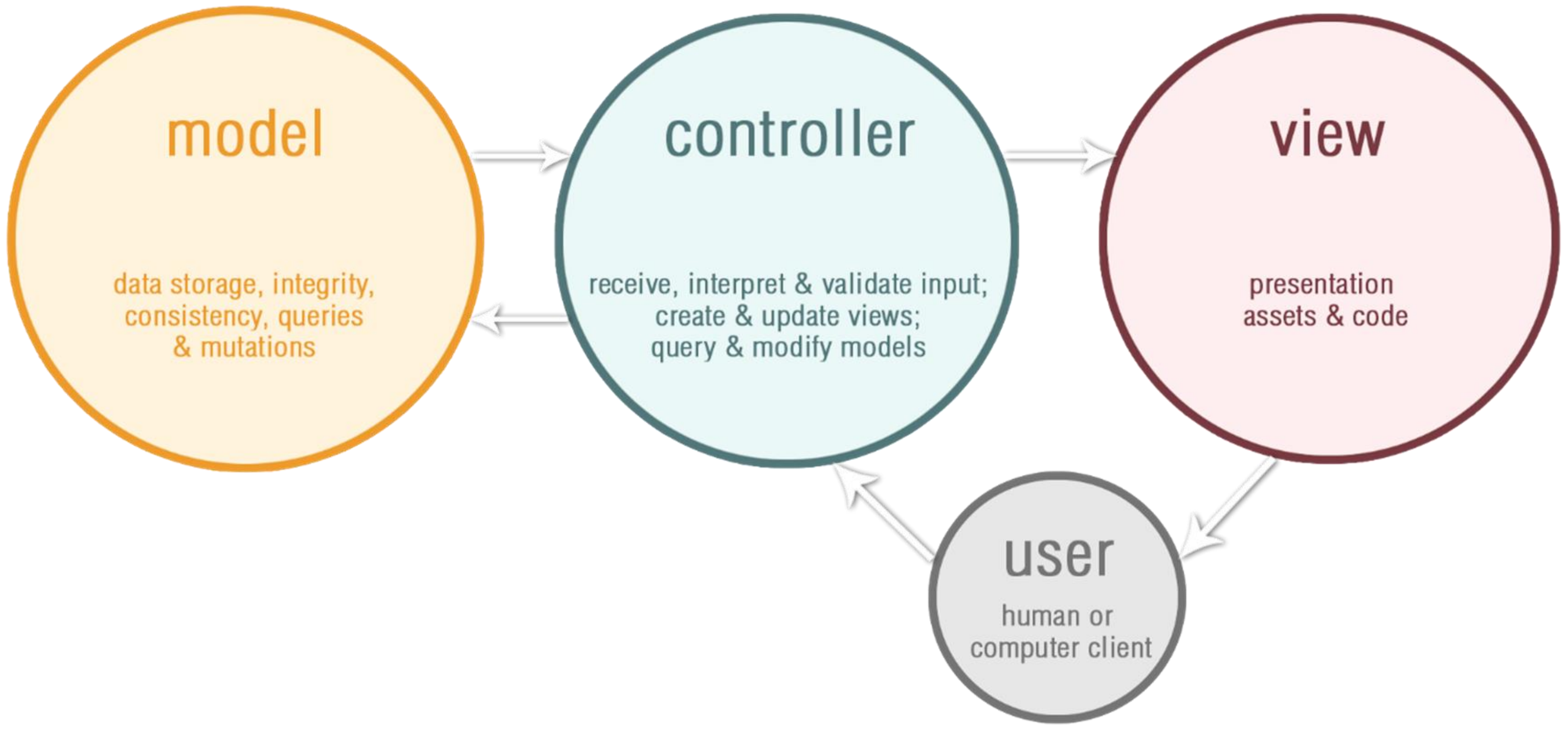
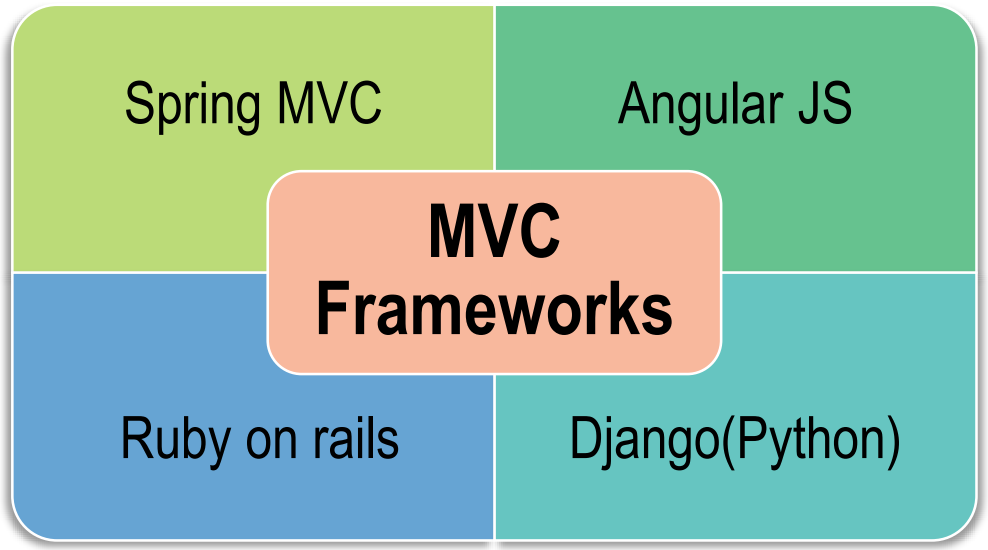
INTRODUCTION TO MVC

WHAT IS MVC?

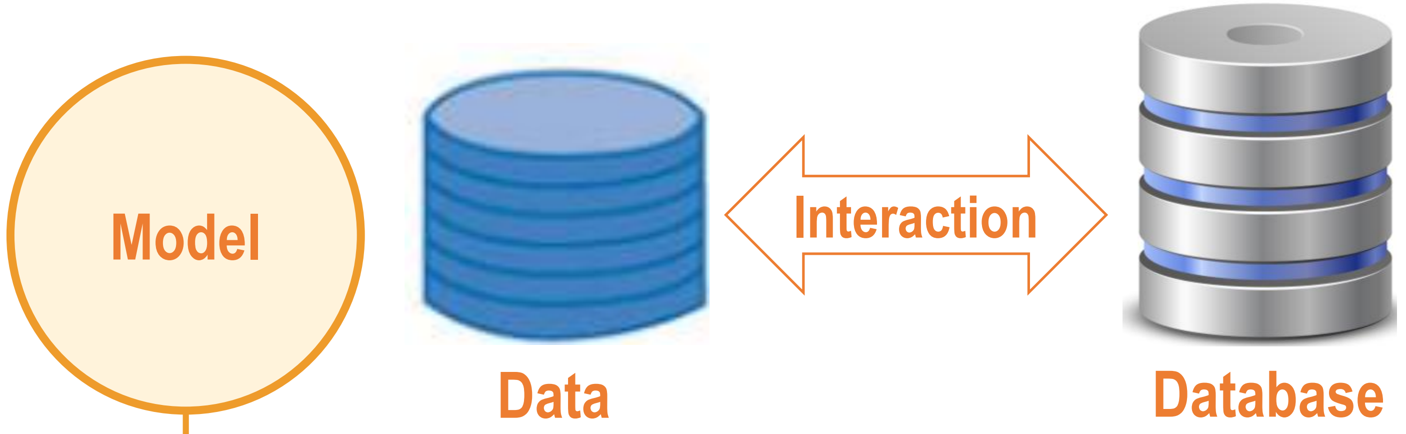

model
data storage, integrity, consistency, queries & mutations
controller
receive, interpret & validate input; create & update views; query & modify models
view
presentation assets & code
user
human or computer client

