1.	HttpServletRequest	Расширяет интерфейс ServletRequest методами, необходимыми для работы с HTTP-запросом.
2.	HttpServletResponse	Расширяет интерфейс ServletResponse методами, необходимыми для формирования HTTP-отклика.
3.	HttpSession	Представляет объект для взаимодействия с HTTP-сессией.

Табл. 2.5. Классы пакета javax.servlet.http

№	Название класса	Описание
1.	Cookie	Предоставляет возможности для манипуляции Cookies
2.	HttpServlet	Базовый класс сервлетов, предназначенных для обработки HTTP-запросов. Расширяет класс javax.servlet.http.GenericServlet и предоставляет удобные методы для обработки HTTP-запросов.

Более детальную информацию о составе пакета javax.servlet.http можно получить из официальной документации

<http://download.oracle.com/javase/6/api/javax/servlet/http/package-summary.html>

Так как наша основная цель – изучение технологий Web-программирования Java, то далее в уроке мы будем, в основном, пользоваться средствами пакета javax.servlet.http.


2.5. Разработка сервлета с использованием среды NetBeans

Теперь, когда мы рассмотрели основные классы, на которых держится технология сервлетов, снова разработаем Hello-приложение, но с большими возможностями и с использованием интегрированной среды разработки NetBeans. Назовем разрабатываемое приложение ServletHelloEx.

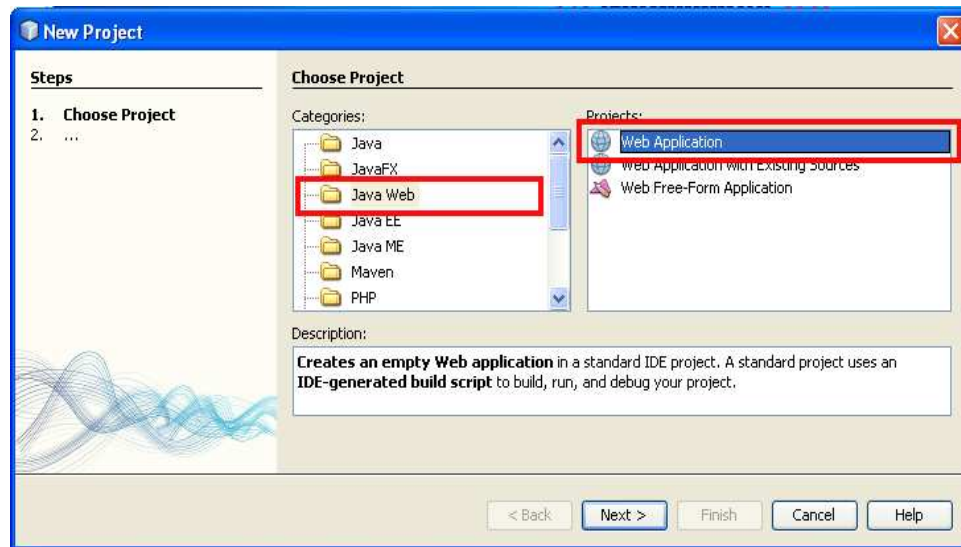
Разработка приложения будет состоять из следующих шагов:

1. Создание Web-проекта в NetBeans.
2. Создание класса сервлета.
3. Реализация обработки запроса GET.
4. Тестирование обработки запроса GET.
5. Реализация обработки запроса POST
6. Тестирование обработки запроса POST.

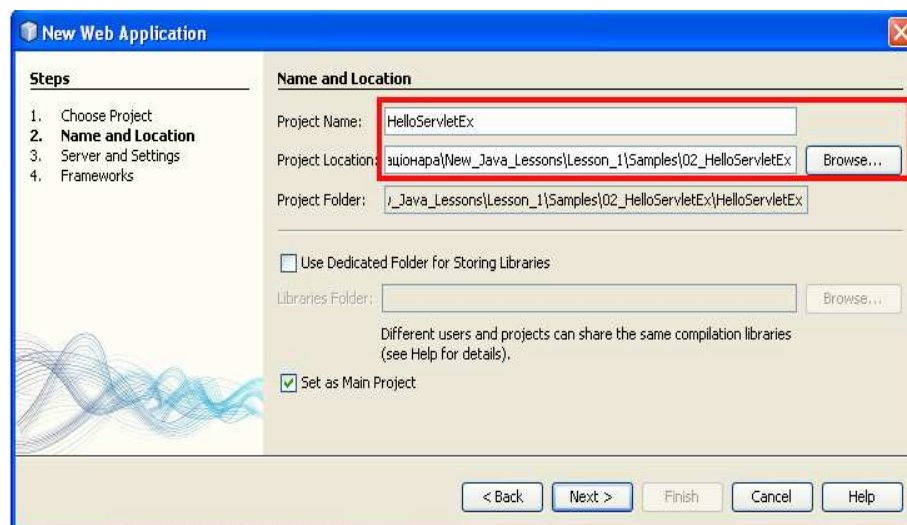
Шаг 1. Создание Web-проекта в NetBeans

Для создания Web-проекта в NetBeans следует воспользоваться командой главного меню File \ New Project (или опцией на панели инструментов – ). После этого пользователю представляется мастер New Project.

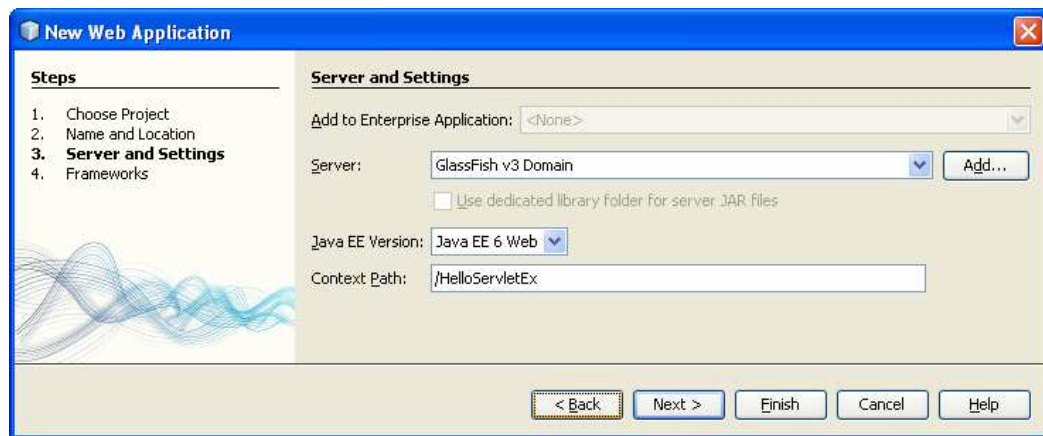
На первом шаге работы мастера следует указать тип проекта. В нашем случае следует выбрать категорию `Java Web` и проект `Web Application`:



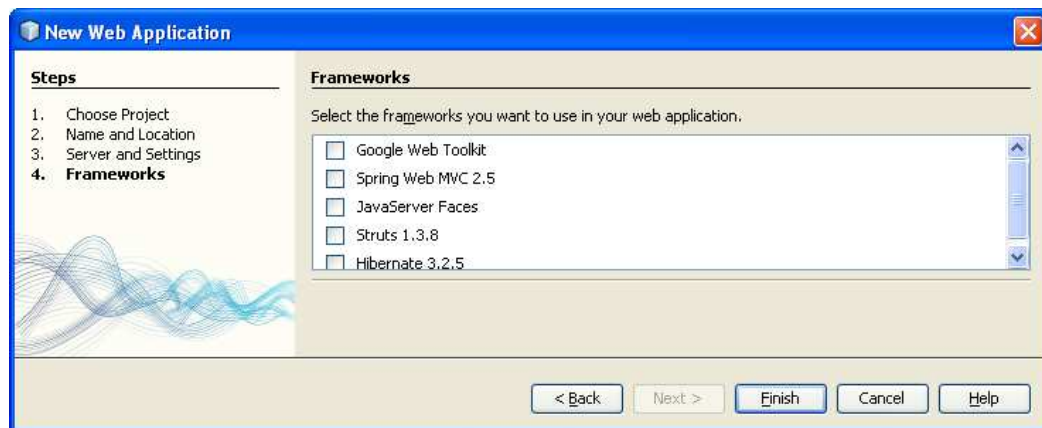
На следующем шаге указываем название проекта и папку, где он будет размещен:



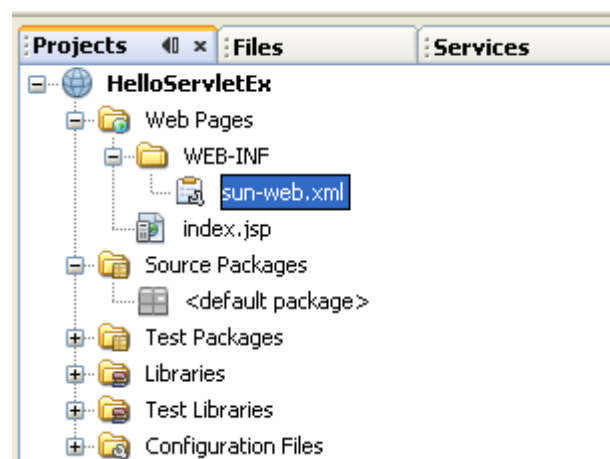
Далее следует сделать настройки сервера приложений. Мы здесь оставим значение все полей по умолчанию. Но обратим внимание, что в качестве Web-сервера (или, точнее, сервера приложений) для этого проекта буде использоваться Glass Fish – встроенный сервер приложений среды Net Beans. Также следует обратить внимание на значение параметра `Context Path`: он будет использоваться в URL для доступа к приложению:



На шаге 4 мы не будем выбирать дополнительных библиотек:




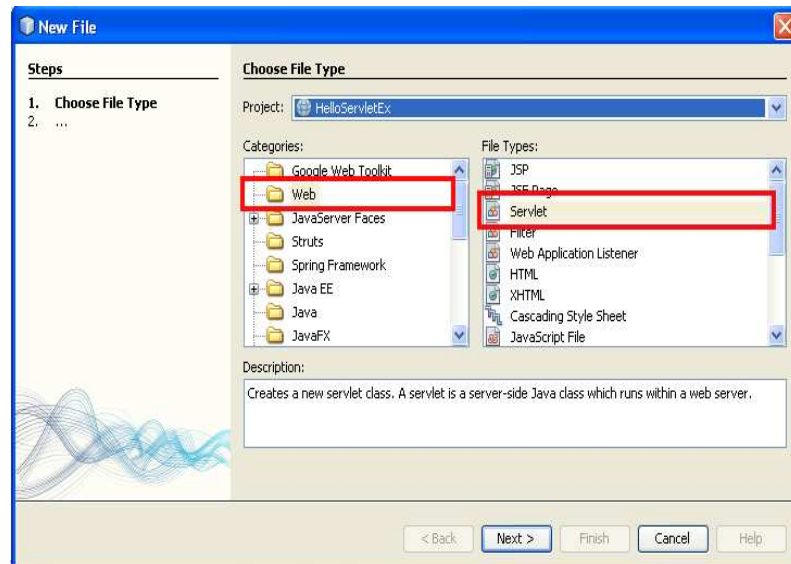
После нажатия на кнопку Finish получим Web-проект со следующей структурой папок:



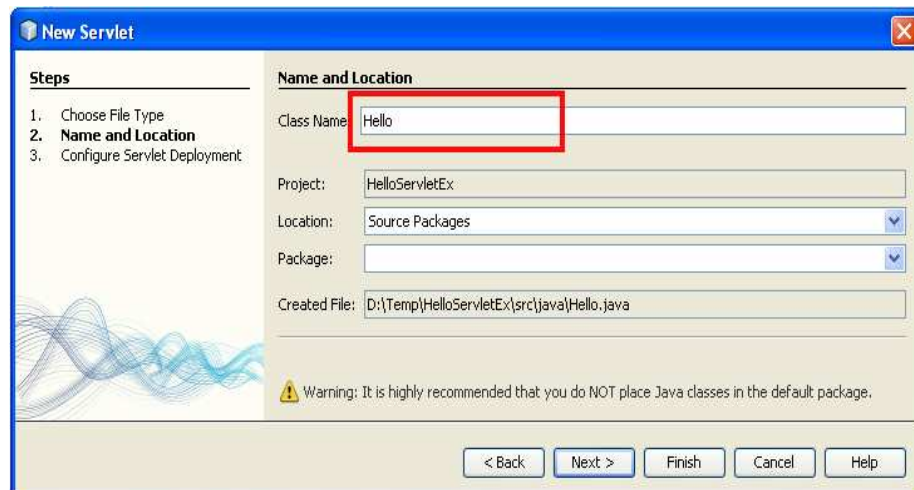
Из созданного проекта удалим файл `index.jsp`, так как он не будет использоваться в этом приложении.

Шаг 2. Создание класса сервлета

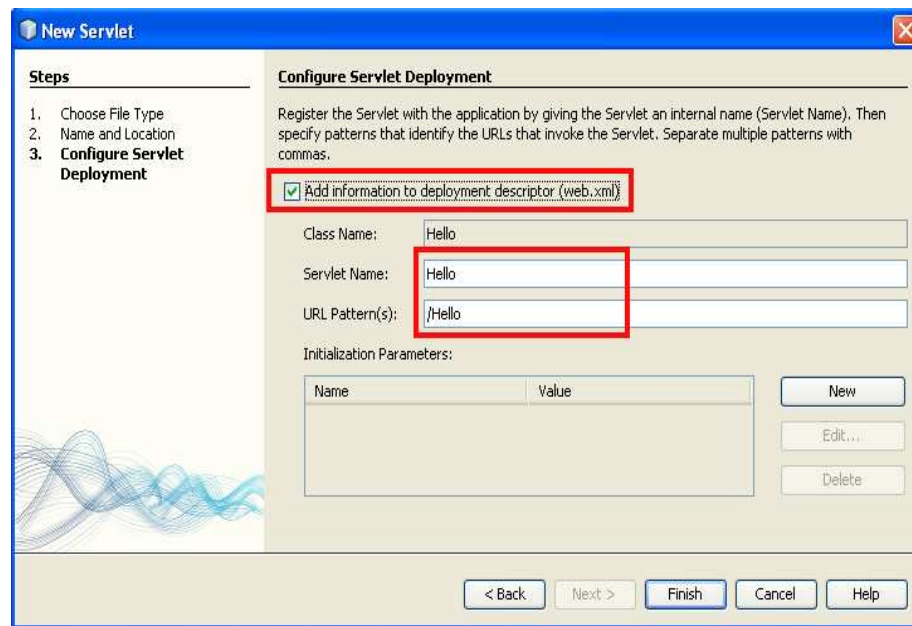
Теперь добавим к проекту сервлет. Для этого воспользуемся командой главного меню `File / New File` (или опцией панели инструментов – ). На шаге выбора типа файла следует указать категорию `Web` и тип файла `Servlet`:



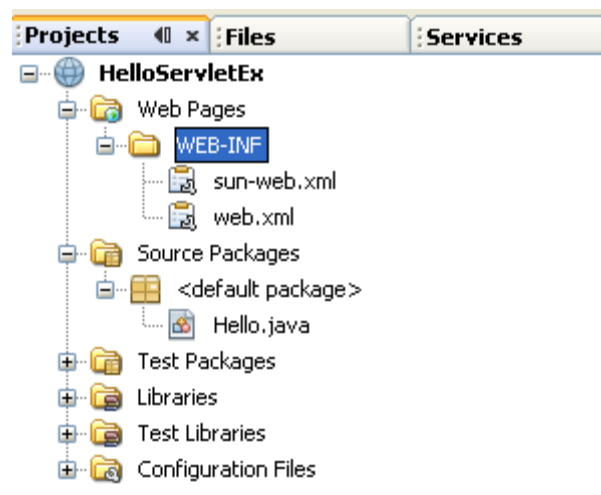
На следующем шаге указываем имя файла – `Hello`:



На шаге 3 следует выбрать опцию `Add Information to Deployment Descriptor (web.xml)`. Для параметров `Servlet Name`, `Class Name` и `Initialization Parameters` оставим значения по умолчанию. Настройки, сделанные на этом шаге, будут записаны в дескриптор развертывания разрабатываемого приложения.



В результате получим Web-проект со следующей структурой:



Модифицируем класс сервлета так, чтобы остались только методы `doGet()` и `doPost()`:

```
public class Hello extends HttpServlet {  
  
    @Override  
    protected void doGet(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {  
  
    }  
  
    @Override  
    protected void doPost(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {  
  
    }  
}
```

Шаг 3. Реализация обработки запроса GET

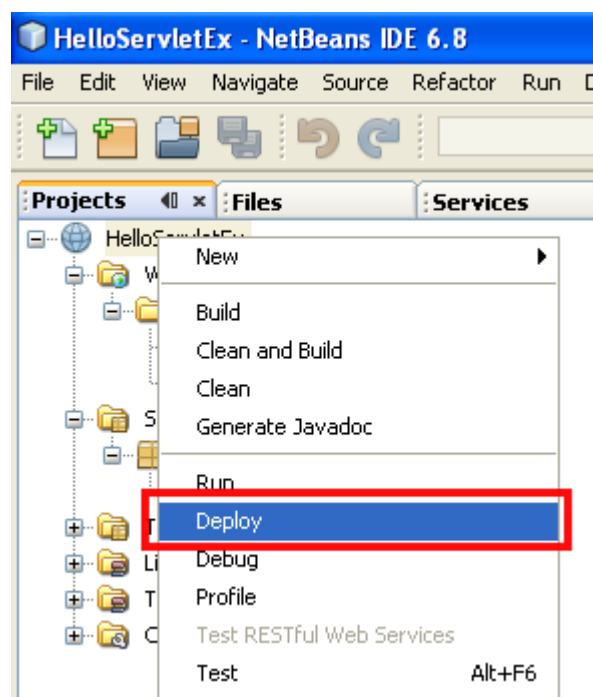
Как мы знаем, за обработку HTTP-запроса GET отвечает метод `doGet()`. Реализуем этот метод так, что бы он возвращал клиенту HTML-форму с полем для ввода имени пользователя:

```
@Override
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {

    // установка типа содержимого HTTP-ответа
    response.setContentType("text/html");
    // получение потока записи в HTTP-ответ
    PrintWriter writer = response.getWriter();
    try {
        // формирование HTML-формы
        writer.println("<form method='post'>");
        writer.println("<b>Enter your name:</b>");
        writer.println("<input type='text' name='userName' />");
        writer.println("<input type='submit' name='submit' value='Enter' />");
        writer.println("</form>");
    } finally {
        // закрытие потока HTTP-ответа
        writer.close();
    }
}
```

