№2 Spring MVC

Spring MVC: паттерн проектирование Front Controller, класс DispatcherServlet, компоненты Controller, View, ViewResolver, Handler Mapping and HandlerAdapter и принципы их взаимодействия. Жизненный цикл запроса.

Фреймворк **Spring MVC** обеспечивает архитектуру паттерна Model — View — Controller (Модель — Отображение (далее — Вид) — Контроллер) при помощи слабо связанных готовых компонентов. Паттерн MVC разделяет аспекты приложения (логику ввода, бизнес-логику и логику UI), обеспечивая при этом свободную связь между ними.

Model (Модель) инкапсулирует (объединяет) данные приложения, в целом они будут состоять из POJO («Старых добрых Java-объектов», или бинов).

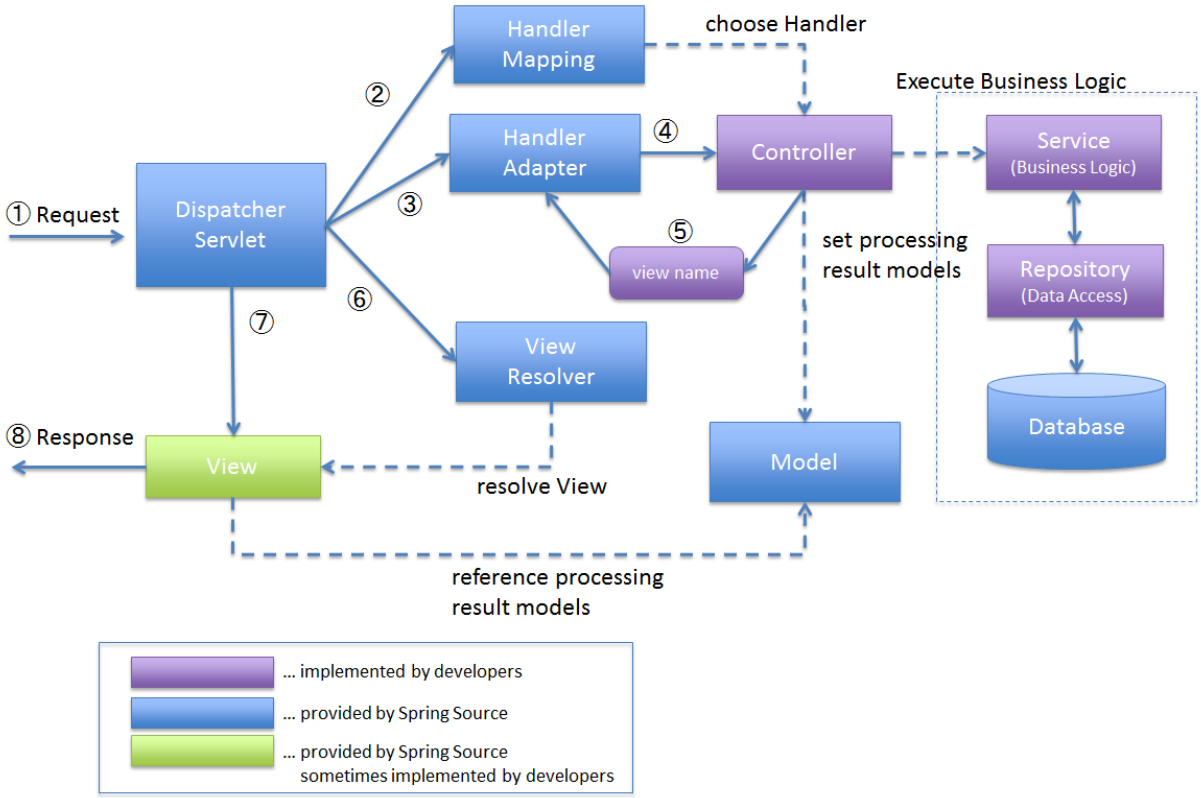
View (Отображение, Вид) отвечает за отображение данных Модели, — как правило, генерируя HTML.

Controller (Контроллер) обрабатывает запрос пользователя, создаёт соответствующую Модель и передаёт её для отображения в Вид.