MVC DESIGN PATTERN



MVC design pattern separates application's **logic** or **functionality**

MODEL



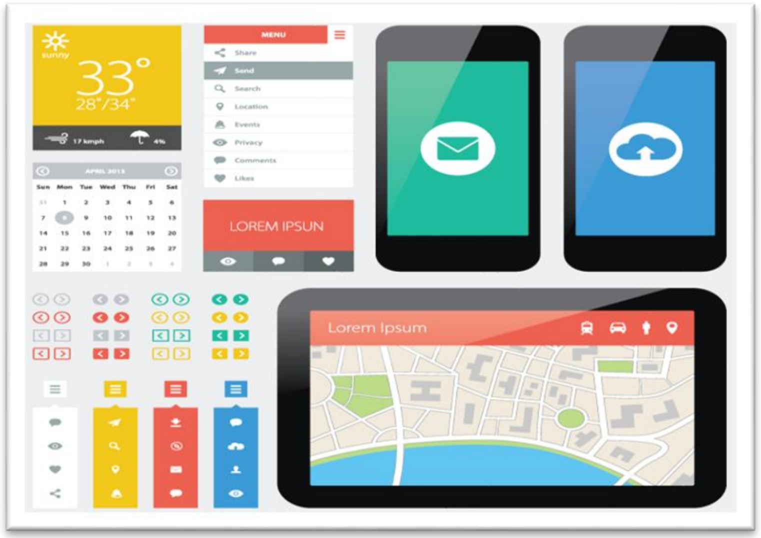
- Contains the data and interacts with database
- Acts as a carrier of data
- Updates the view

Example: Java Class

VIEW

View

- User interface
- Created using various presentation technologies- HTML/CSS, JSP and Thymeleaf
- Communicates with controller



Controller

Controller

- Responsible for handling the requests
- Receives the request from the user
- Collects data from model and send it to the view

Example: Servlet

ADVANTAGES OF MVC ARCHITECTURE

- Promotes faster development process
- Loosely couples the components
- Low maintenance cost
- Reduces the code redundancy

INTRODUCTION TO SPRING MVC

SPRING MVC FRAMEWORK

Spring MVC Framework

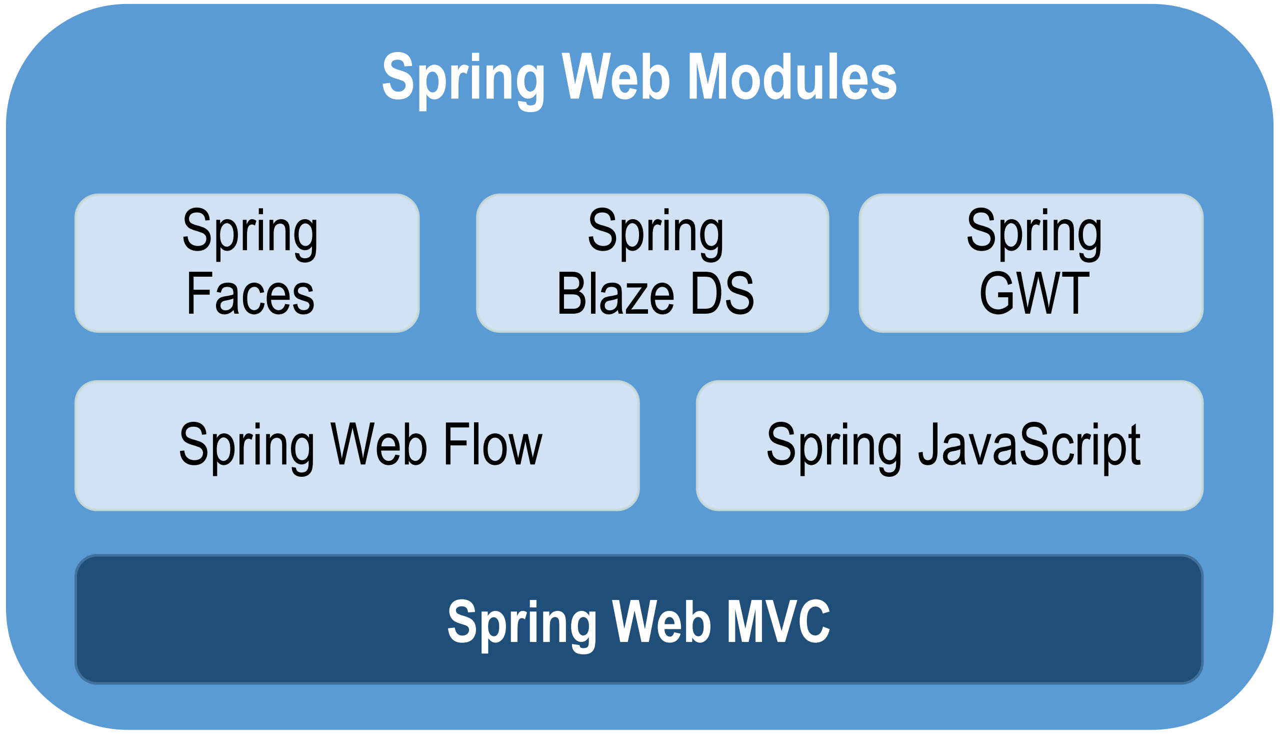
- A light weight web framework
- Provides MVC based architecture for developing web applications
- Makes flexible and loosely coupled web applications

SPRING MVC

Spring MVC

- Designed around a controller servlet called the **DispatcherServlet**
- Every request to the web application has to go through the Dispatcher Servlet
- Eases developer's effort to facilitate the development of web applications

SPRING MVC SUPPORT



SPRING MVC FEATURES

- Provides a powerful configuration of both framework and application classes as Java Beans
- Code looks clean and loosely coupled
- Provides specialized object to fulfill the different roles
- Provides internal implementation of various design pattern

SPRING MVC FEATURES

- Provides model transfer flexibility, specific validation and binding
- Allows specific local and theme resolution
- Supports JSP and JSTL
- Suppprts other view technologies such as Freemarker and Thyemeleaf

APPLICATIONCONTEXT AND WEB APPLICATIONCONTEXT

ServletContext

Represents the servlet’s environment within its container

01 Application Context

02 Web Application Context

APPLICATIONCONTEXT AND WEB APPLICATIONCONTEXT

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

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

APPLICATIONCONTEXT AND WEB APPLICATIONCONTEXT

WebApplication Context

Servlet-specific context which gets loaded based on DispatcherServlet

Dispatcher servlets can serve the web pages via Controller and another instance can be used to implement a stateless REST web service.

