

Занятие № 16.

Spring Framework.

Введение в Spring Web MVC

Spring Framework и его модули

Введение в Spring MVC

Практика

Spring Framework и его модули

Spring Framework (или коротко Spring) — универсальный фреймворк с открытым исходным кодом для Java-платформы. Также существует форк для платформы .NET Framework, названный Spring.NET.

Первая версия была написана Родом Джонсоном, который впервые опубликовал её вместе с изданием своей книги «Expert One-on-One Java EE Design and Development».

Несмотря на то, что Spring Framework не обеспечивал какую-либо конкретную модель программирования, он стал широко распространённым в Java-сообществе главным образом как альтернатива и замена модели Enterprise JavaBeans. Spring Framework предоставляет большую свободу Java-разработчикам в проектировании; кроме того, он предоставляет хорошо документированные и лёгкие в использовании средства решения проблем, возникающих при создании приложений корпоративного масштаба.

Между тем, особенности ядра **Spring Framework** применимы в любом Java-приложении, и существует множество расширений и усовершенствований для построения веб-приложений на Java Enterprise платформе. По этим причинам Spring приобрёл большую популярность и признаётся разработчиками как стратегически важный фреймворк.

Spring Framework обеспечивает решения многих задач, с которыми сталкиваются Java-разработчики и организации, которые хотят создать информационную систему, основанную на платформе Java. Из-за широкой функциональности трудно определить наиболее значимые структурные элементы, из которых он состоит. **Spring Framework** не всецело связан с платформой Java Enterprise, несмотря на его масштабную интеграцию с ней, что является важной причиной его популярности.

Spring Framework, вероятно, наиболее известен как источник расширений (features), нужных для эффективной разработки сложных бизнес-приложений вне тяжеловесных программных моделей, которые исторически были доминирующими в промышленности. Ещё одно его достоинство в том, что он ввел ранее неиспользуемые функциональные возможности в сегодняшние господствующие методы разработки, даже вне платформы Java.

Этот фреймворк предлагает последовательную модель и делает её применимой к большинству типов приложений, которые уже созданы на основе платформы Java. Считается, что **Spring Framework** реализует модель разработки, основанную на лучших стандартах индустрии, и делает её доступной во многих областях Java.

Модули Spring Framework

Spring Framework может быть рассмотрен как коллекция меньших фреймворков или фреймворков во фреймворке. Большинство этих фреймворков может работать независимо друг от друга, однако они обеспечивают большую функциональность при совместном их использовании. Эти фреймворки делятся на структурные элементы типовых комплексных приложений:

- **Inversion of Control-контейнер**: конфигурирование компонентов приложений и управление жизненным циклом Java-объектов.
- **Фреймворк аспектно-ориентированного программирования**: работает с функциональностью, которая не может быть реализована возможностями **объектно-ориентированного программирования** на Java без потерь.
- **Фреймворк доступа к данным**: работает с **системами управления реляционными базами данных** на Java-платформе, используя **JDBC**- и **ORM**-средства и обеспечивая решения задач, которые повторяются в большом числе Java-based environments.
- **Фреймворк управления транзакциями**: координация различных **API** управления транзакциями и инструментарий настраиваемого управления транзакциями для объектов Java.
- **Фреймворк MVC**: каркас, основанный на **HTTP** и **сервлетах**, предоставляющий множество возможностей для расширения и настройки (*customization*).
- **Фреймворк удалённого доступа**: конфигурируемая передача Java-объектов через сеть в стиле **RPC**, поддерживающая **RMI**, **CORBA**, **HTTP**-based протоколы, включая **web-сервисы (SOAP)**.
- **Фреймворк аутентификации и авторизации**: конфигурируемый инструментарий процессов аутентификации и авторизации, поддерживающий много популярных и ставших индустриальными стандартами протоколов, инструментов, практик через дочерний проект **Spring Security** (ранее известный как **Acegi**).
- **Фреймворк удалённого управления**: конфигурируемое представление и управление Java-объектами для локальной или удалённой конфигурации с помощью **JMX**.
- **Фреймворк работы с сообщениями**: конфигурируемая регистрация объектов-слушателей сообщений для прозрачной обработки сообщений из **очереди сообщений** с помощью **JMS**, улучшенная отправка сообщений по стандарту JMS API.
- **Тестирование**: каркас, поддерживающий классы для написания модульных и интеграционных тестов.

Архитектура

Архитектура Spring представлена следующей схемой

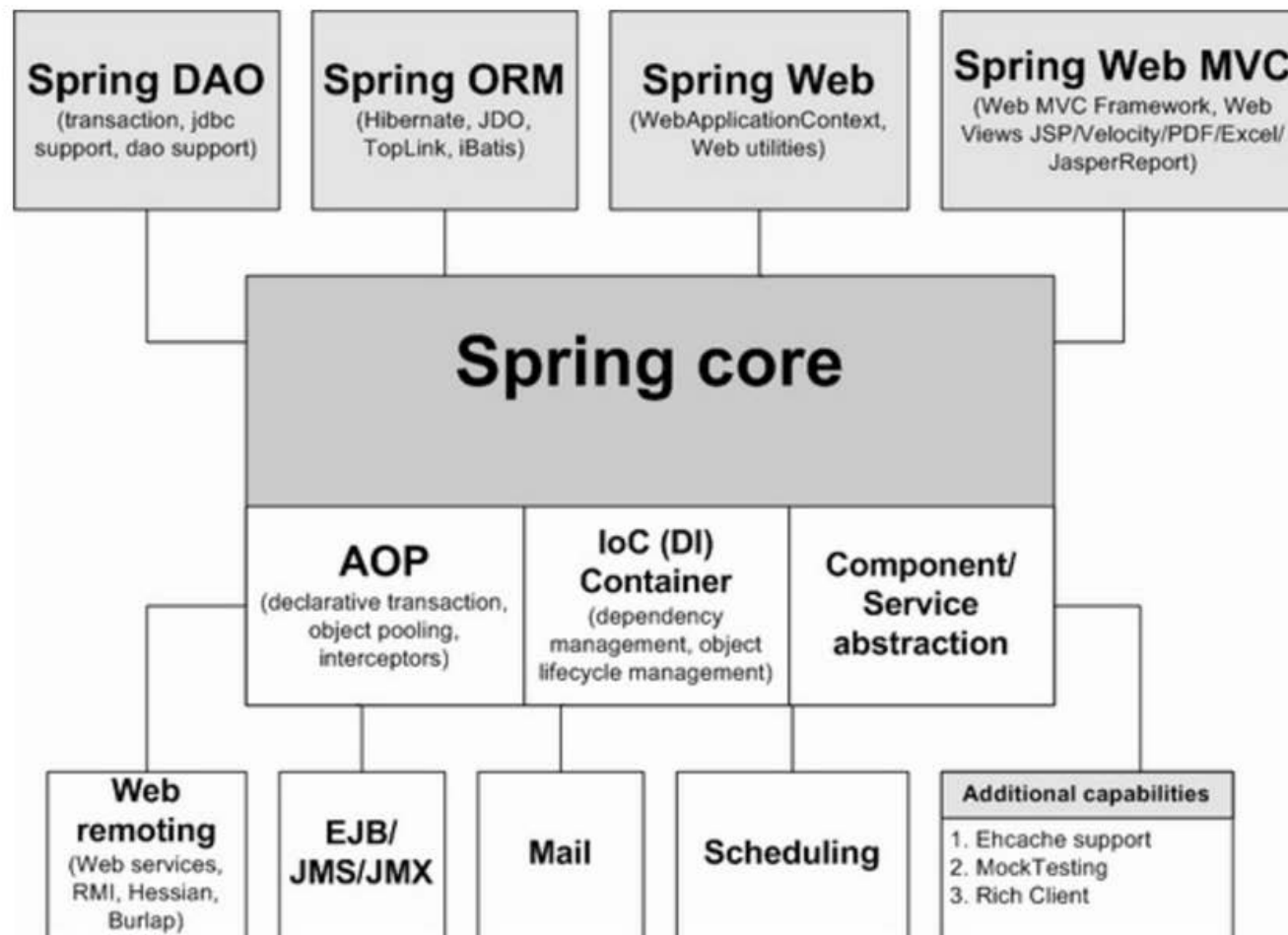


Схема Spring

Inversion of Control

Центральной частью **Spring Framework** является контейнер **Inversion of Control**, который предоставляет средства конфигурирования и управления объектами Java с помощью отражения. Контейнер отвечает за управление жизненным циклом объекта: создание объектов, вызов методов инициализации и конфигурирование объектов путём связывания их между собой.

