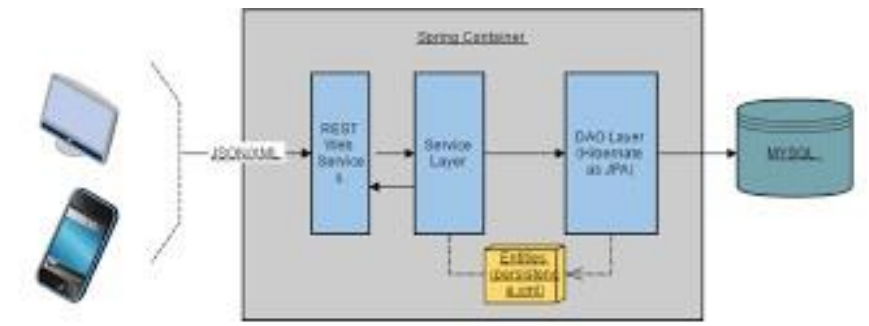
- bookappspringhib
  - Deployment Descriptor: bookappspringhib
  - JAX-WS Web Services
  - Java Resources
    - src
      - com.app.model.controller
        - BookController.java
      - com.app.model.dao
        - Book.java
        - BookDao.java
        - BookDaoImpl.java
      - com.app.model.service
        - BookService.java
        - BookServiceImpl.java
    - Libraries
  - JavaScript Resources
  - build
  - WebContent
    - META-INF
    - WEB-INF
      - lib
      - views
        - fc-configuration.xml



spring HIBERNATE

book isbn	book title	book author	book price	
1338	c is c@@22	gjhgjh	777.0	delete update

Add new Book

Step 1: DAO, DTO

Dao Layer

```
import java.util.Date;
@Entity
@Table(name="b")
public class Book {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)

    private int id;
    @Column(nullable=false, unique=true)
    private String isbn;
    private String title;
    private String author;
    private Double price;
    private Date pubDate;
```

```
public interface BookDao {
    public List<Book> getAll();
    public Book add(Book book);
    public Book delete(int bookId);
    public Book update(Book book);
    public Book getBookById(int bookId);
    public Book getBookByIsbn(String isbn);

}
```

DaoImpl

```java
@Repository
public class BookDaoImpl implements BookDao {

    @Autowired
    private SessionFactory factory;

    private Session getSession(){
        return factory.getCurrentSession();
    }
    @Override
    public List<Book> getAll() {
        return getSession().createQuery("from Book").list();
    }

    @Override
    public Book add(Book book) {
         getSession().save(book);
         return book;
    }

    @Override
    public Book delete(int bookId) {
        Book book=getBookById(bookId);
        if(book!=null)
            getSession().delete(book);
        return book;
    }
```

DaoImpl

```java
@Override
public Book update(Book book) {
    getSession().merge(book);
    return book;
}

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

@Override
public Book getBookByIsbn(String isbn) {
    return null;
}
```

Step 2: Service layer

Service Layer

```java
public interface BookService {
    public List<Book> getAll();
    public Book add(Book book);
    public Book delete(int bookId);
    public Book update(Book book);
    public Book getBookById(int bookId);
    public Book getBookByIsbn(String isbn);
}
```

```java
import java.util.List;

@Service(value="bs")
@Transactional
public class BookServiceImpl implements BookService{

    @Autowired
    private BookDao dao;
    @Override
    public List<Book> getAll() {
        return dao.getAll();
    }

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

    @Override
```

Step 3: Spring Hibernate Configuration

Configuration

```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:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context" xmlns:tx="http://www.springframework.org/schema/t
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spr
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-
        http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.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.iris.bookapp"></context:component-scan>
    <bean id="dataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="url" value="jdbc:mysql://localhost:3306/iris2" />
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="username" value="root" />
        <property name="password" value="root" />
    </bean>
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <property name="packagesToScan">
            <list>
                <value>com.iris.bookapp.model.persistance</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.hbm2ddl.auto">update</prop>
                <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
```

Data source configuration

Hibernate session factory configuration

```
driver=com.mysql.cj.jdbc.Driver
url=jdbc:mysql://localhost:3306/iris2?useSSL=false
username=root
password=root
```