Шаг 5. Тестирование обработки запроса GET

Для запуска приложения, в первую очередь, необходимо развернуть на Web-сервере. Для этого воспользуемся командой `Deploy` из контекстного меню проекта:



Эта команда работает следующим образом:

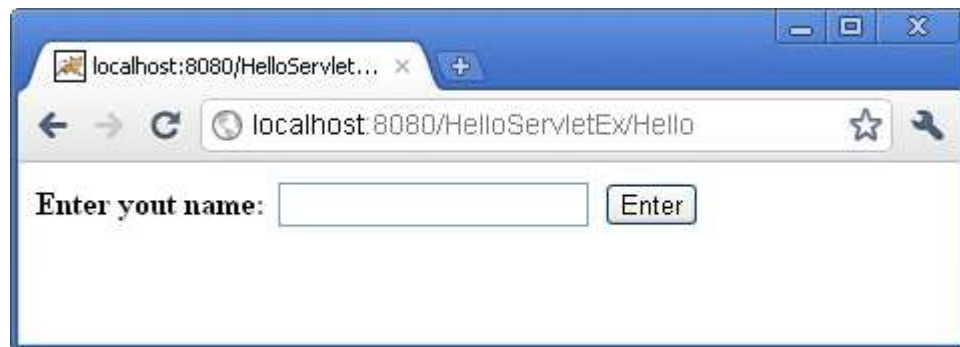
1) если не запущен сервер приложений (в данном случае Glass Fish), он запускается;

2) приложение разворачивается на сервере.

Теперь укажем в браузере URL

<http://localhost:8080/HelloServletEx/Hello>

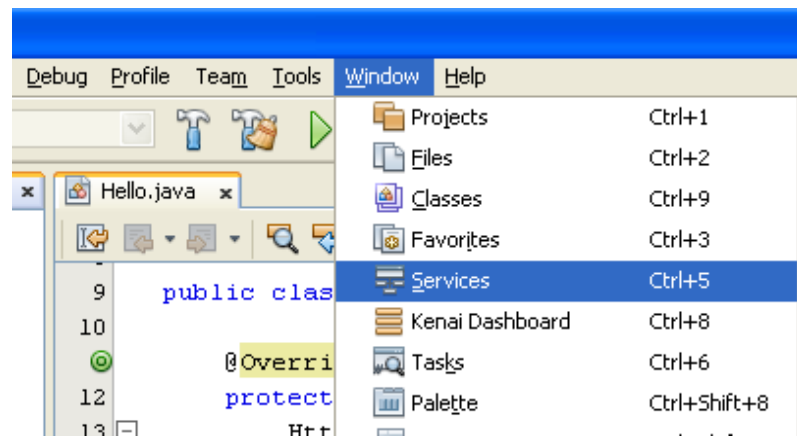
и получим результат выполнения запроса GET к сервлету:



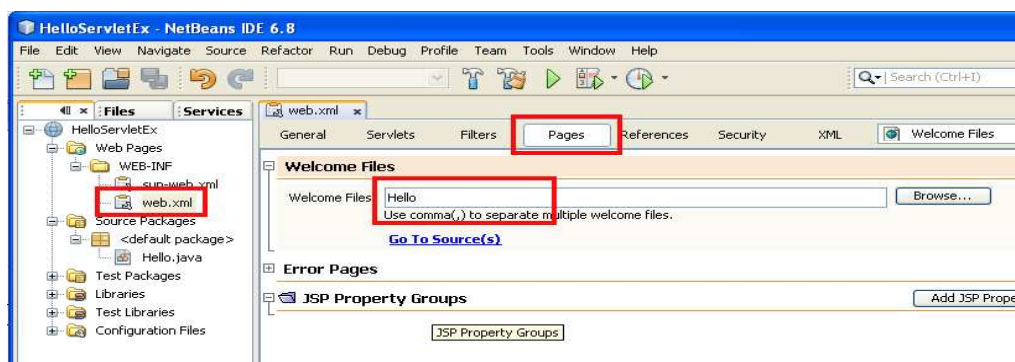
Если при переходе по указанной ссылке возникла ошибка, надо удостовериться, что запущен сервер Glass Fish и если да, то проверить его настройки. Для этого можно воспользоваться представлением Services среды NetBeans:



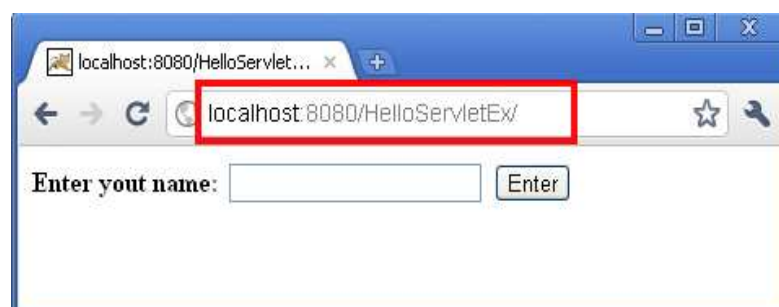
Если это представление неактивно, его можно отобразить с помощью команды главного меню Window / Services:



Наше приложение можно запустить и более удобным способом. Для этого надо указать в дескрипторе развертывания `web.xml` в качестве стартового ресурса сервлет `Hello`:



Теперь для запуска приложения воспользуемся командой главного меню `Run / Run Project` или клавишей `F6` – приложение загрузится в интернет-браузере:



Обратим внимание, что теперь доступ к сервлету мы получаем без указания его псевдонима – сервлет-контейнер сам переадресовывает ему запрос как стартовому ресурсу приложения.

Шаг 6. Реализация обработки запроса POST

Как мы видели выше, HTML-форма, которая генерируется в методе `doGet()`, отправляется на сервер методом POST, поэтому обработку её полей следует реализовать в методе `doPost()` сервлета:

```
@Override
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {

    // установка типа содержимого HTTP-ответа
    response.setContentType("text/html");
    // получение потока записи в HTTP-ответ
    PrintWriter writer = response.getWriter();
    try {
        // получаем имя, указанное пользователем,
        // из HTTP-запроса
        String userName = request.getParameter("userName");
        if (userName == null || userName.length() == 0) {
            // отображаем сообщение об ошибке
            writer.println(
                "<div style='color:red'>Error: incorrect user name</div>");
            writer.println("<form method='get'>");
            writer.println("<input type='submit' value='Try Again' />");
            writer.println("</form>");
        } else {
            // отображаем приветствие пользователя
            writer.println("Hello <b>" + userName + "</b>");
            writer.println("<form method='get'>");
            writer.println("<input type='submit' value='Back' />");
            writer.println("</form>");
        }
    } finally {
        // закрытие потока HTTP-ответа
        writer.close();
    }
}
```

Шаг 7. Тестирование обработки запроса POST

Для разворачивания модифицированной версии приложения на сервере воспользуемся командой `Deploy` NetBeans и обновим страницу в интернет-браузере. Укажем имя пользователя в текстовом поле и подтвердим ввод кнопкой `Enter`:



Теперь оставим поле ввода имени пустым и воспользуемся кнопкой Enter:



Таким образом, мы научились разрабатывать в среде NetBeans простейшие Web-приложения. В следующих разделах урока мы рассмотрим наиболее важные приемы работы с сервлетами.

2.6. Получение данных с запроса

Класс `javax.servlet.http.HttpServlet` определяет удобные методы для обработки каждого типа HTTP-запроса. Как мы видели из предыдущего примера, обработка запросов GET и POST осуществляется соответственно методами `doGet()` и `doPost()`. Этот класс также определяет отдельные методы для обработки других видов HTTP-запросов: `doDelete()`, `doHead()`, `doOptions()`, `doPut()`, `doTrace()`. В силу того, что на практике чаще всего

приходиться работать с HTTP-запросами GET и POST, мы сосредоточим свое внимание на методах обработки только этих запросов.

Каждый из методов `doGet()` и `doPost()` принимает два параметра: объект запроса (`request`) типа `HttpServletRequest` и объект ответа (`response`) типа `HttpServletResponse`.