APPLICATIONCONTEXT V/S WEB APPLICATIONCONTEXT

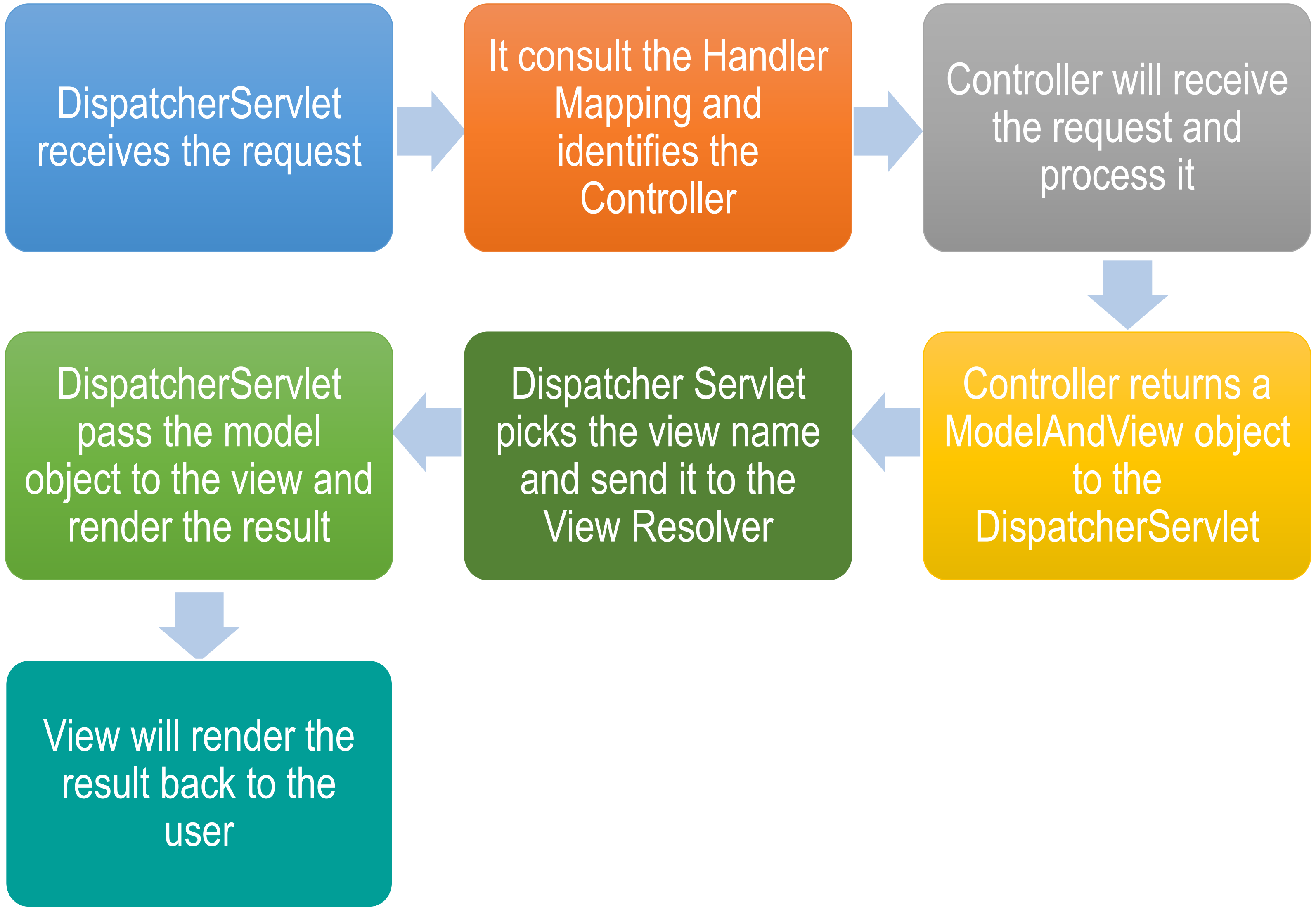
ApplicationContext

- Root-context which contains bean configuration
- Used across the entire application as a single instance
- One per webapplication