Configuration

```
16          <property name="username" value="root" />
17          <property name="password" value="root" />
18      </bean>
19      <bean id="sessionFactory"
20          class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
21          <property name="dataSource" ref="dataSource" />
22          <property name="packagesToScan">
23              <list>
24                  <value>com.iris.bookapp.model.persistance</value>
25              </list>
26          </property>
27          <property name="hibernateProperties">
28              <props>
29                  <prop key="hibernate.hbm2ddl.auto">update</prop>
30                  <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
31                  <prop key="hibernate.show_sql">true</prop>
32                  <prop key="hibernate.format_sql">true</prop>
33              </props>
34          </property>
35      </bean>
36      <bean class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor" />
37      <bean id="transactionManager"
38          class="org.springframework.orm.hibernate4.HibernateTransactionManager">
39          <property name="sessionFactory" ref="sessionFactory"></property>
40      </bean>
41
42      <tx:annotation-driven transaction-manager="transactionManager" />
43
44  </beans>
```

Hibernate factory configruation

Exception translation

Configuration of hibernate tx manager

asking spring for declerative tx

Testing

```
1  public class Tester {
2
3      public static void main(String[] args) {
4
5          ApplicationContext ctx=new ClassPathXmlApplicationContext("beans.xml") ;
6          BookService bs=ctx.getBean("bs", BookService.class);
7
8          /*List<Book> allBooks=bs.getAll();
9          for(Book temp: allBooks){
0              System.out.println(temp);
1          }*/
2          Book book=new Book("454", "a"    , "b", 99.0, new Date());
3          try{
4              bs.add(book);
5          }catch(DataAccessException ex){
6              System.out.println("handled...");
7          }
8      }
9
0  }
```

Step 4: Spring Hibernate Java Configuration

```java
@Configuration
@ComponentScan(basePackages={"com.bookapp"})
@EnableAspectJAutoProxy
@PropertySource(value="db.properties")
@EnableTransactionManagement
public class ModelConfig {

    @Autowired
    private Environment environment;

    @Bean(name="dataSource")
    public DataSource getDataSource(){
        DriverManagerDataSource ds=new DriverManagerDataSource();
        ds.setDriverClassName(environment.getProperty("driver"));
        ds.setUrl(environment.getProperty("url"));
        ds.setUsername(environment.getProperty("username"));
        ds.setPassword(environment.getProperty("password"));
        return ds;
    }

    @Bean
    public LocalSessionFactoryBean getSessionFactory(){
        LocalSessionFactoryBean sf=new LocalSessionFactoryBean();
        sf.setDataSource(getDataSource());
        sf.setPackagesToScan("com.bookapp.model.persistance");
        sf.setHibernateProperties(getHibernateProperties());
        return  sf;
    }
```

```java
    public Properties getHibernateProperties() {
        Properties properties=new Properties();
        properties.setProperty("hibernate.hbm2ddl.auto", "validate");
        properties.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQLDialect");
        properties.setProperty("hibernate.show_sql", "true");
        properties.setProperty("hibernate.formate_sql", "true");
        return properties;
    }

    @Bean
    public PersistenceExceptionTranslationPostProcessor
    getPersistenceExceptionTranslationPostProcessor(){
        PersistenceExceptionTranslationPostProcessor ps=
                new PersistenceExceptionTranslationPostProcessor();
        return ps;
    }

    @Bean(name="transactionManager")
    //@Autowired
    public HibernateTransactionManager getHibernateTransactionManager(SessionFactory factory){
        HibernateTransactionManager tm=new HibernateTransactionManager();
        tm.setSessionFactory(factory);
        return tm;
    }
}
```

Step 5: Spring MVC basics

Hello world controller

```
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";
        }

    }
```

Web layer configuration

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

<!-- CONFIG INTERNAL RESO BEAN RESOLVER -->
<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>
```

Front Controller Configuration

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app 3 0.x
    version="3.0">
    <display-name>app</display-name>

    <servlet>
        <servlet-name>fc</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/fc-configuration.xml</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>fc</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

