virtusa
Accelerating Business Outcomes

Spring MVC

C T O L

Rajkumar S

Agenda

Introduction

Available frameworks....!!!!

Problem Area

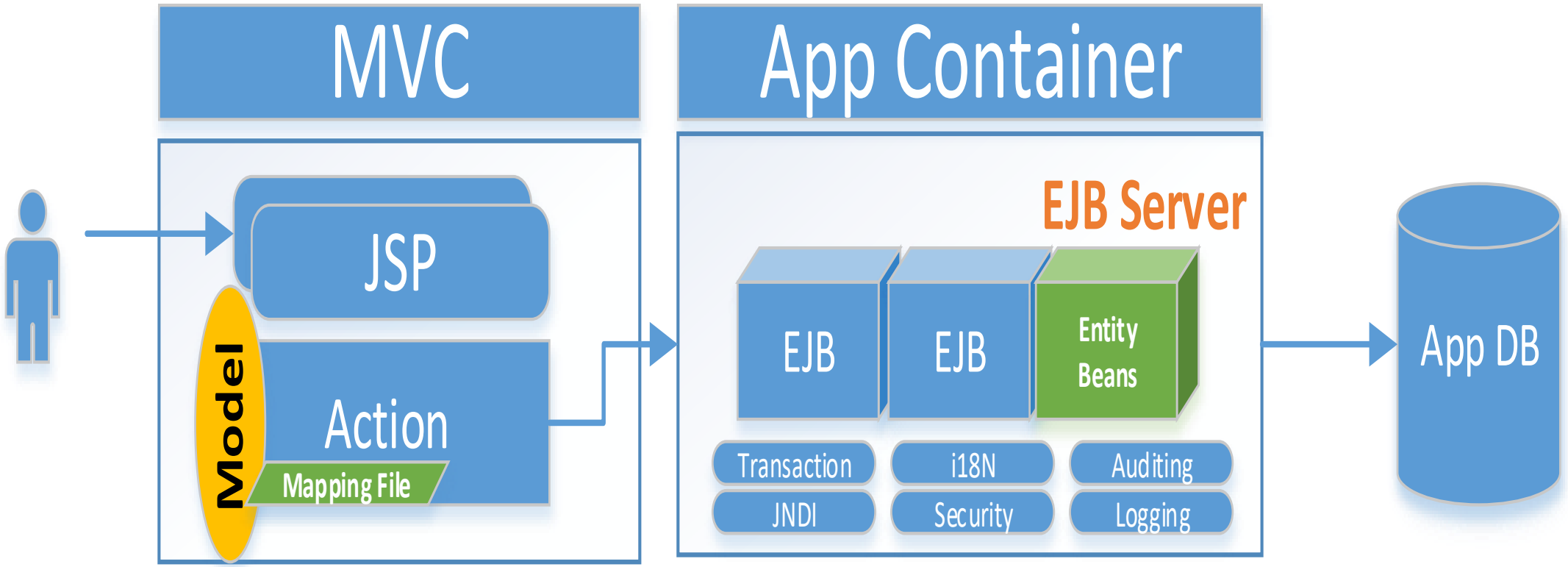
- Mixing application logic and markup is bad practice
  - Harder to change and maintain
  - Error prone
  - Harder to re-use

```
public void doGet( HttpServletRequest request, HttpServletResponse response )
{
    PrintWriter out = response.getWriter();

    out.println( "<html>\n<body>" );

    if ( request.getParameter( "foo" ).equals( "bar" ) )
        out.println( "<p>Foo is bar!</p>" );
    else
        out.println( "<p>Foo is not bar!</p>" );

    out.println( "</body>\n</html>" );
}
```