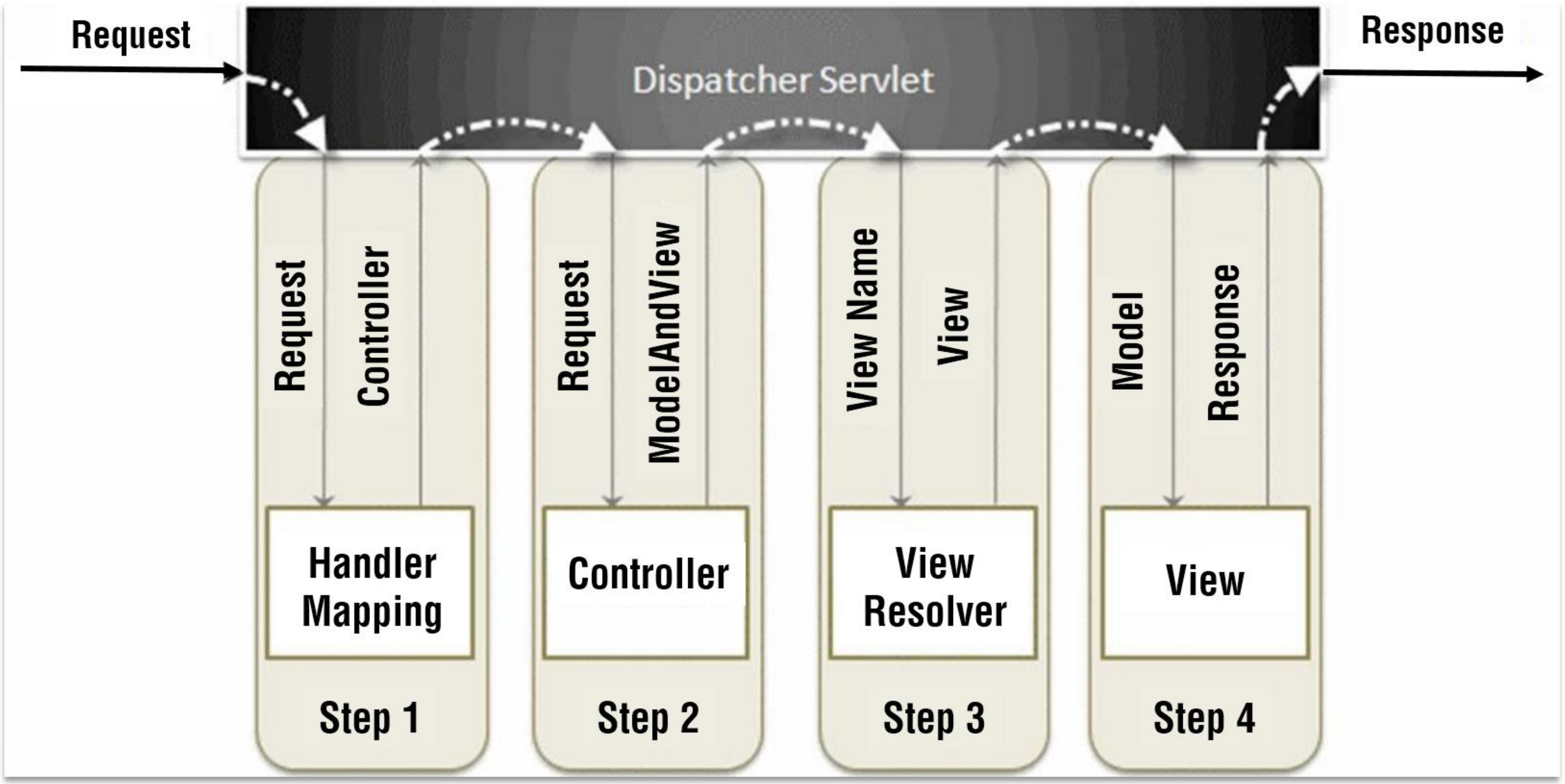
WebApplicationContext

- Spring application can have multiple WebApplicationContext
- Dispatcher servlets is specified in the application’s web.xml.
- Clear separation between middle-tier services

FLOW OF REQUEST IN SPRING MVC FRAMEWORK


DispatcherServlet receives the request
It consult the Handler Mapping and identifies the Controller
Controller will receive the request and process it
Controller returns a ModelAndView object to the DispatcherServlet
Dispatcher Servlet picks the view name and send it to the View Resolver
DispatcherServlet pass the model object to the view and render the result
View will render the result back to the user

FLOW OF REQUEST IN SPRING MVC FRAMEWORK



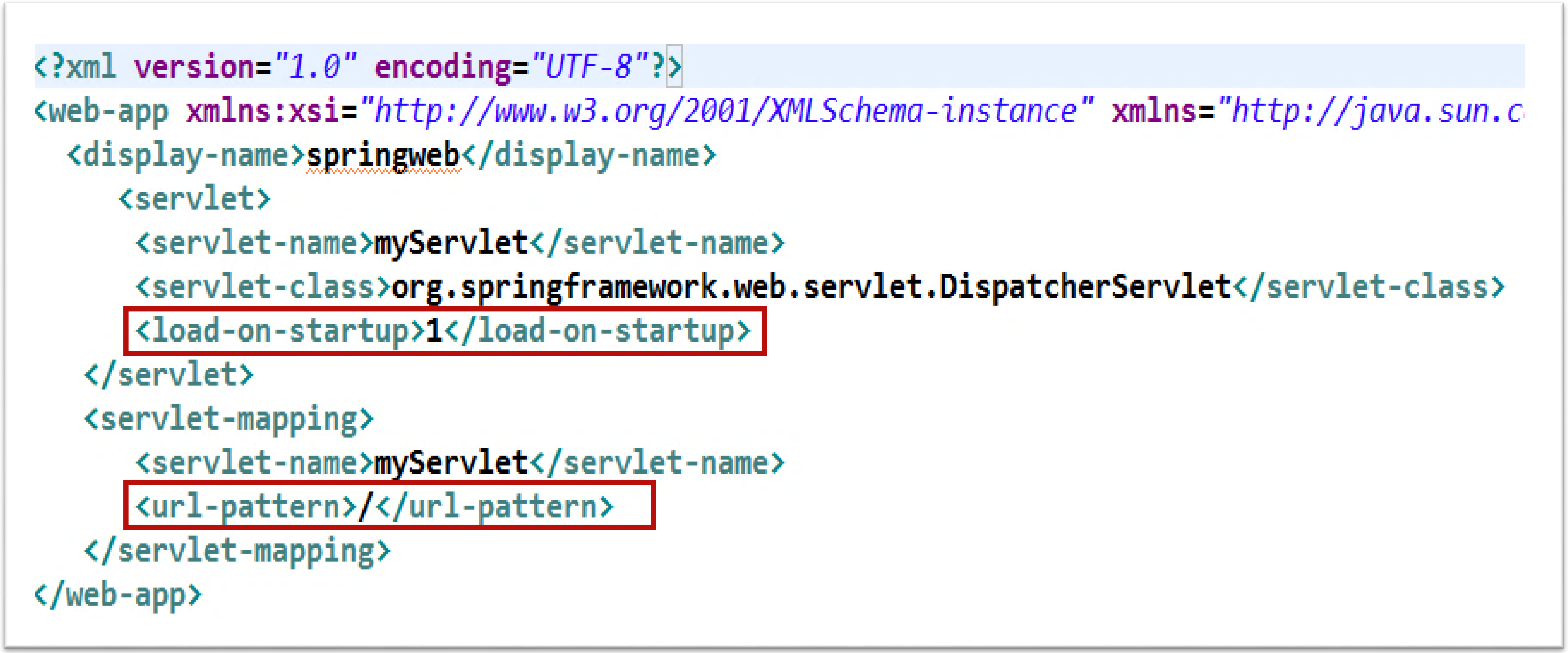
SPRING MVC CONFIGURATION

CONFIGURING SPRING MVC (TRADITIONAL WAY)