Spring MVC построен вокруг центрального сервлета, который распределяет запросы по контроллерам, а также предоставляет другие широкие возможности при разработке веб приложений.

На самом деле DispatcherServlet — полностью интегрированный сервлет в Spring IoC контейнер и таким образом получает доступ ко всем возможностям Spring.

Обработка запросов в DispatcherServlet показана на рисунке



Используется паттерн «Front Controller». - **Front Controller - один обработчик для обработки всех типов запросов приложения (может быть веб-приложение , рабочий стол)**

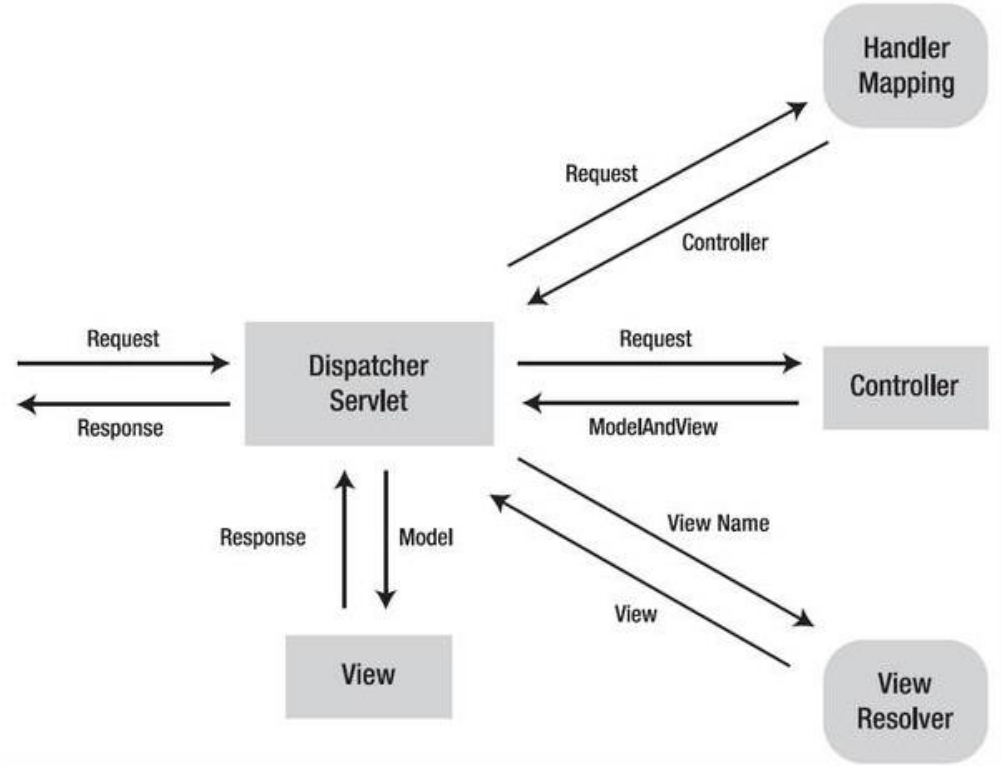
DispatcherServlet анализирует запросы и направляет их соответствующему контроллеру для обработки. В Spring MVC может существовать произвольное количество экземпляров DispatcherServlet, предназначенных для разных целей. Каждый экземпляр DispatcherServlet имеет собственную конфигурацию WebApplicationContext

DispatcherServlet — это обычный сервлет (наследуется от базового класса HttpServlet). Вам необходимо указать мэппинг запросов, которые будут обрабатываться в DispatcherServlet, путем указания URL.

По определению **HandlerMapping** — интерфейс, который реализуется объектами, которые определяют отображение между запросами и объектами обработчиков.

ViewResolver — распознаватель представлений. Интерфейс ViewResolver поддерживает распознавание представлений на основе логического имени, возвращаемого контроллером. Для поддержки различных механизмов распознавания представлений предусмотрено множество классов реализации. Например, класс UrlBasedViewResolver поддерживает прямое преобразование логических имен в URL. Класс ContentNegotiatingViewResolver поддерживает динамическое распознавание представлений в зависимости от типа медиа, поддерживаемого клиентом (XML, PDF, JSON и т.д.). Существует также несколько реализаций для интеграции с различными технологиями представлений, такими как

FreeMarker (FreeMarkerViewResolver), Velocity (VelocityViewResolver) и JasperReports (JasperReportsViewResolver).