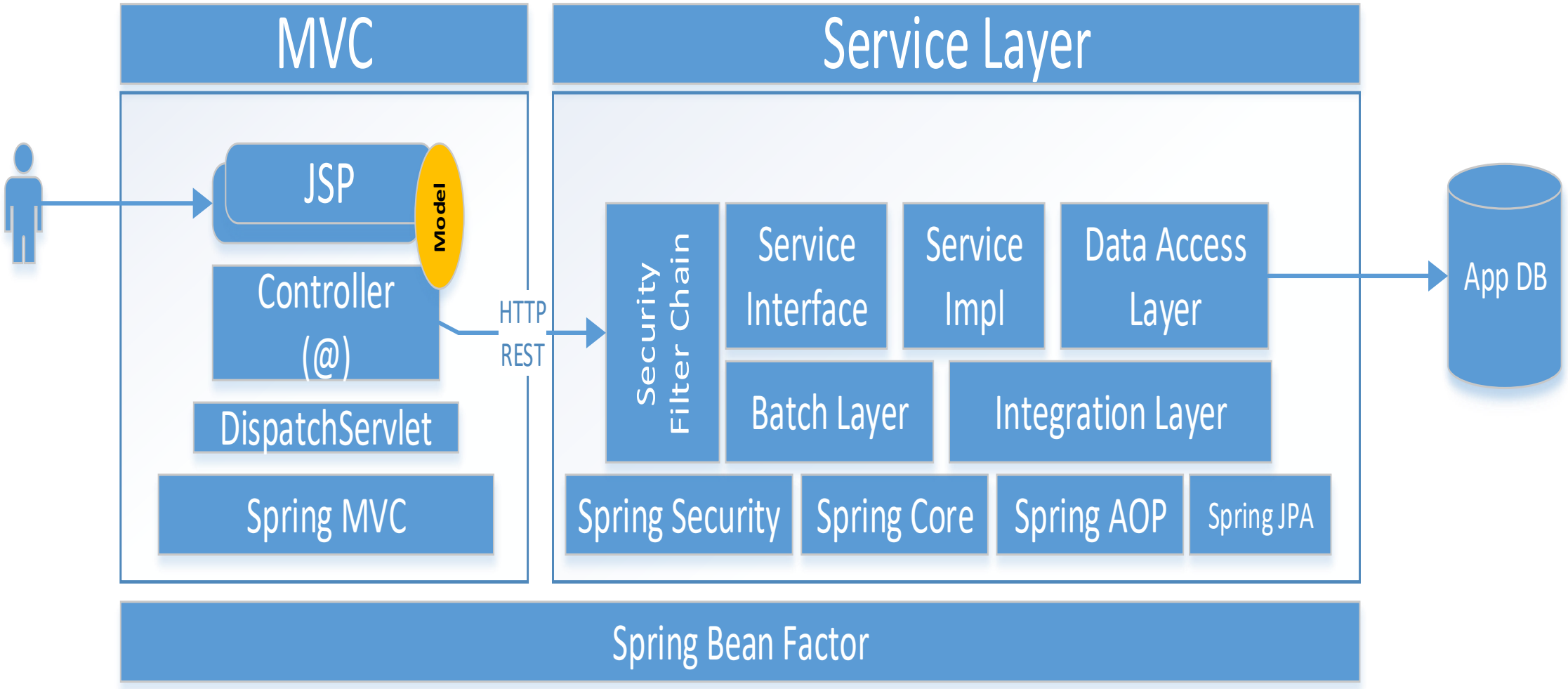
Introduction to Spring MVC

- A web framework built around the principles of Spring
- POJO based and Interface driven
- Based on a Dispatcher Servlet / Front Controller pattern
  - MVC stands for Model-View-Controller
- Very lightweight and unobtrusive compared to other frameworks
- Built from the shortcomings of Struts 1
- Support for:
  - Themes
  - Locales/i18n
  - Restful services
  - Annotation based configuration
  - Seamless integration with other Spring Services/Beans

Architecture – Pre Spring Era



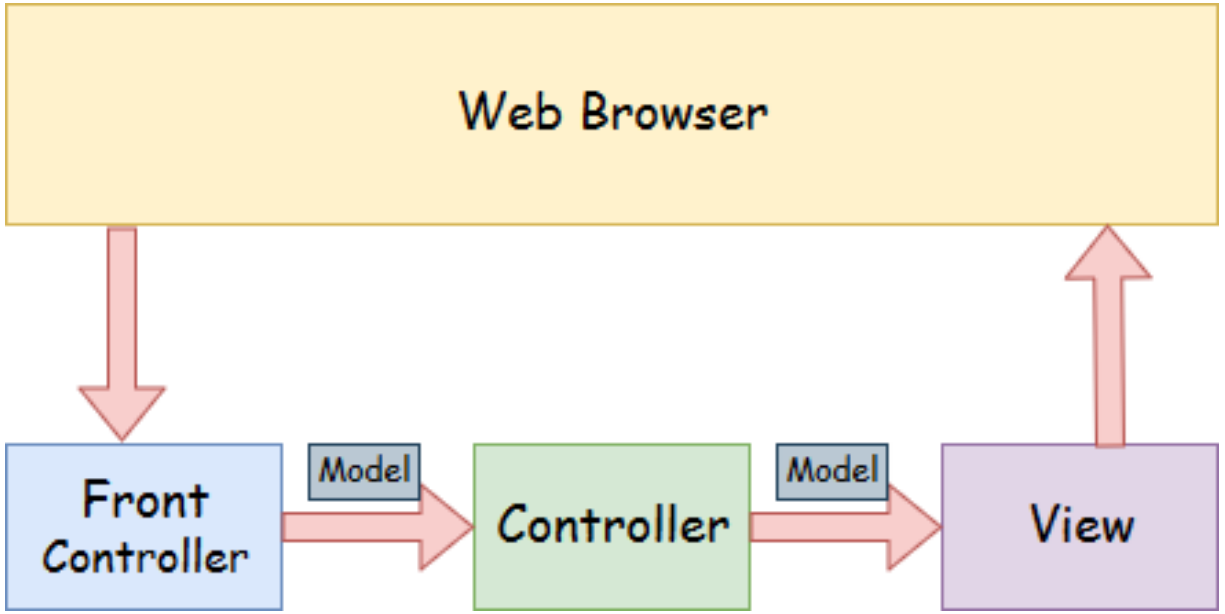
Architecture – Post Spring Adoption



Advantage of Spring MVC Framework

- **Separate roles** - The Spring MVC separates each role, where the model object, controller, command object, view resolver, DispatcherServlet, validator, etc. can be fulfilled by a specialized object.
- **Light-weight** - It uses light-weight servlet container to develop and deploy your application.
- **Powerful Configuration** - It provides a robust configuration for both framework and application classes that includes easy referencing across contexts, such as from web controllers to business objects and validators.
- **Rapid development** - The Spring MVC facilitates fast and parallel development.
- **Reusable business code** - Instead of creating new objects, it allows us to use the existing business objects.
- **Easy to test** - In Spring, generally we create JavaBeans classes that enable you to inject test data using the setter methods.
- **Flexible Mapping** - It provides the specific annotations that easily redirect the page.

