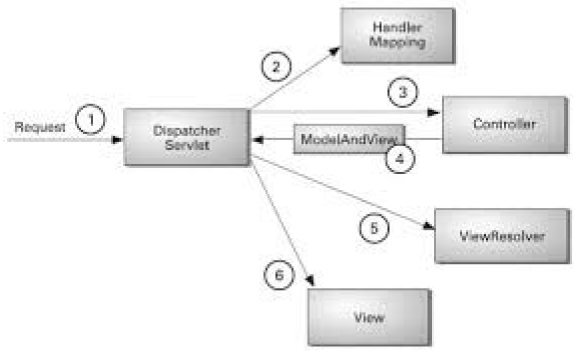
Spring MVC

Spring 4.X with Annoation



Agenda:

1. Hello World
2. Spring 4 mvc form processing example 1
3. Discussion on Map vs ModelMap vs Model
4. Spring mvc crud application
5. Spring mvc crud application With prepoputated values
   @ModelAttribute annotation on an method
6. Spring mvc form validation
7. Interceptror

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Spring 4 hello world

Step:

1. create maven project add to pom file (attached pom)

2. map controller in web.xml

```
<servlet>
   <servlet-name>dispatcher</servlet-name>
   <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
   </servlet-class>
   <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/spring-servlet.xml</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
 </servlet>

 <servlet-mapping>
   <servlet-name>dispatcher</servlet-name>
   <url-pattern>/</url-pattern>
 </servlet-mapping>
```

3. create dispatcher-servlet.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
    http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-
4.0.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context-4.0.xsd">

    <context:component-scan base-package="com.controller" />

    <mvc:annotation-driven />

    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix">
            <value>/WEB-INF/views/</value>
        </property>
        <property name="suffix">
            <value>.jsp</value>
        </property>
    </bean>

</beans>
```

com.controller

4. create hello world controller

```
@Controller
@RequestMapping("/")
public class HelloWorldController {

    @RequestMapping(method = RequestMethod.GET)
    public String sayHello(ModelMap model) {
        model.addAttribute("greeting", "Hello World from Spring 4 MVC");
        return "welcome";
    }

    @RequestMapping(value="/helloagain", method = RequestMethod.GET)
    public String sayHelloAgain(ModelMap model) {
        model.addAttribute("greeting", "Hello World Again, from Spring 4 MVC");
        return "welcome";
        }

    }
```

5. create hello world view welcome.jsp

```
Greeting : ${greeting}
```

Play with the code !

Discussion on Map<String, Object> vs ModelMap vs Model ModelAndView
----

ModelMap
===========

ModelMap subclasses LinkedHashMap, and provides
some additional conveniences to

make it a bit
easier to use by controllers

(Spring give so that u get confused!)

1. addAttribute can be called with just a value,
   and the map key is then inferred from the type.

```
List<Book>blist=new ArrayList<Book>();

Book c=new Book();

ModelMap m=new ModelMap();
m.addAttribute(blist);


bookList
```

2. The addAttribute methods all return the ModelMap,
   so you can chain method called together,
   e.g. modelMap.addAttribute('x', x).addAttribute('y',y)

3. The addAttribute methods checks that the values aren't
   null

The generic type of ModelMap is fixed at Map<String, Object>,
which is the only one that makes sense for a view model.

```
    <<Model>>
    ==========
    which provides nothing other than the addAttribute methods,
    and is implemented by the ExtendedModelMap class
```

problem el is not reconized:
----
http://stackoverflow.com/questions/793983/jsp-el-expression-is-not-evaluated

```
<%@ page isELIgnored="false" %>
```

Spring MVC annotations
============================
@Controller
@RequestMapping
@PathVariable
@RequestParam
@RequestHeader
@ModelAttribute

@PathVariable
----
http://localhost:8080/app-01-spring/hello/delete/22

```
@Controller
@RequestMapping(value="/hello/*")
public class Hello2Controller {

	@RequestMapping(value="/delete/{sid}", method=RequestMethod.GET)
	public String sayHello(@PathVariable ("sid")int s){
		Foo foo=new Foo();
		System.out.println(s);
		return "hello";
	}