Для определения HTTP-метода, с использованием которого был создан запрос, предназначен метод `HttpServletRequest`

```
java.lang.String getMethod()
```

Из объекта запроса наиболее часто приходится получать данные, которые приходят с HTML-формой или как параметры строки запроса. Для этого используются следующие методы, определенные в интерфейсе `javax.servlet.ServletRequest` (этот интерфейс расширяется интерфейсом `HttpServletRequest`).

Для получения значения параметра за его именем следует использовать метод

```
java.lang.String getParameter(java.lang.String name)
```

Если параметр с указанным именем отсутствует в запросе, возвращается значение `null`. Этот метод нецелесообразно использовать, если для заданного параметра возвращается множество значений (например, в случае выбора нескольких пунктов списка на HTML-форме), так как в этом случае он возвратит только первое значение из набора. Для считывания множества значений параметра предназначен метод:

```
java.lang.String[] getParameterValues(java.lang.String name)
```

Что бы получить коллекцию имен всех параметров запроса надо воспользоваться методом:

```
java.util.Enumeration<java.lang.String> getParameterNames()
```

Иногда бывает удобно получить словарь имен всех параметров запроса и их значений. Для этой цели предназначен метод:

```
java.util.Map<java.lang.String, java.lang.String[]> getParameterMap()
```

Он возвращает словарь, ключами которого являются имена параметров, а значениями – массивы строк (это позволяет считывать параметры с множественными значениями).

В случае инициализации запроса методом GET имена и значения параметров представляются в строке запроса. Если же запрос сформирован методом POST параметры приходят на сервер в теле запроса. Тело запроса можно считывать напрямую с помощью бинарного потока, который можно получить методом:

```
ServletInputStream getInputStream() throws java.io.IOException
```

Этот метод генерирует исключение `IllegalStateException`, если для запроса уже был вызван метод `getReader()`, который возвращает символьный поток для считывания тела запроса:

```
java.io.BufferedReader getReader() throws java.io.IOException
```

Метод `getReader()` также генерирует исключение `IllegalStateException`, если для запроса уже был вызван метод `getInputStream()`.

2.7. Построение ответа

За построение ответа в случае сервлета общего назначения отвечает объект типа `ServletResponse`, а в случае HTTP-сервлета – объект типа `HttpServletResponse`.

Для получения типа MIME содержание тела ответа используются метод

```
java.lang.String getContentType()
```

Для установки этого значения предназначен метод

```
void setContentType(java.lang.String type)
```

С его помощью можно также установить значение кодировки символов. Например, для ответа, возвращающего HTML-страницу, можно установить значение типа контента следующим образом:

```
response.setContentType("text/html;charset=UTF-8");
```

Значение кодировки символов, используемое в теле запроса, можно получить с помощью метода:

```
java.lang.String getCharacterEncoding()
```

Установить это значение можно соответствующим set-методом:

```
void setCharacterEncoding(java.lang.String charset)
```

Получить поток бинарный поток вывода для записи в тело ответа можно с помощью метода:

```
ServletOutputStream getOutputStream() throws java.io.IOException
```

Если предварительно был вызван метод `getWriter()` генерируется исключение `IllegalStateException`. Для получения объекта символьного потока предназначен метод:

```
java.io.PrintWriter getWriter() throws java.io.IOException
```

Он также генерирует исключение `IllegalStateException`, если раньше был вызван метод `getOutputStream()`.

2.8. URL запроса

Когда клиент посылает запрос Web-серверу, сервлет-контейнер должен определить, какой Web-компонент (в нашем случае сервлет) будет его обрабатывать. Это обеспечивается путем установки соответствия между URL и Web-компонентом. В общем случае URL запроса имеет вид:

```
http://[host]:[port][context_path][alias]?[query_string]
```

Контекстный путь (`context_path`) определяет приложение на Web-сервере. Он должен начинаться с символа «/» и заканчиваться строкой. Контекстный путь сохраняется в файле `sun-web.xml`.

Псевдоним (`alias`) определяет Web-компонент, который будет обрабатывать запрос. Он начинается с символа «/» и заканчивается строкой или паттерном (например, `*` или `*.jsp`). Псевдоним Web-элемента указывается в дескрипторе развертывания (файл `web.xml`). Отметим, что в спецификации сервлетов 3.0 есть возможность указывать параметры развертывания с помощью аннотации `@WebServlet`, которой обозначается класс сервлета (этот подход продемонстрирован в примере `TestUrl`, что рассматривается ниже в этом параграфе).

Интерфейсы `ServletRequest` и `HttpServletRequest` определяют ряд методов, которые позволяют получить разного рода информацию из URL запроса (табл. 2.6).

**Табл. 2.6. Методы интерфейсов
`ServletRequest` и `HttpServletRequest` для работы с URL**

№	Метод	Описание
1.	<code>getScheme()</code>	Схема, с помощью которой был сформирован запрос, например: <code>http</code> , <code>https</code> , <code>ftp</code> .
2.	<code>getServerName()</code>	Имя хоста, на котором работает сервер.
3.	<code>getServerPort()</code>	Номер порта, который прослушивается сервером.
4.	<code>getContextPath()</code>	Контекстный путь Web-приложения.
5.	<code>getServletPath()</code>	Псевдоним Web-элемента, который будет обрабатывать запрос.
6.	<code>getQueryString()</code>	Строку параметров запроса.
7.	<code>getRequestURI()</code>	Часть URL запроса, включающая контекстный путь Web-приложения и псевдоним Web-компонента.
8.	<code>getRequestURL()</code>	URL запроса без строки параметров.

Для исследования специфики работы этих методов разработано программу – `TestUrl` (исходный код вы можете найти в папке `Samples\03_TestUrl`). Результат работы этой программы наглядно демонстрирует предназначение методов, которые работают с URL:



2.9. Включение ресурса в сервлет

Часто бывает необходимо при формировании HTTP-ответа включать некоторые элементы во многие страницы приложения (например, верхняя и нижняя части страницы, главное меню). Генерацию содержимого этих общих элементов удобно «поручить» отдельным Web-компонентам (сервлетам, JSP-страницам, HTML-страницам).

Для включения других Web-элементов в тело сервлета предназначен класс `RequestDispatcher`, который принимает запрос клиента и пересылает его другому ресурсу в пределах Web-сервера. Его объект можно получить с помощью метода, определенного в интерфейсе `ServletRequest`:

```
RequestDispatcher getRequestDispatcher(String path)
```

Метод `getRequestDispatcher()` принимает в качестве параметра URL запрашиваемого ресурса. URL может быть относительным, но он не должен выходить за пределы контекста сервлета. Если URL начинается с символа «/», он интерпретируется относительно контекста Web-приложения. Если указанный ресурс не существует или сервлет-контейнер не реализует `RequestDispatcher`, возвращается значение `null`.

Объект `RequestDispatcher` можно получить и с помощью методов, интерфейса `javax.servlet.ServletContext`:

```
RequestDispatcher getRequestDispatcher(String path)
```

и

```
RequestDispatcher getNamedDispatcher(String name)
```

Первый из этих методов принимает путь к ресурсу. Второй – имя Web-компонента, которым может быть сервлет или JSP-страница (имя Web-компонента указывается в дескрипторе развертывания).

Включение другого ресурса (сервлета, JSP-страницы, HTML-страницы) в тело ответа осуществляется с помощью метода интерфейса `RequestDispatcher`

```
void include(ServletRequest request, ServletResponse response)  
            throws ServletException, java.io.IOException
```

2.10. Переадресация запроса другому Web-компоненту

Переадресация запроса другому ресурсу (сервлету, JSP-странице, HTML-странице) осуществляется с помощью метода

```
void forward(ServletRequest request, ServletResponse response)  
            throws ServletException, java.io.IOException
```

интерфейса `RequestDispatcher`. Использование этого метода позволяет одному сервлету провести подготовительные действия, а другому – сгенерировать ответ.

2.11. Работа с сессиями

Как мы знаем, протокол HTTP не предусматривает хранение состояния клиента между запросами. Это есть одной из основных причин того, что Web-интерфейсы более трудно разрабатывать и они не настолько удобные в использовании, как интерфейсы настольных приложений. Задача хранения клиентского состояния, зачастую, решается с использованием комплекса средств: часть состояния может храниться на Web-странице и передаваться

между запросами в hidden-полях (например, ViewState в ASP.NET), также используется подход хранения состояния в Cookies (которые также сохраняются на стороне клиента), для хранения состояния клиента на стороне сервера используется сессия.

Сессия существует на сервере некоторый период времени. Она предназначена для хранения данных клиента между запросами. Сессия связывается только с одним пользователем, который периодически делает запросы к серверу. Для идентификации пользователя используется специальное ключевое значение, которое хранится в cookies или передается через URL в параметрах запроса, если cookies отключены.

Для работы с сессией предназначен класс `HttpSession`, объект которого можно получить с помощью метода `getSession()` объекта `request`. Этот метод возвращает сессию, ассоциированную с запросом; если с запросом не связана сессия, она создается.