Spring Web Model-View-Controller



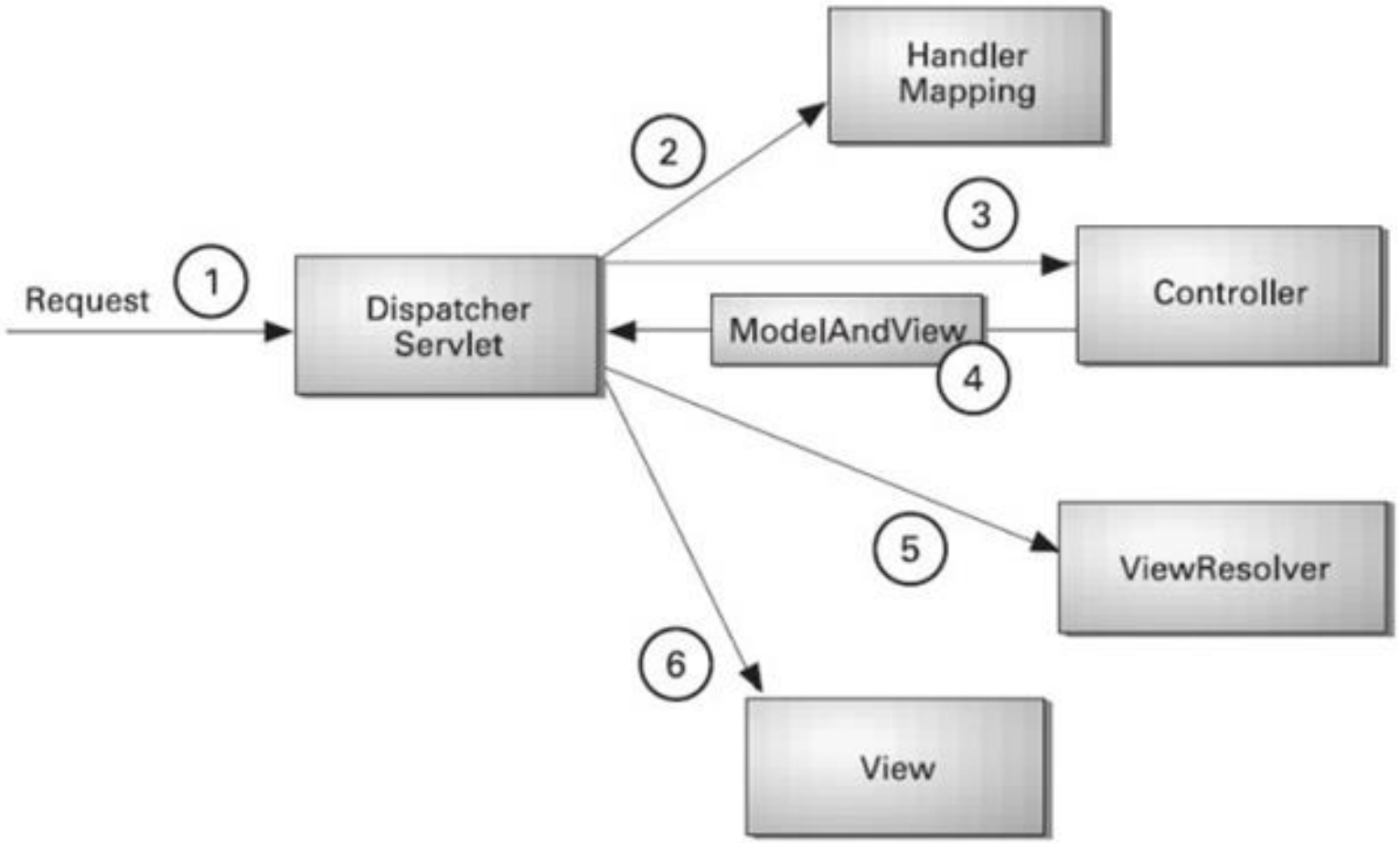
Model - A model contains the data of the application. A data can be a single object or a collection of objects.

Controller - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.

View - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.

Front Controller - In Spring Web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application

Flow of Spring Web MVC



- As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller.
- The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.
- The controller returns an object of ModelAndView.
- The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.

