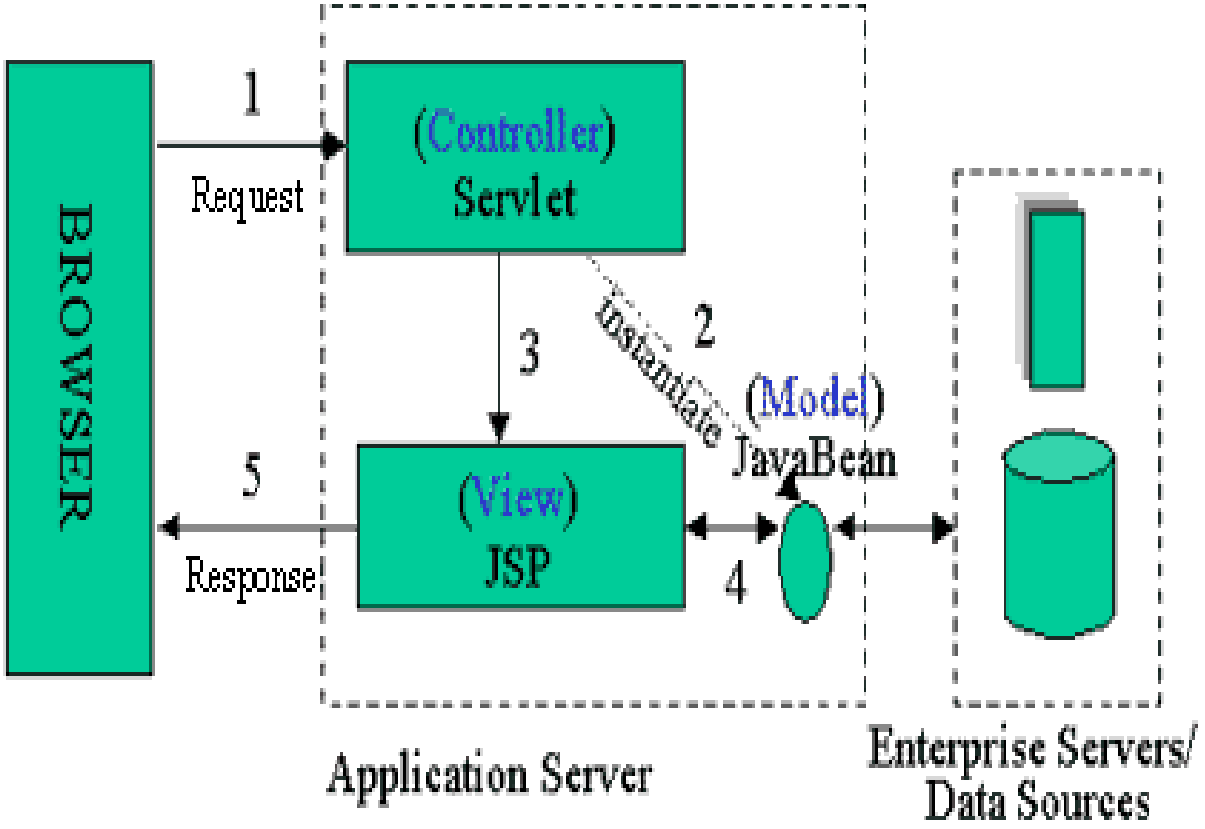
Spring Framework
Module 8 – Spring 3 MVC

Andrey Stukalenko
Last update: March, 2012

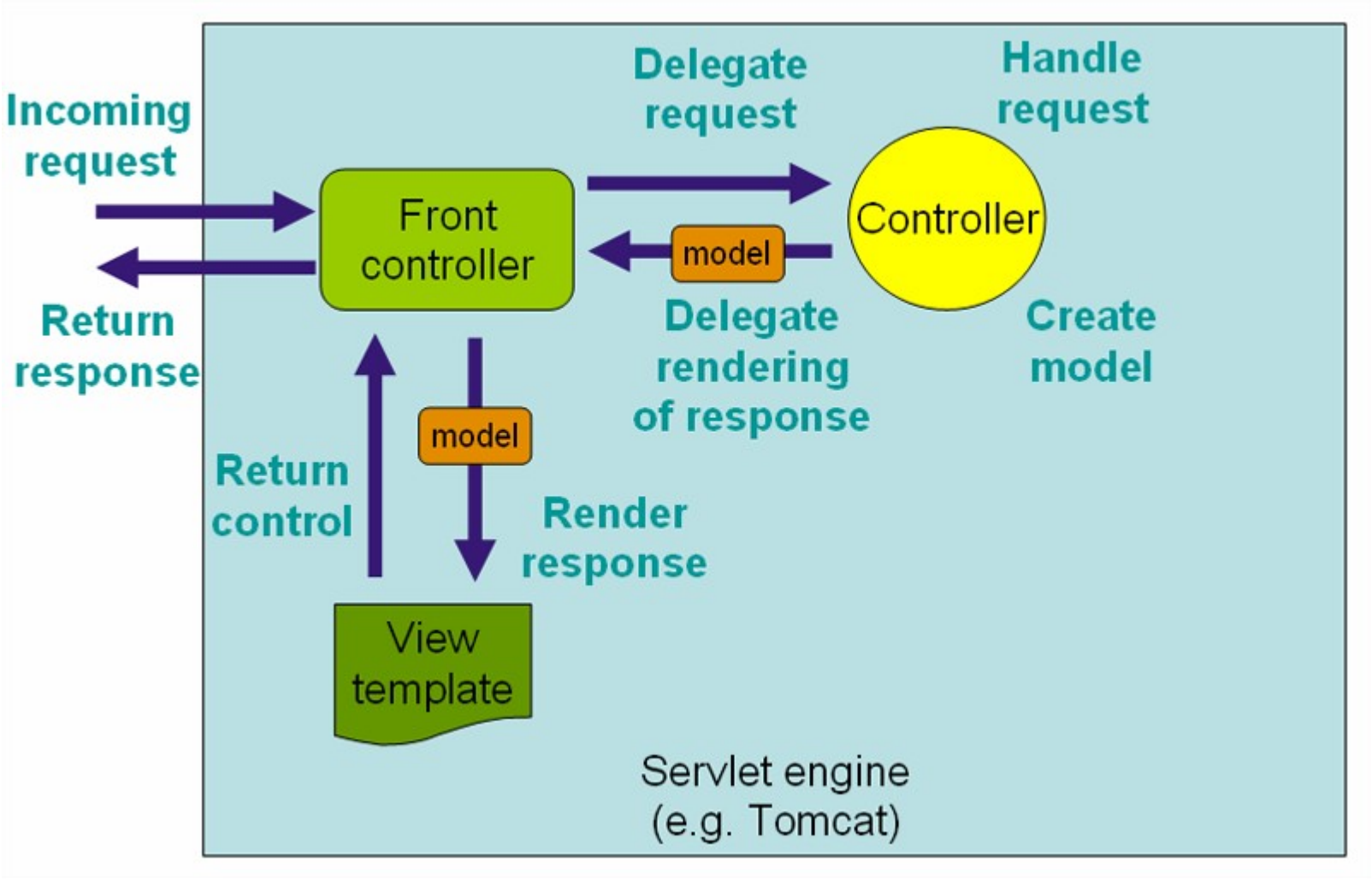
Spirng :: MVC :: Введение

Spring содержит мощный и гибко конфигурируемый MVC веб - фреймворк, но есть и отличия от классической модели:



