SIMBIRSOFT
software development

Введение в Spring
Spring MVC

Что такое Spring?

Spring – универсальная платформа с открытым исходным кодом, которая предоставляет облегченное решение по созданию готовых корпоративных приложений с возможностью использования RMI и Web-служб, отправки сообщений по электронной почте, обработки данных в базе данных, декларативного управления транзакциями, предоставляет среду выполнения MVC, способы интеграции АОП и хорошо структурированную иерархию исключений.

Spring был впервые выпущен в июне 2003 года и получил широкое распространение.

Текущая версия Spring – 4.1.0

Также существует версия для .NET Framework – Spring.NET

Сайт проекта: http://www.springsource.org/

Плюсы Spring

- Spring можно использовать для построение любого приложения на языке Java, что выгодно отличает его от многих других платформ (таких как Apache Struts)
- для использования ядра Spring нужно внести минимальные изменения в код приложения (принцип философии Spring – минимальное воздействие)
- Spring является модульной средой и позволяет использовать отдельные свои части без необходимости вводить остальные
- Возможность работы с POJO (без контейнеров EJB)
- Существует большое количество расширений Spring для построения приложений на Java Enterprise платформе
- Сообщество Spring – одно из лучших сообществ из всех проектов с открытым исходным кодом, списки рассылки и форумы всегда активны.
- У Spring отличная подробная документация
- Spring активно развивается

Возможности Spring

- Использование внедрения зависимостей (DI)
- Поддержка аспектно-ориентированного программирования (в том числе интеграция с AspectJ)
- Язык выражений Spring (SpEL) – позволяет приложению манипулировать объектами Java во время выполнения.
- Встроенная поддержка Bean Validation API – позволяет один раз описать логику проверки достоверности данных и использовать ее как в пользовательском интерфейсе, так и на уровне работы с БД.
- Spring обеспечивает великолепную интеграцию с большинством инструментов доступа к данным (JDBC, Hibernate, MyBatis, JDO, JPA и т.п.)
- Поддержка Object to XML Mapping – преобразование компонентов JavaBean в XML и наоборот (как правило, используется для обмена данными с другими системами)
- Интеграция с JEE – внедрение бинов Spring в компоненты EJB.
- MVC на веб-уровне
- Поддержка электронной почты
- Поддержка планирования заданий
- Поддержка динамических сценариев (Groovy, JRuby, BeanShell)

Модули Spring

Альтернативы Spring

- **JBoss Seam Framework** – платформа, построенная полностью на стандартах JEE (http://www.seamframework.org)
- **Google Guice** – облегченная платформа для управления конфигурацией приложения с помощью DI (http://code.google.com)
- **Pico Container** – исключительно малый контейнер DI (http://www.picocontainer.org)
- **Контайнер JEE 6**

Инверсия управления

Ядро Spring Framework основано на принципе инверсии управления (Inversion Of Control – IoC), при котором создание и управление зависимостями между компонентами становятся внешними.

В общем случае различают два вида IoC:
- внедрение зависимости (Dependency Injection – DI);
- инверсия поиска (Dependency Lookup).

Dependency Injection в свою очередь делится на:
- внедрение зависимостей через конструктор (Constructor DI);
- внедрение зависимостей через метод установки (Setter DI).

В Spring в основном используется внедрение зависимостей, а не инверсия поиска.

Примеры IoC

Традиционный подход:

```
class UserManager {

  private UserDao userDao;

  public UserManager () {
    this.userDao = new UserDaoOracle();
  }
}
```

Constructor DI:

```
class UserManager {

  private UserDao userDao;

  public UserManager (UserDao userDao) {
    this.userDao = userDao;
  }
}
```

Инверсия поиска:

```
class UserManager {

  private UserDao userDao;

  public UserManager (Repository rep) {
    this.userDao = (UserDao)
              rep.get("user-dao");
  }
}
```

Setter DI:

```
class UserManager {

  private UserDao userDao;

  public setUserDao(UserDao userDao) {
    this.userDao = userDao;
  }
}
```

Преимущества DI

- Сокращение объема связующего кода.
- Упрощенная конфигурация приложения (аннотации, XML-файлы).
- Вся информация об общих зависимостях хранится в единственном репозитории (при традиционном подходе она размазана по многочисленным классам).
- Улучшенная возможность тестирования (простая замена зависимостей).
- Стимулирует применение качественных проектных решение – проектирование с использованием интерфейсов.