Traditional Approach

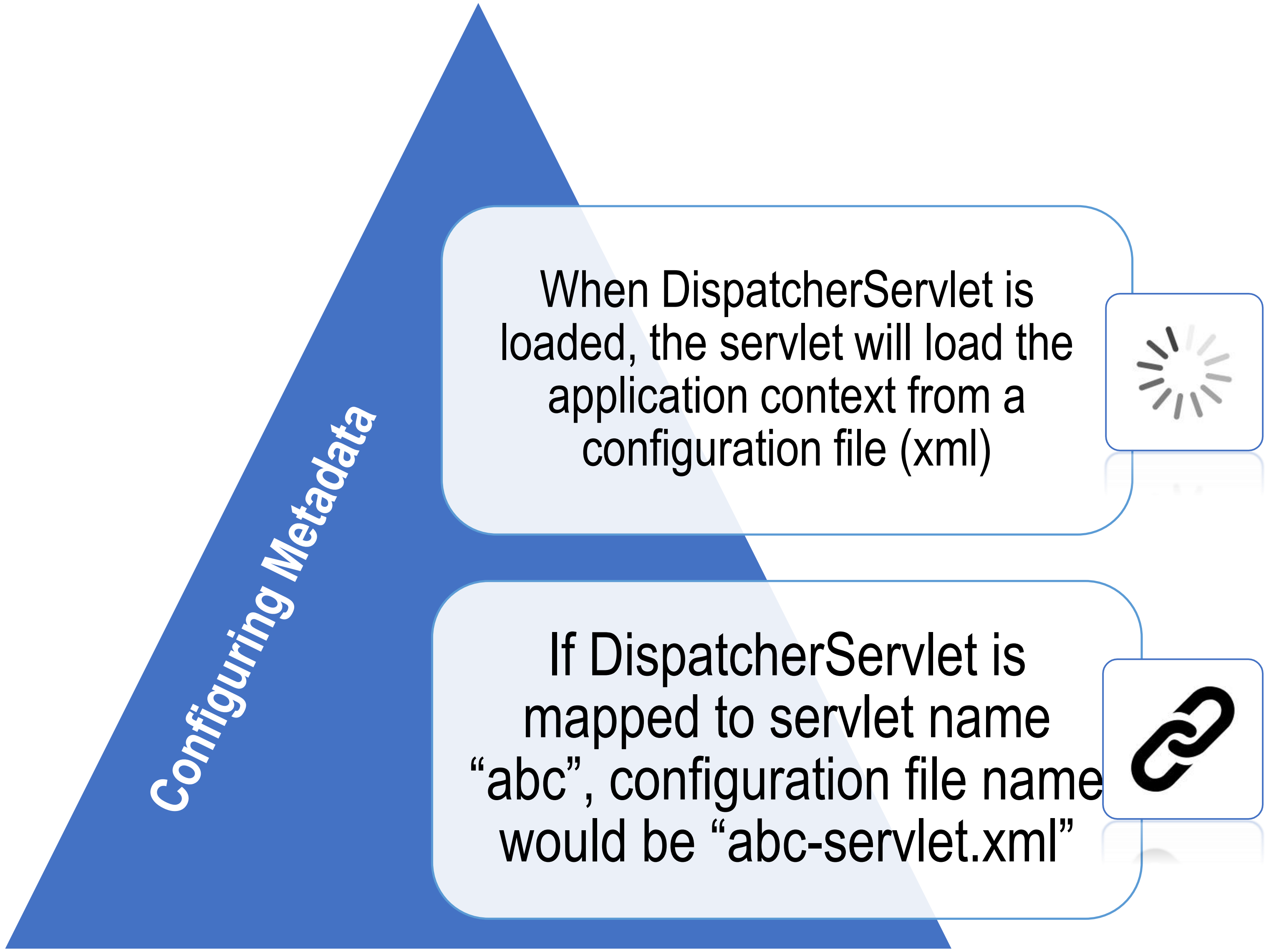
- Spring MVC provides **DispatcherServlet** which acts as a front controller
- Dispatcher servlet is configured in web.xml

CONFIGURING SPRING MVC (TRADITIONAL WAY)



```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.c
  <display-name>springweb</display-name>
     <servlet>
      <servlet-name>myServlet</servlet-name>
      <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
      <servlet-name>myServlet</servlet-name>
      <url-pattern>/</url-pattern>
   </servlet-mapping>
</web-app>
```

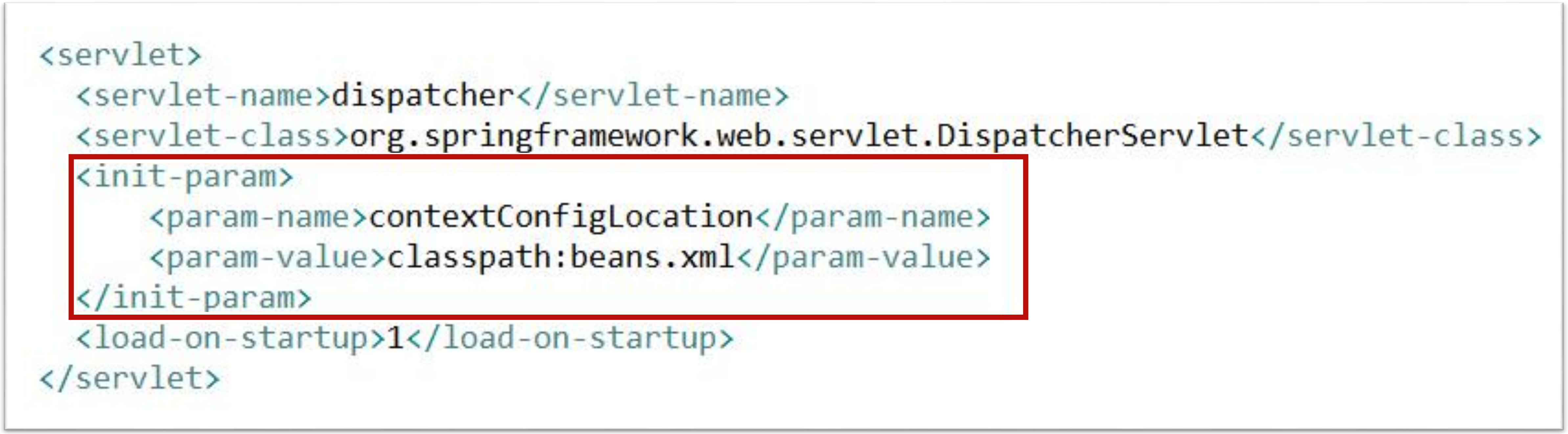
USING DEFAULT SPRING APPLICATIONCONTEXT FILE



When DispatcherServlet is loaded, the servlet will load the application context from a configuration file (xml)

If DispatcherServlet is mapped to servlet name “abc”, configuration file name would be “abc-servlet.xml”