}
```

foo?un=raj&pw=raj

```
@Controller
@RequestMapping("/foo")
public class AnotherController {

	public void foo(@RequestParam("un")String un, @RequestParam("pw")String pw){
		System.out.println("un"+un);
		System.out.println("pw"+pw);
```

```
        }
}
```

What if the name of configuation file name should not be FC-servlet.xml
----

Use init parameter for filterdispacher servlet

```
<servlet>
        ....
        ....
        <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/servletContext.xml</param-value>
        </init-param>

        ...
        ...
</servlet>
```

What happens?

=> This creates a single Spring application context within the setting of the DispatcherServlet and instructs the Servlet container to initialize the DispatcherServlet at startup.

=> When initialized, the DispatcherServlet loads the context configuration from the /WEB-INF/servletContext.xml file and starts the application context

=> Of course, this creates only one application context for your application, which, as previously explained, is not very flexible.

Best practices

=> Seperate context for controller and other beans related to model and service layer

How to do it?
----

service-configuration.xml

=> Use ContextLoaderListener (aka ServletContextListner) that can load some extra configuration files for you!

```
        <context-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/otherContext.xml</param-value>
```

```
	</context-param>

	<listener>
		<listener-class>
			org.springframework.web.context.ContextLoaderListener
		</listener-class>
	</listener>
```

=> NOW WE CAN DEFINE OUR MODEL AND SERVICE LAYER RELATED BEANS IN otherContext.xml
(Refer extra notes)

A big problem!
================

=> beans are created twice!
How to stop!

First, the otherContext.xml configuration adds an exclusion to the scanning: (blacklist approach)

```
  <context:annotation-config></context:annotation-config>

  <context:component-scan base-package="com.demo">
	<context:exclude-filter type="annotation"
	expression="org.springframework.stereotype.Controller"/>
  </context:component-scan>
```

//it is not necesssary to change
survival is not mandatory

- w edwards deming

The servletContext.xml configuration uses a whitelist instead of a blacklist to tell Spring which components to scan for?

```
  <context:annotation-config/>
  <context:component-scan base-package="com.demo" use-default-filters="false">
	<context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"/>
  </context:component-scan>
```

Spring 4 mvc form processing (imp)
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxXXX

backing bean idea!

1. create dynamic web project and add jar

2. map controller in web.xml

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

3. create dispatcher-servlet.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
	xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
						http://www.springframework.org/schema/beans/spring-beans-
4.0.xsd
						http://www.springframework.org/schema/context
						http://www.springframework.org/schema/context/spring-context-
4.0.xsd
						http://www.springframework.org/schema/mvc
						http://www.springframework.org/schema/mvc/spring-mvc-
4.0.xsd">

    <context:component-scan base-package="com.controller" />
    <context:annotation-config />

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

```
</beans>
```

4.

Creating an form

bookform.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<form:form action="addBook" method="post" commandName="book">
	Enter book isbn:<form:input path="isbn"/><br/>
	Enter book title:<form:input path="title"/><br/>
	Enter book author:<form:input path="author"/><br/>
	Enter book price:<form:input path="price"/><br/>
	<input type ="submit"/>
</form:form>
```

booksuccess.jsp

```
${book.title }
```

5. create an backing form bean (it is acting both as backing form bean and dto)
----

com.book.model.persistance

```
public class Book {
	private int id;
	private String isbn;
	private String title;
	private String author;
	private double price;
}
```

```
public interface BookDao{
	public List<Book> getAllBooks();
	public Book getBookById(int bookId);
	public Book addBook(Book book);
	public Book updateBook(Book book);
	public Book removeBook(int bookId);
}
```

```
@Service
public class BookDaoImp implements BookDao {

	private static Map<Integer, Book> books = new HashMap<Integer, Book>();
	static {
		books.put(1, new Book(121, "ABC123", "head first", "katthy", 500.00));
		books.put(1, new Book(11, "ABU123", "head last", "amit", 400.00));
	}

	@Override
	public List<Book> getAllBooks() {
		return new ArrayList<Book>(books.values());
	}

	@Override
	public Book getBookById(int bookId) {
		return books.get(bookId);
	}

	@Override
	public Book addBook(Book book) {
		book.setId(books.size() + 1);
		books.put(book.getId(), book);
		return book;
	}

	@Override
	public Book updateBook(Book book) {
		if (book.getId() <= 0)
			return null;
		else
			books.put(book.getId(), book);
		return book;
	}

	@Override
	public Book removeBook(int bookId) {
		return books.remove(bookId);
	}

}
```

with hibernate/JPa
----

com.book.model.persistance

```
@Entity
public class Book {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int id;
	private String isbn;
	private String title;
	private String author;
	private double price;
}
```