Но при всех своих достоинствах DI не лишена недостатков.
Основной недостаток – бывает трудно сразу определить какая реализация конкретной зависимости к каким объектам привязана (особенно если вы не слишком хорошо ориентируетесь в коде).

BeanFactory

Ядром контейнера DI в Spring является интерфейс фабрики бинов BeanFactory. Этот интерфейс отвечает за управление компонентами, их зависимостями и их жизненным циклом (под бином понимается любой компонент, управляемый контейнером).

Пример XML-конфигурации:

```
<bean id="userDaoBean"
      class="com.simbirsoft.dao.oracle.UserDaoOracle"/>

<bean id="userManagerBean"
      class="com.simbirsoft.dao.oracle.UserDaoOracle"
      p:userDao-ref="userDaoBean"/>
```

Пример конфигурации с помощью аннотаций:

```
@Service("userManager")
class UserManagerImpl implements UserManager {

    @Injected
    private UserDao userDao)
}
```

MVC

MVC – это шаблон, который часто используется при реализации уровня презентаций в приложении. Главный принцип шаблона MVC состоит в определении архитектуры с четкой ответственностью для каждого компонента.

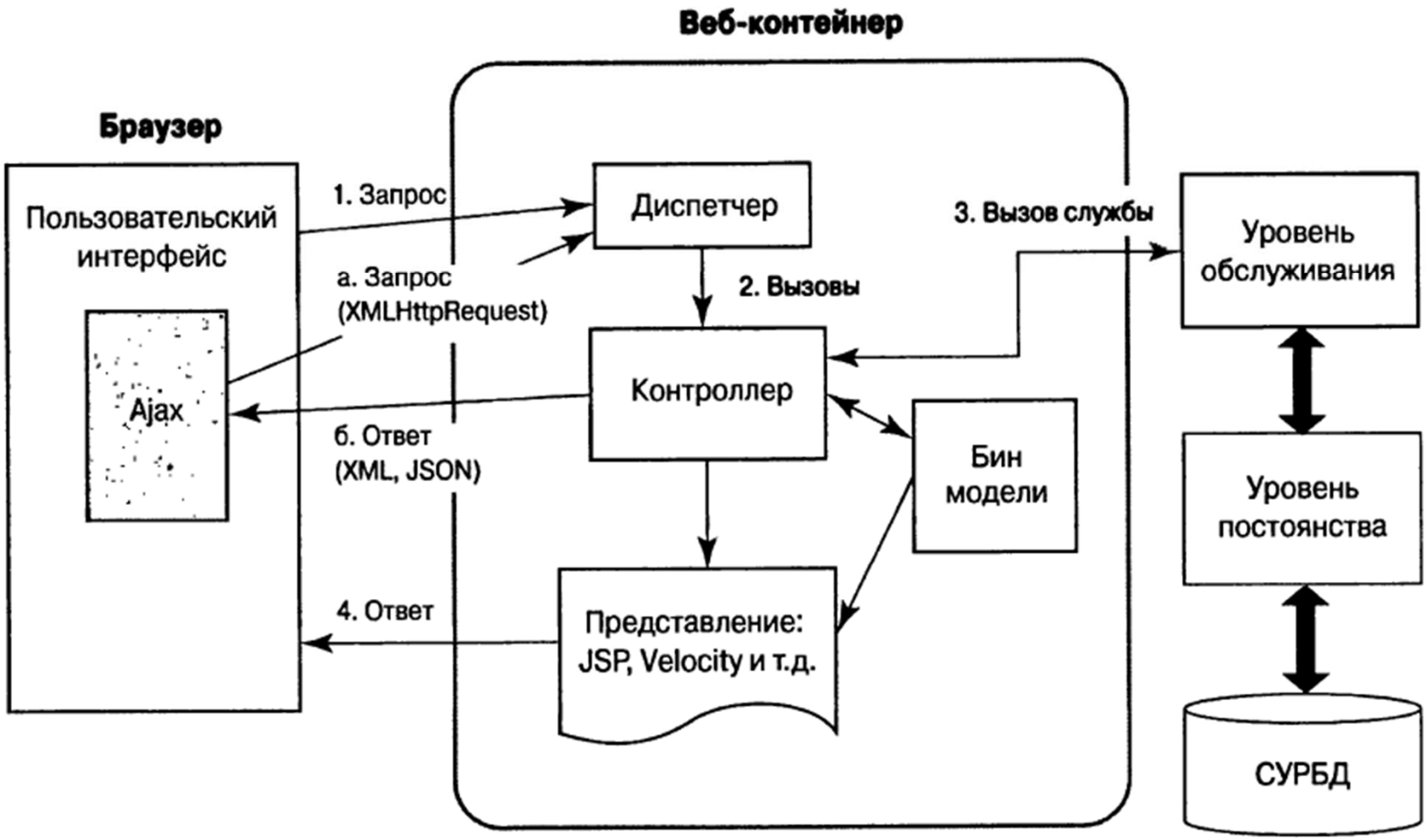
В шаблоне MVC разделяют три участника.