Spirng :: MVC :: Введение

В Spring реализована более гибкая модель:



Spirng :: MVC :: Сравнение со Struts

- ***Модель MVC у Spring похожа на модель Struts, хотя и не наследуется от модели Struts***. Spring Controller похож на Struts Action в том что он многопоточный сервисный обьект, с единственным экземпляром выполняемым от всех клиентов. Однако, у Spring MVC есть отличительные преимущества перед Struts. Например:
  - Spring предоставляет ***ясное разделение*** между контроллерами, моделями JavaBean и видами.
  - ***Гибкость Spring MVC***. В отличии от Struts, который требует чтобы Action и Form наследовали определенные классы(тем самым не давая вам возможности наследовать ваши собственные классы), Spring MVC полностью основывается на интерфейсах. И более того, любую часть Spring MVC вы можете заменить своей реализацией нужного интерфейса. При этом в Spring предоставлены готовые реализации всех нужных интерфейсов.

Spirng :: MVC :: Сравнение со Struts

LUXOFT TRAINING

- Spring, помимо контроллеров, ***предоставляет "перехватчики" (interceptors)*** с помощью которых легко реализовать логику общую для всех вызовов.
- Spring по-настоящему ***независим от конкретной реализации представления***. Вас не заставляют писать *JSP* если вы не хотите, в замен вы можете использовать *Velocity*, *XSLT* или любой другой подход к отображению. Если вы хотите использовать собственный подход к отображению, например свой язык шаблонов, вы легко можете это сделать реализовав как вам нужно интерфейс **`SpringView`**.
- ***Контроллеры*** Spring ***настраиваются через IoC*** также, как и любые другие объекты. Это делает их легкими в тестировании, и удобными при интеграции их с другими объектами, управляемыми Spring.
- ***Web-слой*** от Spring MVC обычно ***легко протестировать***, благодаря отсутствию обязательного наследования конкретного класса и независимости контроллера от сервлета диспетчеризации.

Spirng :: MVC :: Сравнение со Struts

- ***Web слой становится тонким слоем поверх бизнес объектов***, что является хорошей практикой. Struts и прочие ориентированные в первую очередь на web фреймворки никак не помогают вам при реализации ваших бизнес объектов, а Spring предоставляет комплексный фреймворк для всех слоев вашего приложения.

- Как и в Struts версии 1.1 и выше, в приложении Spring MVC у вас может быть столько управляющих сервлетов сколько вам нужно.

- Следующий пример показывает как простой контроллер Spring может получать доступ к бизнес объектам определенным в том же контексте приложения. Этот контроллер ищет объект **`Order`** и возвращает его результатом метода **`handleRequest`**():

Spirng :: MVC :: Пример использования

```
public class OrderController  implements Controller {
   private OrderRepository repo;
   @Autowired
   public OrderController(OrderRepository orderRepository) {
       repo = orderRepository;
   }
   public ModelAndView handleRequest(
       HttpServletRequest request,
       HttpServletResponse response)
       throws ServletException, IOException
   {
       String id = request.getParameter("id");
       Order order = repo.getOrderById(id);
       return new ModelAndView("orderView", "order", order);
   }
}
```

Spirng :: MVC :: Пример использования

```
@Controller
public class OrderController {
   @Autowired
   private OrderRepository repo;
   public OrderController(OrderRepository orderRepository) {
       repo = orderRepository;
   }

  @RequestMapping("/viewOrder")
  public String viewOrder(int orderId, Model model)
   {
       Order order = repo.getOrderById(id);
       model.put(order);
       return "orderView";
   }
}
```

Spirng :: MVC :: Пример использования

LUXOFT TRAINING

- Spring IoC отделяет данный контроллер от используемой реализации OrderRepository, она может работать на основе JDBC, так же, как и на основе веб-сервиса.

- Интерфейс может быть реализован с помощью:
  - POJO,
  - тестовой заглушки,
  - прокси
  - удаленного объекта.

- Контроллер не содержит ничего, относящегося к поиску ресурса; ничего кроме кода, необходимого для веб-взаимодействия.

Spirng :: MVC :: Пример использования

Spring MVC также предоставляет поддержку привязки данных (data binding), форм, "мастеров" и более сложных подходов.

Биндинг (binding) – связывание, отображение параметров http-запроса на свойства объектов (Java-бинов) и наоборот. Удобный способ взаимодействия с вводимыми в форму пользователем данными.

Резолвер (resolver, mapping) – объект, реализующий стратегию сопоствления чего-либо. (Например, URL-ов запроса – преставлениям)

Spirng :: MVC :: Архитектура

`WebApplicationContext –` приспособленный для работы в Web`е **`ApplicationContext.`**

`DispatcherServlet –` сервлет, служащий для перехвата всех запросов клиента, реализующий паттерн FrontController

Model – **`java.util.Map`**

Хранит данные в виде пары ключ-значение

View – **`interface View`**

Имплементация этого интерфейса отображает данные

Controller - **`interface Controller`**

Имплементации этого интерфейса обрабатывают запросы пользователя

Spirng :: MVC

«Открытый для расширения, закрытый для изменения»

http://objectmentor.com/resources/articles/ocp.pdf

Управляется запросами (request-driven).

«Точкой входа» является `DispatcherServlet`.

Основан на инверсии управления (IoC) – обеспечивает гибкость изменения компонентов.

Может быть использован совместно с любым другим веб-каркасом.

Spirng :: MVC

Чёткое разделение ролей объектов (контроллер, валидатор, объект формы, элемент модели, представление и т.д.).

Модель – это всегда **`java.util.Map`**. Отсюда следует лёгкая интеграция с любой технологией уровня представления.

Повторное использование кода – бизнес-объекты могут быть элементами модели.