DispatcherServlet in web.xml

- Web applications define servlets in web.xml
- Maps URL patterns to servlets
- WebApplicationContext is an extension of ApplicationContext for features of Servlets and themes

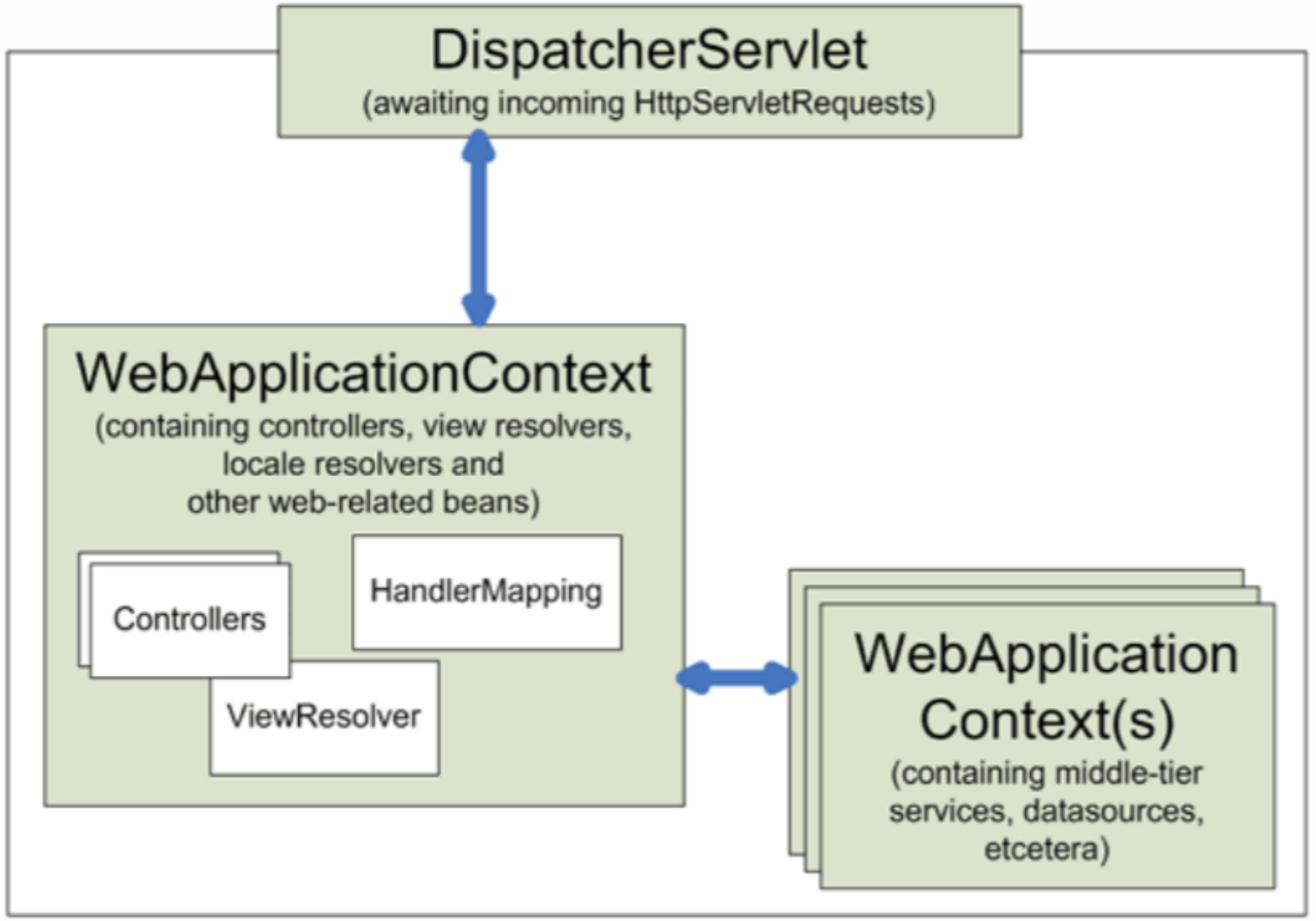
```
<web-app>
...
   <servlet>
      <servlet-name>mvc-dispatcher</servlet-name>
      <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>

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

   <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/mvc-dispatcher-servlet.xml</param-value>
   </context-param>
...
</web-app>
```

The Spring DispatcherServlet

The URL to be “captured” by DispatcherServlet

Finds the file in WEB-INF *[servlet-name]-servlet.xml* to initiate beans

WebApplicationContext Internals



Overriding DispatcherServlet defaults

- DispatcherServlet initiates with default configuration.
- Overriding it through the [servlet-name]-servlet.xml bean
- Configuring ViewResolver is basic step