```
@PathVariable  vs @RequestParam
-----------------------------------
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 {
	@RequestMapping(method=RequestMethod.GET)
	public void foo(@RequestParam("un")String un, @RequestParam("pw")String pw){
		System.out.println("un"+un);
		System.out.println("pw"+pw);
	}
}
```

ContextLoaderListner

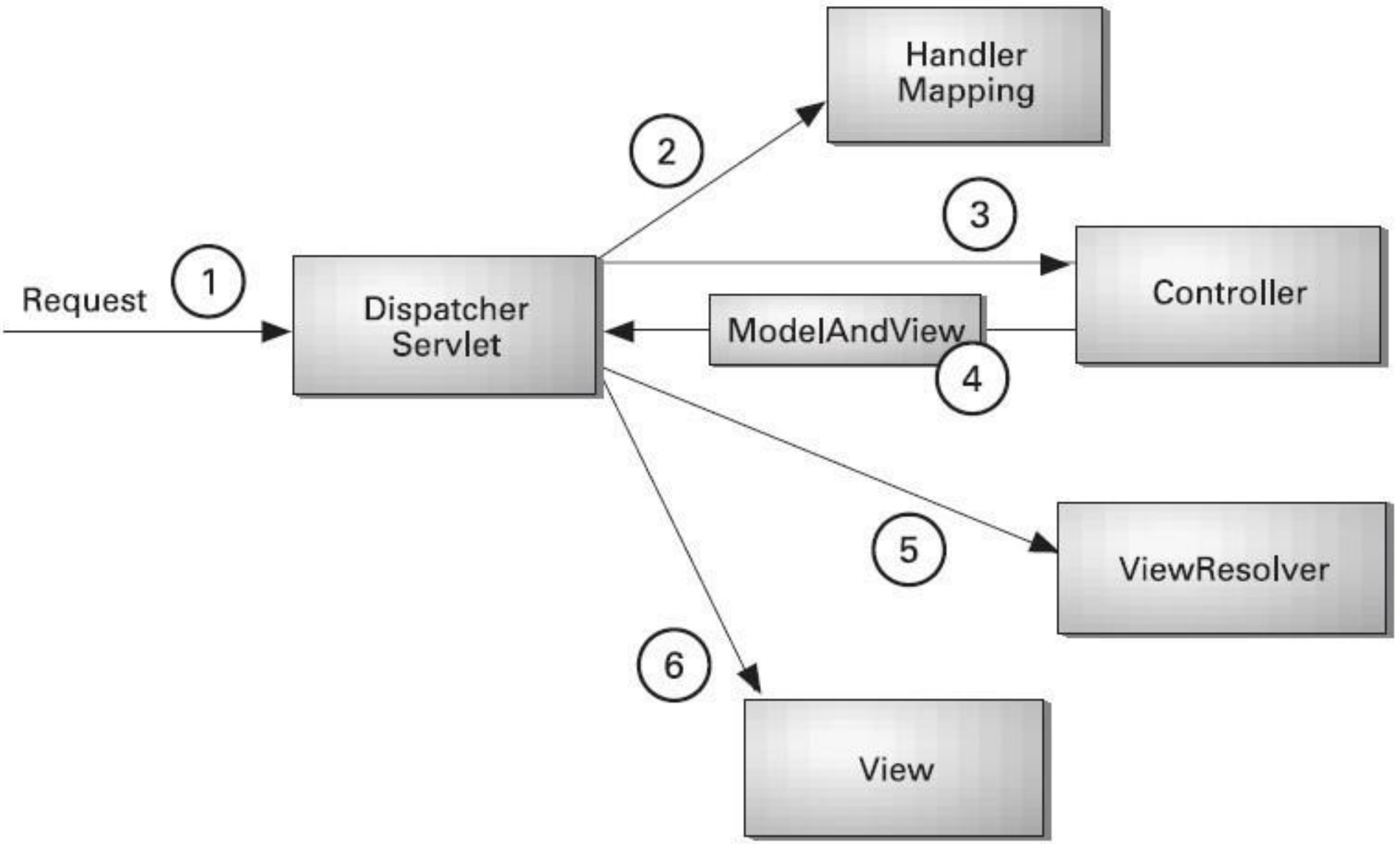
```
=> Use ContextLoaderListener (aka ServletContextListner ) that can
        load some extra configuration files for you!



