Spring WebMVC

Objectives

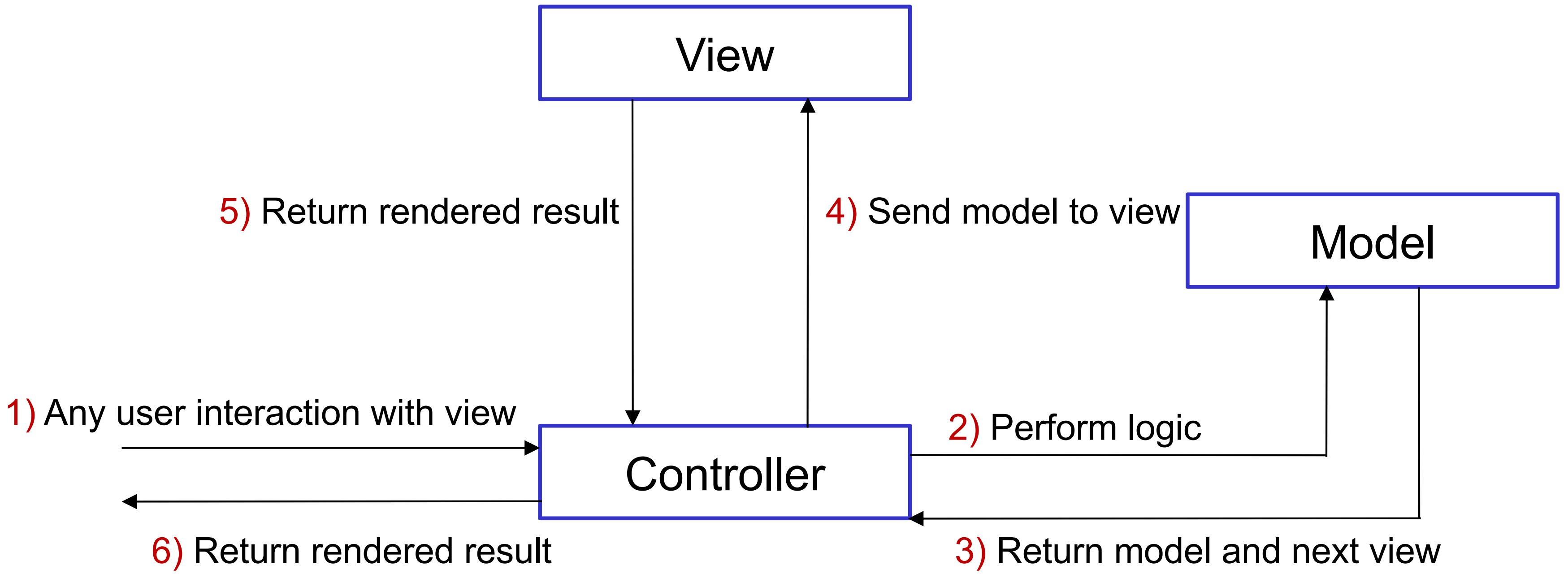
When we are done, you should be able to:

- Configure WebMVC with Spring
- Write and configure a controller class

MVC – Model View Controller

- An architectural design pattern
- Means of separating the presentation (view) from the business logic (model) with an intermediary (controller)
- Used in most UI situations

MVC in a Nutshell


View
Model
Controller
1) Any user interaction with view
2) Perform logic
3) Return model and next view
4) Send model to view
5) Return rendered result
6) Return rendered result

MVC in a Nutshell [cont.]