Design persistance layer, service layer:

```
public interface BookDao {
	public List<Book> getAllBooks();
	public Book getBookById(int bookId);
	public Book addBook(Book book);
	public Book updateBook(Book book);
	public Book removeBook(int bookId);
}



@Repository
public class BookDaoImp implements BookDao {

	@PersistenceContext
	private EntityManager em;

	@Override
	public List<Book> getAllBooks() {
		return em.createQuery("from Book").getResultList();
	}

	@Override
	public Book getBookById(int bookId) {
		return em.find(Book.class, bookId);
	}

	@Override
	public Book addBook(Book book) {
		em.persist(book);
		em.flush();
		return book;
	}

	@Override
	public Book updateBook(Book book) {
		return em.merge(book);
	}

	@Override
	public Book removeBook(int bookId) {
		Book book = em.find(Book.class, bookId);
		if (book != null)
			em.remove(bookId);
		return book;
	}
}
```

com.book.model.service

```
public interface BookService {
                public List<Book> getAllBooks();
                public Book getBookById(int bookId);
                public Book addBook(Book book);
                public Book updateBook(Book book);
                public Book removeBook(int bookId);
        }


@Service
@Transactional
public class BookServiceImp implements BookService {

        @Autowired
        private BookDao dao;

        @Override
        public List<Book> getAllBooks() {
                return dao.getAllBooks();
        }

        @Override
        public Book getBookById(int bookId) {
                return dao.getBookById(bookId);
        }

        @Override
        public Book addBook(Book book) {
                return dao.addBook(book);
        }

        @Override
        public Book updateBook(Book book) {
                return dao.updateBook(book);
        }

        @Override
        public Book removeBook(int bookId) {
                return dao.removeBook(bookId);
        }

}
```

----

6. create an controller

com.book.controllers

```
@Controller
@RequestMapping(value="/addBook")
public class BookController {

	@Autowired
	private BookService service;

	@RequestMapping(method=RequestMethod.GET)
	public String showBookForm(ModelMap map){
		Book book=new Book();
		map.addAttribute("book",book);
		return "bookform";
	}

	@RequestMapping(method=RequestMethod.POST)
	public ModelAndView submittedBookForm(Book book){
		service.addBook(book);
		System.out.println("book is added");
		return new ModelAndView("booksuccess" ,"book",book);
	}
}
```

Hashtable

nested form bean?

Note:
xxxxxxxxxx

If we user map.addAttribute("command",book);
then no need to apply commandName="book" in
<form:form action="add.htm" method="post" commandName="book">

persistance.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
	<persistence-unit name="curd" transaction-type="RESOURCE_LOCAL">
	<provider>org.hibernate.ejb.HibernatePersistence</provider>
		<class>com.book.model.persistance.Book</class>
	</persistence-unit>
</persistence>
```

spring configuration file: spring-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
	xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:tx="http://www.springframework.org/schema/tx"
	xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
		http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd
		http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">



	<context:annotation-config />
	<context:component-scan base-package="com" />
	<mvc:annotation-driven />




	<bean id="dataSource"
		class="org.springframework.jdbc.datasource.DriverManagerDataSource">

		<property name="driverClassName" value="com.mysql.jdbc.Driver" />
		<property name="url" value="jdbc:mysql://localhost:3306/foo" />
		<property name="username" value="root" />
		<property name="password" value="root" />
	</bean>

	<bean
		class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
		id="entityManagerFactory">
		<property name="persistenceUnitName" value="curd" />
		<property name="dataSource" ref="dataSource" />

	</bean>

	<tx:annotation-driven />

	<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
		<property name="entityManagerFactory" ref="entityManagerFactory" />
	</bean>


	<bean
		class="org.springframework.web.servlet.view.InternalResourceViewResolver">
		<property name="prefix">
			<value>/WEB-INF/views/</value>
		</property>
```

```
			<property name="suffix">
				<value>.jsp</value>
			</property>
		</bean>
</beans>
```