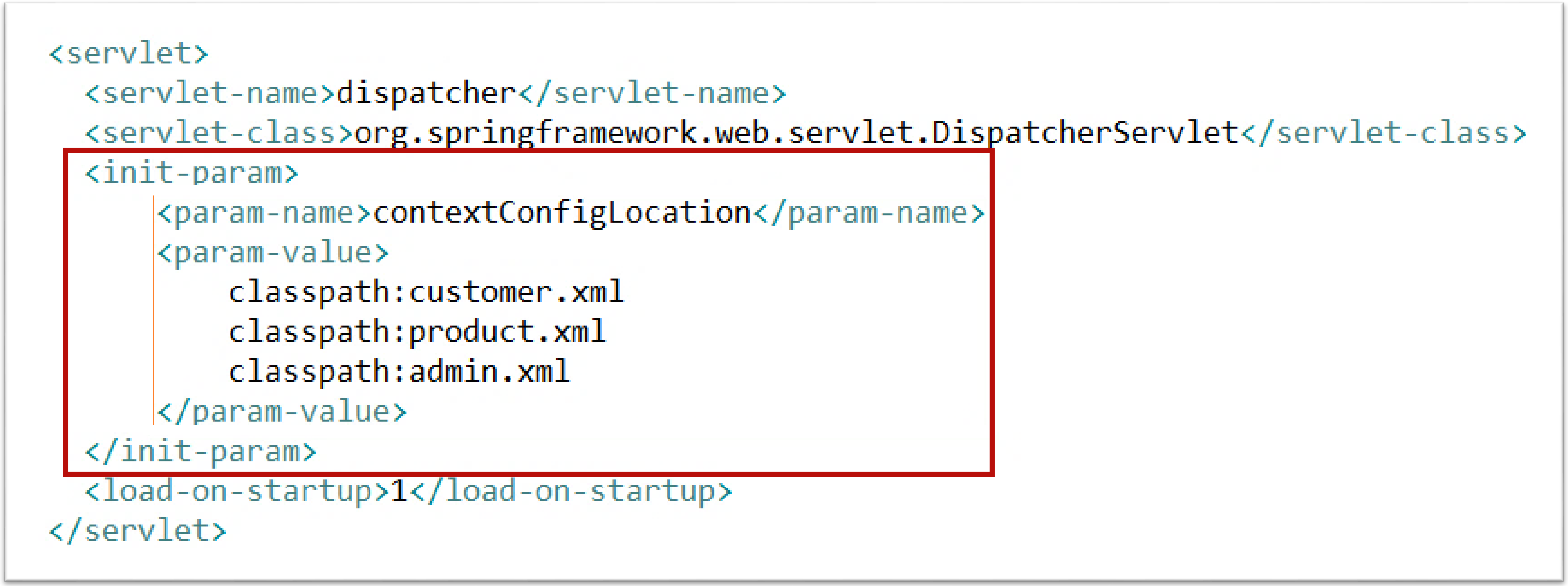
USING SINGLE SPRING APPLICATIONCONTEXT FILE



```
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:beans.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

USING MULTIPLE SPRING APPLICATIONCONTEXT FILES

The best practice is to split large configuration file into multiple small files.



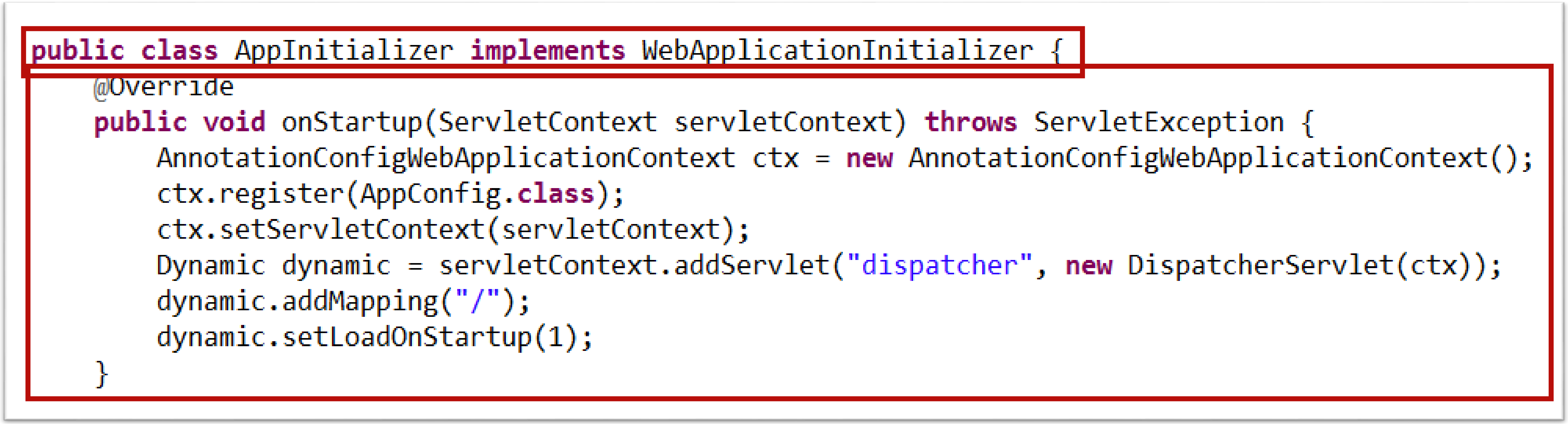
```
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>
          classpath:customer.xml
          classpath:product.xml
          classpath:admin.xml
      </param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

CONFIGURING SPRING MVC (PROGRAMMATIC WAY)

Programmatic Approach

- Standard Java EE Servlet Configuration in a Servlet 3.0 or later environment
- Create a class by implementing an interface called **WebApplicationInitializer**

CONFIGURING SPRING MVC (PROGRAMMATIC WAY)



```java
public class AppInitializer implements WebApplicationInitializer {
    @Override
    public void onStartup(ServletContext servletContext) throws ServletException {
        AnnotationConfigWebApplicationContext ctx = new AnnotationConfigWebApplicationContext();
        ctx.register(AppConfig.class);
        ctx.setServletContext(servletContext);
        Dynamic dynamic = servletContext.addServlet("dispatcher", new DispatcherServlet(ctx));
        dynamic.addMapping("/");
        dynamic.setLoadOnStartup(1);
    }
```

CONFIGURING SPRING MVC (PROGRAMMATIC WAY)

```
public class AppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {
@Override
protected Class<?>[] getRootConfigClasses() {
return null;
}
@Override
protected Class<?>[] getServletConfigClasses() {
return new Class[] { AppConfig.class };
}
@Override
protected String[] getServletMappings() {
return new String[] { "/" };
}
}
```

ENABLING THE MVC JAVA CONFIG

```
@Configuration
@EnableWebMvc
public class AppConfig {
}
```

MULTIPLE CONFIGURATION CLASS

```
@Configuration
@Import({ CustomerConfig.class, ProductConfig.class })
public class AppConfig {
}
```

SPRING MVC ANNOTATIONS

SPRING MVC ANNOTATIONS

Spring MVC provides the **auto detection** features for annotations

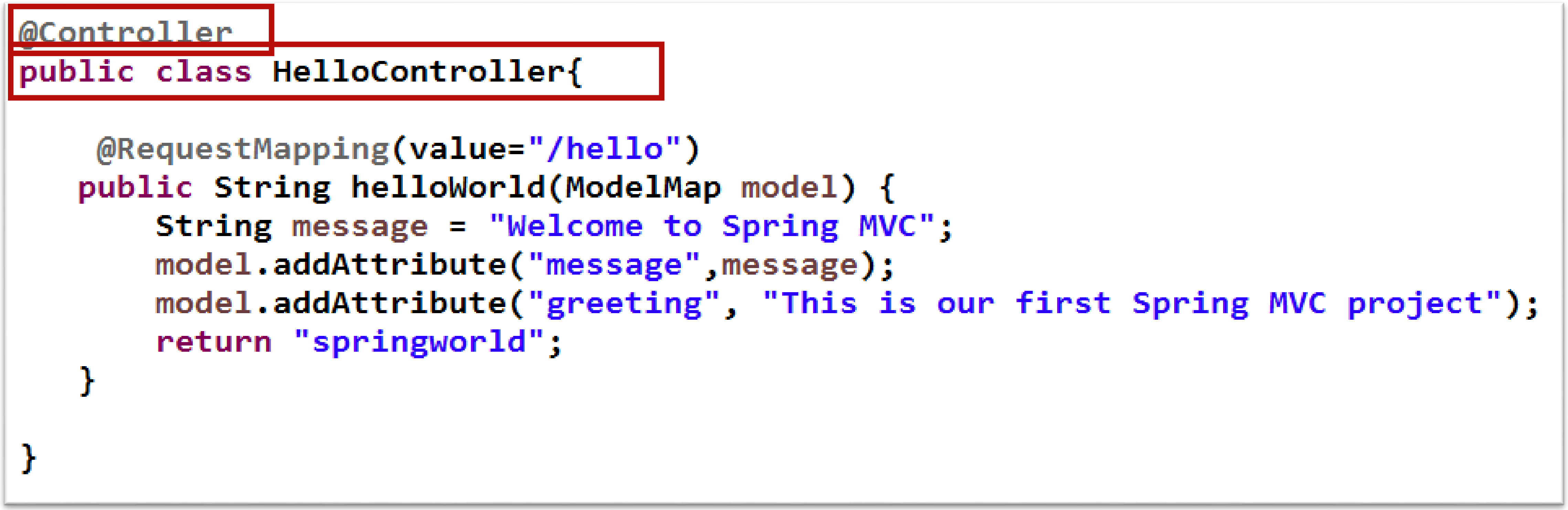
Add “**component scan**” in spring-context and provide the base-package