Адресация в Контроллере

```
@Controller
@RequestMapping("/hello")
public class HelloController {
    @RequestMapping(method = RequestMethod.GET)
    public String printHello(ModelMap model) {
        model.addAttribute("message", "Hello Spring MVC Framework!");
        return "hello";
    }
}
```

Аннотация @**Controller** определяет класс как контроллер Spring MVC., @**RequestMapping** указывает, что все методы в данном Контроллере относятся к URL-адресу "/hello".

```
@Controller
public class HelloController {
    @RequestMapping(value = "/hello", method = RequestMethod.GET)
    public String printHello(ModelMap model) {
        model.addAttribute("message", "Hello Spring MVC Framework!");
        return "hello";
    }
}
```

Итого: Аннотации @RequestMapping определяют обработчики для классов Controller и / или для методов контроллера. Эти аннотации ищутся среди стереотипных классов DispatcherServlet. Основная идея аннотаций @RequestMapping состоит в том, чтобы определить первичное сопоставление путей на уровне класса и сузить методы, заголовки, параметры и типы медиа-запросов в методах.

В **@RequestMapping** доступны следующие параметры:

Параметр/тип	Использование/описание
name (String)	Присваивает имя сопоставленю
value (String[])	URI отображения пути Также поддерживаются шаблоны пути в стиле ant (например, /path/*.do). Могут содержать заполнители (например, / $ {connect}) для локальных свойств и / или системных свойств и переменных среды. Путь реализует шаблоны URI, которые предоставляют доступ к выбранным частям URL-адреса с помощью шаблонов, переменных, заполнителей и матричных переменных На уровне метода относительные пути (например, edit.do) поддерживаются в первичном отображении, выраженном на уровне типа.
method (RequestMethod[])	GET, POST, HEAD, OPTIONS, PUT, PATCH, DELETE, TRACE.
params (String[])	последовательность выражений стиля param = value. Выражения могут быть отменены с помощью оператора !=, param! = value.
headers (String[])	Последовательность выражений стиля My-Header = myValue. Поддерживается указание только имени заголовка (например, My-Header) (разрешено иметь любое значение). Поддерживается отрицание имени заголовка (например, "!My-Header") (указанный заголовок не должен присутствовать в запросе). Также поддерживает символы подстановки медиа-типа (*) для заголовков, таких как Accept и Content-Type.
consumes (String[])	Используемые медиа типы в маппинге запроса Отображается только в том случае, если {@code Content-Type} соответствует одному из этих типов мультимедиа. Отрицание выражения (например, !text xml) также поддерживается.

produces (String[])	Производимые медиа типы в меппинге запроса. Отображается если {@code Accept} соответствует одному из типов мультимедиа. Отрицание выражения (также поддерживается.

Все эти параметры могут использоваться как на уровне типа, так и на уровне метода. При использовании на уровне типа все параметры уровня метода наследуют сужение родительского уровня

Поддерживаемые типы аргументов методов.

```
@Controller
public class HelloController {

    @RequestMapping(value = "/hello", method = RequestMethod.GET)
    public String printHello(ModelMap model) {
        model.addAttribute("message", "Hello Spring MVC Framework!");
        return "hello";
    }
}
```

Типы аргументов представлены в следующей таблице:

Аргумент метода	Использование и описание
ServletRequest / HttpServletRequest	Внедряет запрос
ServletRespoce / HttpServletResponce	Внедряет ответ
HttpSession	Внедряет сеанс HTTP, связанный с запросом сервлета. Если ноль, Spring создает новый. Параметр synchronizeOnSession должен быть установлен на AbstractController или в RequestMappingHandlerAdapter, если сеансы должны совместно использоваться несколькими запросами.
WebRequest / NativeWebRequest	Добавляет оболочку для доступа только к параметрам запроса и атрибутам запроса /сеанса.
Locale	Внедряет локаль запроса, используя настроенный LocaleResolver.
InputStream / Reader OutputStream / Writer	Обеспечивает прямой доступ к запросу / ответу.
HttpMethod	Внедряет текущий метод запроса
Principal	Используя контекст безопасности Spring, внедряет аутентифицированную учетную запись.
HttpEntity<?>	Spring преобразует и внедряет входящий запрос в пользовательский тип, используя HttpMessageConverter. Он также обеспечивает доступ к заголовкам запросов