persistance.xml

```
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
		<persistence-unit name="curd" transaction-type="RESOURCE_LOCAL">
		<provider>org.hibernate.ejb.HibernatePersistence</provider>
			<class>com.rock.model.persitance.Book</class>
			<properties>
			<property name="hibernate.dialect" value="org.hibernate.dialect.DerbyDialect"/>
				<property name="hibernate.hbm2ddl.auto" value="create" />
				<property name = "hibernate.show_sql" value = "true" />
			</properties>
		</persistence-unit>
</persistence>
```

best practice
----
db.properties

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/foo
jdbc.username=root
jdbc.password=root
```

```
		<bean
			class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
			<property name="locations" value="classpath:db.properties"></property>
		</bean>
```

```
		<bean id="dataSource"
			class="org.springframework.jdbc.datasource.DriverManagerDataSource">

			<property name="driverClassName" value="${jdbc.driverClassName}" />
			<property name="url" value="${jdbc.url}" />
			<property name="username" value="${jdbc.username}" />
			<property name="password" value="${jdbc.password}" />
		</bean>
```

Spring mvc crud application

```
@Controller
@RequestMapping
public class BookController3 {

	@Autowired
	private BookService service;

	@RequestMapping(value="viewAll" , method=RequestMethod.GET)
	public ModelAndView viewAll(){
		ModelAndView m=new ModelAndView();
		m.setViewName("showAllbooks");
		m.addObject("books",service.getAllBooks());
		return m;
	}

	@RequestMapping(value="addBook", method=RequestMethod.GET)
	public String showBookForm(ModelMap map){
		Book book=new Book();
		map.addAttribute("book",book);
		return "bookform";
	}

	@RequestMapping(value="addBook", method=RequestMethod.POST)
	public String submittedBookForm(Book book){
		service.addBook(book);
		return "redirect:viewAll";
	}
}
```

showAllbooks.jsp

```
<%@taglib prefix="frm" uri="http://www.springframework.org/tags/form"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>



<body>
	<table>
		<thead>
			<tr>
				<th>book isbn</th>
				<th>book title</th>
				<th>book author</th>
				<th>book price</th>
			</tr>
```

```
            </thead>

            <tbody>

                <c:forEach var="b" items="${books}">
                    <tr>
                        <td>${b.isbn}</td>
                        <td>${b.title}</td>
                        <td>${b.author}</td>
                        <td>${b.price}</td>
                    </tr>
                </c:forEach>

            </tbody>
        </table>
        <a href="addBook">Add new Book</a>
</body>
```

=> validation
=>@ModelAttribute: annotation used in spring mvc:
  can be applied at 2 places, cant be applied on class
  ----
  method level
  method argument

@ModelAttribute annotation

Spring mvc crud application With prepoputated values
    @ModelAttribute annotation on an method
----

  two use of @ModelAttribute

  1. Annotated inside metod argument
  2. Annotated over method, that method is guranteed to call before any request.

2 new requirments:
----
1. We want pre populated values for book type
2. add pulish date for the book

```
@Entity
public class Book {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int id;
	private String isbn;
	private String title;
	private String author;
	private double price;

	@Enumerated(EnumType.STRING)
	private BookType bookType;