Spirng :: MVC :: Пример использования

LUXOFT TRAINING

- Несколько базовых подклассов контроллеров, отвечающих различным потребностям.
- Набор стратегий биндинга и валидации (исключения – уровня приложения, проперти – не только строки).
- Набор стратегий поиска (resolvers) ответственных контроллеров и нужных представлений.

Spirng :: MVC :: WebApplicationContext

LUXOFT TRAINING

- Это расширение **`ApplicationContext`**, имеющее некоторые особенности, необходимые для web-приложений (например, ассоциация с **`ServletContext`**).
- Добавляет три разновидности цикла жизни бинов, имеющих смысл только в веб-контексте (request, scope, global).
- Бины некоторых типов могут жить только внутри **`WebApplicationContext`**.

Spirng :: MVC :: WebApplicationContext

- Специальные типы бинов
  - Контроллеры
  - Мапперы обработчиков
  - Резолверы представлений
  - Резолверы локалей
  - Резолверы тем
  - Обработчики загрузки и файлов
  - Обработчики исключений

Spirng :: MVC :: WebApplicationContext

- Настройка
  - Для инициализации контекста нужно добавить **`ContextLoaderListener`** в web.xml
  - При инициализации контекста создаются бины (кроме «lazy-init» бинов), определённые в файлах **applicationContext.xml** и **[servlet-name]-servlet.xml** (для каждого **`DispatcherServlet`**)
  - Набор файлов, из которых следует создавать бины, может быть изменён с помощью задания параметра **`contextConfigLocation`** в дескрипторе приложения.

Spirng :: MVC :: WebApplicationContext

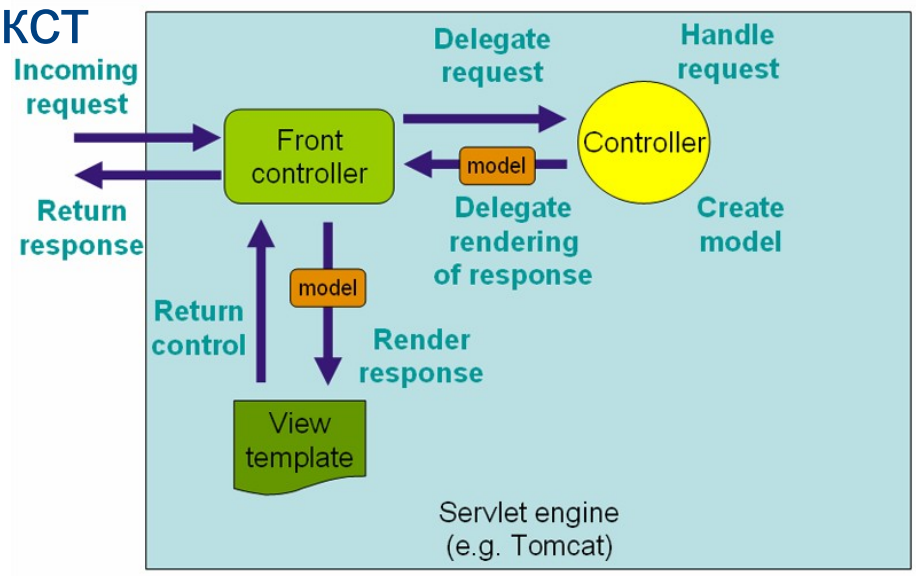
LUXOFT TRAINING

Пример:

```
<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        /WEB-INF/context.xml
        /WEB-INF/handlers.xml
        /WEB-INF/myContext.xml
        /WEB-INF/validators.xml
        /WEB-INF/commonBeans.xml
    </param-value>
</context-param>
```

Spirng :: MVC :: DispatcherServlet

- - это **`HttpServlet`**
- Реализация шаблона «Front Controller»
- Является «точкой входа» для запросов, управляя запросами к обработчикам
- Имеет свой **`WebApplicationContext`**, и скрывает внутреннюю логику работы с ним.
- Наследует корневой контекст
- Знает о **`ServletContext`**