<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
```

Step 6: Spring MVC java configuration

```java
@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/");

    }
}
```

```java
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[]{"/"};
    }
}
```

Step 7: Spring MVC, Hibernate


Handler
Mapping
2
3
Request
1
Dispatcher
Servlet
Controller
ModelAndView
4
5
ViewResolver
6
View

```
throws InterruptedException {
    long base = System.currentTimeMillis();
    //x long now = 0;

    if (millis < 0) {
        throw new IllegalArgumentException("timeout value is negative");
    }

    if (millis == 0) {
        while (isAlive())
        {  wait(0);
        }
    } else {
        while (isAlive())
        { long delay = millis - now; 700-0
            if (delay <= 0) {
                break;
            }
            wait(delay);
            now = System.currentTimeMillis() - base;
        }
    }
```

//case 1: 0 or nothing

//+ve case

x---
x+1

```
Thread.currentThread.join(0);
hread.currentThread.join();

hread.currentThread.join(700);
```

```java
@Controller
public class BookController {

    @Autowired
    private BookService bookService;

    @RequestMapping(value="showallbooks", method=RequestMethod.GET)
    public String showAllBooks(ModelMap map){
        map.addAttribute("books", bookService.getAllBooks());
        return "showallbooks";
    }

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

    @RequestMapping(value="updatebook", method=RequestMethod.GET)
    public String updateBook(ModelMap map, HttpServletRequest request){
        int id=Integer.parseInt(request.getParameter("id"));
        Book book=bookService.getBookById(id);
        map.addAttribute("book", book);
        return "addbook";
    }
```

```java
@RequestMapping(value="deletebook", method=RequestMethod.GET)
public String deleteBook(HttpServletRequest request){
    int id=Integer.parseInt(request.getParameter("id"));

    bookService.removeBook(id);
    return "redirect:/showallbooks";
}
//@valid must ==>BindingResult
@RequestMapping(value="addbook", method=RequestMethod.POST)
public String saveBook( @ModelAttribute(value="book") @Valid Book book, ModelMap map,
        BindingResult bindingResult  ){
    if(bindingResult.hasErrors()){
        return "addbook";
    }else{
        if(book.getId()==0)
            bookService.addBook(book);
        else
            bookService.updateBook(book);
        return "redirect:/showallbooks";
    }

}
```

Display All Books

```
<body>
<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>${b.price}</td>
                    <td><a href="deletebook?id=${b.id}">delete</a></td>
                    <td><a href="updatebook?id=${b.id}">update</a></td>
                </tr>
            </c:forEach>
        </tbody>
    </table>
    <a href="addbook">Add new Book</a>
</body>
```

Add/ Edit a Book

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<form:form action="addbook" method="post" modelAttribute="book">
<form:hidden path="id"/>
    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 price:<form:input path="price"/><br/>
    <input type ="submit"/>
</form:form>
</body>
```

```java
@Controller
public class HelloController {

    @RequestMapping("/welcome/{countryName}/{userName}")
    public ModelAndView helloWorld(@PathVariable Map<String,String> pathVars) {

        String name = pathVars.get("userName");
        String country = pathVars.get("countryName");

        ModelAndView model = new ModelAndView("HelloPage");
        model.addObject("msg","hello "+name+ " You are from "+country);

        return model;
    }
}
```

```java
  @RequestMapping("/welcome/countryName/{userName}")
  public ModelAndView helloWorld(@PathVariable("userName") String name) {

      ModelAndView model = new ModelAndView("HelloPage");
      model.addObject("msg","hello "+name);

      return model;
  }
```

```java
 @RequestMapping(value="/submitAdmissionForm.html", method = RequestMethod.POST)
 public ModelAndView submitAdmissionForm(@RequestParam("studentName") String name, @RequestParam("studentHobby") String

     ModelAndView model = new ModelAndView("AdmissionSuccess");
     model.addObject("msg","Details submitted by you:: Name: "+name+ ", Hobby: " + hobby);

     return model;
 }
```

```
some imp points:
-----------------
for converting string ==> desire data formate
-------------------------------------
@DateTimeFormat(pattern = "dd/MM/yyyy")

must use  <mvc:annotation-driven


auto populate some field:
-------------------------
Enter Book Type: <form:select path="pubName" items="${pubNammes}"/><br/>


@ModelAttribute(value="pubNammes")
        public List<String> getGender(){
                return ....;
}

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>