SPRING MVC ANNOTATIONS

Annotation	Use	Description
@Controller	Type	Stereotype annotation
@RequestMapping	Method, Type	Used to map a method with a URL
@PathVariable	Method	Binds method parameter to URI template variable
@RequestParam	Parameter	Used to read the form data
@ModelAttribute	Parameter, Method	Used to preload the model with the value returned from the method.
@RequestHeader	Parameter	Used to read RequestHeader
@SessionAttributes	Type	Maintains the session

@CONTROLLER ANNOTATION

@Controller



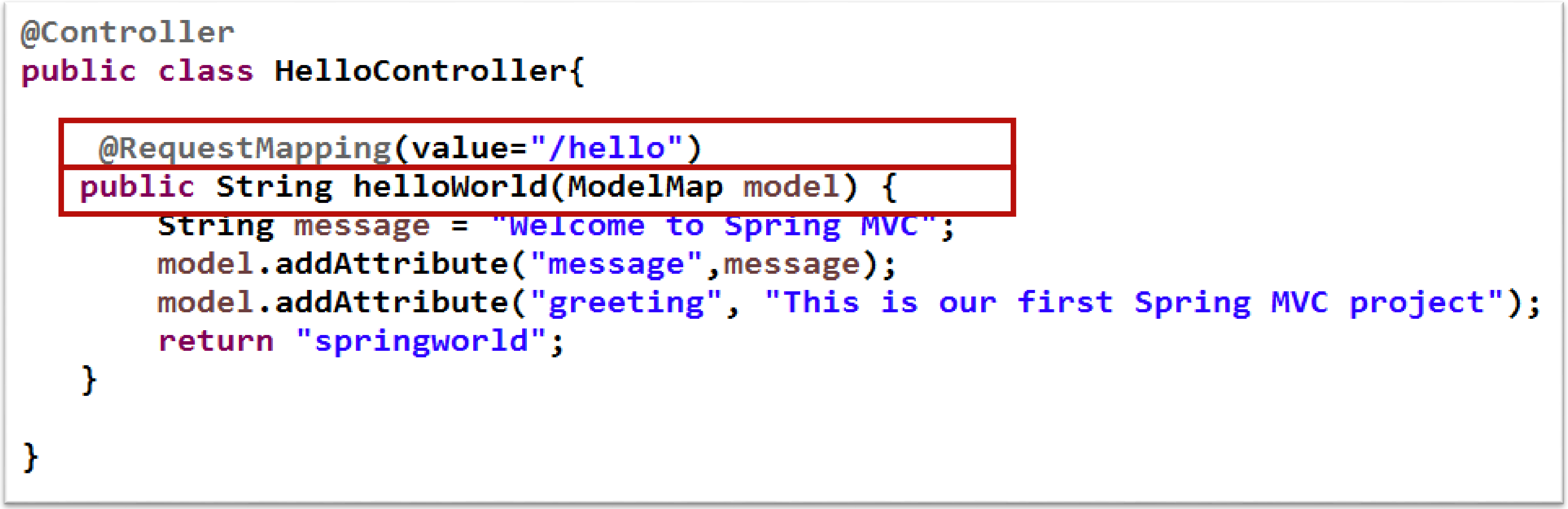
```
@Controller
public class HelloController{

     @RequestMapping(value="/hello")
    public String helloWorld(ModelMap model) {
        String message = "Welcome to Spring MVC";
        model.addAttribute("message",message);
        model.addAttribute("greeting", "This is our first Spring MVC project");
        return "springworld";
    }

}
```

@REQUESTMAPPING ANNOTATION

@RequestMapping



```
@Controller
public class HelloController{

    @RequestMapping(value="/hello")
   public String helloWorld(ModelMap model) {
       String message = "Welcome to Spring MVC";
       model.addAttribute("message",message);
       model.addAttribute("greeting", "This is our first Spring MVC project");
       return "springworld";
   }

}
```

@PATHVARIABLE ANNOTATION

@PathVariable

It obtains some placeholder from the **Uniform Resource Identifier**.

http://localhost:7080/springmvc/hello/101?p1=100&p2=200

```
@RequestMapping("/hello/{id}")
public String getDetails(
            @PathVariable(value="id") String id,
            ){ ....... }
```

@REQUESTPARAM ANNOTATIONS

@RequestParam

http://localhost:7080/springmvc/hello/101?p1=100&p2=200

```
public String getDetails(
            @RequestParam(value="p1", required=true) String p1,
            @RequestParam(value="p2", required=false) String p2
            )
{ ... }
```

@REQUESTPARAM ANNOTATIONS

@RequestParam

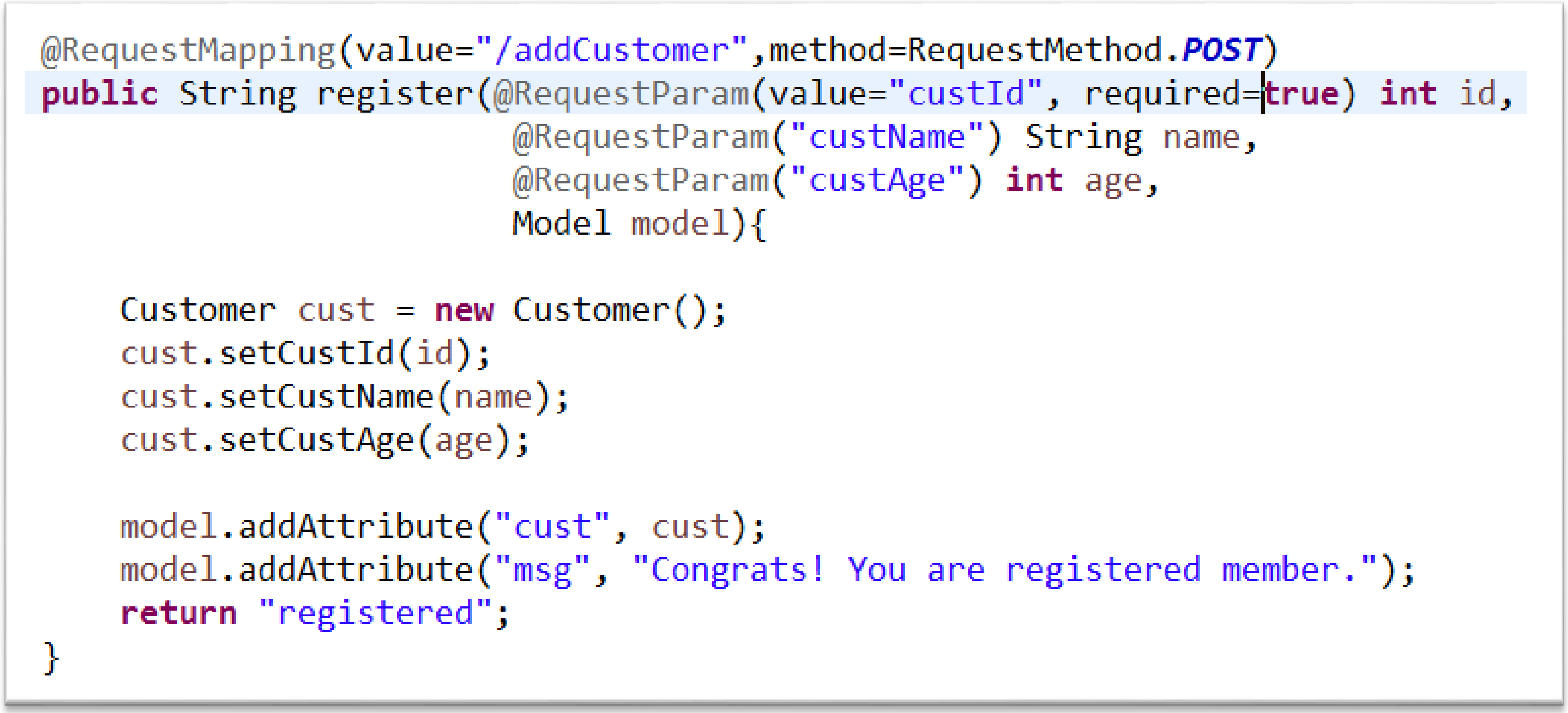
- defaultValue
- name
- required
- value