	@DateTimeFormat(pattern = "dd/MM/yyyy")
	@Temporal(TemporalType.DATE)
	private Date pubDate;




public enum BookType {
	IT, MGT
}
```

==

MOST IMPORTANT: About joda-time

=> Add @DateTimeFormat(iso=ISO.DATE) to the Date type to automatically parse string value from input field.

=> But you need to formate it on jsp using fmt:formateDate tag

The @DateTimeFormat required:

Note: @DateTimeFormat(iso=ISO.DATE) required

1. joda-time dependancies
2. mapping in config file <mvc:annotation-driven/>

Why <mvc:annotation-driven/>

=> The <mvc:annotation-driven/> is a Spring 3 configuration
element that greatly simplifies Spring MVC setup.

=> This tag registers the \93HandlerMapping\94 and \93HandlerAdapter\94
 required to dispatch requests to your @Controller annotated classes.

In addition, it applies sensible defaults based on what is present
in your classpath. Such defaults include (among others) :

=> Support for formatting Number fields with @NumberFormat
 annotation

=> Support for formatting Date, Calendar, and Joda Time fields with
@DateTimeFormat annotation, if Joda Time is on the classpath

=> Support for validating @Controller annotated class inputs
with @Valid annotation, if a JSR-303 Provider is on the classpath

=> Support for reading and writing XML, if JAXB is on the
 classpath Support for reading and writing JSON,
if Jackson is on the classpath

==

Change to controller:

```
@Controller
@RequestMapping
public class BookController {

	@Autowired
	private BookService service;

	@RequestMapping(value="viewAll" , method=RequestMethod.GET)
	public ModelAndView viewAll(){
		ModelAndView m=new ModelAndView();
		m.setViewName("showAllbooks");
		m.addObject("books",service.getAllBooks());
		return m;
	}

	@RequestMapping(value="addBook", method=RequestMethod.GET)
	public String showBookForm(ModelMap map){
		Book book=new Book();
		map.addAttribute("book",book);
		return "bookform";
	}

	@RequestMapping(value="addBook", method=RequestMethod.POST)
	public String submittedBookForm(@ModelAttribute(value="book") Book book){
		service.addBook(book);
		return "redirect:viewAll";
	}

	@ModelAttribute(value="booktypes")
	public BookType[] getGender(){
		return BookType.values();
	}
}
```

bookform.jsp

```
<form:form action="addBook" method="post" commandName="book">
	Enter book isbn:<form:input path="isbn"/><br/>
	Enter book title:<form:input path="title"/><br/>
	Enter book author:<form:input path="author"/><br/>
	Enter Book Type: <form:select path="bookType" items="${booktypes}"/><br/>
	Enter book price:<form:input path="price"/><br/>
	Enter book publish date:<form:input path="pubDate"/><br/>
	<input type ="submit"/>
</form:form>
```

showAllbooks.jsp

```
<body>

	<table>
		<thead>
			<tr>
				<th>book isbn</th>
				<th>book title</th>
				<th>book author</th>
				<th>book type</th>
				<th>book price</th>
				<th>book publish date</th>
			</tr>
		</thead>

		<tbody>

			<c:forEach var="b" items="${books}">
				<tr>
					<td>${b.isbn}</td>
					<td>${b.title}</td>
					<td>${b.author}</td>
					<td>${b.bookType}</td>
```

```
                                                <td>${b.price}</td>
                                                <td>${b.pubDate}</td>
                                        </tr>
                                </c:forEach>

                        </tbody>
                </table>
                <a href="addBook">Add new Book</a>
</body>
```

----
validation framework in java!

Spring mvc form validation using JSR 303
Hibernate validator
----

modified book class

```
@Entity
public class Book {
        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        private int id;

        @NotEmpty(message="isbn can not be empty")
        private String isbn;

        @NotEmpty(message="title can not be empty")
        private String title;

        @NotEmpty(message="author can not be empty")
        private String author;


        private double price;
        @Enumerated(EnumType.STRING)
        private BookType bookType;

        @Past
        @DateTimeFormat(pattern = "dd/MM/yyyy")
        @Temporal(TemporalType.DATE)
        private Date pubDate;
```

controller

```
@Controller
@RequestMapping
public class BookController {

	@Autowired
	private BookService service;

	@RequestMapping(value="viewAll" , method=RequestMethod.GET)
	public ModelAndView viewAll(){
		ModelAndView m=new ModelAndView();
		m.setViewName("showAllbooks");
		m.addObject("books",service.getAllBooks());
		return m;
	}

	@RequestMapping(value="addBook", method=RequestMethod.GET)
	public String showBookForm(ModelMap map){
		Book book=new Book();
		map.addAttribute("book",book);
		return "bookform";
	}