Объекты, создаваемые контейнером, также называются управляемыми объектами (beans). Обычно конфигурирование контейнера осуществляется путём загрузки XML-файлов, содержащих определение bean'ов и предоставляющих информацию, необходимую для создания bean'ов.

Объекты могут быть получены одним из двух способов:

- **Поиск зависимости** — шаблон проектирования, в котором вызывающий объект запрашивает у объекта-контейнера экземпляр объекта с определённым именем или определённого типа.
- **Внедрение зависимости** — шаблон проектирования, в котором контейнер передает экземпляры объектов по их имени другим объектам с помощью **конструктора**, свойства или **фабричного метода**.

Основные преимущества IoC контейнеров:

1. управление зависимостями
2. упрощение повторного использования классов или компонентов
3. упрощение unit-тестирования
4. более "чистый" код (Классы больше не занимаются инициализацией вспомогательных объектов. Не стоит, конечно "перегибать палку", управляя созданием абсолютно всех объектов через IoC. В IoC контейнер лучше всего выносить те интерфейсы, реализация которых может быть изменена в текущем проекте или в будущих проектах.)

Введение в Spring MVC

Модель-представление-контроллер

Spring имеет собственную MVC-платформу веб-приложений, которая не была первоначально запланирована. Разработчики Spring решили написать её как реакцию на то, что они восприняли как неудачность конструкции (тогда) популярного Apache Struts, а также других доступных веб-фреймворков. В частности, по их мнению, было недостаточным разделение между слоями представления и обработки запросов, а также между слоем обработки запросов и моделью.

Класс `DispatcherServlet` является основным контроллером фреймворка и отвечает за делегирование управления различным интерфейсам, на всех этапах выполнения HTTP-запроса. Об этих интерфейсах следует сказать более подробно.

Как и Struts, Spring MVC является фреймворком, ориентированным на запросы. В нем определены стратегические интерфейсы для всех функций современной запросно-ориентированной системы. Цель каждого интерфейса — быть простым и ясным, чтобы пользователям было легко его заново имплементировать, если они того пожелают. MVC прокладывает путь к более чистому front-end-коду. Все интерфейсы тесно связаны с Servlet API. Эта связь рассматривается некоторыми как неспособность разработчиков Spring предложить для веб-приложений абстракцию более высокого уровня. Однако эта связь оставляет особенности Servlet API доступными для разработчиков, облегчая все же работу с ним. Наиболее важные интерфейсы, определенные Spring MVC, перечислены ниже:

- **HandlerMapping** : выбор класса и его метода, которые должны обработать данный входящий запрос на основе любого внутреннего или внешнего для этого запроса атрибута или состояния.
- **HandlerAdapter** : вызов и выполнение выбранного метода обработки входящего запроса.
- **Controller** : включен между Моделью (Model) и Представлением (View). Управляет процессом преобразования входящих запросов в адекватные ответы. Действует как ворота, направляющие всю поступающую информацию. Переключает поток информации из модели в представление и обратно.
- **View** : ответственно за возвращение ответа клиенту в виде текстов и изображений. Некоторые запросы могут идти прямо во View, не заходя в Model; другие проходят через все три слоя.
- **ViewResolver** : выбор, какое именно View должно быть показано клиенту.
- **HandlerInterceptor** : перехват входящих запросов. Сопоставим, но не эквивалентен сервлет-фильтрам (использование не является обязательным и не контролируется DispatcherServlet-ом).
- **LocaleResolver** : получение и, возможно, сохранение локальных настроек (язык, страна, часовой пояс) пользователя.
- **MultipartResolver** : обеспечивает Upload — загрузку на сервер локальных файлов клиента.

Spring MVC предоставляет разработчику следующие возможности:

- . Ясное и прозрачное разделение между слоями в MVC и запросах.
- . Стратегия интерфейсов — каждый интерфейс делает только свою часть работы.
- . Интерфейс всегда может быть заменен альтернативной реализацией.
- . Интерфейсы тесно связаны с Servlet API.
- . Высокий уровень абстракции для веб-приложений.

В веб-приложениях можно использовать различные части Spring Framework, а не только Spring MVC.

Практика

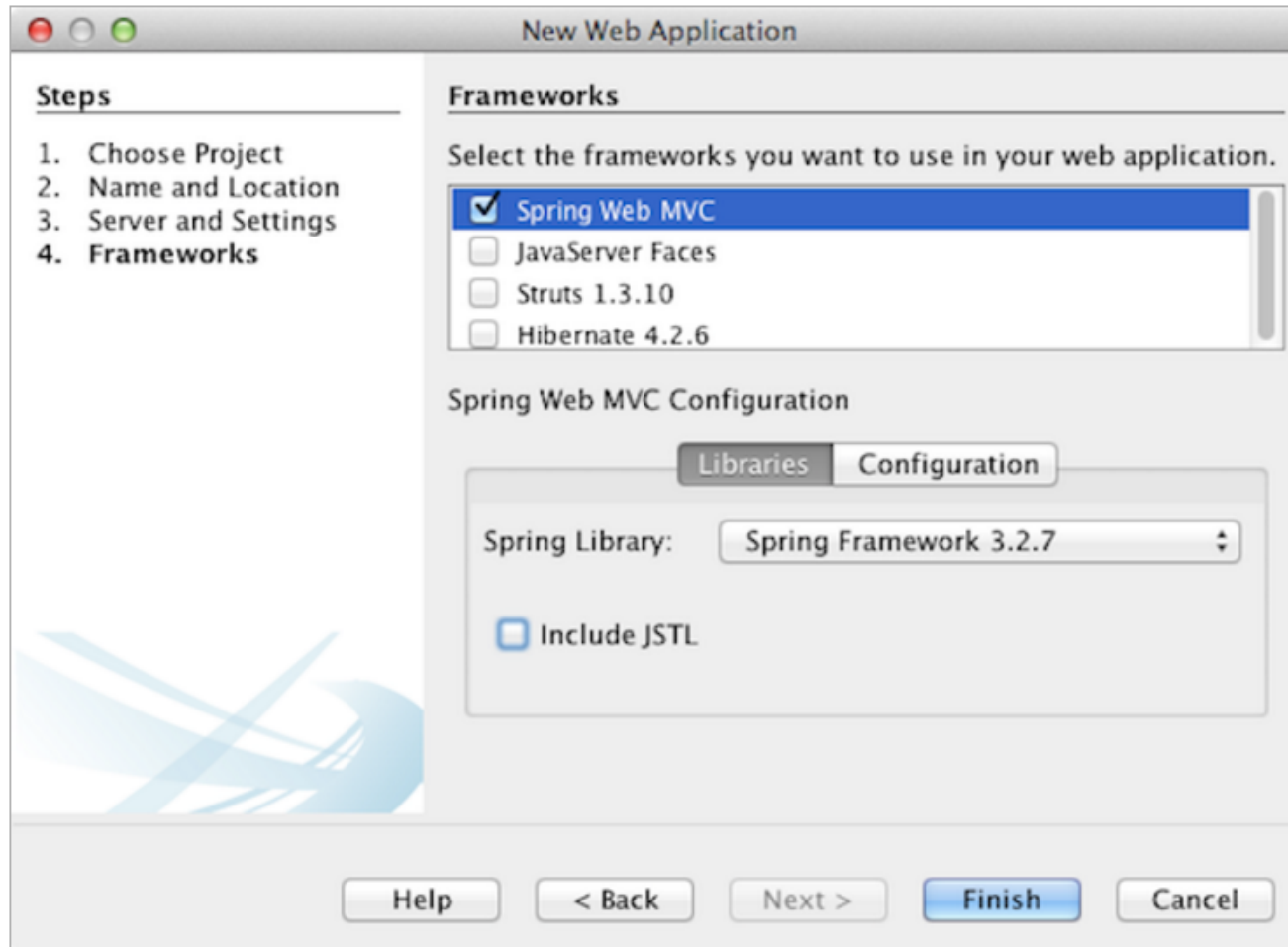
Создание схемы проекта с поддержкой веб-модели MVC Spring

Начните с создания проекта веб-приложения с поддержкой платформы Spring.

