Spring MVC Introduction

"Code with Passion!"

Topics

- Introduction to Spring MVC
- DispatchServlet, Context configuration
- SpringMVC interfaces

Introduction to Spring MVC

What is Spring MVC?

- MVC-based Web application framework that takes advantage of design principles of Spring framework
  - Dependency Injection
  - Interface-driven design
  - POJO style everywhere
  - Test-driven development (TDD)

Features of Spring MVC

- Clear separation of roles
  - Controller, validator, command object, form object, model object, DispatcherServlet, handler mapping, view resolver, and so on
  - These roles are represented by Java interfaces and their implementations - each of these roles can be pluggable
- Powerful and straightforward configuration
  - Both framework and application classes are configured as POJOs
- Adaptability, non-intrusiveness, and flexibility
  - Example: define any controller method signature you need with @Controller annotation for a given scenario

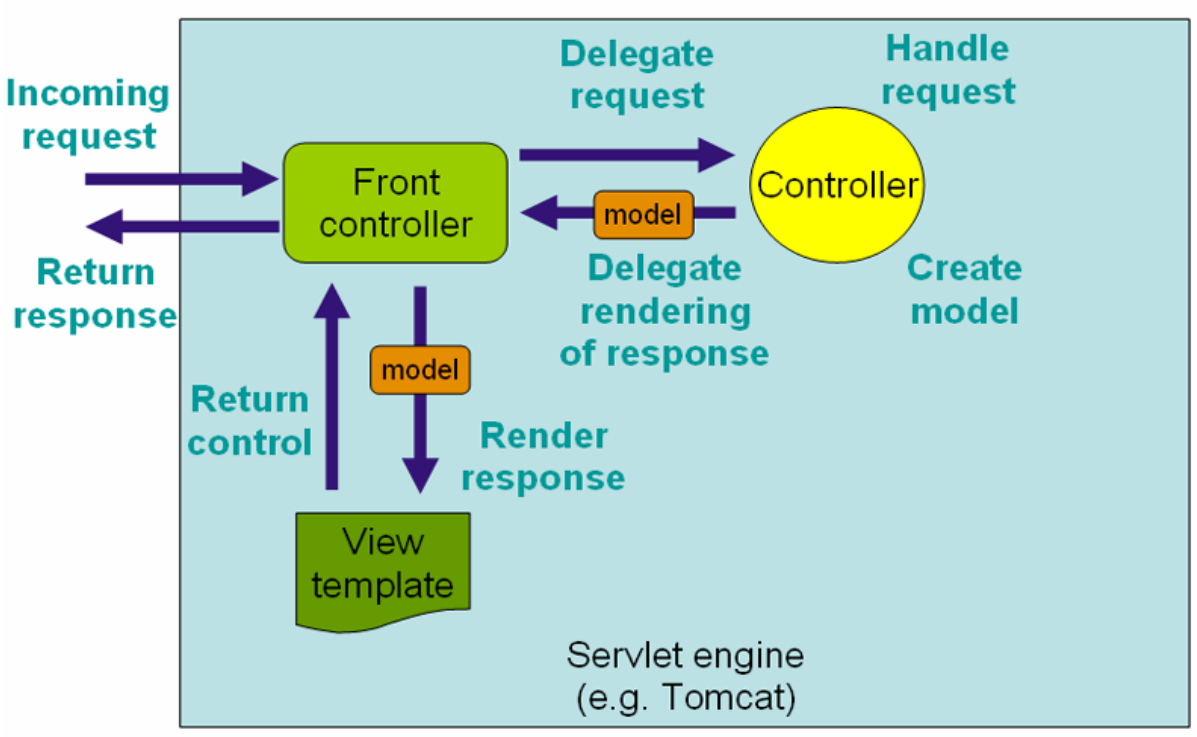
Features of Spring MVC (Continued)