	@RequestMapping(value="addBook", method=RequestMethod.POST)
	public String submittedBookForm(@Valid Book book, BindingResult result){
		if (result.hasErrors()) {
			return "bookform";
		}
		else{
		service.addBook(book);
		return "redirect:viewAll";
		}
	}

	@ModelAttribute(value="booktypes")
	public BookType[] getGender(){
		return BookType.values();
	}
}
```

bookform.jsp

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


<form:form action="addBook" method="post" commandName="book">
	Enter book isbn:<form:input path="isbn"/><form:errors path="isbn" class="error"/><br/>
	Enter book title:<form:input path="title"/><form:errors path="title" class="error"/><br/>
	Enter book author:<form:input path="author"/><form:errors path="author" class="error"/><br/>
	Enter Book Type: <form:select path="bookType" items="${booktypes}"/><br/>
	Enter book price:<form:input path="price"/><form:errors path="isbn" class="error"/><br/>
	Enter book publish date:<form:input path="pubDate"/><form:errors path="pubDate"
class="error"/><br/>
	<input type ="submit"/>
</form:form>
```

put style in head section :

```
<style>
.error {
color: #EF1313;
font-style: italic;
}
</style>
```

Putting messages from external file
----
messages.properties

```
NotEmpty.book.isbn=isbn can not be blank
```

How spring come to know about it?

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

restful web service

```
	@RequestMapping(value = "/api/messages/{id}", method = RequestMethod.GET, produces =
MediaType.APPLICATION_JSON_VALUE)
	public ResponseEntity<Message> getMessageById(@PathVariable("id") Integer id) {
		Message message = service.getMessageById(id);
		if (message == null) {
			return new ResponseEntity<Message>(HttpStatus.NOT_FOUND);
		} else
			return new ResponseEntity<Message>(message, HttpStatus.OK);
	}

	@RequestMapping(value = "/api/Messages", method = RequestMethod.POST, consumes =
MediaType.APPLICATION_JSON_VALUE, produces = MediaType.APPLICATION_JSON_VALUE)
	public ResponseEntity<Message> createMessage(@RequestBody Message message) {
		Message savedMessage = service.addMessage(message);
		return new ResponseEntity<Message>(savedMessage, HttpStatus.CREATED);
	}

	@RequestMapping(value = "/api/Messages/{id}", method = RequestMethod.PUT, consumes =
MediaType.APPLICATION_JSON_VALUE, produces = MediaType.APPLICATION_JSON_VALUE)
	public ResponseEntity<Message> updateMessage(@RequestBody Message message) {
		Message updatedMessage = service.updateMessage(message);
		return new ResponseEntity<Message>(updatedMessage, HttpStatus.OK);
	}

	@RequestMapping(value = "/api/Messages/{id}", method = RequestMethod.DELETE)
	public ResponseEntity<Message> deleteMessage(@PathVariable("id") Integer id)throws Exception
{
		service.removeMessage(id);
		return new ResponseEntity<Message>(HttpStatus.NO_CONTENT);
	}
```

Reference:
----
http://www.baeldung.com/spring-mvc-static-resources
https://www.mkyong.com/spring-mvc/spring-mvc-how-to-include-js-or-css-files-in-a-jsp-page/
mkyong
http://codetutr.com/2013/03/24/simple-spring-mvc-web-application-using-gradle/
Spring in action
http://codetutr.com/2013/04/06/spring-mvc-form-submission/
http://georgemao.wordpress.com/2013/02/14/comparsion-struts-2-vs-spring-3-mvc/
http://viralpatel.net/blogs/spring-3-mvc-handling-forms
http://viralpatel.net/blogs/spring-mvc-hashmap-form-example/
http://viralpatel.net/blogs/spring-mvc-multi-row-submit-java-list/
http://www.giuseppeurso.eu/en/check-authentication-using-spring-mvc-and-handler-interceptor/
http://viralpatel.net/blogs/spring-mvc-interceptor-example/

http://viralpatel.net/blogs/spring-mvc-cookie-example/
http://stackoverflow.com/questions/18791645/how-to-use-session-attributes-in-spring-mvc
http://stackoverflow.com/questions/3423262/what-is-modelattribute-in-spring-mvc
http://www.keepsnowballing.com/2013/06/spring-mvc-jquery-sample-tutorial.html
http://www.codebeach.com/2008/06/spring-mvc-application-architecture.html
http://www.intertech.com/Blog/understanding-spring-mvc-model-and-session-attributes/