Для сохранения некоторого значения в сессии используется метод

```
void setAttribute(String name, Object value)
```

который принимает имя объекта (`name`) и ссылку на сам объект (`value`). Для получения объекта, сохраненного в сессии, за его именем (ключом) предназначен метод

```
Object getAttribute(String name)
```

Коллекцию ключей всех объектов, сохраненных в сессии, можно получить с помощью метода

```
java.util.Enumeration<String> getAttributeNames()
```

Удаление указанного атрибута можно осуществить с помощью метода

```
void removeAttribute(String name)
```

Для уничтожения сессии предназначен метод

```
void invalidate()
```

2.12. Работа с cookies

Cookies – это еще один способ управления состоянием. Но, в отличие от сессии, cookies хранятся на стороне клиента в виде массива строк. Каждое значение с cookie-набора содержит следующую информацию:

- имя cookie-набора;
- значение cookie-набора;
- истечение строка действия cookie-набора;
- домен и путь cookie-набора.

Строк действия cookie-набора определяет, когда он будет удален из компьютера клиента. Если это значение не указано, cookie-набор удаляется по завершению сеанса работы с браузером.

Домен и путь cookie-набора определяют, когда cookie-набор будет включен в заголовок HTTP-запроса: если URL запроса соответствует этим значениям, набор пересылается серверу, а в противном случае – нет.

Cookie-набор представляет класс `javax.servlet.http.Cookie`. Массив объектов `Cookie`, связанных с запросом, можно получить с помощью метода интерфейса `HttpServletRequest`:

```
Cookie[] getCookies()
```

Для установки значения cookie-набора используется метод интерфейса `HttpServletResponse`:

```
void addCookie(Cookie cookie)
```

Ниже в этом уроке рассматривается практический пример использования cookie-набора для хранения состояния на стороне клиента.

2.13. Пример разработки web-приложения с использованием сервлетов

Для демонстрации возможностей технологии сервлетов предлагается пример сравнительно несложного web-приложения, идея которого заключается в следующем.

2.13.1. Идея разработки

Допустим, разработчик программного обеспечения должен каждый день отчитываться о проделанной работе: он должен указывать в отчете проект, над которым он работал, выполненные задания и количество затраченного времени. На основании его отчетов, ему начисляется зарплата за месяц. Предлагается разработать web-приложение, позволяющее максимально упростить ежедневный отчет и формирование сводного отчета за рабочий месяц.

2.13.2. Постановка требований

Для начала опишем и реализуем основную функциональную возможность приложения – формирование дневного отчета. Требования к этой функциональности опишем в виде истории пользователя (или прецедента)¹.

Названия прецедента: «Формирование дневного отчета».

Основной исполнитель: Разработчик.

Цели исполнителя: максимально быстро и безошибочно сделать отчет о выполненной проектной задаче.

Результат: добавление к дневному отчету информации о выполненной проектной задаче.

Основной успешный сценарий:

1. Пользователь заходит на страницу dayreport. На странице отображаются поля для ввода отчета о реализованной задаче. На этой странице также отображаются задачи, о которых пользователь уже отчитался за текущий рабочий день, и суммарное количество отработанных за день часов согласно отчету.

2. Пользователю предлагается сделать отчет по текущей дате.

3. В списке проектов, в которых пользователь берет участие, активным является тот проект, задачу по которому пользователь создавал последней. Если «последнего проекта» определить не удалось, в списке доступных проектов отображается пустое значение.

4. Пользователь выбирает проект (или оставляет значение по умолчанию без изменений), указывает количество затраченного времени в часах и делает краткое описание выполненной задачи.

Правила валидации введенных пользователем данных:

¹ Используется развернутый формат описания прецедента, который предлагается в [Larman_UML_2.0, С. 95-135]

- Должен быть указан проект, по которому делается отчет.
- Количество затраченного времени должно быть положительным дробным числом с одним знаком после запятой в диапазоне от 0 (исключительно) до 24 (включительно).
- Описание задачи должно быть непустым значением, длина которого не превышает 1000 символов.

5. После заполнения всех необходимых полей пользователь нажимает кнопку «Сохранить» для добавления задачи к дневному отчету:

- В случае ошибки в заполненных данных, пользователю отображается соответствующее сообщение об ошибке (при этом значения в заполненных полях не исчезают).
- В случае успешной проверки корректности введенных данных, задача отображается в списке задач дневного отчета. Также, соответственно меняется суммарное количество дневного рабочего времени.

Альтернативные неуспешные сценарии:

Пользователь отменяет ввод данных:

1. Пользователь может отменить введенные данные с помощью кнопки «Отмена».

Поля формы ввода данных в таком случае принимают значения по умолчанию.

Список проектов пользователя пуст:

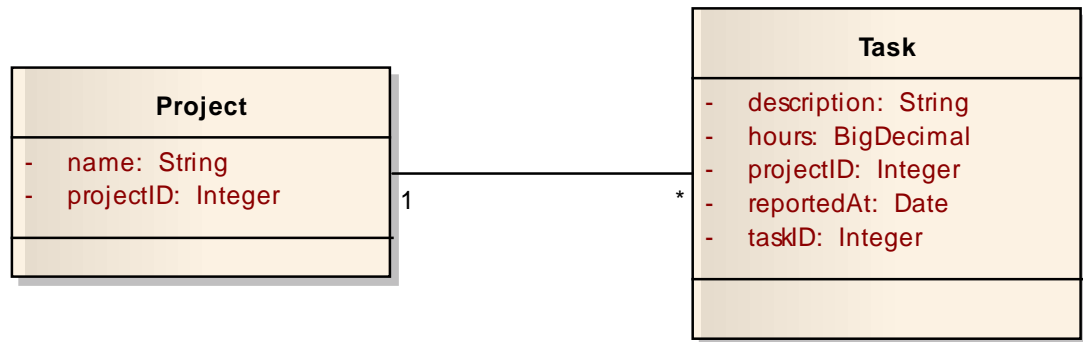
1. Пользователь не может сохранить введенные данные, так как список его проектов пуст.

Частота использования: практически постоянно.

Нельзя не согласиться, что функциональные возможности приложения, изложенные в истории «Формирование дневного отчета», не удовлетворят даже самого неприхотливого пользователя, так как в них, например, не представлены возможности просматривать и корректировать отчеты, сделанные не в текущем рабочем дне. Но эти функциональные возможности целесообразно реализовывать с помощью более мощных и удобных средств Web-программирования Java, в частности, Java Server Pages.

2.13.3. Разработка доменной модели

Следующим важным шагом процесса разработки нашего приложения есть построение доменной модели предметной области, из которой потом будет сформирована модель базы данных:



2.13.4. Создание базовой инфраструктуры проекта

Для создания базовой инфраструктуры проекта выполним следующие действия:

1. Создадим Web-проект в среде NetBeans под названием DR (сокращение от Developer Reporting).
2. Удалим файл `index.jsp`, который находится в паке `Web Pages`.
3. В корневой папке проекта создадим папку `sql`, где разместим `sql`-скрипты для работы с базой данных приложения.
4. В корневой папке проекта создадим папку `lib`, где разместим `jar`-файл JDBC-драйвера для MySQL.
5. Подключим `jar`-файл JDBC-драйвера к нашему проекту.
6. Создаем пакет `dr.domainmodel.entities`, в этом пакете разместим классы-сущности – `Project` и `Task`, представляющие записи с одноименных таблиц базы данных.
8. Создаем пакет `dr.domainmodel.repository.base`, в этом пакете разместим интерфейсы репозитариев².
9. Создаем пакет `dr.domainmodel.repository.impl.mysql`, в этом пакете разместим реализации репозитариев, специфичные для СУБД MySQL.

Мы не будем здесь описывать детали технической реализации шагов 1-9, так как это не касается основной темы урока. Результаты выполнения этих действий нужно проанализировать самостоятельно, исследовав исходные коды примера.

² Репозиторий – архитектурный паттерн, решающий задачу организации работы с источниками данных.

2.13.5. Разработка бизнес-логики и графического интерфейса приложения

С целью упрощения изложения материала разделим разработку бизнес-логики и графического интерфейса рассматриваемого приложения на следующие этапы:

- 1) разработка формы добавления задачи к отчету;
- 2) проверка корректности введенных данных и сохранение задачи в базе данных;
- 3) отображения списка задач, выполненных за текущий день.

К уроку прилагаются варианты исходных кодов проекта на момент окончания каждого из этапов разработки (папка `Samples\04_DR_Servlet`).

Этап 1. Разработка формы добавления задачи к отчету

Теперь приступим непосредственно к разработке web-компонентов нашего приложения. Для начала создадим пакет `dr.servlet`, где будем размещать сервлеты.

Создадим сервлет `DayReport` (с URL `/DayReport`), отвечающий за формирование и просмотр дневного отчета разработчика. Модифицируем дескриптор развертывания приложения (т.е. файл `web.xml`) так, чтобы сервлет `DayReport` был стартовой страницей приложения:

```
<welcome-file-list>
  <welcome-file>DayReport</welcome-file>
</welcome-file-list>
```

Запустим проект на выполнение, чтобы убедиться в его работоспособности. По умолчанию URL приложения будет таким <http://localhost:8080/DR/>.