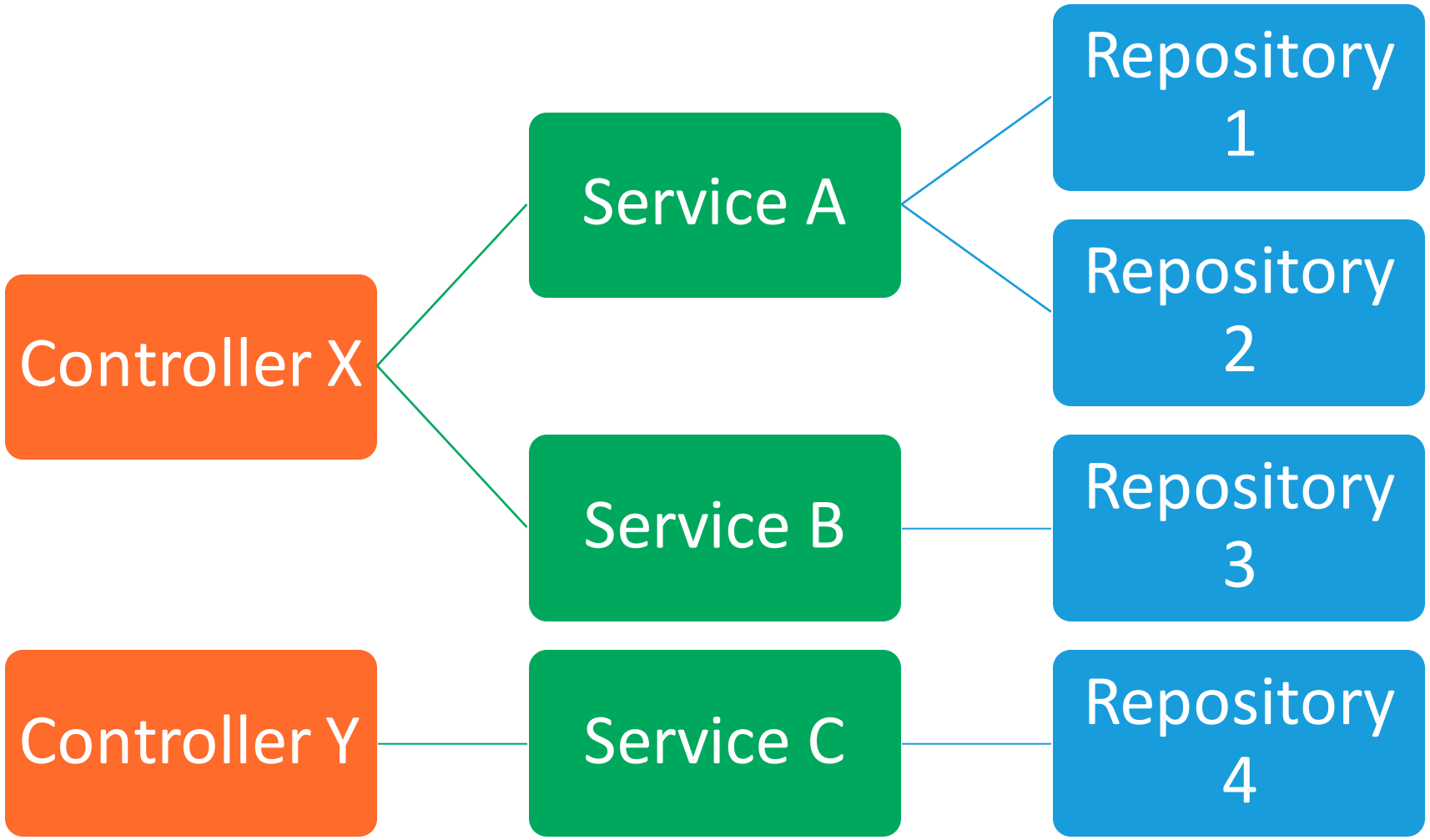
Different types of ViewResolver. Following 2 basic ones:

- InternalResourceViewResolver (for jsp, css, images etc)
- ContentNegotiatingViewResolver (for ContentType response, useful for REST APIs)

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

From Assignment 1:
if the Controller returns "**index**", InternalResourceViewResolver tries to find file as view **/WEB-INF/pages/index.jsp**

Spring Framework Components



Controller

- Handles incoming requests and building the response
- Business logic should not be handled here
- Works with the Service and Repository tier for business logic and data gathering
- Annotated with @Controller
- Handles Exceptions and routes the view accordingly

```
@Controller
public class SampleController {
    @GetMapping("/sample")
    public String showForm() {
        return "sample";
    }
}
```

Service

- Annotated with @Service
- The Service tier describes the verbs (actions) of a system
- Where the business logic resides
- Ensures that the business object is in a valid state
- Where transactions often begin (two phase commits)
- Often has the same methods as the Repository, but different focus

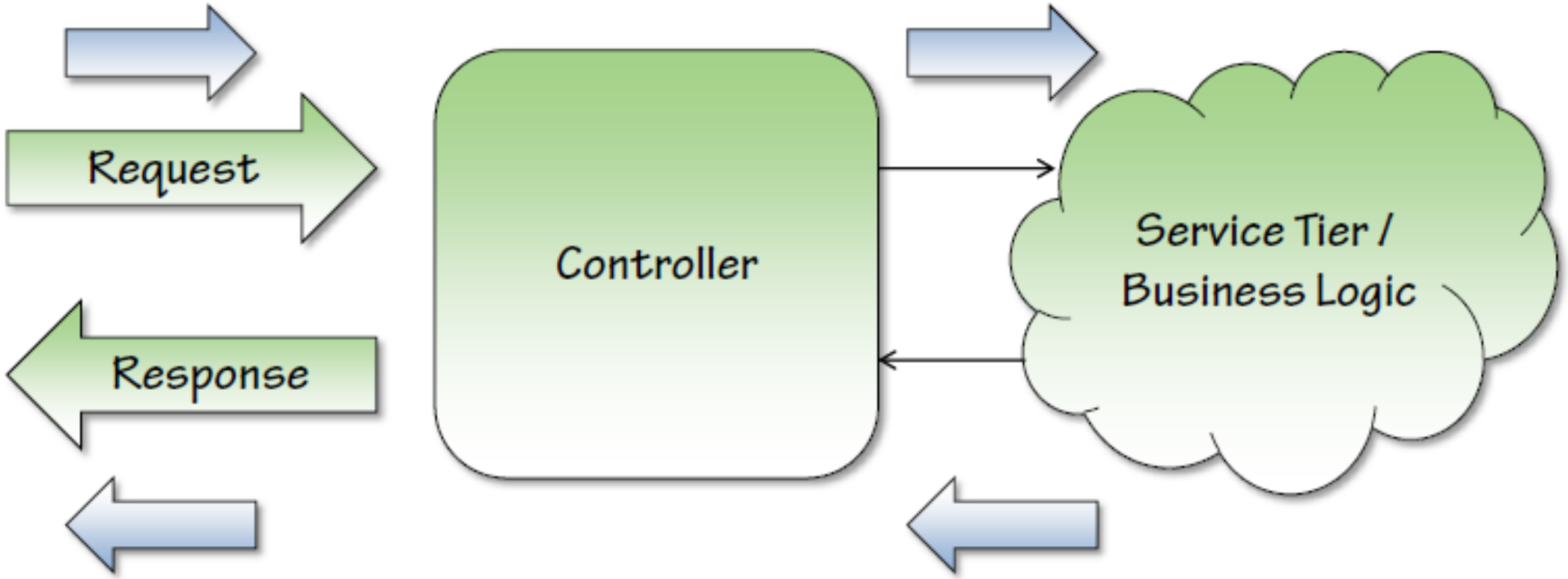
```
@Service("ms")
public class MathService {

		public int add(int x, int y) {
			return x + y;
		}

		public int subtract(int x, int y) {
			return x - y;
		}
}
```