Spring mvc hello world! java configuration
----
http://websystique.com/springmvc/spring-4-mvc-helloworld-tutorial-annotation-javaconfig-full-example/

```
 <mvc:annotation-driven validator="validator" />
 <bean id="messageSource"
        class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
       <property name="basename" value="classpath:messages" />
   </bean>

   <bean id="validator" class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">
       <property name="validationMessageSource" ref="messageSource"/>
   </bean>
```

Root Application Context vs WebApplicationContext

```
   ContextLoaderListener     |-------- WebApplicationContext 1
Root Application Context----|--------- WebApplicationContext 2
                            |---------- WebApplicationContext 3
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
        version="2.0" xmlns="http://java.sun.com/xml/ns/persistence">
        <persistence-unit name="demo" transaction-type="RESOURCE_LOCAL">
                <properties>
                        <property name="javax.persistence.jdbc.driver"
value="org.apache.derby.jdbc.ClientDriver" />
                        <property name="javax.persistence.jdbc.url"
value="jdbc:derby://localhost:1527/demodb" />
```

```
					<property name="javax.persistence.jdbc.user" value="root" />
					<property name="javax.persistence.jdbc.password" value="root" />
					<property name="hibernate.hbm2ddl.auto" value="create"/>
					<property name = "hibernate.show_sql" value = "true" />
			</properties>
	</persistence-unit>
</persistence>
```

Spring mvc hello world! java configuration
----

xml vs Java configuration?
----
	xml + java

=> java configration
=. spring boot : zero configuration
		java configration

Step 1: first we need to replace dispatcher-servlet.xml with java code

	What we have mentioned in dispatcher-servlet.xml?

	1. which package to scan
	2. view resolver

configuration for bootstrapping

```
@Configuration // replacement of xml file, telling spring it is configuration file
@ComponentScan(basePackages={"com"})
@EnableWebMvc
public class AppConfig extends WebMvcConfigurerAdapter{

   @Bean
   public InternalResourceViewResolver getInternalResourceViewResolver() {
      InternalResourceViewResolver resolver = new InternalResourceViewResolver();
      resolver.setPrefix("/WEB-INF/pages/");
      resolver.setSuffix(".jsp");
      return resolver;
   }

	@Override
	public void addResourceHandlers(ResourceHandlerRegistry registry) {
		   // Don't forget the ending "/" for location or you will hit 404.
	   registry.addResourceHandler("/resources/**").addResourceLocations("/resources/");

	}
}
```

Or even we can replace everything from web.xml

```
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
                <version>2.2</version>
                        <!-- ignore missing web.xml error -->
                        <configuration>
                                <failOnMissingWebXml>false</failOnMissingWebXml>
                        </configuration>
</plugin>
```

web.xml spring config, model

now how to replace web.xml?

```
public class WebInitilizer extends
                AbstractAnnotationConfigDispatcherServletInitializer {

        @Override
        protected Class<?>[] getRootConfigClasses() {

                return null;
        }

        @Override
        protected Class<?>[] getServletConfigClasses() {

                return new Class[]{AppConfig.class};
        }

        @Override
        protected String[] getServletMappings() {

                return new String[]{"/"};
        }

}
```

hello world controller:

```
@Controller
@RequestMapping("/")
public class HelloWorldController {
          @RequestMapping(method = RequestMethod.GET)
```

```
                public String sayHello(ModelMap model) {
                    model.addAttribute("greeting", "Hello World from Spring 4 MVC");
                    return "welcome";
                }
 }
```

http://www.kubrynski.com/2014/01/understanding-spring-web-initialization.html
https://www.mkyong.com/spring-mvc/gradle-spring-4-mvc-hello-world-example-annotation/