This is an alias for the name attribute.

This is a Boolean attribute to indicate whether the parameter is required.

@REQUESTPARAM ANNOTATIONS

@RequestParam: Example



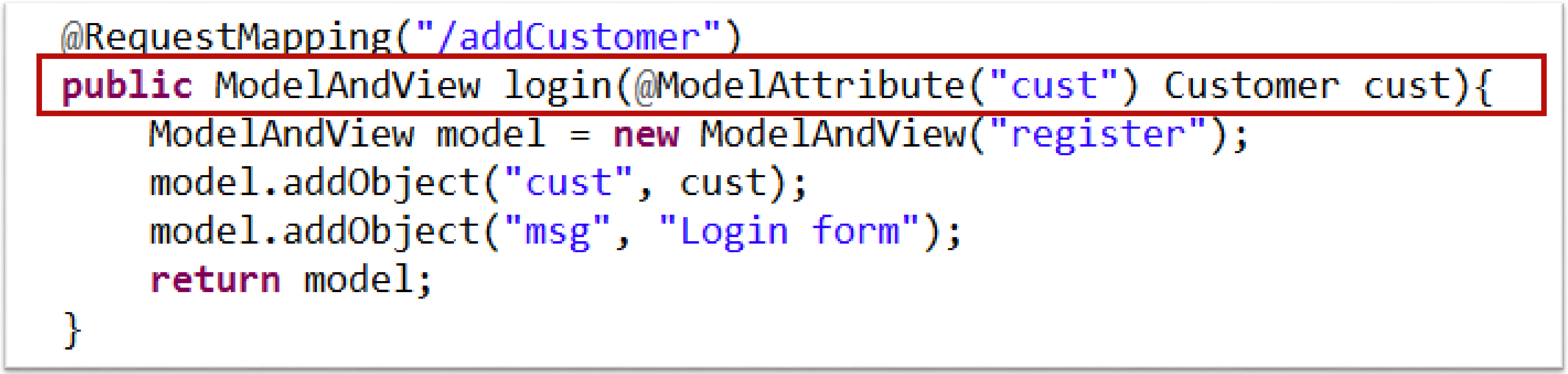
```
@RequestMapping(value="/addCustomer",method=RequestMethod.POST)
public String register(@RequestParam(value="custId", required=true) int id,
                        @RequestParam("custName") String name,
                        @RequestParam("custAge") int age,
                        Model model){

    Customer cust = new Customer();
    cust.setCustId(id);
    cust.setCustName(name);
    cust.setCustAge(age);

    model.addAttribute("cust", cust);
    model.addAttribute("msg", "Congrats! You are registered member.");
    return "registered";
}
```

@MODELATTRIBUTE ANNOTATION

@ ModelAttribute : Example



```
@RequestMapping("/addCustomer")
public ModelAndView login(@ModelAttribute("cust") Customer cust){
    ModelAndView model = new ModelAndView("register");
    model.addObject("cust", cust);
    model.addObject("msg", "Login form");
    return model;
}
```

VIEW RESOLVER

VIEW RESOLVERS

View Resolver

- Used to address the view
- Renders model in a browser without thinking of a specific View Technology
- Maps a logical view name to an appropriate implementation of View

VIEW RESOLVERS – A BIRD’S EYE VIEW

Common View Resolvers

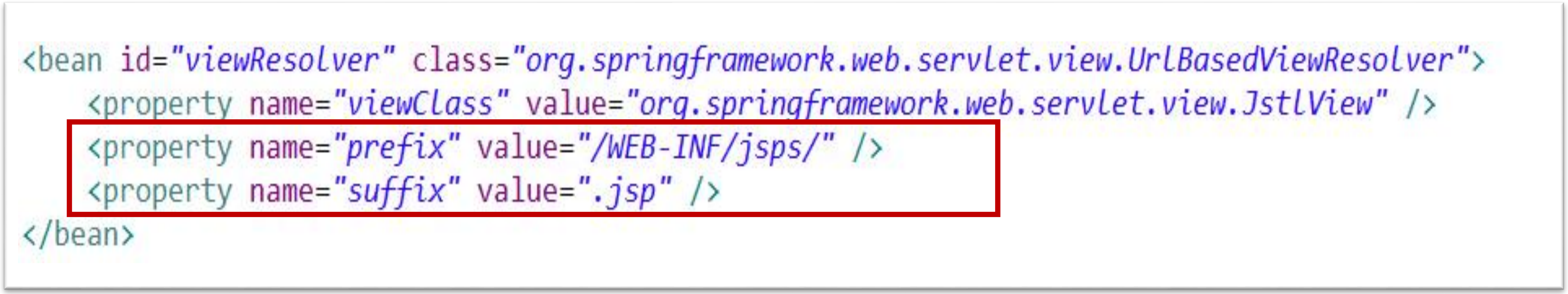
- UrlBasedViewResolver
- InternalResourceViewResolver
- ResourceBundleViewResolver
- VelocityViewResolver

EXAMPLE: URLBASEDVIEWRESOLVER

Simple implementation of View Resolver interface

Preferred if logical name matches with actual view resource

EXAMPLE: URLBASEDVIEWRESOLVER



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

EXAMPLE: INTERNALRESOURCEVIEWRESOLVER

XML Configuration

```
<context:component-scan base-package="org.learn" />

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

Java Configuration

```
@Bean
public ViewResolver viewResolver() {
        InternalResourceViewResolver vr = new InternalResourceViewResolver();
        vr.setPrefix("/WEB-INF/view/");
        vr.setSuffix(".jsp");
        return vr;
}
```