Map Model MadelMap	Создает BindingAwareModelMap для использования в представлении
RedirectAttributes	Вставляет и повторно заполняет карту атрибутов и атрибутов flash, поддерживаемых при перенаправлении запроса
Errors BindingResult	Вставляет результаты проверки аргумента, расположенного непосредственно перед списком аргументов.
SessionStatus	позволяет пометить с помощью setComplete (Boolean), завершение сеанса. Этот метод очищает атрибуты сеанса, определенные на уровне типа с помощью @SessionAttributes
UriComponentsBuilder	Внедряет конструктор URL-адресов Spring UriComponentsBuilder.

Поддерживаемые аннотации для аргументов метода

Разработан набор собственных аннотаций для аргументов обработчика метода. Они должны рассматриваться как дескрипторы, которые настраивают веб-поведение методов контроллера в отношении входящих запросов или ответа, который еще предстоит построить.

Пакет

org. springframework. web.bind. annotation.*

@PathVariable	Вставляет переменную шаблона URI в аргумент.
@ MatrixVariable	Вставляет пары имя-значение, расположенные в сегментах пути URI, в аргумент.
@RequestParam	Вставляет определенный параметр запроса в аргумент
@RequestHeader	Вводит HTTP-заголовок определенного запроса в аргумент.
@RequestBody	Разрешает прямой доступ к запросу, вставляя его в аргумент
@RequestPart	Вставляет содержимое определенной части (метаданные, файл -данные .) закодированного запроса multipart / form-data в аргумент соответствующего типа (MetaData, MultipartFile ...
@ ModelAttribute	Заполняет автоматически атрибут Модели с использованием шаблона URI. Эта привязка работает до обработки метода

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class HomeController {
    @RequestMapping(value="/index")
    public ModelAndView getRequestExample(@RequestParam("param1") String
param){
        ModelAndView mav = new ModelAndView();
        mav.setViewName("index");
        mav.addObject("param1", param);
        return mav; }
}
```

Поддерживаемые типы возврата

Spring MVC позволяет указать либо ответ, отправленный обратно клиенту, либо необходимую конфигурацию для цели или заполнения переменных промежуточного уровня View.

Поддерживаемые типы возврата	Использование
Model	Spring MVC создает реализацию интерфейса модели для метода-обработчика. Объекты Model заполняются вручную в методе-обработчике или с помощью @ModelAttribute. Представление для отображения должно быть сопоставлено с запросом с помощью RequestToViewNameTranslator
ModelAndView	Объект-обертка для модели с представлением и именем представления. Если указано имя представления, Spring MVC попытается разрешить связанный вид. В противном случае встроенный вид отображается. Объекты Model заполняются вручную в методе или с помощью @ModelAttribute.
Map	Позволяет реализовать пользовательскую модель. Представление для отображения должно быть сопоставлено с запросом с помощью RequestToViewNameTranslator.
View	Позволяет реализовать пользовательскую модель. Представление для отображения должно быть сопоставлено с запросом с помощью RequestToViewNameTranslator..
String	Если аннотация @ResponseBody не указана в методе-обработчике, возвращаемая строка обрабатывается как имя представления (идентификатор представления).

HttpEntity<?> / ResponseEntity<?>	Объекты-обертки для простого управления заголовками ответа и телом, преобразованным в Spring (с HttpMessageConverters).
HttpHeaders	Обертка для объекта HEAD ответов
Callable<?>	Может производить асинхронно типизированный объект, когда поток контролируется Spring MVC.
DeferredResult<?> ListenableFuture<?>	
void	Когда представление разрешается извне с помощью RequestToViewNameTranslator или когда метод печатает непосредственно в ответе.

Шаблоны URI

Шаблоны URI позволяют настраивать универсальные URI с шаблонами и переменными для конечных точек контроллера.

```
@RequestMapping (value = "/*", ...)
```

Подстановочный знак может соответствовать символу, слову или последовательности слов.

Пример:/portal/1, /portal/add, /portal/get-for

Ограничением будет использование другого слеша в последней последовательности:/ portal/add/2

/* - Все ресурсы и каталоги на уровне

/** - Все ресурсы и каталоги на уровне и подуровнях

/portal/**/add/*/1

Сравнение шаблонов путей выполняется Spring MVC, когда заданный URL-адрес соответствует нескольким зарегистрированным шаблонам пути, чтобы выбрать, какой обработчик будет отображать запрос. Будет выбран шаблон, который считается наиболее конкретным. Первый критерий - это число подсчитанных переменных и подстановочных знаков в сравниваемых шаблонах пути: шаблон, имеющий наименьшее количество переменных и подстановочных знаков, считается наиболее конкретным.

Шаблон с двойными подстановочными знаками всегда менее специфичен, чем шаблон без таковых.

Рассмотрим следующую иерархию, идущую от самой подходящей к наименее определенной:

/portal/users
/portal/{users}
/portal/*
/portal/{users}/{as}
/portal/default/*/{users}
/portal/{users}/*
/portal/**/*
/portal/**

ViewResolvers

В dispatcher-context.xml мы определили bean-компонент viewResolver:

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

Бин viewResolver - это конкретный экземпляр предопределенного класса, используемый для обслуживания view.

Эта возможность предотвращает создание дополнительных отображений. В нашей конфигурации мы определили хранилище представлений (/WEB-INF/jsp/*.jsp), и мы можем напрямую ссылаться на index.jsp с помощью индекса String.

Рекомендуется настроить репозиторий JSP в / WEB-INF, чтобы эти JSP не могли быть указаны публично. Вместо JSP-шаблонов мы могли бы использовать Velocity или Freemarker соответственно, используя преобразователи представлений VelocityViewResolver или FreeMarkerViewResolver.

Кроме того, ContentNegotiatingViewResolver позже может быть использован когда будем создавать API REST.

@PathVariable

@PathVariable используется для чтения переменных в шаблонах URI

```
@RequestMapping(value="/example/{param}")
public HttpEntity<String> example(@PathVariable("param") String parameter) {
    return new HttpEntity<String>(parameter);
}
```

Если бы вызвали URI/portal/example/one, он был бы отображен как ответ от возвращенного HTTPEntity: String one.

Или можно было бы сделать так

```
@RequestMapping(value="/example/{param}")
public HttpEntity<String> example(@PathVariable String  parameter) {
    return new HttpEntity<String>(parameter);
}
```

Не предоставляя значение для аннотации, Spring MVC по умолчанию будет искать в шаблоне URI переменную с тем же именем, что и целевой аргумент.

Другие Mapping аннотации

Спецификация класса @RequestMapping может уточняется с помощью аннотации: @GetMapping. @GetMapping в паре с классом уровня @RequestMapping указывает, что при получении запроса HTTP GET этот метод будет вызван для обработки запроса.

@GetMapping - это относительно новая аннотация, появившаяся в Spring 4.3. До Spring 4.3 могли использовать аннотацию @RequestMapping уровня метода:

```
@RequestMapping(method=RequestMethod.GET)
```

Очевидно, что @GetMapping более лаконичен и специфичен для метода HTTP, на который он нацелен. Однако, @GetMapping - всего лишь одна из семейства аннотаций отображения запросов. В Таблице перечислены все аннотации отображения запросов, доступные в Spring MVC.

Таблица

Аннотации	Описание
@RequestMapping	Обработка запросов общего назначения
@GetMapping	Обработка GET запросов
@PostMapping	Обработка POST запросов
@PutMapping	Обработка PUT запросов
@DeleteMapping	Обработка DELETE запросов
@PatchMapping	Обработка PATCH запросов

Новые аннотации сопоставления запросов имеют все те же атрибуты, что и @RequestMapping, так что вы можете использовать их везде, где использовали @RequestMapping.

Обычно @RequestMapping используется на уровне класса. А более конкретные @GetMapping, @PostMapping и т.д. аннотации используются на каждом из методов-обработчиков.