Проведем инициализацию сервлета `DayReport`, для этого переопределим метод `init()` базового класса. Обратим внимание, что код метода `init()` размещен в блоке `try-catch`, поскольку здесь возможны исключения, в частности, загрузка JDBC-драйвера может вызвать исключение `ClassNotFoundException`. Любое исключение в этой части кода перехватывается блоком `catch`, где оно трансформируется в исключение

UnavailableException, что свидетельствует о невозможности создания (значит и о недоступности) сервлета.

```
@Override
public void init(ServletConfig config) throws ServletException {
    super.init(config);
    try {
        // загрузка JDBC-драйвера MySQL
        Class.forName("com.mysql.jdbc.Driver");
    } catch (Throwable e) {
        // сервлет недоступен,
        // если в процессе инициализации сервлета возникла ошибка
        throw new UnavailableException("Servlet could not be created");
    }
}
```

Теперь приступим к реализации метода doGet() сервлета DayReport. Этот метод, как мы знаем, отвечает за обработку HTTP-запроса GET. В коде этого метода мы пока только отображаем HTML-форму для добавления новой задачи в отчет.

```
private final String DB_CONNECTION_URL
    = "jdbc:mysql://localhost/DR?user=root";

@Override
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    try {
        // создание репозитория для таблицы Project
        ProjectRepository projectRepository
            = new MySqlProjectRepository(DB_CONNECTION_URL);
        // создание репозитория для таблицы Task
        TaskRepository taskRepository
            = new MySqlTaskRepository(DB_CONNECTION_URL);
        // получение списка всех проектов,
        // в которых берет участие пользователь
        List<Project> projects = projectRepository.select();

        // определение типа содержимого HTTP-ответа
        response.setContentType("text/html;charset=UTF-8");
        // получение потока для формирования HTTP-ответа
        PrintWriter writer = response.getWriter();
        try {
            // генерация формы ввода данных о новой задаче
            renderAddNewTaskForm(writer, projects);
        } finally {
            // закрытие потока HTTP-ответа
            writer.close();
        }
    } catch (Throwable e) {
        // любое исключение представить как ServletException
        throw new ServletException(e);
    }
}
```


За генерацию HTML-разметки отвечает метод `renderAddNewTaskForm()`:

```
/**
 * Генерация HTML-представления формы
 * добавления новой задачи к отчету
 *
 * @param writer - поток HTTP-ответа
 * @param projects - список проектов пользователя
 */
private void renderAddNewTaskForm(PrintWriter writer, List<Project> projects)
{
    writer.println("<div>");
    // HTML-форма
    writer.println("<form method='POST'>");
    writer.println("<h2>Add New Task</h2>");
    // выпадающий список для выбора проекта
    writer.println("<b>Project </b>");
    writer.println("<select name='"+ PROJECT_ATTRIBUTE + "'>");
    for(Project p : projects) {
        writer.println("<option value='"
            + p.getProjectID() + "'>"
            + p.getName()
            + "</option>");
    }
    writer.println("</select>");
    writer.println("&nbsp;&nbsp;&nbsp;");

    // поле для ввода значения количества затраченных часов
    writer.println("<b>Hours </b>");
    writer.println("<input type='text' name='"
        + HOURS_ATTRIBUTE
        + "' value=''/>");
    writer.println("<br/>");


    // поле для ввода описания проделанной работы
    writer.println("<b>Task Description </b>");
    writer.println("<br/>");
    writer.println("<textarea name='"
        + DESCRIPTION_ATTRIBUTE +
        "' rows='5' cols='50' /></textarea>");
    writer.println("<br/>");

    // кнопка подтверждения введенных данных
    writer.println("<input type='submit' name='save' value='Save' />");

    // кнопка отклонения введенных данных
    writer.println("<input type='submit' name='cancel' value='Cancel' />");

    writer.println("</form>");
    writer.println("</div>");
}
```

Получаем следующий результат работы приложения:



The screenshot shows a web browser window with the title 'Developer Reporting System'. The address bar displays 'localhost:8080/DR/'. The main content area features a heading 'Add New Task'. Below this, there are two input fields: 'Project' with a dropdown menu showing 'Easy Payment' and a blue arrow, and 'Hours' with an empty text box. Underneath these is a label 'Task Description' followed by a large, empty text area. At the bottom of the form are two buttons: 'Save' and 'Cancel'.

Форма приложения представлена следующей HTML-разметкой:

```

1 <div>
2 <form method='POST'>
3 <h2>Add New Task</h2>
4 <b>Project    </b>
5 <select name='ProjectID'>
6 <option value='1'>Easy Payment</option>
7 <option value='2'>Virtual Market</option>
8 <option value='3'>SCP</option>
9 </select>
10     <br>
11     <b>Hours    </b>
12 <input type='text' name='hours' />
13 <br>
14 <b>Task Description    </b>
15 <br>
16 <textarea name='description' rows='5' cols='50' />
17 </textarea>
18 <br>
19 <input type='submit' name='save' value='Save' />
20 <input type='submit' name='cancel' value='Cancel' />
21 </form>
22 </div>
23

```

Мы можем справедливо возмутиться по поводу продемонстрированного подхода верстки HTML-страниц. Действительно, формируя HTML из Java-кода, маловероятно, что можно получить качественные HTML-страницы при рациональных затратах времени и усилий. Но на этом этапе разработки нашего приложения мы ограничимся использованием только такого метода, так как более удобные технологии формирования статической HTML-разметки выходят за пределы данного урока.

Как мы видим, полученная нами HTML-страница имеет существенный недостаток: она не имеет верхней и нижней части. Эти фрагменты HTML-

разметки будут одинаковыми для всех страниц приложения, поэтому есть смысл поручить их генерацию отдельным сервлетам. С этой целью создадим сервлеты PageHeader и PageFooter, отвечающие за генерацию соответственно нижней и верхней части страницы приложения:

```
/**
 * Сервлет, генерирующий верхнюю часть страницы
 */
public class PageHeader extends HttpServlet {

    /**
     * На основе HTTP-запроса формирует URL приложения.
     * Например, http://localhost:8080/DR
     * @param request - HTTP-запрос
     * @return URL приложения
     */
    protected String getApplicationPath(HttpServletRequest request) {
        return request.getScheme()
            + "://" + request.getServerName()
            + ":" + request.getServerPort()
            + request.getContextPath();
    }

    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {

        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<link href='"
            + getApplicationPath(request)
            + "/content/styles/style.css' rel='stylesheet'"
            + " type='text/css'/>");
        out.println("<title>");
        out.println("Developer Reporting System");
        out.println("</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<div id='page-head'>");
        out.println("<h1>Developer Reporting System</h1>");
        out.println("</div>");
        out.println("<div id='page-content'>");
        // нельзя закрывать поток out,
        // так как он закрывается в сервлете DayReport
    }

    @Override
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        processRequest(request, response);
    }
}
```

```
}  
...  
/**  
 * Сервлет, генерирующий нижнюю часть страницы  
 */  
public class PageFooter extends HttpServlet {  
    protected void processRequest(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {  
  
        PrintWriter out = response.getWriter();  
        out.println("</div>");  
        out.println("<div id='page-footer'>");  
        out.println("Reporting Software &copy; 2010");  
        out.println("</div>");  
        out.println("</body>");  
        // нельзя закрывать поток out,  
        // так как он закрывается в сервлете DayReport  
    }  
    @Override  
    protected void doGet(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {  
        processRequest(request, response);  
    }  
    @Override  
    protected void doPost(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {  
        processRequest(request, response);  
    }  
}
```

Обратим внимание на некоторые важные моменты в реализации сервлетов PageHeader и PageFooter. Во-первых, в этих сервлетах нельзя закрывать поток вывода данных в HTTP-ответ, так как он закрывается в сервлете, включающем эти сервлеты, – DayReport. Во-вторых, клиент не должен иметь возможности вызывать эти сервлеты из командной строки браузера через URL, поэтому в дескрипторе развертывания web.xml мы только указываем имена сервлетов PageHeader и PageFooter, но не связываем с ними URL:

```
<web-app>  
...  
  <servlet>  
    <servlet-name>PageHeader</servlet-name>  
    <servlet-class>dr.servlet.PageHeader</servlet-class>  
  </servlet>  
  <servlet>  
    <servlet-name>PageFooter</servlet-name>  
    <servlet-class>dr.servlet.PageFooter</servlet-class>  
  </servlet>  
...  
</web-app>
```

Теперь, когда мы имеем заголовок страницы, поработаем над её внешним видом: в папке `Web Pages` нашего проекта создадим папку `content` и вложенную в нее папку `styles`. В папке `styles` поместим файл `style.css`, в котором реализуем CSS-стили для страниц нашего приложения. Стили приложения укажем при формировании заголовка страницы.