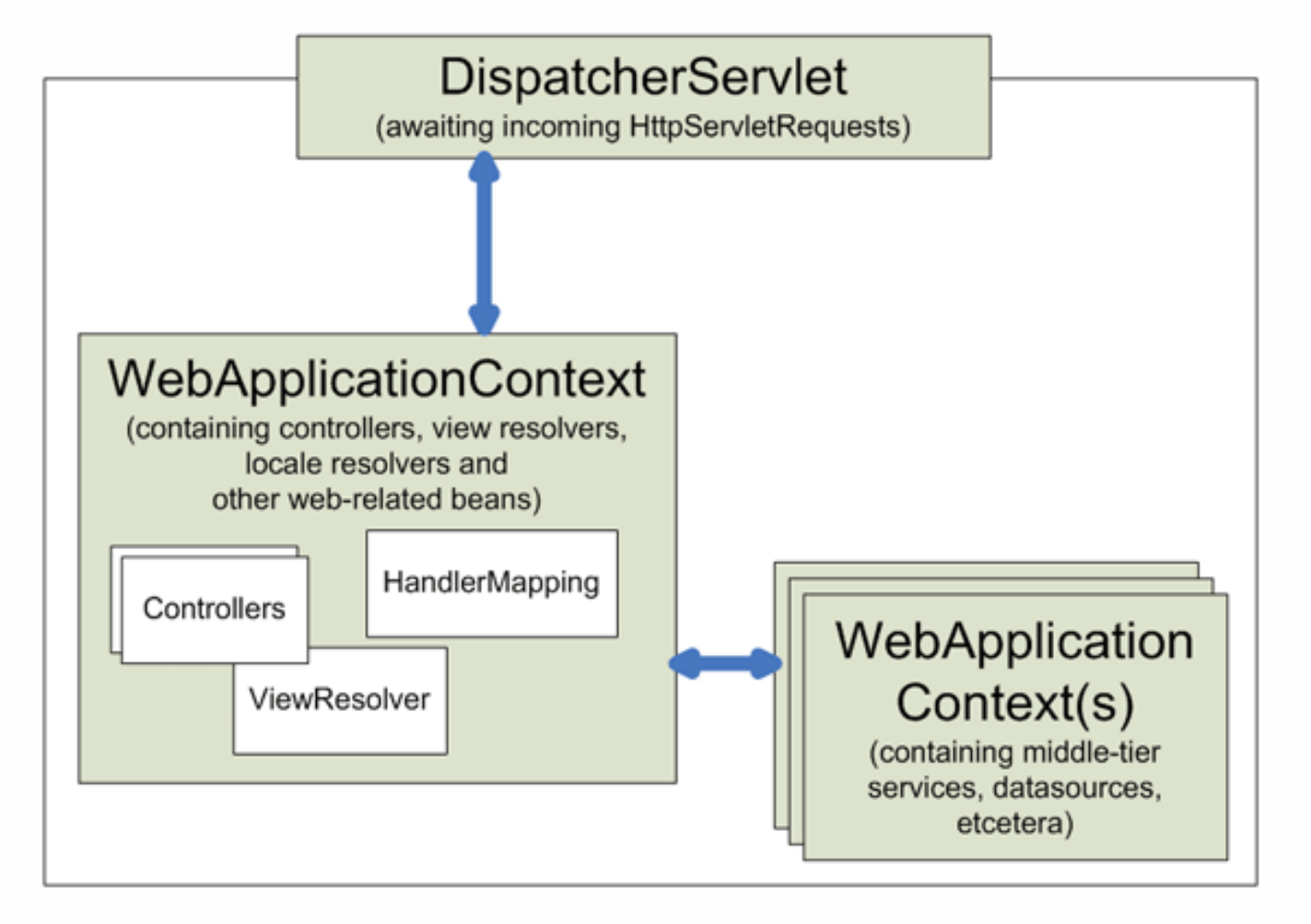
Spirng :: MVC :: DispatcherServlet

- Настройка:

```
<servlet>
    <servlet-name>viewer</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>viewer</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Spirng :: MVC :: DispatcherServlet

- Структура контекста:



Spirng :: MVC :: DispatcherServlet

- Обработка запроса:
  - **`WebApplicationContext`** ассоциируется с запросом для возможности использования контроллерами.
  - Резолверы локалей и тем также ассоциируются с запросом.
  - Ищется обработчик, запускается цепочка выполнения (перехватчики и контроллеры).
  - Ищется и отображается представление на основе обновлённой модели.

Spirng :: MVC :: DispatcherServlet

Spirng :: MVC :: Web & RESTful-service контроллеры

LUXOFT TRAINING

- Выполняют всю работу по:
  - Интерпретации пользовательского ввода (запросов)
  - Построению модели
  - Передаче модели в представление для отображения

Spirng :: MVC :: Web & RESTful-service контроллеры

- Базовый интерфейс:

```
package org.springframework.web.servlet.mvc;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;

public interface Controller {
    ModelAndView handleRequest(
        HttpServletRequest request,
        HttpServletResponse response
    ) throws Exception;
}
```

Spirng :: MVC :: Web & RESTful-service контроллеры

LUXOFT TRAINING

- AbstractController:
  - Предоставляет доступ к **`ServletContext`** и **`ApplicationContext`**
  - Берёт на себя работу с некоторыми HTTP-заголовками
  - Обеспечивает управление кэшированием и синхронизацией
  - Является суперклассом для всех остальных реализаций

LUXOFT TRAINING

- Свойства AbstractController`a:
  - **`supportedMethods`** – обрабатываемые HTTP-методы (по-умолчанию GET,POST)
  - **`requiresSession`** – будет ли выброшено исключение при отсутствии сессии, ассоциированной с запросом
  - **`synchronizeOnSession`** – синхронизировать ли вызовы основного метода на объекте сессии
  - 4 свойства, управляющие политиками «устаревания» и кэширования

Spirng :: MVC :: AbstractController

- Workflow (and that defined by interface):
  - handleRequest() will be called by the DispatcherServlet ;
  - Inspection of supported methods (ServletException if request method is not support) ;
  - If session is required, try to get it (ServletException if not found) ;
  - Set caching headers if needed according to the cacheSeconds property ;
  - Call abstract method handleRequestInternal() (optionally synchronizing around the call on the HttpSession), which should be implemented by extending classes to provide actual functionality to return ModelAndView objects.

Spirng :: MVC :: AbstractController

- Пример использования AbstractController`a:

```
package samples;
public class SampleController extends AbstractController {
    public ModelAndView handleRequestInternal(
        HttpServletRequest request,
        HttpServletResponse response
    ) throws Exception {
        ModelAndView mav = new ModelAndView("hello");
        mav.addObject("message", "Hello World!");
        return mav;
    }
}
```

- .xml:

```
<bean id="sampleController" class="samples.SampleController">
    <property name="supportedMethods" value="GET"/>
</bean>
```

Spirng :: MVC :: @Controller

- С помощью аннотации @Controller
- Помним про инструкцию контексту:

```
<context:component-scan base-package=foo.bar.web"/>
 @Controller
 public class HelloWorldController {

     @RequestMapping("/helloWorld")
     public ModelAndView helloWorld() {
         ModelAndView mav = new ModelAndView();
         mav.setViewName("helloWorld");
         mav.addObject("message", "Hello World!");
         return mav;
     }
 }
```

Spirng :: MVC :: @Controller

Классическое узкое место создания аннотированных контроллеров – проблемы с использованием AOP в целом и декларативного управления транзакциями в частности

Стандартные JDK Dynamic Proxies – только на интерфейсы

Чтобы заработало для контроллеров – надо выделять интерфейс

@RequestMappings – в интерфейс (т.к. механизм мэппинга будет видеть **прокси**), что крайне неудобно

В качестве альтернативы - `proxy-target-class="true"` и использовать CGLib

Поэтому (вспоминаем AOP и TX) – в случае с веб-приложением лучше СРАЗУ делать все на классах

Spirng :: MVC :: @RequestMapping

Настройка мэппинга, с помощью аннотации @RequestMapping:

Задается для:

- Контроллера (класса):
  - абсолютный путь;
- Метода контроллера:
  - В случае если задан путь для класса – относительный путь ;
  - Абсолютный в случае, если для класса не задан;

Spirng :: MVC :: Маппинг, шаблоны URL

- Поддержка шаблонов URL ;
- Для привязки переменной (параметра) к части URL-шаблона используется @PathVariable ;
- Использование для реализации RESTful-сервисов ;

Spirng :: MVC :: Маппинг, шаблоны URL

- Примеры использования:

```
@RequestMapping(value="/owners/{ownerId}/pets/{petId}”)
public String findPet(@PathVariable String ownerId,
   @PathVariable String petId, Model model) {
 // implementation
}
```