1. Выберите New Project ("Создать проект") из меню File ("Файл") среды IDE. Выберите категорию "Java Web", затем выберите проект "Веб-приложение". Нажмите кнопку "Далее".
2. В поле "Имя проекта" введите HelloSpring. Нажмите кнопку "Далее".
3. На третьем экране "Сервер и параметры настройки" отключите параметр "Enable Contexts and Dependency Injection", поскольку в данном учебном курсе не используется спецификация JSR-299.
4. Убедитесь, что в разворачивающемся списке 'Сервер' выбран сервер GlassFish. Нажмите кнопку "Далее".
Версия Java EE зависит от версии выбранного сервера. Если выбран сервер GlassFish Server 4.0, в качестве версии Java EE по умолчанию указывается Java EE 7 Web.

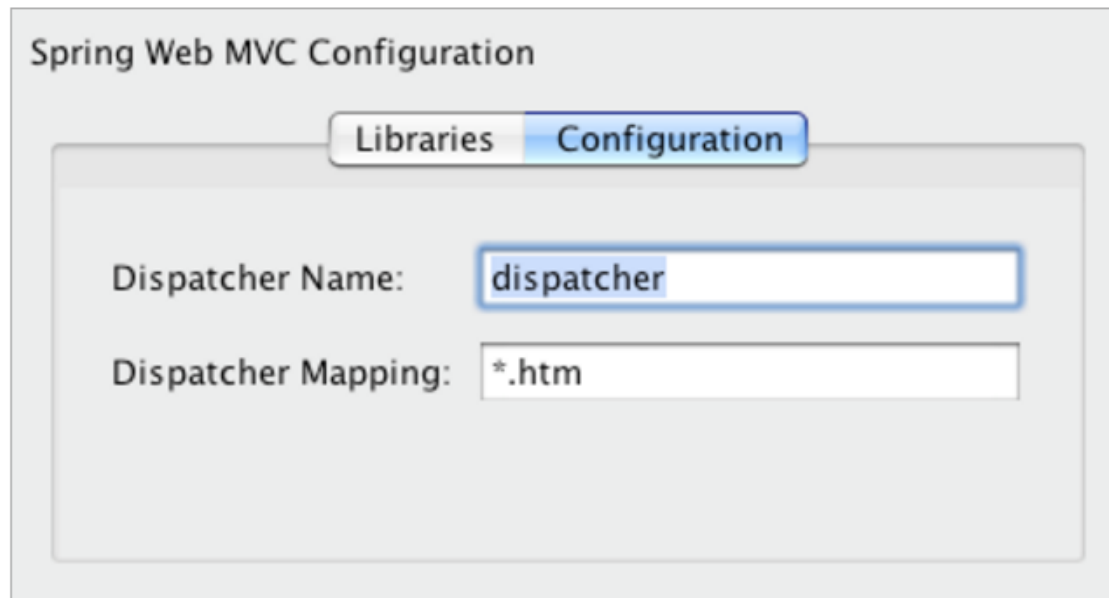
5. На четвертом экране мастера на панели "Frameworks" выберите "Spring Web MVC".

6. Выберите **Spring Framework 3.x** в списке "Библиотека Spring".



⚠ Обратите внимание, что IDE позволяет добавить библиотеку Spring 4.x в проект, но в данном учебном курсе используется компонент SimpleFormController, не поддерживаемый на платформе Spring 4.x. Кроме того, если выбран вариант Spring Web MVC, следует помнить, что во время создания проекта в путь к классу по умолчанию добавляется библиотека JSTL (JavaServer Pages Standard Tag Library). Отключите этот параметр (как показано на снимке экрана), поскольку в этом учебном курсе не требуется JSTL.

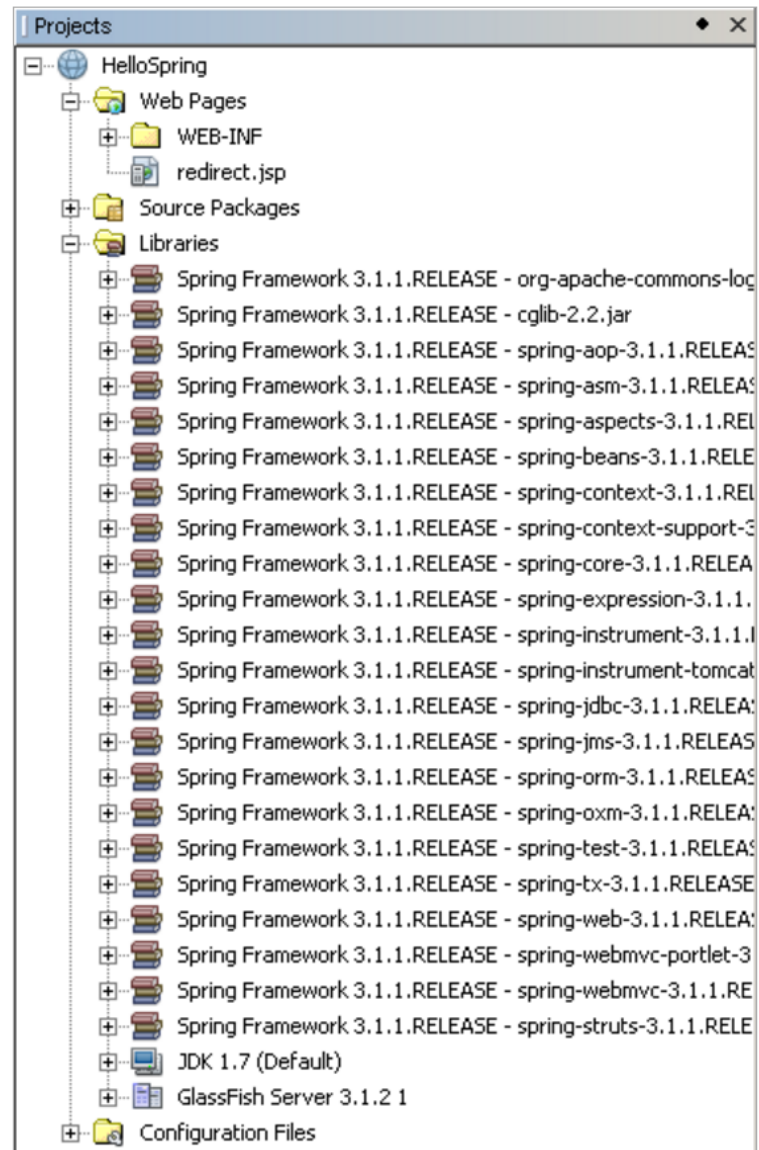
7. Выберите вкладку "Настройка" и обратите внимание, что в мастере можно настроить имя и отображение сервлета обработчика Spring.



The image shows a dialog box titled "Spring Web MVC Configuration". It has two tabs: "Libraries" and "Configuration", with "Configuration" being the active tab. Inside the "Configuration" tab, there are two text input fields. The first is labeled "Dispatcher Name:" and contains the text "dispatcher". The second is labeled "Dispatcher Mapping:" and contains the text "*.htm".


8. Нажмите кнопку "Завершить". В результате создается проект для всего приложения, в т.ч. все метаданные, а также сценарий сборки проекта Ant, с которым можно ознакомиться в окне "Файлы". Структуру шаблона можно просмотреть в окне "Проекты". Также следует отметить, что по умолчанию в редакторе среды IDE открываются четыре файла: dispatcher-servlet.xml, applicationContext.xml, redirect.jsp и index.jsp.

9. В окне 'Проекты' разверните узел 'Библиотеки' нового проекта и обратите внимание, что файлы JAR Spring включены в путь к классу проекта.