Для включения `PageHeader` и `PareFooter` в содержимое сервлета `DayReport` разработаем метод `renderResource()`:

```
/**
 * Включение содержимого ресурса resourceName в HTTP-ответ
 * @param request - HTTP-запрос
 * @param response - HTTP-ответ
 * @param resourceName - имя ресурса
 */
private void renderResource(
    HttpServletRequest request,
    HttpServletResponse response,
    String resourceName)
    throws ServletException, IOException {
    // получения диспатчера для ресурса resourceName
    RequestDispatcher requestDispatcher
        = request.getServletContext().getNamedDispatcher(resourceName);
    // включение в HTTP-ответ содержимого ресурса resourceName
    requestDispatcher.include(request, response);
}
```

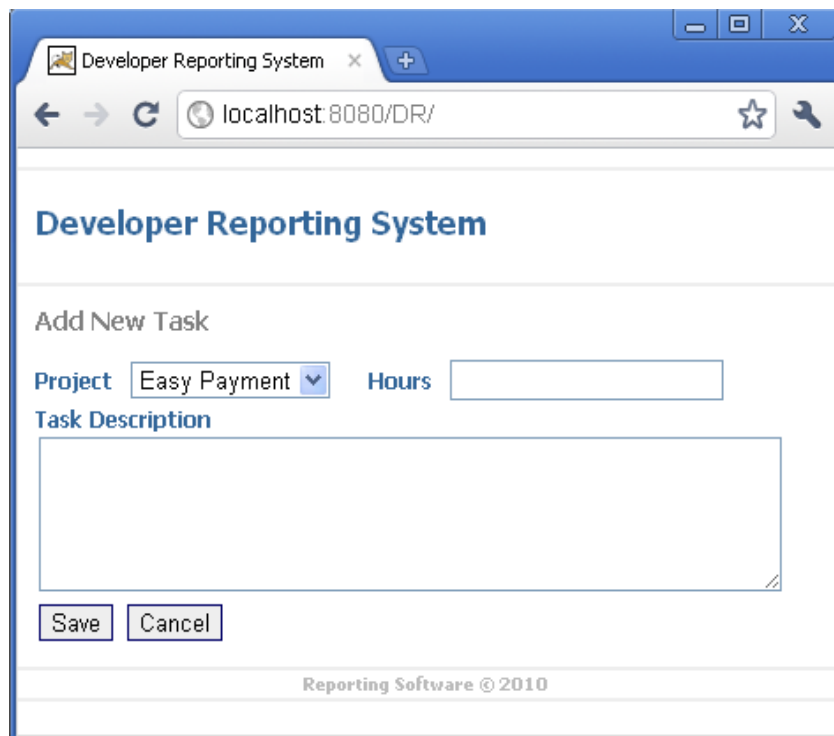
Теперь соответствующим образом модифицируем метод `doGet()` сервлета `DevReport`:

```
@Override
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {

    try {
        // создание репозитория для таблицы Project
        ProjectRepository projectRepository
            = new MySqlProjectRepository(DB_CONNECTION_URL);
        // создание репозитория для таблицы Task
        TaskRepository taskRepository
            = new MySqlTaskRepository(DB_CONNECTION_URL);
        // получение списка всех проектов,
        // в которых берет участие пользователь
        List<Project> projects = projectRepository.select();
        // определение типа содержимого HTTP-ответа
        response.setContentType("text/html;charset=UTF-8");
        // получение потока для формирования HTTP-ответа
        PrintWriter writer = response.getWriter();
        try {
            // генерация верхней части страницы
            renderResource(request, response, "PageHeader");
            // генерация формы ввода данных о новой задаче
            renderAddNewTaskForm(writer, projects);
        }
    }
}
```

```
// генерация нижней части страницы
renderResource(request, response, "PageFooter ");
} finally {
    // закрытие потока HTTP-ответа
    writer.close();
}
} catch (Throwable e) {
    // любое исключение представить как ServletException
    throw new ServletException(e);
}
}
```

В результате получим страницу:



Исходные коды примера на текущем этапе реализации находятся в папке Samples\04_DR_Servlet\DR_Version_01\.

Этап 2. Проверка корректности введенных данных и сохранение задачи в базе данных

Приступим к реализации возможности добавления новой задачи к отчету. Как мы видели выше, на странице создается HTML-форма, которая отправляется на сервер методом POST. Поэтому код обработки этой формы реализуем в методе doPost() сервлета DayReport.

Одним из необходимых условий при работе с данными, которые вводит пользователь, является проверка их корректности на стороне сервера. С этой целью добавим в проект класс ModelState. Основное предназначение этого

класса – хранение состояния полей HTML-формы и информации о корректности представленных в ней. Изменим метод `renderAddNewTaskForm()` так, что бы он принимал объект `ModelState` в качестве параметра и отображал состояний полей формы, а также, при необходимости, сообщения об ошибках в данных, введенных пользователем:

```
/**
 * Генерация HTML-представления формы
 * добавления новой задачи к отчету
 *
 * @param writer - поток HTTP-ответа
 * @param projects - список проектов пользователя
 * @param modelState - текущее состояние модели
 */
private void renderAddNewTaskForm(PrintWriter writer,
    List<Project> projects, ModelState modelState) {
    writer.println("<div>");
    // HTML-форма
    writer.println("<form method='POST'>");
    writer.println("<h2>Add New Task</h2>");
    // сообщения валидации
    writer.println("<div class='validationsummary'>");
    if (!modelState.isValid()) {
        writer.println("<b>Validation Summary</b>");
    }
    writer.println("<ul>");
    for (String message : modelState.getValidationMessages()) {
        writer.println("<li>" + message + "</li>");
    }
    writer.println("</ul>");
    writer.println("</div>");
    // выпадающий список для выбора проекта
    writer.println("<b>Project </b>");
    // считывание значения ProjectID из состояния модели
    String project = modelState.is(PROJECT_ATTRIBUTE)
        ? modelState.get(PROJECT_ATTRIBUTE).getValue() : "";
    writer.println("<select name='" + PROJECT_ATTRIBUTE + "'>");
    for (Project p : projects) {
        writer.println("<option value='" + p.getProjectID() + "'"
            + (p.getProjectID().toString().equals(project)
                ? "selected" : "")
            + ">" + p.getName() + "</option>");
    }
    writer.println("</select>");
    writer.println("&nbsp;&nbsp;&nbsp;");
    // поле для ввода значения количества затраченных часов
    writer.println("<b>Hours </b>");
    // считывание значения Hours из состояния модели
    String hours = modelState.is(HOURS_ATTRIBUTE)
        ? modelState.get(HOURS_ATTRIBUTE).getValue() : "";
    writer.println("<input type='text' name='"
        + HOURS_ATTRIBUTE
        + "' value='" + hours + "'/>");
    writer.println("<br/>");
    // поле для ввода описания проделанной работы
    writer.println("<b>Task Description </b>");
    writer.println("<br/>");
```

```

// считывание значения Description из состояния модели
String description = modelState.is(DESCRIPTION_ATTRIBUTE)
    ? modelState.get(DESCRIPTION_ATTRIBUTE).getValue() : "";
writer.println("<textarea name='"
    + DESCRIPTION_ATTRIBUTE
    + "' rows='5' cols='50' />" +
    description
    + "</textarea>");
writer.println("<br/>");
// кнопка подтверждения введенных данных
writer.println("<input type='submit' name='save' value='Save' />");
// кнопка отклонения введенных данных
writer.println("<input type='submit' name='cancel' value='Cancel' />");
writer.println("</form>");
writer.println("</div>");
}

```

Разработаем метод `doRequest()`, предназначенный для генерации HTML страницы с учетом текущего состояния модели:

```

/**
 * Отображение страницы
 * с учетом текущего состояния модели
 * @param response - HTTP-ответ
 * @param request - HTTP-запрос
 * @param modelState - состояние модели
 */
private void doRequest(HttpServletResponse response,
    HttpServletRequest request, ModelState modelState)
    throws IOException, SQLException, ServletException {
    // создание репозитория для таблицы Project
    ProjectRepository projectRepository
        = new MySqlProjectRepository(DB_CONNECTION_URL);
    // создание репозитория для таблицы Task
    TaskRepository taskRepository
        = new MySqlTaskRepository(DB_CONNECTION_URL);
    // определение типа содержимого HTTP-ответа
    response.setContentType("text/html;charset=UTF-8");
    // получение списка всех проектов,
    // в которых берет участие пользователь
    List<Project> projects = projectRepository.select();
    PrintWriter writer = response.getWriter();
    try {
        // включение верхней части страницы
        renderResource(request, response, "PageHeader");
        // вывод в поток ответа HTML-верстки
        // формы ввода данных о новой задаче
        renderAddNewTaskForm(writer, projects, modelState);
        // включение нижней части страницы
        renderResource(request, response, "PageFooter");
    } finally {
        writer.close();
    }
}

```

Модифицируем метод `doGet()`, чтобы он использовал метод `doRequest()`:

```

@Override
protected void doGet(HttpServletRequest request,