Repository

- Annotated with @Repository
- The Repository tier describes the nouns (data) of a system
- Focused on persisting and interacting with the database
- One-to-one mapping with an Object
- Often a one-to-one mapping with a database table

```
@Repository ("employeeDao")
public class EmployeeDAOImpl implements EmployeeDAO
{
   public EmployeeDTO createNewEmployee()
   {
      EmployeeDTO e = new EmployeeDTO();
      e.setId(1);
      e.setFirstName("Lokesh");
      e.setLastName("Gupta");
      return e;
   }
}
```

What is Controller



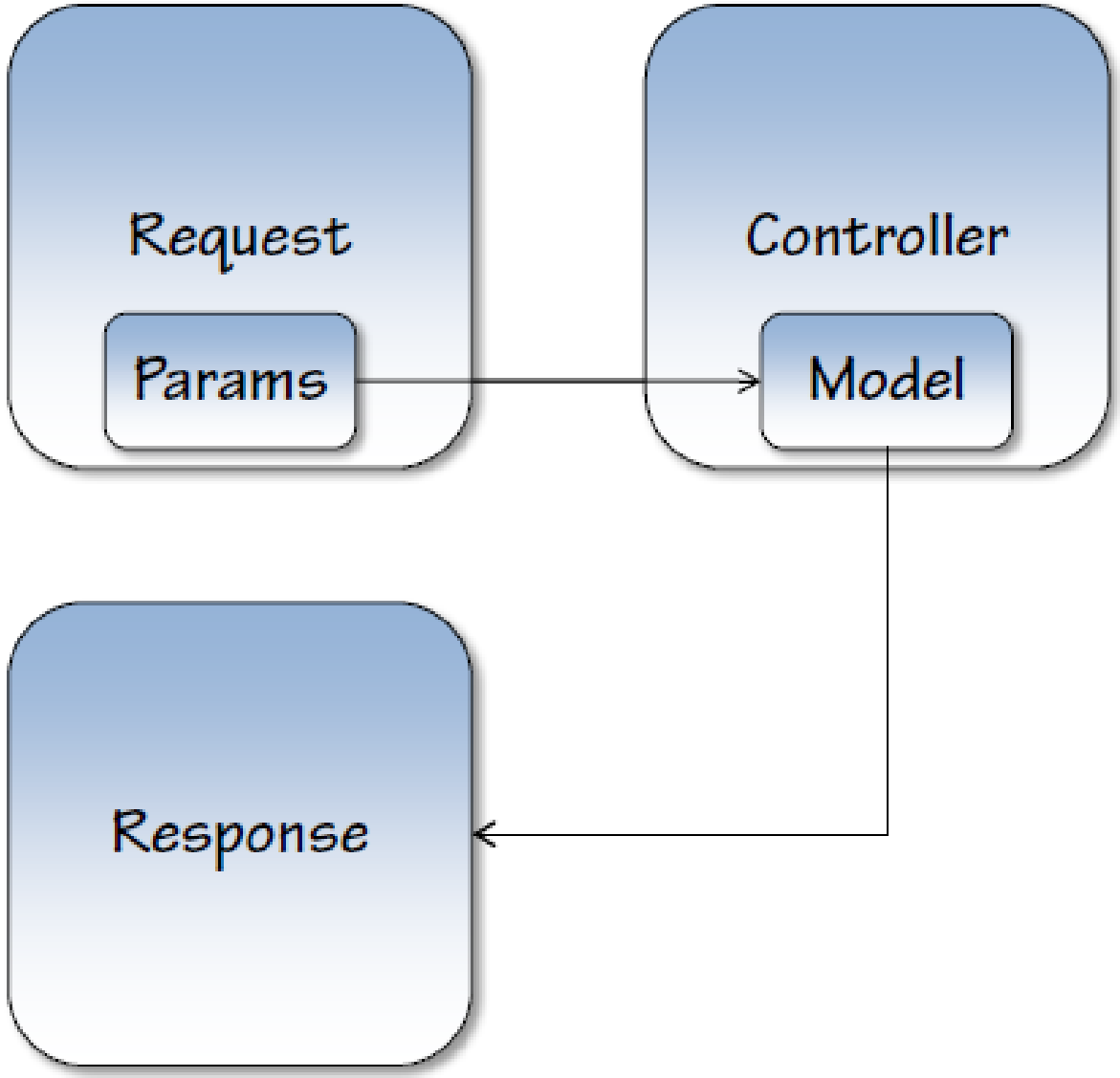
What is Controller - Responsibilities

- Interpret user input and transform to input to a model
- Provide access to business logic
- Determines view based off of logic
- Interprets Exceptions from the business logic / service tier