- *Модель* – представляет бизнес-данные.
- *Представление* – отображает данные пользователя в желаемом формате.
- *Контроллер* – обрабатывает запросы к действиям, осуществляемые пользователем в пользовательском интерфейсе, обновляет модель и направляет пользователей на соответствующие представления.

Обработка запроса

1. **Запрос** - клиент отправляет HTTP-запрос серверу
2. **Вызов диспетчера** - сервер направляет запрос диспетчеру (сервлету) SpringMVC
3. **Вызов контроллера** - диспетчер направляет запрос соответствующему контроллеру на основе информации HTTP-запроса и конфигурации веб-приложения
4. **Вызов службы** - контроллер взаимодействует с уровнем обслуживания (сервисами)
5. **Ответ** - контроллер обновляет модель и в зависимости от результата возвращает соответствующее представление пользователю

Шаблон MVC


Веб-контейнер
Браузер
Пользовательский интерфейс
Ajax
1. Запрос
а. Запрос (XMLHttpRequest)
б. Ответ (XML, JSON)
4. Ответ
Диспетчер
2. Вызовы
Контроллер
Бин модели
Представление: JSP, Velocity и т.д.
3. Вызов службы
Уровень обслуживания
Уровень постоянства
СУРБД

Пример. Модель данных


```
@Entity
@Table(name = "USERS")
public class User implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @NotNull
    @Size(min = 3, max = 50, message="{validation.user.name.size.message}")
    @Column(name = "name")
    private String name;

    @NotNull
    @Column(name = "email")
    private String email;

    @Column(name = "birth")
    @Type(type = "org.jadira.usertype.dateandtime.joda.PersistentDateTime")
    @DateTimeFormat(iso = ISO.DATE)
    private DateTime birth;
```

Файл /src/main/java/com/simbircite/demo/model/User.java

DAO UserRepository


```
/**
 * Интерфейс репозитория (DAO) для работы с моделью данных User.
 * Реализовывать интерфейс не нужно, об этом позаботится платформа Spring.
 * Используется в сервисе. Репозиторий не работает с транзакциями.
 * Не содержит бизнес-логику.
 */
public interface UserRepository extends PagingAndSortingRepository<User, Long> {

    /**
     * SELECT *
     * FROM users
     * WHERE email = ?
     * ORDER BY name
     */
    List<User> findByEmailOrderByNameAsc(String email);

}
```

Файл /src/main/java/com/simbircite/demo/repository/UserRepository.java

Сервис UserService


```
@Service
@Transactional
public class UserService {

    @Injected
    private UserRepository userRep;

    public void add(User user) {
        userRep.save(user);
    }

    public void update(User user) {
        userRep.save(user);
    }

    public void delete(User user) {
        userRep.delete(user);
    }

    @Transactional(readOnly = true)
    public User getById(Long id) {
        return userRep.findOne(id);
    }

    @Transactional(readOnly = true)
    public Iterable<User> getAll() {
        return userRep.findAll();
    }
}
```