```
@Controller
@RequestMapping("/owners/{ownerId}")
public class RelativePathUriTemplateController {
  @RequestMapping("/pets/{petId}")
  public void findPet(@PathVariable String ownerId,
   @PathVariable String petId, Model model) {
    // implementation
  }
}
```

Spirng :: MVC :: @RequestMapping

value – задает URL шаблон. Поддерживаются:

- параметризованные (/somepath/{someVar}) URL шаблоны
- Ant-style (/somepath/*.do)
- их комбинации (/owners/*/pets/{petId})
- при отсутствии параметра – мэппинг производится по имени метода

method – привязка к типу запроса:

- RequestMethod.POST
- RequestMethod.GET

params – задает необходимость наличия (или отсутствия) определенных параметров в запросе

- “myParam = myValue” – наличие параметра с заданным значением
- “myParam” – наличие параметра с произвольным значением
- “!myParam” – отсутствие параметра в запросе

- headers – спецификация по заголовкам запроса (например, "content-type=text/*»)

Spirng :: MVC :: @RequestMapping, @RequestParam

LUXOFT TRAINING

- Привязка к параметрам запроса осуществляется с помощью аннотации @RequestParam
- По-умолчанию, таки параметры являются обязательными, но можно сделать опциональными с помощью инструкции required=false
- Пример:

```
@RequestMapping(method = RequestMethod.GET)
public String setupForm(
   @RequestParam("petId") int petId,
   ModelMap model)

@RequestMapping(method = RequestMethod.GET)
public String showItem(
   @RequestParam(value=”id”, required=false) int id,
   ModelMap model)
```

Spirng :: MVC :: @RequestMapping

Допустимые параметры методов-обработчиков:

- Request / Response объекты Servlet API. Можно использовать подтипы (например, ServletRequest или HttpServletRequest) ;
- Объект сессии Servlet API (HttpSession). Наличие такого атрибута принудительно создает сессию в случае ее отсутствия ;
- WebRequest и NativeWebRequest (пакета org.springframework.web.context.request) позволяет получить доступ к параметрам запроса без зависимости на Servlet или Portlet API ;
- Java.util.Locale для ссылки на текущую установленную локализацию;
- InputStream / Reader для доступа к содержимому запроса ;
- OutputStream / Writer – для генерации содержимого ответа ;
- java.security.Principal – содержит текущего авторизованного пользователя ;
- Аннотированние @PathVariable параметры для доступа к переменным из URI шаблона ;
- Аннотированние @RequestParam параметры для доступа к параметрам запроса ;

Spirng :: MVC :: @RequestMapping

Допустимые параметры методов-обработчиков:

- @RequestHeader аннотированные параметры для доступа к соответствующим заголовкам запроса ;
- @RequestBody – для получения тела запроса ;
- java.util.Map / org.springframework.ui.Model / org.springframework.ui.ModelMap – для доступа к модели, которая будет видима View ;
- org.springframework.validation.Errors / org.springframework.validation.BindingResult – для доступа / назначения результатов валидации объекта. Этот атрибут должен следовать **сразу** за объектом, для которого проводится валидация, т.к. для каждого объекта создается и привязывается отдельный результат валидации ;
- org.springframework.web.bind.support.SessionStatus – для контроля за состоянием сессии и закрытия сессии (что автоматически приведет к чистке объектов в сессии) ;

Spirng :: MVC :: @RequestMapping

Допустимые возвращаемые значения:

- ModelAndView – для возврата модели и имени View ;
- Model – для возврата модели. Имя View будет вычисляться с помощью RequestToViewNameTranslator ;
- Map – аналогично Model ;
- View – view объект. Модель будет автоматически подставлена (изменения, сделанные в модели, принятой через параметр обработчика будут включены) ;
- String – логическое имя view, которе будет обработано ViewResolver_ом;
- void – в случае если метод сам отвечает за формирование response или в случае если имя представления вычисляется с помощью RequestToViewNameTranslator;
- Если метод аннотирован @ResponseBody – возвращаемый тип будет записан в тело HTTP ответа;
- Любой другой тип будет рассматриваться как элемент модели, который будет иметь имя определенное в @ModelAttribute уровня метода или автоматически сгенерированное по типу возвращаемого значения;

Spirng :: MVC :: @ModelAttribute

Два сценария использования:

- Аннотация к методу – для вычисления соответствующего атрибута модели;
- Аннотация к параметру обработчика – привязывает атрибут модели к соответствующему параметру. Так обычно получается доступ к данным, введенным пользователем в форме (например);

Вызов @ModelAttribute методов происходит непосредственно **перед** вызовом соответствующего обработчика для заполнения модели.

Best Practices: в @ModelAttribute методах проверять на наличие заполненного атрибута модели и возвращать имеющийся

Spirng :: MVC :: @SessionAttribute

@SessionAttribute - указывается на уровне класса контроллера;

Определяет – какие атрибуты модели должны быть сохранены в сессии для передачи между запросами;

Spirng :: MVC :: @SessionAttribute

```
@Controller
@RequestMapping("/owners/{ownerId}/pets/{petId}/edit")
@SessionAttributes("pet")
public class EditPetForm {
    @ModelAttribute("types")
    public Collection<PetType> populatePetTypes() {
        return this.clinic.getPetTypes();
    }
    @RequestMapping(method = RequestMethod.POST)
    public String processSubmit(
            @ModelAttribute("pet") Pet pet,
            BindingResult result, SessionStatus status) {
        new PetValidator().validate(pet, result);
        if (result.hasErrors()) {
            return "petForm";
        }
        else {
            this.clinic.storePet(pet);
            status.setComplete();
            return "redirect:owner.do?ownerId=" + pet.getOwner().getId();
        }
    }
}
```

Spirng :: MVC :: Исключения

LUXOFT TRAINING

- Обработка ошибок:
  - Интерфейс **`HandlerExceptionResolver`** плюс одна готовая реализация **`SimpleMappingExceptionResolver`**.
  - Сопоставляет классы исключений именам представлений на основе **`Properties`**.
  - Ловит исключения, выброшенные из обработчиков, передаёт исключение через модель.
  - Является более гибким механизмом по сравнению с настройкой **web.xml**, т.к. можно сделать свою реализацию.

Spirng :: MVC :: Исключения

LUXOFT TRAINING

Обработка ошибок. Пример:

```
<bean id="exceptionResolver"
      class="org.springframework.web.servlet.
             handler.SimpleMappingExceptionResolver">
    <property name="exceptionMappings">
        <value>
            java.lang.Throwable=error
            java.lang.SecurityException=error403
        </value>
    </property>
</bean>
```

Spirng :: MVC :: @ExceptionHandler

- В контроллере можно задать метод, который будет вызываться в случае, если какой-то из обработчиков контроллера выкинул соответствующий Exception
- Параметры метода аналогичны параметрам обработчиков