@Controller

```
@Controller
public class HelloController {

    @RequestMapping(value ="/greeting")
    public String sayHello (Model model) {
```

Passing Params

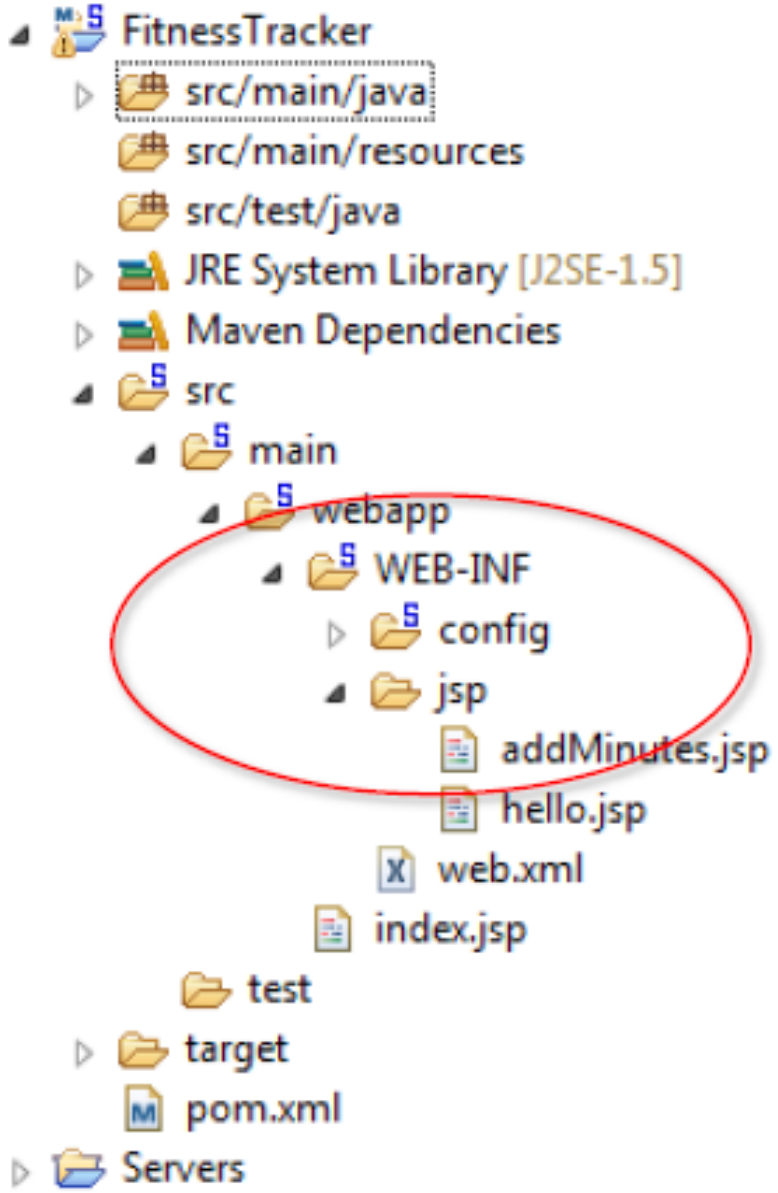


@ModelAttribute

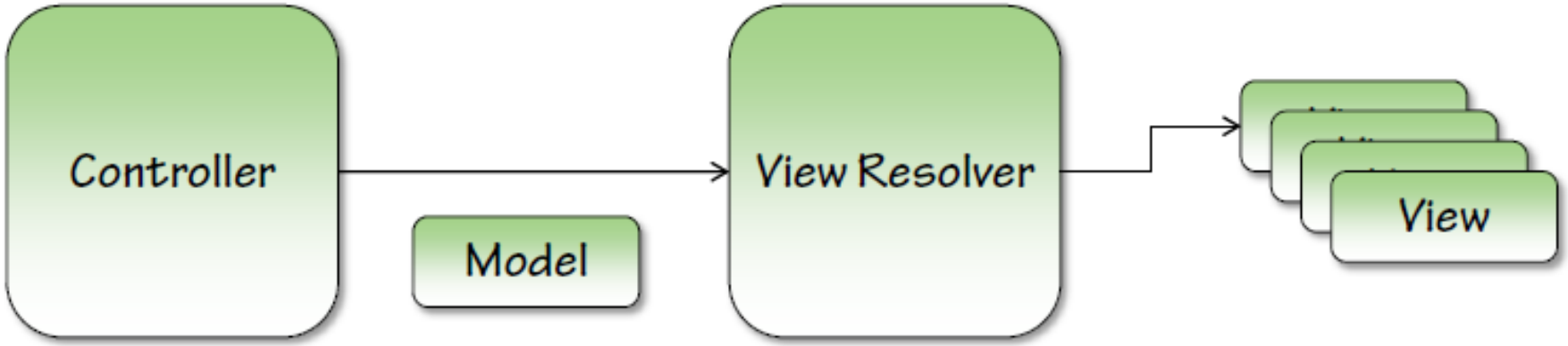
- Used with an HTTP GET
- Used with an HTTP POST
- Works with POJOs
- Can be validated with a Binding Result