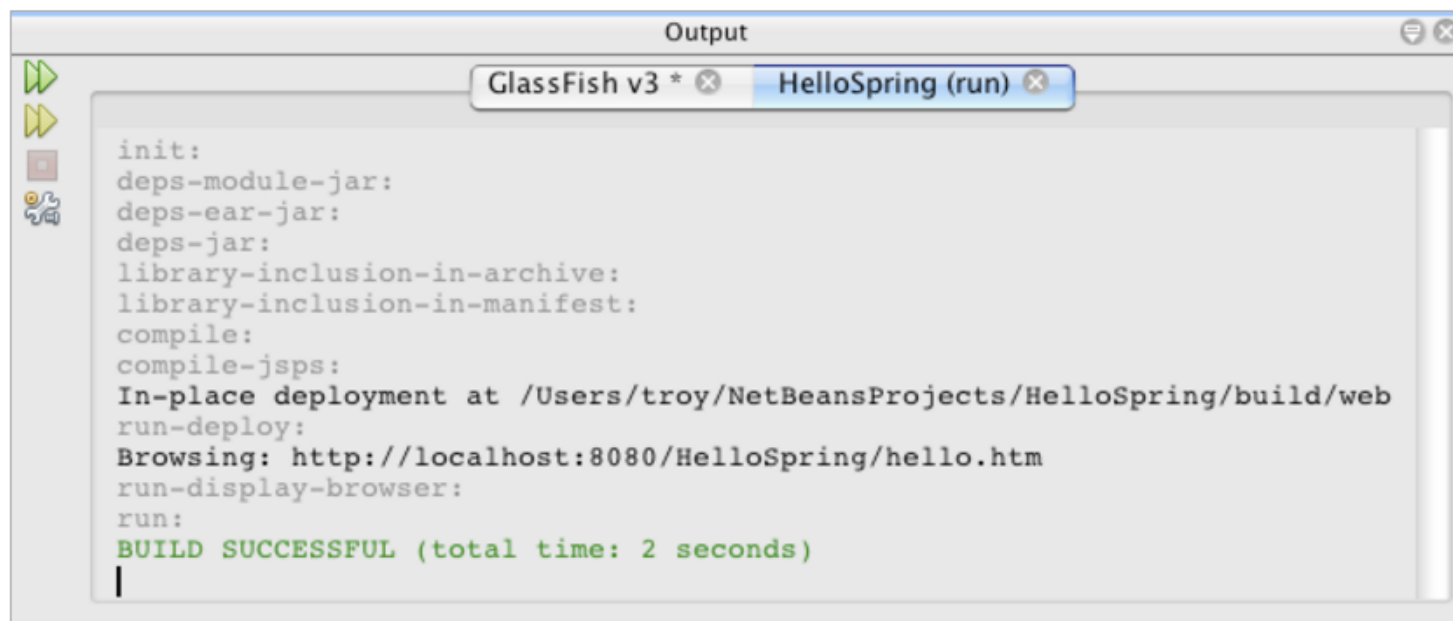


Выполнение схемы проекта

Перед изменением файлов проекта следует попытаться запустить созданный проект в среде IDE:

Нажмите кнопку 'Запустить проект' () на главной панели инструментов IDE. В среде IDE автоматически запускается сервер GlassFish, если он не был запущен, проект компилируется и разворачивается на сервере.

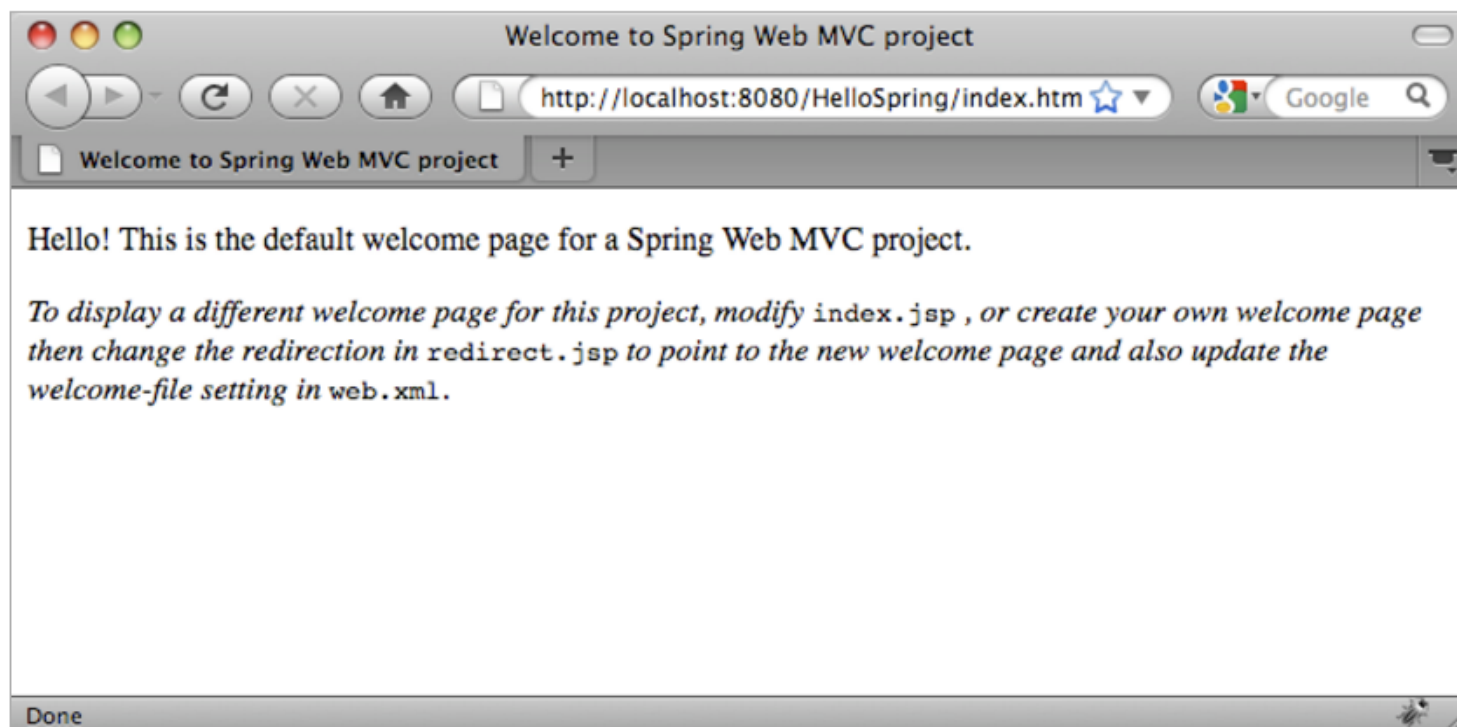
Обратите внимание на данные, отображаемые в окне "Вывод" среды IDE. В конце этих данных отображается сообщение BUILD SUCCESSFUL.



The screenshot shows the 'Output' window of an IDE. It contains two tabs: 'GlassFish v3 *' and 'HelloSpring (run)'. The 'HelloSpring (run)' tab is active and displays the following text:

```
init:
deps-module-jar:
deps-ear-jar:
deps-jar:
library-inclusion-in-archive:
library-inclusion-in-manifest:
compile:
compile-jsps:
In-place deployment at /Users/troy/NetBeansProjects/HelloSpring/build/web
run-deploy:
Browsing: http://localhost:8080/HelloSpring/hello.htm
run-display-browser:
run:
BUILD SUCCESSFUL (total time: 2 seconds)
```

В среде IDE запускается браузер по умолчанию, и отображается содержимое представления страницы приветствия (`/WEB-INF/jsp/index.jsp`).



TIP При выполнении проекта в среде IDE он компилируется и разворачивается на сервере, а затем открывается в браузере по умолчанию. Более того, среда IDE предоставляет возможность "Развертывание при сохранении", которая включена по умолчанию для веб-проектов. При сохранении файлов в редакторе проект автоматически компилируется и разворачивается на сервере. Для просмотра изменений достаточно просто обновить страницу в браузере.

Ключ к пониманию произошедших событий представлен в дескрипторе развертывания проекта (web.xml). Чтобы открыть этот файл в редакторе исходного кода, щелкните правой кнопкой мыши узел WEB-INF > web.xml в окне 'Проекты' и выберите 'Правка'. Точка входа для приложения по умолчанию — redirect.jsp:

```
<welcome-file-list>
  <welcome-file>redirect.jsp</welcome-file>
</welcome-file-list>
```

В файле redirect.jsp содержится оператор перенаправления, направляющий все запросы в index.htm:

```
<% response.sendRedirect("index.htm"); %>
```

Обратите внимание, что в дескрипторе развертывания все шаблоны URL-адресов, соответствующие выражению *.htm отображаются на DispatcherServlet Spring.

```
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>2</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```



Полностью определенное имя сервлета диспетчера сервлета

`org.springframework.web.servlet.DispatcherServlet`. Это класс из библиотеки Spring, которая была добавлена в путь к классам проекта при его создании. Чтобы проверить это, разверните узел "Библиотеки" в окне "Проекты". Найдите файл `spring-webmvc-3.1.1.RELEASE.jar`, затем разверните его и найдите `org.springframework.web.servlet > DispatcherServlet`.

`DispatcherServlet` обрабатывает входящие запросы на основе параметров настройки из файла `dispatcher-servlet.xml`. Откройте файл `dispatcher-servlet.xml`, щелкнув его вкладку в редакторе. Обратите внимание на следующий код.