```
@Controller
public class SimpleController {
  @ExceptionHandler(IOException.class)
  public String handleIOException(
   IOException ex,
   HttpServletRequest request)
  {
    return ClassUtils.getShortName(ex.getClass());
  }
}
```

Spirng :: MVC :: View, основные идеи

LUXOFT TRAINING

- Отображают модель, построенную в контроллере;
- «Из коробки» могут быть JSP-страницами, Velocity-шаблонами, XSLT-представлениями, имплементациями интерфейса **`View`**, или **`Tiles ;`**
- Для сопоставления имён (с учётом локалей) объектам представлений используются реализации интерфейса **`ViewResolver ;`**
- Наибольшую популярность в качестве представлений в последнее время получили **`Tiles`**

Spirng :: MVC :: ViewResolver

Для вычисления вью по результатам вызова обработчиков контроллера используется ViewResolver

Должен уметь вычислять View по прямому указанию (обработчик вернул имя представления, View или ModelAndView) или неявно на основании каких-либо соглашений

Стандартные реализации:

- InternalResourceViewResolver – сопоставление имен представлений внутренним ресурсам приложения, например JSP ;
- XmlViewResolver, ResourceBundleViewResolver – сопоставления на основе .xml- или .properties-файла (последний – поддерживает i18n) ;
- BeanNameViewResolver – поиск по имени бина. Для этого все View должны быть созданы как бины в контексте ;
- UrlBasedViewResolver – напрямую преобразует имя представления в URL представления ;

Spirng :: MVC :: ViewResolver

Реализации для поддержки различных view-технологий:

- `FreeMarkerViewResolver`
- `VelocityViewResolver`
- `VelocityLayoutViewResolver`
- `JasperReportsViewResolver`
- `XsltViewResolver`

Spirng :: MVC :: ViewResolver

Пример конфигурации:

```
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.
   InternalResourceViewResolver">

   <property name="viewClass"
      value="org.springframework.web.servlet.view.JstlView"/>
   <property name="prefix" value="/WEB-INF/jsp/"/>
   <property name="suffix" value=".jsp"/>
</bean>
```

Spirng :: MVC :: ViewResolver

LUXOFT TRAINING

- Все готовые реализации **`ViewResolver`** по умолчанию кэшируют распознанные **`view`**
- Изменить это поведение можно, установив свойство **`cache`** в **`false`**.
- Однократно форсировать распознавание представления можно, вызвав метод **`removeFromCache(String viewName, Locale loc)`**

Spirng :: MVC :: ViewResolver

LUXOFT TRAINING

- Все view-резолверы реализуют интерфейс **`Ordered`**. Это позволяет иметь несколько резолверов, срабатывающих в определённом порядке.
- Для этого нужно установить свойсто **`order`**. Если не указано явно – такой резолвер будет срабатывать последним.
- Это может быть нужно для переопределения некоторых view в особых случаях, или если один резолвер не поддерживает все используемые реализации **`View`**

Spirng :: MVC :: ViewResolver

- Пример:
  - **applicationContext.xml:**

```
<bean id="jspViewResolver"
class="org.springframework.web.servlet.view.
  InternalResourceViewResolver">
    <property name="viewClass”
value="org.springframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/WEB-INF/jsp/"/>
    <property name="suffix" value=".jsp"/>
</bean>
<bean id="excelViewResolver“
  class="org.springframework.web.servlet.view.
  XmlViewResolver">
    <property name="order" value="1"/>
    <property name="location" value="/WEB-INF/views.xml"/>
</bean>
```

Spirng :: MVC :: I10n

- В JSP страницах пользоваться тэгом **`spring:message`**. Вместо сообщений указываются их ключи, хранящиеся в **`.properties`**-файле;
- Задать **`MessageSource`** в контексте;
- Тексты сообщений вынести в **`.properties`**-файлы, названные соответственно поддерживаемым локалям;
- Задать способ определения локали – реализацию **`LocaleResolver`** – в контексте (не обязательно, есть резолвер по умолчанию);

Spirng :: MVC :: I10n

```
<%@ taglib prefix="spring"
          uri="http://www.springframework.org/tags" %>
<html>
    <head>
        <title>
            <spring:message code="error.page"/>
        </title>
    </head>
    <body>
        <h1><spring:message code="exception.thrown"/></h1>
        . . .
    </body>
</html>
```

Spirng :: MVC :: I10n

applicationContext.xml:

```
<bean id="messageSource"
class="org.springframework.context.support.
  ResourceBundleMessageSource">
    <property name="basename" value="messages"/>
</bean>
```

messages_ru_RU.properties:

```
exception.thrown=Было выброшено исключение
error.page=Информация об ошибке
```

Spirng :: MVC :: I10n :: LocaleResolver

LUXOFT TRAINING

- **`AcceptHeaderLocaleResolver`** (резолвер по умолчанию) – смотрит на заголовок HTTP-запроса «accept-language»;
- **`CookieLocaleResolver`** – берёт значение локали из cookie;
- **`SessionLocaleResolver`** – берёт значение локали из сессии;
- **`FixedLocaleResolver`** – берёт значение локали, заданное при конфигурировании бина;

Spirng :: MVC :: I10n :: LocaleResolver

Вы можете:

- Изменить настройки браузера (в случае AcceptHeader-резолвера) ;
- Вызвать метод RequestContextUtils.getLocaleResolver(request).setLocale(request, response, locale). Это создаст cookie или атрибут сессии;
- Напрямую в коде контроллера через Locale;
- Задать у HandlerMapping перехватчик типа LocaleChangeInterceptor. Локаль указывать как параметр запроса (самый простой способ поддержки переключения языков);

Spirng :: MVC :: I10n :: LocaleResolver

```
<bean id="localeChangeInterceptor"
class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
    <property name="paramName" value="siteLanguage"/>
</bean>

<bean id="localeResolver"
class="org.springframework.web.servlet.i18n.CookieLocaleResolver"/>
<bean id="urlMapping"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="interceptors">
        <list>
            <ref bean="localeChangeInterceptor"/>
        </list>
    </property>
    <property name="mappings">
        <value>/**/*.view=someController</value>
    </property>
</bean>
```

Spirng :: MVC :: Темы

Тема – это совокупность статических ресурсов, влияющих на внешний вид веб-приложения.

Как правило, сюда входят изображения и файлы стилей (CSS).

Spirng :: MVC :: Темы

Вы можете:

- На страницах пользоваться тэгом **`spring:theme`** вместо жёстких ссылок на ресурсы ;
- Задать **`ThemeSource`** в контексте ;
- Ссылки на отличающиеся ресурсы (картинки, стили) вынести в .properties-файлы, названные соответственно темам ;
- Задать способ определения темы – реализацию **`ThemeResolver`** – в контексте ;

Spirng :: MVC :: Темы

Пример jsp-файла:

```
<%@ taglib prefix="spring"
  uri=http://www.springframework.org/tags %>
<html>
    <head>
        <link rel="stylesheet" type="text/css"
              href="<spring:theme code="css"/>" />
    </head>
    <body background="<spring:theme code="bg"/>">
        . . .
    </body>
</html>
```

Spirng :: MVC :: Темы

Пример:

applicationContext.xml:

```
<bean id="themeSource"
class="org.springframework.ui.context.support.Res
  ourceBundleThemeSource"/>
```

cool.properties:

```
css=/themes/cool/style.css
bg=/themes/cool/img/bg.jpg
```

Spirng :: MVC :: Темы

`CookieThemeResolver` – берёт имя темы из cookie ;

`SessionThemeResolver` – берёт имя темы из сессии ;

`FixedThemeResolver` – берёт название темы, заданное при конфигурировании бина ;

Spirng :: MVC :: Темы

Однократное изменение темы:

- Вызвать метод **`RequestContextUtils.getThemeResolver(request).setThemeName(themeName)`**. Это создаст cookie или атрибут сессии ;
- Напрямую в коде контроллера ;
- Задать у **`HandlerMapping`** перехватчик типа **`ThemeChangeInterceptor`**. Имя темы указывать как параметр запроса ;

Spirng :: MVC :: Темы :: ThemeResolver

LUXOFT TRAINING

```
<bean id="themeChangeInterceptor"
class="org.springframework.web.servlet.theme.ThemeChangeInterceptor">
    <property name="paramName" value="lookAndFeel"/>
</bean>
<bean id="themeResolver"
    class="org.springframework.web.servlet.theme.CookieThemeResolver"/>

<bean id="urlMapping"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="interceptors">
        <list>
            <ref bean="themeChangeInterceptor"/>
        </list>
    </property>
    <property name="mappings">
        <value>/**/*.view=someController</value>
    </property>
</bean>
```

Spirng :: MVC :: Темы + I10n

Может быть нужно, если темы содержат например картинки с надписями.

Достаточно для каждой темы создать дополнительные .properties-файлы, названные соответственно локалям. Например:

```
cool.properties
cool_ru_RU.properties
cool_en_US.properties
dark.properties
dark_ru_RU.properties
dark_en_US.properties
```

Spirng :: MVC :: Multipart

- Поддержка Multipart – это возможность загружать файлы из html-форм
- Атрибут формы enctype должен быть выставлен в multipart/form-data
- Файлы загружаются через

```
<input type="file" />
```

- По умолчанию обработка загрузки файлов отключена в Spring (для возможности самостоятельной обработки)

Spirng :: MVC :: Multipart

Пакет **`org.springframework.web.multipart`** :

- Интерфейс MultipartResolver
- Реализации CommonsMultipartResolver и CosMultipartResolver
- Необходимые библиотеки:
  - commons-io.jar и commons-fileupload.jar ;
  - Или cos.jar ;

Spirng :: MVC :: Multipart

- Создать в контексте приложения бин – реализацию `MultipartResolver`
- Каждый поступающий запрос будет проверяться на наличие «файлов» и оборачиваться в `MultipartHttpServletRequest` при необходимости

- Пример:

```
<bean id="multipartResolver”
class="org.springframework.web.multipart.
    commons.CommonsMultipartResolver”>
   <property name="maxUploadSize" value="100000"/>
</bean>
```

Spirng :: MVC :: Multipart :: MultipartResolver

LUXOFT TRAINING

- **`maxInMemorySize`** – размер «буфера» в памяти, прежде чем файлы начнут записываться на диск. По умолчанию 10KB ;
- **`maxUploadSize`** – максимальный суммарный размер файлов. По умолчанию не ограничен;
- **`uploadTempDir`** – временное хранилище загружаемых файлов. По умолчанию – временная директория контейнера ;
- **`defaultEncoding`** – кодировка для разбора заголовков «частей». Имеет меньший приоритет, чем указанная в запросе явно ;

Spirng :: MVC :: Multipart

Пример формы:

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

<form:form action="someRequest.do” enctype="multipart/form-data”>
        <input type=“text” name=“name”/>
        <input type="file" name="file"/>
        <input type="submit"/>
</form:form>
```

Spirng :: MVC :: Multipart

Пример контроллера:

```
@Controller
public class FileUpoadController {

    @RequestMapping(value = "/form", method = RequestMethod.POST)
    public String handleFormUpload(@RequestParam("name") String name,
        @RequestParam("file") MultipartFile file) {

        if (!file.isEmpty()) {
            byte[] bytes = file.getBytes();
            // store the bytes somewhere
           return "redirect:uploadSuccess";
       } else {
           return "redirect:uploadFailure";
       }
    }
}
```

Spirng :: MVC :: Tag Library

Существуют 2 библиотеки:

- Spring
- Spring-form

Их основная задача – упрощение механизмов биндинга, предоставляемых API JSP-страниц, объекта формы на элементы формы и обратно;

Почти все тэги имеют атрибут **`htmlEscape`**, который позволяет управлять экранированием HTML или JS-кода. В расчёт также берётся значение парметра **`defaultHtmlEscape`** из **`web.xml`**

Spirng :: MVC :: Tag Library

LUXOFT TRAINING

Подключение реализуется следующим образом

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


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

Spirng :: MVC :: Tag Library

Темы и локали:

- Тэг **`theme`** - подставляет вместо себя путь к ресурсу, имя которого задано в атрибуте code, используя **`ThemeResolver`** и **`ThemeSource`**.
- Тэг **`message`** - подставляет вместо себя локализованное сообщение, имя которого задано в атрибуте code, используя **`LocaleResolver`** и **`MessageSource`**.

Spirng :: MVC :: Tag Library

Тэг **`bind`**:

- Имеет атрибут **`path`**, содержащий «путь» к свойству объекта формы в dot-нотации, подлежащему биндингу;
- Предпринимает попытку биндинга, предоставляет объект **`BindStatus`** для последующего анализа;
- Свойства **`error`**, **`errors`**, **`errorCodes`**, **`errorsMessage`**, **`expression`**, **`path`** ;
- **`value`**, **`valueType`** ;

Spirng :: MVC :: Tag Library

Тэг **`hasBindErrors`**:

- Имеет атрибут **`name`**, содержащий имя объекта формы, поля которого биндились ранее;
- Содержимое тэга будет выведено при наличии ошибок биндинга. Внутри тэга ошибки доступны через объект **`Errors`**.

Spirng :: MVC :: Tag Library

Пример применения тэгов **bind, hasBindErrors:**

```
<form method="post" action="testPage.html">
    <spring:bind path="commandBean.fieldOne">
        <input type="text" name="fieldOne" value="${status.value}"/>
        <c:if test="${status.error}"> ... </c:if>
    </spring:bind>
    <spring:bind path="commandBean.fieldTwo">
        <input type="text" name="fieldTwo" value="${status.value}" />
        <c:if test="${status.error}">
            <c:forEach items="${status.errorMessages}" var="msg">${msg}<br/>
            </c:forEach>
        </c:if>
    </spring:bind>
    <spring:hasBindErrors name="commandBean">
         There were ${errors.errorCount} error(s) in total:
         <c:forEach var="errMsgObj" items="${errors.allErrors}">
             <spring:message code="${errMsgObj.code}"
                             text="${errMsgObj.defaultMessage}"/><br/>
```

Spirng :: MVC :: Tag Library

Тэг **`transform:`**

- Имеет атрибут **`value`**, содержащий объект, который подлежит преобразованию
- Преобразует указанный объект в строку, используя **`PropertyEditors`**, чтобы при отправке формы было возможно обратное преобразование
- Может использоваться только внутри тэга **`bind`**

Spirng :: MVC :: Tag Library

Пример применения тэга **transform**:

```
<form method="post">
  <spring:bind path="contract.contractType">
    <select name="<c:out value="${status.expression}"/>">
      <c:forEach items="${contractTypes}" var="type">
        <spring:transform value="${type}" var="typeString"/>
        <option value="<c:out value="${typeString}"/>"
          <c:if test="${status.value == typeString}"/> selected</c:if>>
          <c:out value="${typeString}"/>
        </option>
      </c:forEach>
    </select>
  </spring:bind>
</form>
```

Spirng :: MVC :: Tag Library

Тэг **`form:`**

- Имеет атрибут **`commandName`**, содержащий имя объекта формы. При описании «путей» элементов формы, первый элемент (это имя) опускается.
- Генерирует html-элемент **`<form>`**

Spirng :: MVC :: Tag Library

Тэг **`errors`**:

Имеет атрибуты:

- **`commandName`**, содержащий свойство для которого выводятся ошибки биндинга (* = все ошибки) ;
- **`cssClass`**, содержащий присваиваемый блоку css-класс ;

- Генерирует html-элемент **`<span>`**, в котором выводятся ошибки биндинга через
 ;
- Можно создавать несколько, в разных местах страницы, для вывода ошибок биндинга отдельных свойств ;

Spirng :: MVC :: Tag Library

Пример применения тэга **`errors`**:

```
<form:form action="someRequest.do" commandName="someBean">
    <form:errors path="*" cssClass="errorText" />
    . . .
<form:form>
```

Spirng :: MVC :: Tag Library

Поля ввода:

- Тэг **`input`**
- Тэг **`hidden`**
- Тэг **`textarea`**
- Тэг **`password`**

- Генерируют **`input`** соответствующего типа;
- В атрибут **`value`** помещается значение свойства объекта формы, указанного в атрибуте **`path`**;
- В атрибут **`name`** помещается «путь» свойства для возможности биндинга при отправке;

Spirng :: MVC :: Tag Library

Пример применения тэгов полей ввода:

```
<form:form action="someRequest.do" commandName="user">
    <form:input path="name" />
    <form:password path="passwd" />
    <form:hidden path="id" />
    <form:textarea path="notes" />
    . . .
<form:form>
```

Spirng :: MVC :: Tag Library

Переключатели:

- Тэг **`radiobutton`**
- Тэг **`checkbox`**

- Генерируют соответствующий html-элемент;
- Используется несколько раз с одним и тем же значением **`path`** но разными **`value`**. После биндинга выбран будет тот, у которого совпадают значение свойства, указанного в **`path`** и значение **`value`**;

Spirng :: MVC :: Tag Library

Пример применения тэгов переключателей:

```
<form:form action="someRequest.do" commandName="user">
    Intrests:
    Pets  <form:checkbox path="prefs.interests"
                value="pets"/>
    Games <form:checkbox path="prefs.interests"
                value="games"/>
    . . .
    Gender:
    <form:radiobutton path="sex" value="M"/>
    <form:radiobutton path="sex" value="F"/>
    . . .
<form:form>
```

Spirng :: MVC :: Tag Library

Группы переключателей:

- Тэг **`radiobuttons`**
- Тэг **`checkboxes`**

Атрибуты:

- **`path`** – «путь» к свойству объекта формы;
- **`items`** – **`java.util.Map`**, элемент модели;

Генерируют несколько соответствующих html-элементов

`key` из **`Map`** используется как **`value`** элемента. **`value`** из **`Map`** используется как текст. Выбран будет тот, у которого совпадают значение свойства, указанного в **`path`** и значение **`value`**.

Spirng :: MVC :: Tag Library

Пример применения тэгов групп переключателей :

```
<form:form action="someRequest.do" commandName="user">
    Interests:
    <form:checkboxes path="prefs.interests"
                items="${availInterests}"/>
    Gender:
    <form:radiobuttons path="sex" items="${availGenders}"/>
    . . .
<form:form>
```

Spirng :: MVC :: Tag Library

LUXOFT TRAINING

Пример применения тэгов групп переключателей :

```
<form:form action="someRequest.do" commandName="user">
    Interests:
    <form:checkboxes path="prefs.interests"
                items="${availInterests}"/>
    Gender:
    <form:radiobuttons path="sex" items="${availGenders}"/>
    . . .
<form:form>
```

Spirng :: MVC :: Tag Library

Список:

- Тэг `select`
- Тэг `options`
- Тэг `option`

Логика биндинга повторяет логику для переключателей.

Spirng :: MVC :: Tag Library

LUXOFT TRAINING

Примеры применения тэгов списков :

```
<form:form action="someRequest.do" commandName="user">
        Gender:
        <form:select path="sex" />
            <form:options items="${availGenders}"/>
        </form:select>
<form:form>

<form:form action="someRequest.do" commandName="user">
        Gender:
        <form:select path="sex" />
            <form:option value="M">Male</form:option>
            <form:option value="F">Female</form:option>
        </form:select>
<form:form>
```

Упражнения

№: 9 : «Разработка web - приложения на базе Spring MVC»

- 45 мин – самостоятельная работа;
- 15 мин – обсуждение;

TODO

Валидация

JSR-303

LUXOFT TRAINING

Вопросы!?