Файл /src/main/java/com/simbircite/demo/service/UserService.java

Контроллер


```
/**
 * В Spring MVC существует целая иерархия контроллеров,
 * базовый интерфейс среди которых, - Controller.
 *
 * Все контроллеры в Spring MVC реализуют этот интерфейс.
 * Однако нам реализовывать его не нужно будет.
 */
@Controller /* объявляем, что данный класс является контроллером */
@RequestMapping("/users")
public class UserController {

    @Injected /* внедряем бин сервиса для работы с пользователями */
    private UserService userService;

    /**
     * Метод получения списка пользователей.
     * Вызывается при обращении к URL user при помощи метода GET
     * (http://localhost/users/list)
     */
    @RequestMapping(value = "/users/list", method = RequestMethod.GET)
    public ModelAndView getList() {
        /* Возвращает вид с логическим именем users.
         * В этот вид передаются данные:
         * - "users" список всех пользователей
         */
        ModelAndView mav = new ModelAndView();
        mav.setViewName("users/list"); /* шаблон модели в файле /WEB-INF/views/users/list.jspx */
        mav.addObject("users", userService.getAll());

        return mav;
    }
```

Файл /src/main/java/com/simbircite/demo/controller/UserController.java

Контроллер


```
/** Метод для сохранения параметров существующего пользователя.
   * Вызывается при сабмите формы.
   * Перед сохранением параметров происходит проверка корректности данных.
   * Если ошибок нет, то происходит переход на форму просмотра пользователя.
   * Если есть, то выводится сообщения об ошибках.
   * Параметры модели пользователя заполняются автоматически
   * атрибутами модели вида (формы) с именем "user". */
@RequestMapping(value = "{id}", params = "form", method = RequestMethod.POST)
public String update(
        @Valid @ModelAttribute("user") User user,
        BindingResult bindingResult,
        Model uiModel,
        HttpServletRequest httpServletRequest,
        RedirectAttributes redirectAttributes,
        Locale locale
) {
      if (bindingResult.hasErrors()) {
         uiModel.addAttribute("message", new Message("error",
               messageSource.getMessage("users.save.fail", new Object[] {}, locale)));
         uiModel.addAttribute("user", user);
         return “users/update”;
      }
      uiModel.asMap().clear();
      redirectAttributes.addFlashAttribute("message", new Message("success",
            messageSource.getMessage("users.save.success", new Object[] {}, locale)));

      userService.update(user);

      return "redirect:users/”+ UrlUtil.encodeUrlPathSegment(user.getId().toString(), httpServletRequest);
 }
```

Файл /src/main/java/com/simbircite/demo/controller/UserController.java

Представление списка пользователей


```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<div xmlns:jsp=http://java.sun.com/JSP/Page
      xmlns:c=http://java.sun.com/jsp/jstl/core
      xmlns:joda="http://www.joda.org/joda/time/tags"
      xmlns:form="http://www.springframework.org/tags/form" >

   <jsp:directive.page contentType="text/html;charset=UTF-8"/>
   <jsp:output omit-xml-declaration="yes"/>

    <h1>Users Listing</h1>

      <c:if test="${empty users}">
        List is empty
      </c:if>
      <c:if test="${not empty users}">
        <table>
          <thead>
            <tr>
              <th>Name</th>  <th>Email</th> <th>Birth Date</th>
            </tr>
          </thead>
          <tbody>
           <c:forEach items="${users}" var="u">
            <tr>
              <td><c:out value="${u.name}"/></td>
              <td><c:out value="${u.email}"/></td>
              <td><joda:format value="${u.birth}" pattern="dd.MM.yyyy"/></td>
              <td><a href='user/${u.id}/edit'>Edit</a></td>
              <td><a href='user/${u.id}/delete'>Delete</a></td>
            </tr>
           </c:forEach>
          </tbody>
        </table>
      </c:if>

      <a href='user/new'>Add new user</a>
</div>
```