```



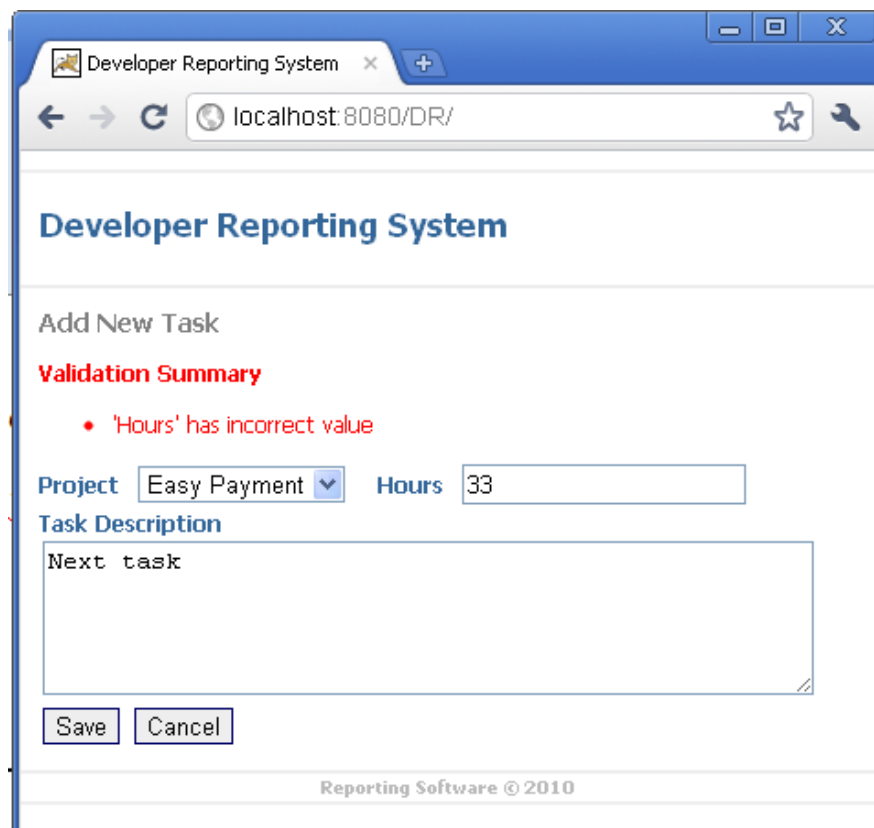
```
HttpServletResponse response) throws ServletException, IOException {
    try {
        // состояние модели по умолчанию
        ModelState modelState = new ModelState();
        // отображение страницы
        doRequest(response, request, modelState);
    } catch (Throwable e) {
        // любое исключение представить как ServletException
        throw new ServletException(e);
    }
}
```

Теперь приступим к реализации метода `doPost()`. В первую очередь, при его вызове следует проверить, с помощью какой кнопки (Save или Cancel) HTML-форма была отправлена на сервер. Если форма была отправлена с помощью кнопки Save, тогда происходит проверка введенных пользователем данных. В случае успеха, новый пункт отчета сохраняется в базе данных, а в случае неудачи – пользователю представляются сообщения об ошибках. Собственно отображение страницы, как и в случае метода `doGet()`, осуществляется методом `doRequest()`.

```
@Override
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    try {
        ModelState modelState;
        // проверяем с помощью какой кнопки
        // HTML-форма была отправлена на сервер
        if (request.getParameter("save") != null) {
            // форма была отправлена на сервер с помощью кнопки Save
            // считывание полей формы из запроса
            modelState = readModelFromRequest(request);
            // проверка корректности введенных данных
            validateModel(modelState);
            if (modelState.isValid()) {
                // введенные данные корректные
                // получение объекта Task из состояния модели
                Task task = readTaskFromModelState(modelState);
                // установить текущую дату для отчета
                task.setReportedAt(new Date());
                // создание репозитория для таблицы Task
                TaskRepository taskRepository
                    = new MySqlTaskRepository(DB_CONNECTION_URL);
                // сохранить отчет в базе данных
                taskRepository.insert(task);
                // очистить состояние модели
                modelState.clear();
            }
        } else {
            // форма была отправлена на сервер с помощью кнопки Cancel
            // создать состояние модели по умолчанию
            modelState = new ModelState();
        }
    }
}
```

```
// отображение страницы
doRequest(response, request, modelState);
} catch (Throwable e) {
    // любое исключение представить как ServletException
    throw new ServletException(e);
}
}
```

Запустим наше приложение. Для демонстрации возможностей проверки корректности ввода укажем неверное значение в поле `Hours` и попытаемся сохранить данные:



Если пользователь некоторое время выполняет задачи по одному проекту, будет удобно, чтобы этот проект был активным по умолчанию при каждом запросе формы добавления новой задачи. Реализовать такую возможность можно с использованием cookie-набора (познакомиться с реализацией можно с исходных кодов примера).

Таким образом, текущий этап работы над проектом закончен. Исходные коды находятся папке `Samples\04_DR_Servlet\DR_Version_02\`.

Этап 3. Отображения списка задач, выполненных за текущий день

Для отображения на странице списка выполненных задач за текущий день модифицируем метод `doRequest()`:

```
private void doRequest(HttpServletRequest response,
    HttpServletRequest request,
    ModelState modelState)
    throws IOException, SQLException, ServletException {

    // создание репозитория для таблицы Project
    ProjectRepository projectRepository
        = new MySqlProjectRepository(DB_CONNECTION_URL);

    // создание репозитория для таблицы Task
    TaskRepository taskRepository
        = new MySqlTaskRepository(DB_CONNECTION_URL);

    // определение типа содержимого HTTP-ответа
    response.setContentType("text/html;charset=UTF-8");

    // получение списка всех проектов,
    // в которых берет участие пользователь
    List<Project> projects = projectRepository.select();
    // получение списка всех задач,
    // которые пользователь выполнил в текущий день
    List<Task> todayTasks = taskRepository.selectByDate(new Date());

    // вычисление общего количества отработанных часов за текущий день
    BigDecimal totalHours
        = taskRepository.selectTotalHoursByDate(new Date());

    // создание потока для формирования HTTP-ответа клиенту
    PrintWriter writer = response.getWriter();
    try {
        // включение верхней части страницы
        renderResource(request, response, "PageHeader");
        // вывод в поток ответа HTML-верстки
        // формы ввода данных о новой задаче
        renderAddNewTaskForm(writer, projects, modelState);
        // вывод в поток ответа таблицы задач,
        // выполненных за текущий день
        renderTodayTasks(writer, todayTasks, totalHours);
        // включение нижней части страницы
        renderResource(request, response, "PageFooter");
    } finally {
        writer.close();
    }
}
```

Получаем результат:

Developer Reporting System

Add New Task

Project: Easy Payment Hours:

Task Description

Save Cancel

Reported Today

Hours	Reported At	Description
3.0	2010-11-01 21:32:52	Implemented important improvements.
4.0	2010-11-01 21:32:10	Fixed some bugs.

Total Hours: 7.0

Reporting Software © 2010

Исходные коды полученного варианта приложения находятся в папке Samples\04_DR_Servlet\DR_Version_03\.

Домашнее задание

Задача 1

Разработчику программного обеспечения приходится много учиться, чтобы не отставать от бурно развивающихся технологий программирования. Для этого ему приходится постоянно читать разные статьи онлайн. Часто бывает так, что найденную статью прочитать обязательно надо, но не сразу, а когда появится свободное время. В таких ситуациях найденные полезные ссылки нужно где-то сохранять, а когда их накопится большее количество, то удобно систематизировать.

Предлагается разработать Web-программу для хранения ссылок на полезные онлайн-ресурсы. Программа должна обладать такими минимальными возможностями:

- 1) сохранение ссылки;
- 2) добавление комментария к ссылке;
- 3) определение статуса ссылки (например, «Новая», «Прочитанная», «Подлежит удалению» и т.п.);
- 4) удаление ненужных ссылок.

Для хранения данных программы следует использовать базу данных.

Результаты выполнения задачи:

- 1) истории пользователя системы (в развернутом виде);
- 2) доменная модель предметной области (диаграмма классов UML);
- 3) описание архитектуры (диаграмма компонентов UML);
- 4) реализованный программный продукт.

Использованные информационные источники

[Shild_JSE6]

Герберт Шилд. Полный справочник по Java SE 6. Изд. 7-е. / Пер. с англ. – М.: «Издательский дома Вильямс», 2007. – 1034 с.

[Servlet_JSP_Cookbook]

Java сервлеты и JSP: сборник рецептов. Изд. 2-е. / Пер. с англ. – М.: КУДИЦ-ПРЕСС, 2006. – 768с.

[JEE_Tutorial]

<http://download.oracle.com/javase/5/tutorial/doc/>

[Larman_UML_2.0]

Ларман, Крэг. Применение UML 2.0 и шаблонов проектирования, 3-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2007. – 736 с.