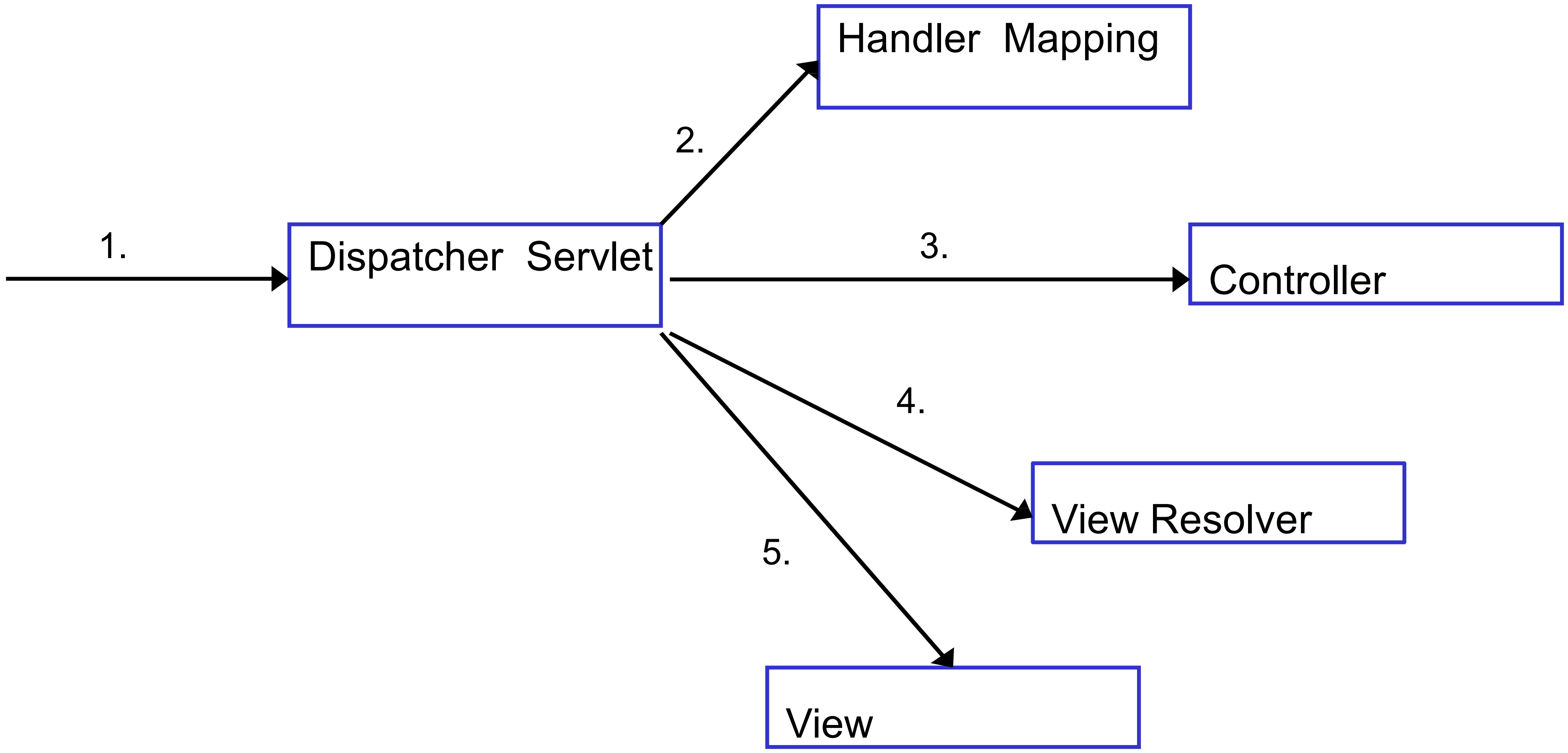
- Model
  - Data that is being modified and used
  - Often service that does the modifying is included
- View
  - The code that dynamically define what goes to the user
- Controller
  - Maps to view and model
- From an architectural standpoint, these are likely components of classes

Intro to Spring MVC

Spring MVC

- A framework implementing MVC
  - Can always go around, but should make MVC easier to accomplish
- Comes in two forms
  - @MVC – configuration using annotations
  - MVC – configuration only uses XML
- @MVC is preferred

Request Life Cycle



`DispatcherServlet`

- Front Controller for SpringMVC
- Delegates to second layer controllers and to view
- Loads `WebApplicationContext`
- Can be initialized without `web.xml` in Servlet 3.0+ environments

Configuration with `web.xml`

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/application-config.xml</param-value>
</context-param>
<servlet>
  <servlet-name>Spring</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/web-config.xml</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>Spring</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

Configuration With Java

- Needs two additional files
  - MVC configuration file
    - Extends `WebMvcConfigurerAdapter`
    - Register view resolvers
    - Register locale resolvers
    - Register interceptors
  - `DispatcherServlet` configuration file
    - Extends `AbstractAnnotationConfigDispatcherServletInitializer`
    - Registers all other configuration files
    - Registers servlet mappings
    - Registers `DispatcherServlet`