Файл /src/main/webapp/WEB-INF/views/users/list.jspx

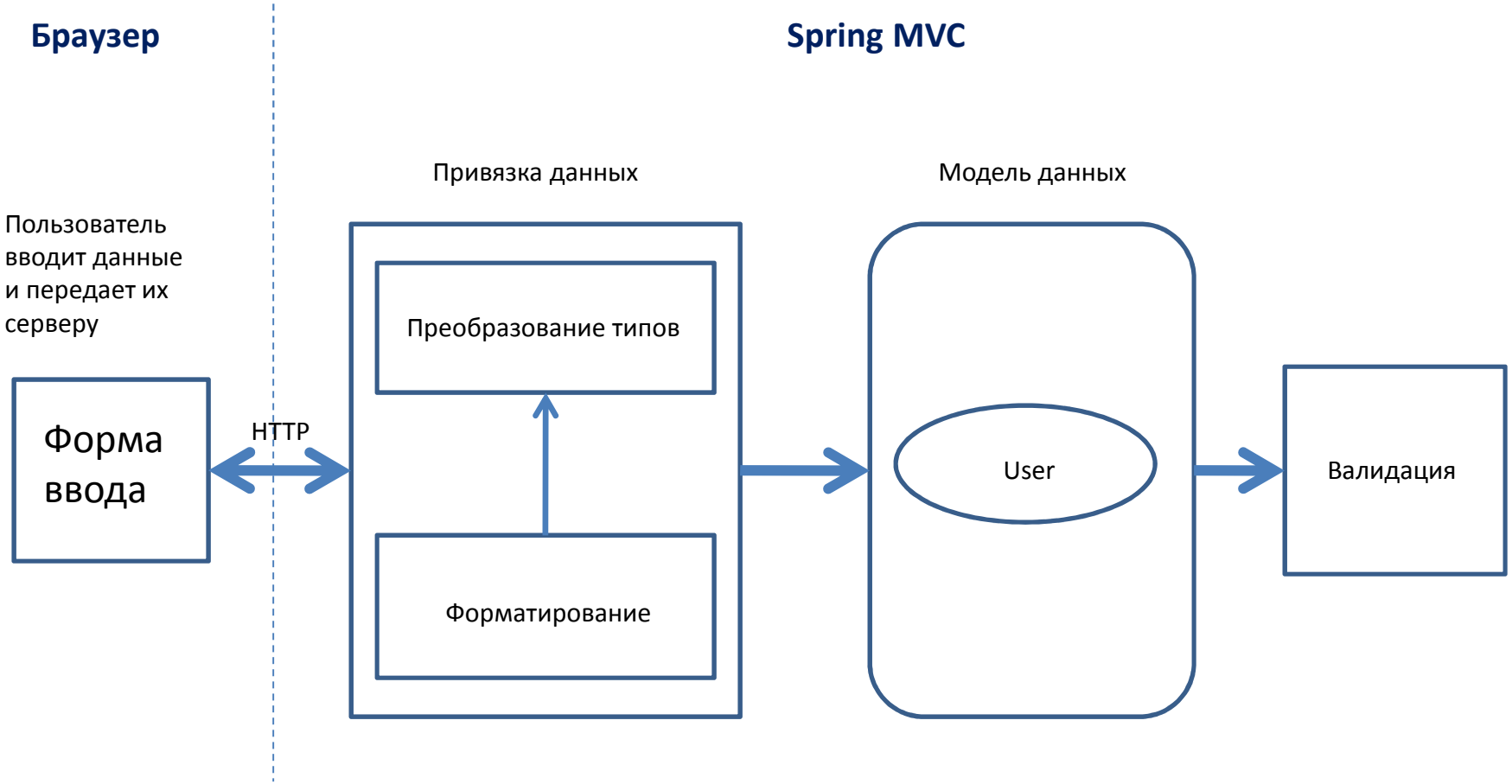
Результат

Users Listing

Name	*Email*	*Birth Date*		
Ivanov	ivan@mail.ru	02.12.1989	Edit	Delete
Kruglov	krug@mail.ru	14.09.1983	Edit	Delete

Add new user

Построение модели



Система преобразования типов

В Spring 3.0 появилась обобщенная система преобразования типов, которая находится в пакете org.springframework.core.convert. Эта система позволяет выполнять преобразования между произвольными типами на любом уровне приложения.