View

- Conventions



Resolving a View




```
@Controller
public class MinutesController {

    @RequestMapping(value = "/addMinutes")
    public String addMinutes(@ModelAttribute ("exercise") Exercise exercise) {

        System.out.println("exercise: " + exercise.getMinutes());

        return "addMinutes";
    }

}
```

View Resolvers

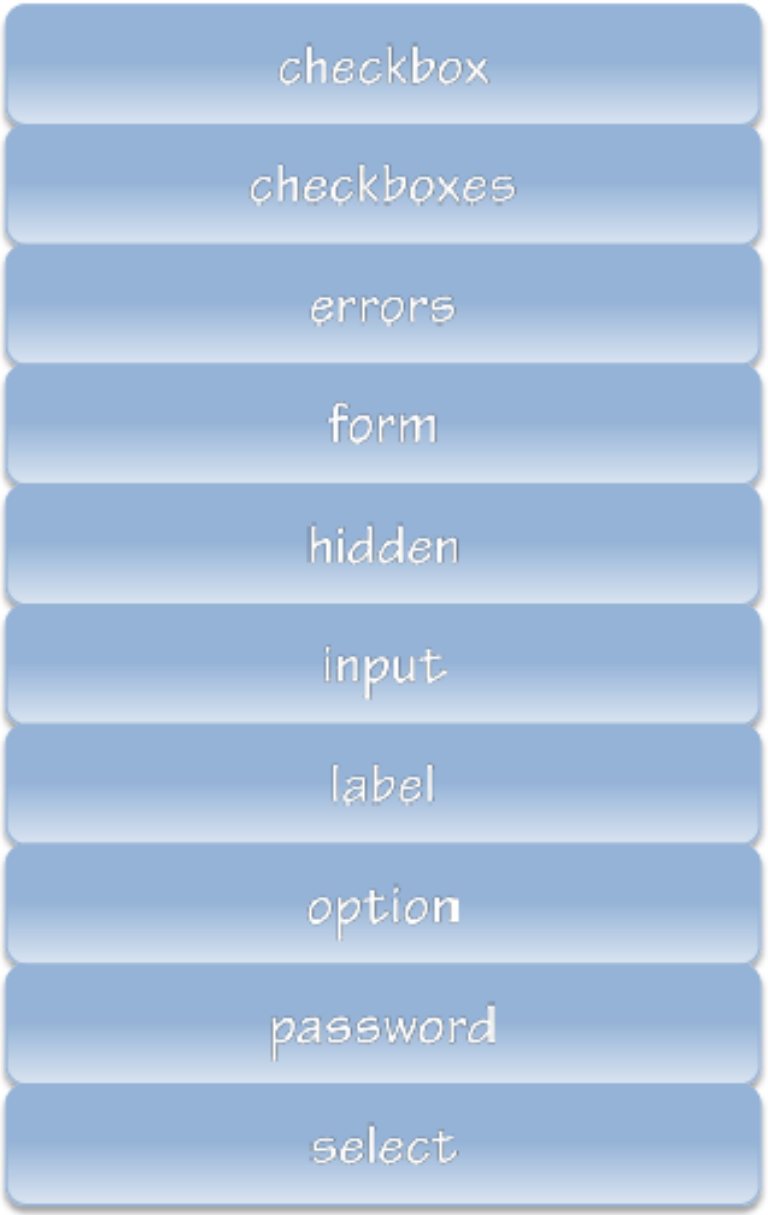
Spring Tag Libraries

- http://static.springsource.org/spring/docs/current/spring-framework-reference/html/spring.tld.html
- http://static.springsource.org/spring/docs/current/spring-framework-reference/html/spring-form.tld.html

spring.tld


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

Spring-form.tld



Interceptors

- Registered and part of the request lifecycle
- Have the ability to pre-handle and post-handle web requests
- Callback methods used to override or change values
- Commonly used for Locale Changing

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

Validation Tags

- All of the form tags have an error class associated with them
- There is a specific errors tag for displaying validation errors

```
<form:form>
      <table>
          <tr>
              <td>First Name:</td>
              <td><form:input path="firstName" /></td>
              <%-- Show errors for firstName field --%>
              <td><form:errors path="firstName" /></td>
          </tr>

          <tr>
              <td>Last Name:</td>
              <td><form:input path="lastName" /></td>
              <%-- Show errors for lastName field --%>
              <td><form:errors path="lastName"  /></td>
          </tr>
          <tr>
              <td colspan="3">
                  <input type="submit" value="Save Changes" />
              </td>
          </tr>
      </table>
  </form:form>
```

Validation Interface

- Validator Interface
- ValidationUtils class
- BindingResult class
- SimpleFormController

THANK YOU