```

JSR 303 Validation API

- ***@NotNull*** – validates that the annotated property value is not *null*
- ***@AssertTrue*** – validates that the annotated property value is *true*
- ***@Size*** – validates that the annotated property value has a size between the attributes *min* and *max*; can be applied to *String*, *Collection*, *Map*, and array properties
- ***@Min*** – *V*alidates that the annotated property has a value no smaller than the *value* attribute
- ***@Max*** – validates that the annotated property has a value no larger than the *value* attribute
- ***@Email*** – validates that the annotated property is a valid email address

  - ***@NotEmpty*** – validates that the property is not null or empty; can be applied to *String*, *Collection*, *Map* or *Array* values
  - ***@NotBlank*** – can be applied only to text values and validated that the property is not null or whitespace
  - ***@Positive*** and ***@PositiveOrZero*** – apply to numeric values and validate that they are strictly positive, or positive including 0
  - ***@Negative*** and ***@NegativeOrZero*** – apply to numeric values and validate that they are strictly negative, or negative including 0
  - ***@Past*** and ***@PastOrPresent*** – validate that a date value is in the past or the past including the present; can be applied to date types including those added in Java 8
  - ***@Future*** *and* ***@FutureOrPresent*** – validates that a date value is in the future, or in the future including the present

8. Spring REST

```java
@RestController//   @RestController=@Controller + @ResponseBody
public class BookResources {

    @Autowired
    private BookService service;

    @RequestMapping(value = "/api/book", method = RequestMethod.GET,
            produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Collection<Book>> getAllBooks() {
        Collection<Book> greetings = service.getAllBooks();
        return new ResponseEntity<Collection<Book>>(greetings, HttpStatus.OK);
    }

    @RequestMapping(value = "/api/book/{id}", method = RequestMethod.GET,
            produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Book> getAnBook(@PathVariable Integer id) {
        Book book = service.getBookById(id);
        if (book == null) {
            return new ResponseEntity<Book>(HttpStatus.NOT_FOUND);
        }

        return new ResponseEntity<Book>(book, HttpStatus.OK);
    }
```

```java
@RequestMapping(value = "/api/book", method = RequestMethod.POST,
        consumes = MediaType.APPLICATION_JSON_VALUE, produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<Book> createBook(@RequestBody Book book) {
    Book savedBook = service.addBook(book);
    return new ResponseEntity<Book>(savedBook, HttpStatus.CREATED);
}

@RequestMapping(value = "/api/book/{id}", method = RequestMethod.PUT,
        consumes = MediaType.APPLICATION_JSON_VALUE, produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<Book> updateBook(@PathVariable Integer id,
        @RequestBody Book book) {

    service.updateBook(book);

    return new ResponseEntity<Book>(HttpStatus.OK);
}

@RequestMapping(value = "/api/book/{id}", method = RequestMethod.DELETE)
public ResponseEntity<Book> deleteBook(@PathVariable("id") Integer id)
        throws Exception {

    service.removeBook(id);

    return new ResponseEntity<Book>(HttpStatus.NO_CONTENT);
}
```

Step 9: Spring Hibernate Testing basics

```java
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations="classpath:spring-context.xml")
@TransactionConfiguration(defaultRollback=true,transactionManager="transactionManager")
public class EmployeeHibernateDAOImplTest {

	@Autowired
	private EmployeeDAO employeeDAO;

	@Test
	public void testGetEmployeeById() {
		Employee emp = employeeDAO.getEmployeeById(1L);

		assertNotNull(emp);
	}

	@Test
	public void testCreateEmployee()
	{
		Employee emp = new Employee();
		emp.setName("Emp123");
		Long key = employeeDAO.createEmployee(emp);

		assertEquals(2L, key.longValue());
	}
}
```