Для добавления собственного преобразователя типа A в тип B нужно создать класс, реализующий интерфейс Converter<A, B> и добавить его в конфигурацию службы преобразования.

```
public class StringToDateTimeConverter implements Converter<String, DateTime>
…
<bean id="conversionService" class="org.springframework.context.support.ConversionServiceFactoryBean">
   <property name="converters">
      <list>
         <bean class="com.simbircite.converters. StringToDateTimeConverter " />
      </list>
…

@Inject
ConversionService conversionService;
…

DateTime startDate = conversionService.convert("01-02-2014", DateTime.class);
```

Spring MVC широко использует службу преобразования. В системе уже присутствует множество стандартных преобразователей. Например, StringToArrayConverter, StringToBoolenConverter и т.п.

Форматирование полей

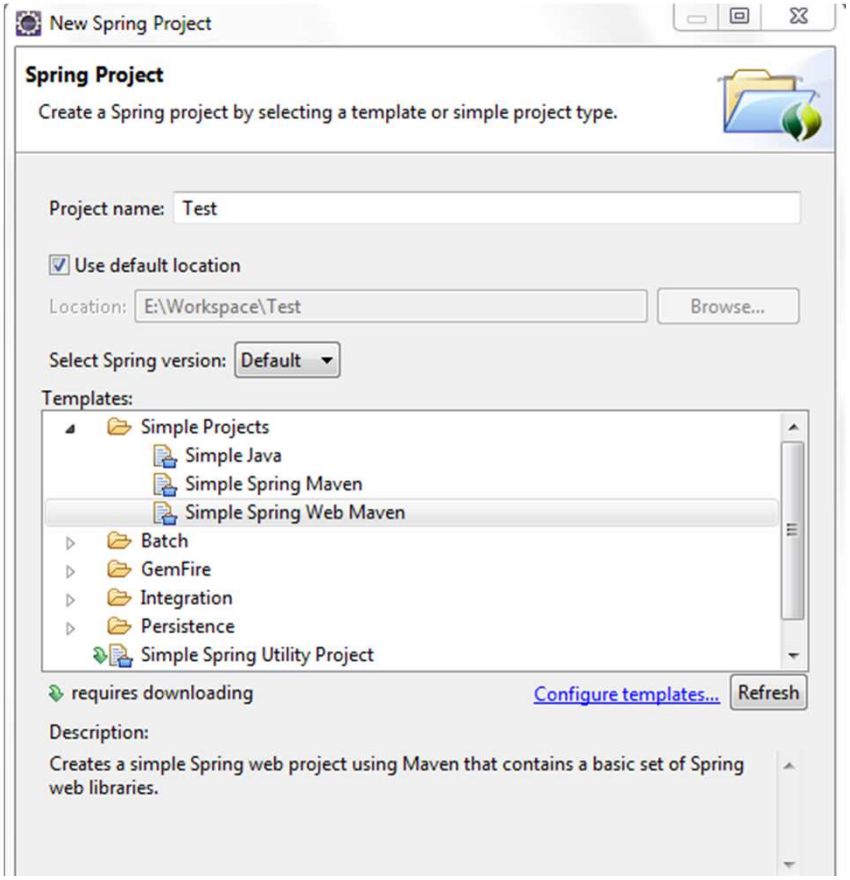
По мимо преобразования типов разработчикам доступно другое важное средство – Formatter SPI, которое позволяет настроить все аспекты, связанные с форматированием полей.