Needed Dependencies

- Maven Dependencies
  - groupId: `org.springframework`
  - artifactId: `spring-web`
  - artifactId: `spring-webmvc`
- Actual (additional) JAR files
  - `aopalliance`
  - `spring-webmvc`
  - `spring-aop`
  - `spring-web`

DispatcherServlet Configuration

```
public class WebAppInitializer extends
   AbstractAnnotationConfigDispatcherServletInitializer {

  protected Class<?>[] getRootConfigClasses() {
    return new  Class[] { JavaConfig.class };
  }

  protected Class<?>[] getServletConfigClasses() {
    return new Class[] { MVCConfig.class };
  }

  protected String[] getServletMappings() {
    return new String[] { "/*.jsp" };
  }

  public void onStartup(ServletContext servletContext) throws
   ServletException {
    super.onStartup(servletContext);
  }
}
```

MVC Configuration – Part 1

```
@Configuration
@ComponentScan(basePackages = "com.di.phonebook")
@EnableWebMvc
public class MvcConfig extends WebMvcConfigurerAdapter {

public void addResourceHandlers(ResourceHandlerRegistry registry)
{
   registry.addResourceHandler("/resources/**").
   addResourceLocations("/resources/");
}

public void configureDefaultServletHandling
  (DefaultServletHandlerConfigurer  configurer) {

  configurer.enable();
}
```

MVC Configuration – Part 2

```
public InternalResourceViewResolver viewResolver() {
  InternalResourceViewResolver resolver = new
    InternalResourceViewResolver();
  resolver.setPrefix("/WEB-INF/views/");
  resolver.setSuffix(".jsp");
  resolver.setAlwaysInclude(true);
  return resolver;
}

public void configureViewResolvers(ViewResolverRegistry registry)
{
  registry.viewResolver(viewResolver());
}

}
```

Controller

```java
import org.springframework.stereotype.Controller;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class BookController {

  @Resource
  private BookService bookSvc;

  @RequestMapping(value="/book")
  public ModelAndView getBook(@RequestParam(value="id") Long id){
    Book book = bookSvc.getBook(id);
    return new ModelAndView("/WEB-INF/jsp/showBook.jsp",
     "book",book);
  }
}
```

Controller Returns

- Three options for returnType
  - `void` – controller forwards us to the default view
  - `String` - the name of the view we are delegating to
  - `ModelAndView` – the model and the name of the view
- `String` vs. `ModelAndView`
  - Returning `String` is newer option
    - Model is already on the `HttpRequest` object
  - `ModelAndView` is older option
    - Still used, gives us more control over model and view contents

Request Processing

- Method annotated with `RequestMapping`
- Can determine if it responds to GET or POST requests
- Can determine where to get input data

@RequestMapping

- Basic setup

```
@RequestMapping(value="/allbooks")
public ModelAndView getBooks()
```

- Pass arguments as added part of URL

```
@RequestMapping(value="/book")
public String getBook(@RequestParam("id") Long id,
Model model)
```

- Pass arguments as part of destination URL

```
@RequestMapping(value="/book/{id}")
public ModelAndView getBook(@PathVariable("id") Long id,
Model model)
```

- Can determine where to get input data

"Double Post" Issue

- Accidental resubmission of page
- Use a URL redirect response instead of a forward response

```
return new ModelAndView("redirect:showBooks.jsp", "book", book);
```

View

- Can be anything
  - Need to have converters
  - Considered to be one of the big advantages of Spring MVC
  - Default is .jsp

```
<body>
<H1>Your book:</H1><br/>
Title: ${book.title }<br/>
Author: ${book.author }<br/>

</body>
```

Lab 5 - WebMVC

Integration with Spring Part II

`WebApplicationContext`

- Ultimately, `WebApplicationContext` is the object we want when integrating Spring with the web tier.
- Usually we use `DispatcherServlet`
- Secondarily we can use our own servlet
  - Need things
    - `ContextLoaderListener`
    - `WebApplicationContextUtils`

ContextLoaderListener

Found in `web.xml`

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

WebApplicationContextUtils

Gives us `ApplicationContext` from which we get Spring-bean

```
protected void init() {
 WebApplicationContext appContext =
  WebApplicationContextUtils.getRequiredWebApplicationContext(
   getServletContext());

 BookService service =
  (BookService)appContext.getBean("bookService", BookService.class);

 List<Book> books = service.getAllBooks();
}
```