```
<bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="/index.htm">indexController</prop>
        </props>
    </property>
</bean>

<bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver"
    p:prefix="/WEB-INF/jsp/"
    p:suffix=".jsp" />

<bean name="indexController"
    class="org.springframework.web.servlet.mvc.ParameterizableViewController"
    p:viewName="index" />
```

В этом файле определены три компонента: `indexController`, `viewResolver` и `urlMapping`. Когда `DispatcherServlet` получает запрос, соответствующий выражению `*.htm`, например, `index.htm`, выполняется поиск контроллера внутри `urlMapping`, способного обработать этот запрос. Выше можно заметить, что существует свойство `mappings`, связывающее `/index.htm` с `indexController`.

Среда выполнения выполняет поиск определения компонента `indexController`, предоставляемого схемой проекта. Обратите внимание, что класс `indexController` расширяет класс `ParameterizableViewController`. Это еще один класс инфраструктуры Spring, который просто возвращает представление. Также обратите внимание, что `p:viewName="index"` указывает логическое имя представления, которое разрешается с помощью `viewResolver` путем добавления `/WEB-INF/jsp/` слева и добавления `.jsp` справа. Это позволяет среде выполнения найти файл в папке приложения и предоставить в ответ представление страницы приветствия (`/WEB-INF/jsp/index.jsp`).

Обзор приложения


Создаваемое приложение состоит из двух страниц JSP (которые называются представлениями в терминологии MVC). Первое представление содержит форму HTML с полем, в которое пользователь вводит свое имя. Второе представление — страница, на которой отображается приветственное сообщение с именем пользователя.

Представления управляются контроллером, который получает запросы к приложению и принимает решение, какие представления вернуть. Также он передает в представления информацию, которая требуется для их отображения (она называется моделью). Контроллер этого приложения называется `HelloController`.

В сложном веб-приложении бизнес-логика не размещается непосредственно в контроллере. Вместо этого контроллером используется другой объект — служба — при каждом обращении к бизнес-логике. В этом приложении бизнес-логика ограничена обработкой приветственного сообщения, и для этой цели создается служба `HelloService`.

Реализация службы

Теперь, после проверки правильности настроек среды, можно начать расширение схемы проекта в соответствии с имеющимися требованиями. Начните с создания класса `HelloService`.

1. Нажмите кнопку 'Создать файл' () на панели инструментов IDE. (В качестве альтернативы нажмите Ctrl-N; ⌘-N в Mac.)
2. Выберите категорию **Java**, затем **Класс Java** и нажмите кнопку "Далее".
3. В мастере создания класса Java введите имя класса **HelloService**, затем введите имя пакета **service**, чтобы создать для класса новый пакет.
4. Нажмите кнопку "Завершить". В среде IDE создается и открывается в редакторе новый класс.

Класс `HelloService` предоставляет очень простую службу. Он принимает в качестве параметра имя и подготавливает и возвращает строку `String`, содержащую это имя. Создайте в редакторе следующий метод `sayHello()` для этого класса (изменения помечены **полужирным шрифтом**).

```
public class HelloService {  
  
    public static String sayHello(String name) {  
        return "Hello " + name + "!";  
    }  
}
```

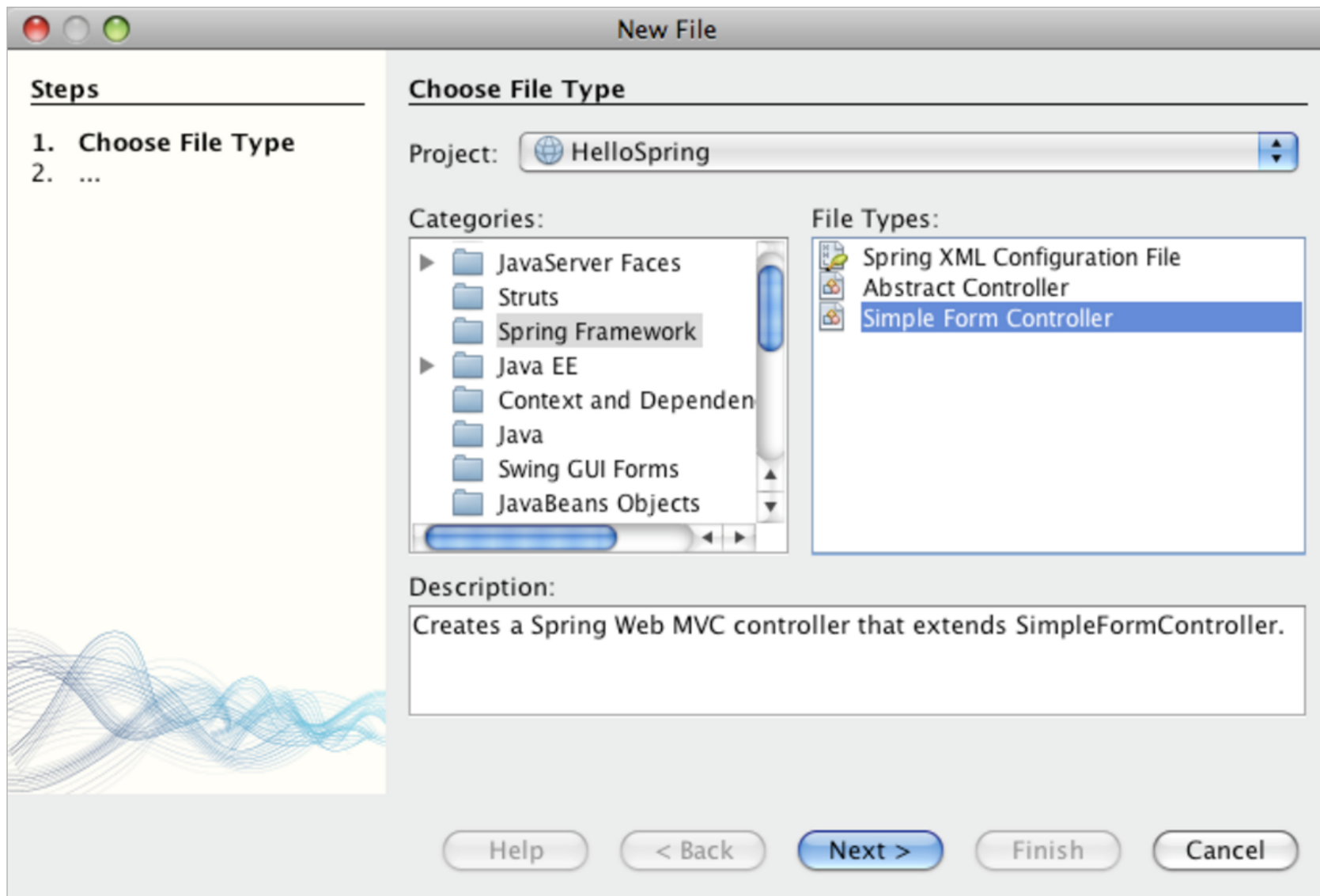
Реализация контроллера и модели


Для обработки пользовательских данных и выбора представления для возврата можно использовать *SimpleFormController*.



Примечание: SimpleFormController устарел в Spring 3.x. Он используется в этом учебном руководстве для выполнения задач демонстрации. Однако следует использовать контроллер с аннотациями вместо файлов XML.

1. Откройте мастер создания файлов, нажав Ctrl-N. Выберите категорию **Spring Framework** и тип файла **Simple Form Controller**.



 IDE NetBeans обеспечивает шаблоны для различных артефактов Spring, включая файл конфигурации Spring XML, `AbstractController` и `SimpleFormController`.

2. Нажмите кнопку "Далее".

3. Назовите класс **HelloController** и создайте для него новый пакет, для этого введите в поле "Package" текст **controller**. Нажмите кнопку "Завершить". В среде IDE создается и открывается в редакторе новый класс.

4. Укажите свойства контроллера, раскомментировав методы установки, отображаемые шаблоном класса по умолчанию.