- Customizable data binding, type conversion, and validation
  - Data binding allows user input to be dynamically bound to the domain model of an application
- Customizable handler mapping and view resolution
  - Handler mapping and view resolution strategies range from simple URL-based configuration, to sophisticated resolution strategies
- Flexible model transfer
  - Model transfer with a name/value Map supports easy integration with any view technology
- Customizable locale and theme resolution
- JSP tag library known as the Spring tag library
- Template technologies: thymeleaf, freemarker, velocity

DispatcherServlet

DispatcherServlet Spring Internal Servlet

- It plays the role of Front controller in Spring MVC
  - > Coordinates the HTTP request life-cycle



Sequence of HTTP Request Handling

1. *DispatchServlet* receives a HTTP request
2. *DispatchServlet* selects a Controller (actually a handler method within a Controller) based on the URL Handler mapping and pass the request to the Controller
3. *Controller* (actually a handler method within a Controller) performs the business logic and set values of Model objects
4. *Controller* (actually a handler method within a Controller) returns a logical view
5. *A ViewResolver, which* provides a particular view (JSP, PDF, Excel, etc.), is selected
6. A view gets displayed using Model objects

DispatcherServlet Configuration

- Configured through *SprintBootServletInitializer (if Spring Boot is used)*

SpringMvc Context Configuration

Spring MVC Configuration Files

- Two (or more) Spring MVC configuration files might be a recommended convention for medium/large project
- Example web configuration files (if Java configuration is used)
  - WebConfiguration.java
  - DataSourceConfiguration.java
  - SecurityConfiguration.java
  - ...

Spring MVC framework beans

- Spring MVC framework beans that can be configured
  - Controllers
  - Handler mappings
  - View resolvers
  - Locale resolvers
  - Theme resolvers
  - Multipart file resolver
  - Handler exception resolvers
  - ...

Configuring SpringMVC App

- When ready-to-use configuration is sufficient
  - Usage of *@EnableWebMvc* with *@Configuration*
- When custom configuration is desired (rarely needed)
  - Extend WebMvcConfigurerAdapter class or
  - Create WebMvcConfigurerAdapter bean

Usage of @EnableWebMvc

- Used with @Configuration class to import the Spring MVC configuration defined in *WebMvcConfigurationSupport*

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackageClasses = { MyConfiguration.class })
public class MyWebConfiguration {

}
```

- *WebMvcConfigurationSupport*
  - Provides common and basic set of SpringMVC configurations
  - Registers basic set of Handler mappings, Exception resolvers, etc
  - Auto-configured in Spring Boot apps

Customize configuration option #1

- Extend the WebMvcConfigurerAdapter base class

```
@Configuration
@ComponentScan(basePackageClasses = { MyConfiguration.class })
public class MyConfiguration extends WebMvcConfigurationSupport {

    @Override
    public void addFormatters(FormatterRegistry formatterRegistry) {
        formatterRegistry.addConverter(new MyConverter());
    }

    // More overridden methods ...
}
```

Customize configuration option #2

- Create WebMvcConfigurerAdapter bean

```
@Bean
WebMvcConfigurerAdapter mvcViewConfigurer() {
        return new WebMvcConfigurerAdapter() {
            @Override
            public void addViewControllers(ViewControllerRegistry registry) {
                registry.addRedirectViewController("/", "/index.html");
            }
        };
}
```

Spring MVC Interfaces

What are Spring MVC Interfaces for?

- Key features of the Spring MVC are modularized through Spring MVC interfaces
- Spring framework comes with built-in implementations of these interfaces
  - Default implementation is pre-selected for most common cases
- Custom implementations can be created and configured
  - This is how Spring framework itself can be extended and customized

Examples of Spring MVC Interfaces

- *HandlerMapping*
  - Mapping of requests to controllers/handlers
- *ViewResolver*
  - Maps symbolic name to a view
- *HandlerExceptionResolver*
  - Maps exceptions to error pages
- *LocaleResolver*
  - Selects locale based on HTTP accept header, cookie

Code with Passion!