Основным интерфейсом является org.springframework.format.Formatter<T>

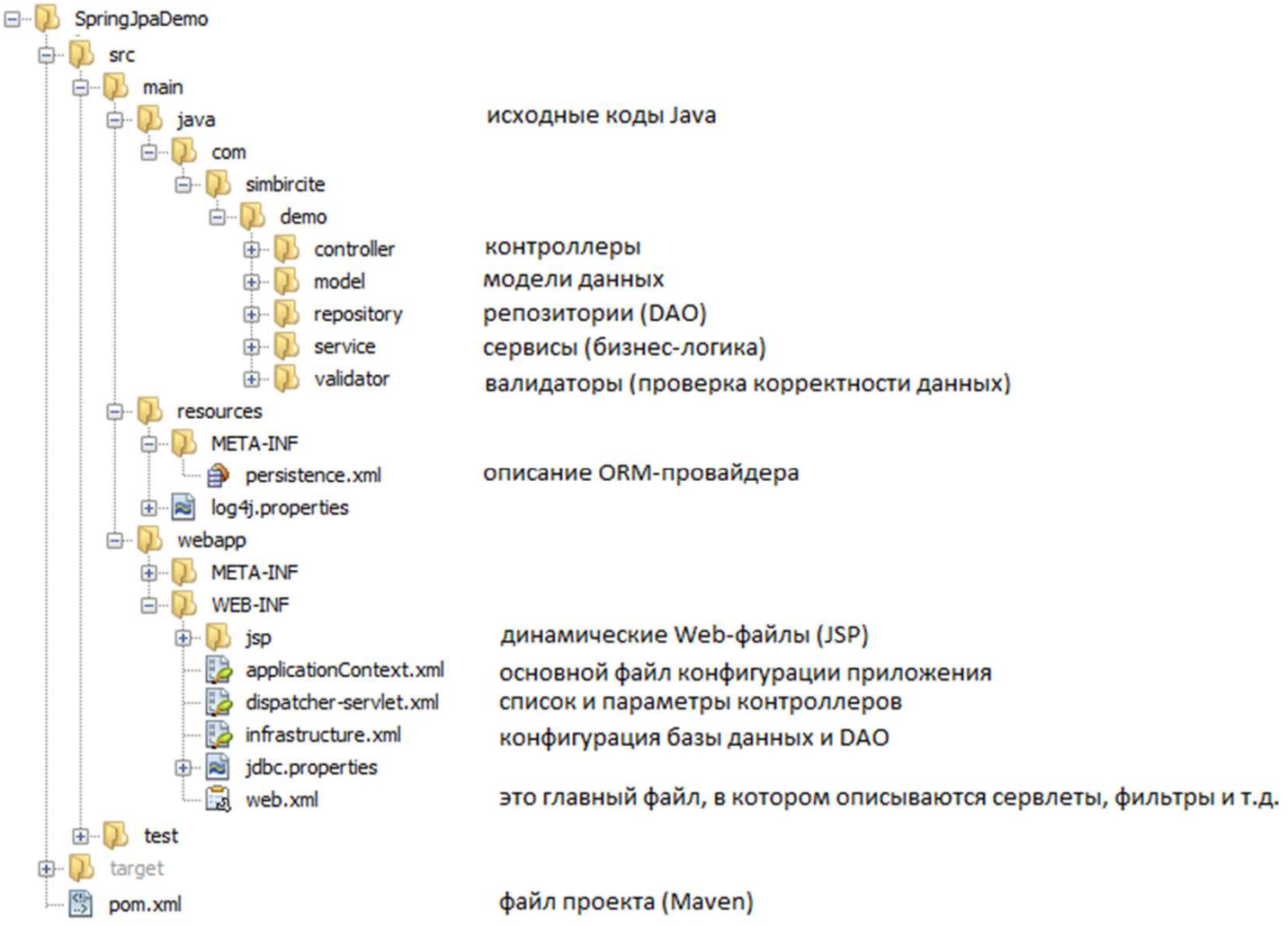
Spring предоставляет несколько реализаций для часто используемых типов. Например, DateFormatter, CurrencyFormatter, NumberFormatter и т.п.

Создание проекта в STS

STS – Spring Tool Suite - плагин для Eclipse, упрощающий разработку Spring-приложений



Структура каталога приложения



Дескриптор веб-развертывания


```
<context-param>
   <!-- Конфигурационные файлы Spring  -->
   <param-name>contextConfigLocation</param-name>
   <param-value>
      classpath*:META-INF/spring/applicationContext.xml
      classpath*:META-INF/spring/applicationContext-infrastructure.xml
   </param-value>
</context-param>
<listener>
   <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<servlet>
   <!-- Сервлет диспетчера. Анализирует запросы и направляет их
        соответствующему контроллеру для обработки -->
   <servlet-name>applicationServlet</servlet-name>
   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
   <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/spring/applicationServlet.xml</param-value>
   </init-param>
</servlet>
<servlet-mapping>
   <servlet-name>applicationServlet</servlet-name>
   <url-pattern>/</url-pattern>
</servlet-mapping>
```

Файл /src/main/webapp/WEB-INF/web.xml

Конфигурация Spring MVC


```
<!-- В этом файле размещены список и параметры контроллеров.

    Контроллеры - это тот слой, в котором должна выполняться логика обработки форм,
    ответы сервера, разбор параметров, принятых со страницы и т.п.
-->
<beans>

    <!-- Сообщаем, что наше MVC приложение основано на аннотациях  -->
    <mvc:annotation-driven/>

    <!-- Определяем пакет, в котором размещены классы контроллеров -->
    <context:component-scan base-package="com.simbircite.demo.controller"/>

    <!-- Определяем правило для определения пути к файлу с JSP шаблоном
         по логическому имени вида.

         Т.е. если вид называется users, то его шаблон описан в файле
         /WEB-INF/views/users.jspx
    -->
    <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/views/"/>
        <property name="suffix" value=".jspx"/>
    </bean>

</beans>
```

Файл /src/main/webapp/WEB-INF/spring/applicationServlet.xml

Настройка контекста приложения


```
<beans>

    <!-- Включаем использование аннотаций для конфигурации Spring -->
    <context:annotation-config/>

    <!-- Определяем бин для валидации модели User  -->
    <bean name="userValidator" class="com.simbircite.demo.validator.UserValidator"/>

    <!-- Определяем бин сервиса для работы с моделью User  -->
    <bean name="userService" class="com.simbircite.demo.service.UserService"/>

</beans>
```

Файл /src/main/resources/META-INF/spring/applicationContext.xml

Конфигурация БД и DAO


```
<beans>
   <!-- Указываем пакет, в котором содержатся классы DAO -->
   <jpa:repositories base-package="com.simbircite.demo.repository" />

   <!-- Определяем менеджер транзакций -->
   <bean id="transactionManager"
       class="org.springframework.orm.jpa.JpaTransactionManager"
       p:entityManagerFactory-ref="entityManagerFactory"/>

   <!-- Подключаем файл jdbc.properties с настройками подключения к БД -->
   <bean id="propertyConfigurer"
       class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"
       p:location="classpath:jdbc.properties" />

   <!-- Определяем источник данных (подключение к БД) -->
   <bean id="dataSource"
       class="org.springframework.jdbc.datasource.DriverManagerDataSource"
       p:driverClassName="${jdbc.driverClassName}"
       p:url="${jdbc.url}" p:username="${jdbc.username}"  p:password="${jdbc.password}" />

   <!-- Определяем менеджер для работы с сущностями JPA -->
   <bean id="entityManagerFactory"
       class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
       p:dataSource-ref="dataSource"  p:persistenceUnitName="persistenceUnit">
     <property name="jpaVendorAdapter">
       <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter" />
     </property>
   </bean>
</beans>
```

Файл /src/main/resources/META-INF/spring/applicationContext-infrastructure.xml

Резюме

В этой лекции был представлен высокоуровневый взгляд на платформу Spring Framework. После прочтения лекции вы должны получить определенное представление о возможностях Spring и о технологиях, которые она использует.

Так же здесь были представлены основные сведения, необходимые для подготовки и запуска Spring, и был рассмотрен пример простого Web-приложения.

Спасибо за внимание!