```
15 public HelloController() {  
16     //Initialize controller properties here or  
17     //in the Web Application Context  
18  
19     //setCommandClass(MyCommand.class);  
20     //setCommandName("MyCommandName");  
21     //setSuccessView("successView");  
22     //setFormView("formView");  
23 }
```

5. Внесите следующие изменения (показаны **полужирным шрифтом**).

```
public HelloController() {  
    setCommandClass(Name.class);  
    setCommandName("name");  
    setSuccessView("helloView");  
    setFormView("nameView");  
}
```

Параметр FormView позволяет задать имя представления, используемого для отображения формы. Это страница, содержащая текстовое поле для ввода имени пользователя. Параметр SuccessView аналогичным образом позволяет задать имя представления, отображаемого при успешной передаче данных. Параметр CommandName задает имя команды в модели. В данном случае, команда — это объект формы со связанными параметрами запроса. Параметр CommandClass определяет имя класса команды. Экземпляр этого класса заполняется и проверяется на допустимость при каждом запросе.

Обратите внимание, что для `Name` в методе `setCommandClass()` отображается ошибка:

```
15 public HelloController() {  
16     //Initialize controller properties here or  
17     //in the Web Application Context  
18  
19     setCommandClass(Name.class);  
20     setCommandName("name");  
21     setSuccessView("helloView");  
22     setFormView("nameView");  
23 }
```

Требуется создать класс `Name` в качестве простого компонента, хранящего данные каждого запроса.

6. В окне 'Проекты', щелкните правой кнопкой мыши узел проекта и выберите 'Создать' > 'Класс Java'. Отображается мастер создания класса Java.

7. Введите имя класса **Name** и выберите в выпадающем списке пакет **controller**.

8. Нажмите кнопку "Завершить". Класс *Name* будет создан и открыт в редакторе.

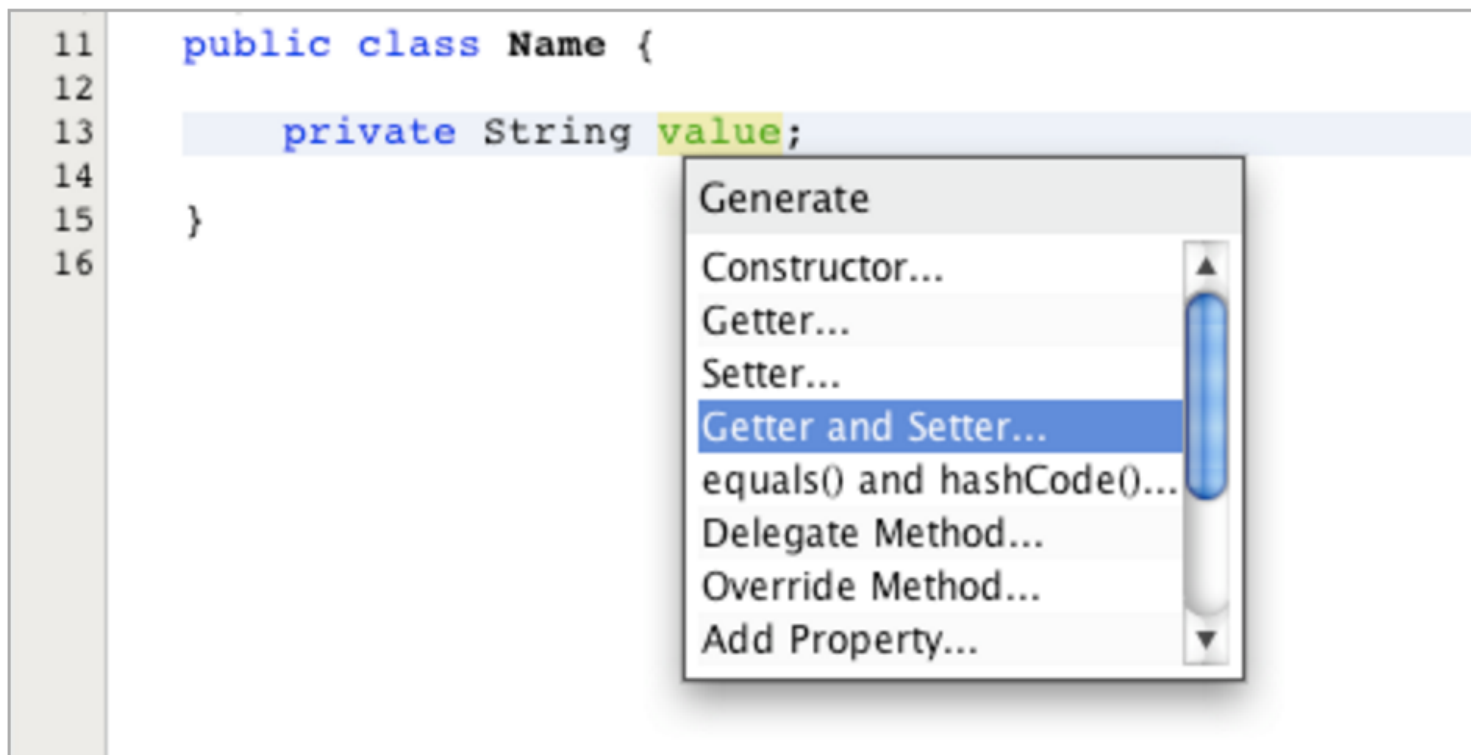
9. В классе Name создайте поле value, затем создайте методы доступа (т.е. методы получения и установки) для этого поля. Сначала объявите поле value:

```
public class Name {  
  
    private String value;  
  
}
```



Чтобы быстро ввести "private", можно ввести "pr" и затем нажать клавишу TAB. Автоматически добавляется модификатор доступа "private". Это пример использования шаблонов кода редактора. Полный список шаблонов кода можно просмотреть, выбрав в меню "Справка" пункт "Таблица сочетаний клавиш".

В среде IDE предусмотрена возможность автоматического создания методов доступа. В редакторе щелкните правой кнопкой мыши в `value` и выберите 'Вставить код' (или нажмите Alt-Insert; Ctrl-I в Mac). Во всплывающем меню выберите пункт "Методы получения и установки".



10. В диалоговом окне выберите параметр `value : String` и нажмите кнопку "OK". Методы `getValue()` и `setValue()` добавляются к классу `Name`:

```
public String getValue() {  
    return value;  
}  
  
public void setValue(String value) {  
    this.value = value;  
}
```

11. Нажмите сочетание клавиш `CTRL+TAB` и выберите `HelloController`, чтобы переключиться обратно к классу `HelloController`. Обратите внимание, что значок ошибки исчез, поскольку класс `Name` теперь существует.

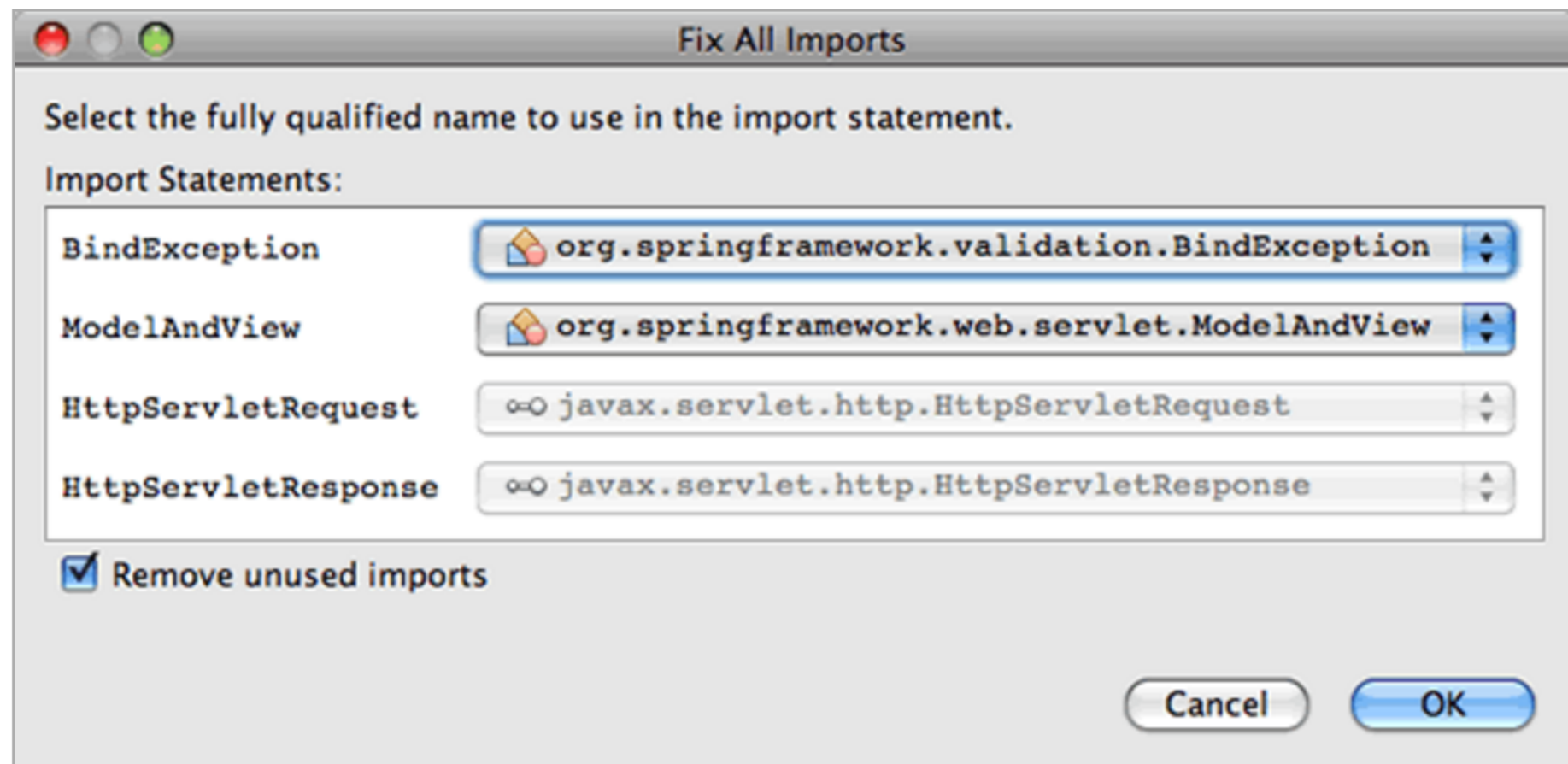
12. Удалите метод `doSubmitAction()` и раскомментируйте метод `onSubmit()`. Метод `onSubmit()` позволяет создать собственный `ModelAndView`, что требуется здесь. Внесите следующие изменения:


```
@Override
protected ModelAndView onSubmit(
    HttpServletRequest request,
    HttpServletResponse response,
    Object command,
    BindException errors) throws Exception {

    Name name = (Name) command;
    ModelAndView mv = new ModelAndView(getSuccessView());
    mv.addObject("helloMessage", helloService.sayHello(name.getValue()));
    return mv;
}
```

Как указано выше, `command` приводится к объекту `Name`. Создается экземпляр `ModelAndView` и с помощью метода получения в `SimpleFormController` создается представление. После этого модель заполняется данными. Единственный элемент модели в данном случае — приветственное сообщение, получаемое из ранее созданной службы `HelloService`. Для добавления к модели приветственного сообщения можно использовать метод `addObject()` под именем `helloMessage`.

13. Исправьте ошибки импорта, щелкнув правой кнопкой мыши в редакторе и выбрав 'Исправить ошибки'.



 **Примечание.** Убедитесь, что `org.springframework.validation.BindException` и `org.springframework.web.servlet.ModelAndView` выбраны в диалоговом окне 'Исправить все выражения импорта'.

14. Нажмите кнопку "OK". В начало файла добавляется следующий оператор импорта:

```
import org.springframework.web.servlet.ModelAndView;
```

Как указано в документации API, этот класс "представляет модель и представление, возвращаемые контроллером для разрешения `DispatcherServlet`. Представление может принимать форму имени представления `String`, которое должно быть разрешено объектом `ViewResolver`; в качестве альтернативы объект `View` может быть указан непосредственно. Модель — это объект `Map`, что позволяет использовать несколько объектов, выбираемых по имени."

Обратите внимание, что на данном этапе исправлены не все ошибки, поскольку класс по-прежнему не может определить класс `HelloService` и использовать его метод `sayHello()`.

15. Внутри `HelloController` объявите частное поле `HelloService`:

```
private HelloService helloService;
```

Затем создайте для поля общедоступный метод установки:

```
public void setHelloService(HelloService helloService) {  
    this.helloService = helloService;  
}
```


Наконец, щелкните правой кнопкой мыши в редакторе и выберите 'Исправить выражения импорта' (Ctrl-Shift-I; ⌘-Shift-I в Mac). В начало файла добавляется следующий оператор импорта:

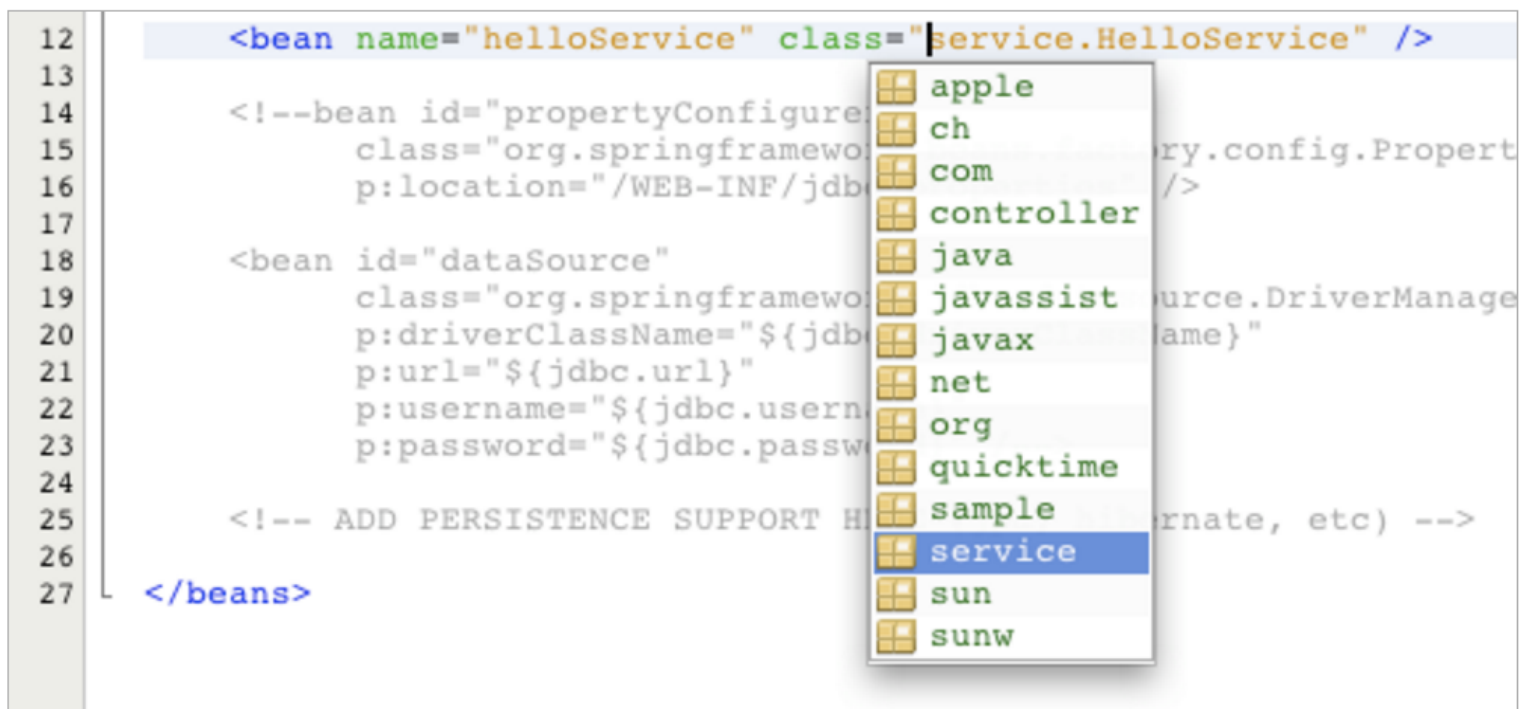
```
import service.HelloService;
```

Все ошибки исправлены.

16. Зарегистрируйте `HelloService` в файле `applicationContext.xml`. Откройте в редакторе файл `applicationContext.xml` и введите следующее определение компонента:

```
<bean name="helloService" class="service.HelloService" />
```

 В состав поддержки Spring в среде IDE входит автозавершение кода в файлах настройки XML для классов Java и ссылок на компоненты. Для вызова автозавершения кода нажмите сочетание клавиш CTRL+ПРОБЕЛ в редакторе:



17. Зарегистрируйте `HelloController` в файле `dispatcher-servlet.xml`. Откройте в редакторе файл `dispatcher-servlet.xml` и введите следующее определение компонента:

```
<bean class="controller.HelloController" p:helloService-ref="helloService"/>
```

Реализация представлений

Для реализации представлений в проекте требуется создать две страницы JSP. Первая из них, `nameView.jsp`, служит страницей приветствия и позволяет пользователю ввести имя. На второй, `helloView.jsp`, отображается приветственное сообщение с введенным именем. Сначала создайте `helloView.jsp`.

1. В окне 'Проекты' щелкните правой кнопкой мыши узел `WEB-INF > jsp` и выберите 'Создать' > JSP. Откроется мастер "New JSP File". Введите имя файла **helloView**.
2. Нажмите кнопку "Завершить". Страница JSP создается в папке `jsp` и открывается в редакторе.
3. В редакторе измените заголовок файла на `Hello` и измените выходное сообщение для получения `helloMessage` объекта `ModelAndView`, созданного в `HelloController`.

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Hello</title>
</head>
<body>
  <h1>${helloMessage}</h1>
</body>
```

4. Тем же способом создайте вторую страницу JSP и назовите ее `nameView`.
5. В редакторе добавьте следующее определение библиотеки тегов Spring к `nameView.jsp`.

```
<%@taglib uri="http://www.springframework.org/tags" prefix="spring" %>
```

При этом импортируется [библиотека тегов Spring](#), содержащая полезные теги для реализации представлений как страниц JSP.

6. Измените содержимое тегов `<title>` и `<h1>` на `Enter Your Name`.

7. После тега `<h1>` введите следующий код:

```
<spring:nestedPath path="name">
  <form action="" method="post">
    Name:
    <spring:bind path="value">
      <input type="text" name="${status.expression}" value="${status.value}">
    </spring:bind>
    <input type="submit" value="OK">
  </form>
</spring:nestedPath>
```

`spring:bind` позволяет привязать свойство компонента. Тег привязки обеспечивает состояние и значение привязки, которые используются в качестве имени и значения поля ввода. Таким образом, при передаче формы платформе Spring будет известен способ извлечения переданного значения. Здесь командный класс (`controller.Name`) имеет свойство `value`, поэтому нужно установить `path` равным `value`.

`spring:nestedPath` позволяет добавить к компоненту слева определенный путь. Поэтому при использовании вместе с `spring:bind` путь к компоненту становится равным `name.value`. Как уже указывалось, имя команды `HelloController` — `name`. Поэтому этот путь ссылается на свойство `value` компонента `name` в контексте страницы.

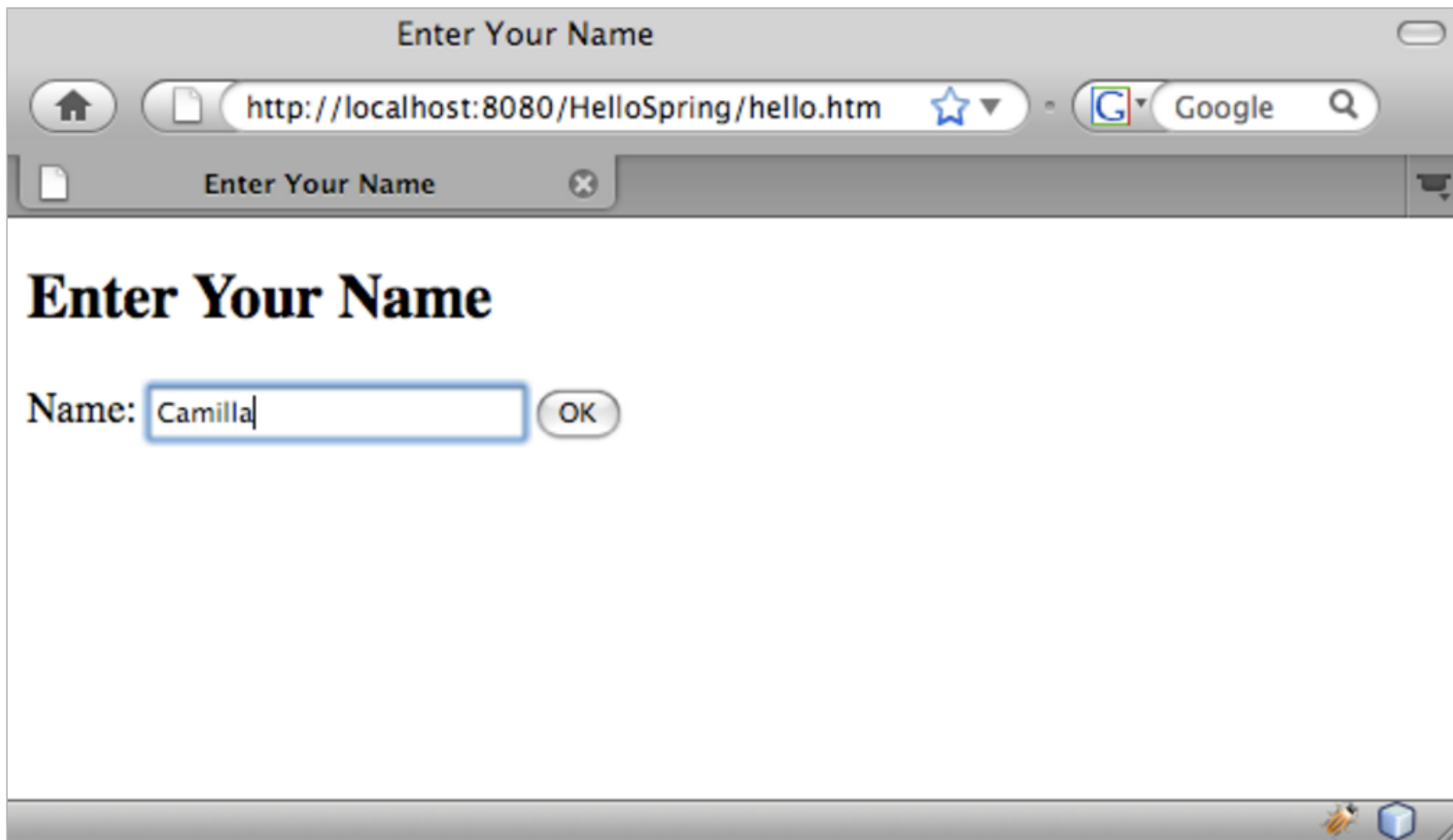
8. Измените относительную точку входа для приложения. В настоящий момент точка входа по-прежнему `index.htm`, что в соответствии с разделом **Выполнение схемы проекта** перенаправляется на `WEB-INF/jsp/index.jsp`. Можно указать точку входа для проекта после его развертывания и запуска. В окне 'Проекты', щелкните правой кнопкой мыши узел проекта и выберите 'Свойства'. Открывается диалоговое окно "Свойства проекта". В области "Категории" выберите "Выполнить". В поле "Относительный URL-адрес" введите `/hello.htm` и нажмите кнопку "OK".

Возникает вопрос: где же располагается отображение `hello.htm` на `HelloController`? Отображение не добавлялось к компоненту `urlMapping`, как в случае страницы приветствия схемы проекта `index.htm`. Такая возможность обеспечивается средствами автоматизации платформы Spring, предоставляемыми следующим определением компонента `dispatcher-servlet.xml`:

```
<bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping"/>
```

Этот компонент отвечает за автоматическое создание отображения URL-адресов для всех зарегистрированных в файле контроллеров. Из полностью определенного имени контроллера (в данном случае `controller.HelloController`) удаляется имя пакета и суффикс `Controller`, и затем результат используется как образец URL-адреса. Таким образом, для `HelloController` создается отображение `hello.htm`. Это средство, однако, не срабатывает для контроллеров, включенных в платформу Spring, например, `ParameterizableViewController`. Для них требуется явное отображение.

9. В окне 'Проекты' щелкните правой кнопкой мыши узел проекта и выберите 'Выполнить'. Проект компилируется, разворачивается и выполняется. Открывается браузер по умолчанию и отображается hello.htm как nameView проекта:



Введите имя в текстовом поле и нажмите ENTER. Отображается `helloView` с приветственным сообщением:

