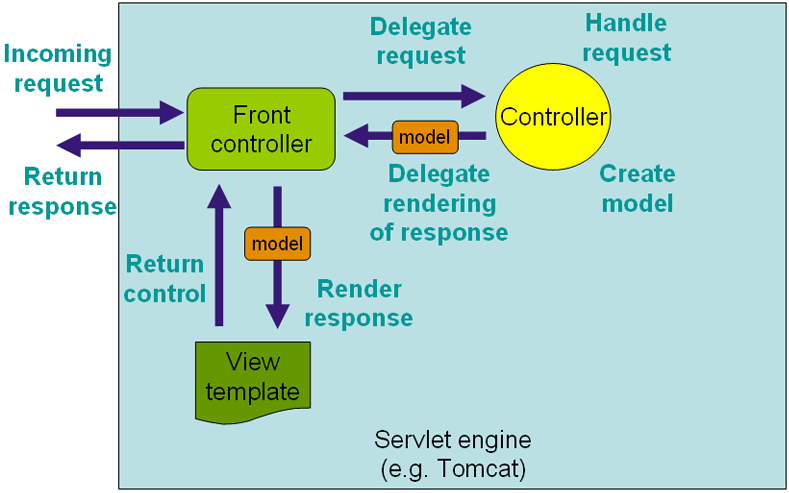
@RequestMapping 是 Spring Web 应用程序中最常被用到的注解之一。这个注解会将 HTTP 请求映射到 MVC 和 REST 控制器的处理方法上。   
  
在这篇文章中，你将会看到 @RequestMapping 注解在被用来进行 Spring MVC 控制器方法的映射可以如何发挥其多才多艺的功能的。   
  
**Request Mapping 基础用法**   
  
在 Spring MVC 应用程序中，RequestDispatcher (在 Front Controller 之下) 这个 servlet 负责将进入的 HTTP 请求路由到控制器的处理方法。   
  
在对 Spring MVC 进行的配置的时候, 你需要指定请求与处理方法之间的映射关系。



要配置 Web 请求的映射，就需要你用上 @RequestMapping 注解。   
  
@RequestMapping 注解可以在控制器类的级别和/或其中的方法的级别上使用。   
  
在类的级别上的注解会将一个特定请求或者请求模式映射到一个控制器之上。之后你还可以另外添加方法级别的注解来进一步指定到处理方法的映射关系。   
  
下面是一个同时在类和方法上应用了 @RequestMapping 注解的示例：

**代码**

1. @RestController
2. @RequestMapping("/home")
3. public class IndexController {
4. @RequestMapping("/")
5. String get() {
6. //mapped to hostname:port/home/
7. return "Hello from get";
8. }
9. @RequestMapping("/index")
10. String index() {
11. //mapped to hostname:port/home/index/
12. return "Hello from index";
13. }
14. }

如上述代码所示，到 /home 的请求会由 get() 方法来处理，而到 /home/index 的请求会由 index() 来处理。   
  
**@RequestMapping 来处理多个 URI**   
  
你可以将多个请求映射到一个方法上去，只需要添加一个带有请求路径值列表的 @RequestMapping 注解就行了。

**代码**

1. @RestController
2. @RequestMapping("/home")
3. public class IndexController {
5. @RequestMapping(value = {
6. "",
7. "/page",
8. "page\*",
9. "view/\*,\*\*/msg"
10. })
11. String indexMultipleMapping() {
12. return "Hello from index multiple mapping.";
13. }
14. }

如你在这段代码中所看到的，@RequestMapping 支持统配符以及ANT风格的路径。前面这段代码中，如下的这些 URL 都会由 indexMultipleMapping() 来处理： 

* localhost:8080/home
* localhost:8080/home/
* localhost:8080/home/page
* localhost:8080/home/pageabc
* localhost:8080/home/view/
* localhost:8080/home/view/view

**带有 @RequestParam 的 @RequestMapping**   
  
@RequestParam 注解配合 @RequestMapping 一起使用，可以将请求的参数同处理方法的参数绑定在一起。   
  
@RequestParam 注解使用的时候可以有一个值，也可以没有值。这个值指定了需要被映射到处理方法参数的请求参数, 代码如下所示：

**代码**

1. @RestController
2. @RequestMapping("/home")
3. public class IndexController {
5. @RequestMapping(value = "/id")
6. String getIdByValue(@RequestParam("id") String personId) {
7. System.out.println("ID is " + personId);
8. return "Get ID from query string of URL with value element";
9. }
10. @RequestMapping(value = "/personId")
11. String getId(@RequestParam String personId) {
12. System.out.println("ID is " + personId);
13. return "Get ID from query string of URL without value element";
14. }
15. }

在代码的第6行，id 这个请求参数被映射到了 thegetIdByValue() 这个处理方法的参数 personId 上。   
  
如果请求参数和处理方法参数的名称一样的话，@RequestParam 注解的 value 这个参数就可省掉了, 如代码的第11行所示。   
  
@RequestParam 注解的 required 这个参数定义了参数值是否是必须要传的。

**代码**

1. @RestController
2. @RequestMapping("/home")
3. public class IndexController {
4. @RequestMapping(value = "/name")
5. String getName(@RequestParam(value = "person", required = false) String personName) {
6. return "Required element of request param";
7. }
8. }

在这段代码中，因为 required 被指定为 false，所以 getName() 处理方法对于如下两个 URL 都会进行处理： 

* /home/name?person=xyz
* /home/name

@RequestParam 的 defaultValue 取值就是用来给取值为空的请求参数提供一个默认值的。

**代码**

1. @RestController
2. @RequestMapping("/home")
3. public class IndexController {
4. @RequestMapping(value = "/name")
5. String getName(@RequestParam(value = "person", defaultValue = "John") String personName) {
6. return "Required element of request param";
7. }
8. }

在这段代码中，如果 person 这个请求参数为空，那么 getName() 处理方法就会接收 John 这个默认值作为其参数。   
  
**用 @RequestMapping 处理 HTTP 的各种方法**   
  
Spring MVC 的 @RequestMapping 注解能够处理 HTTP 请求的方法, 比如 GET, PUT, POST, DELETE 以及 PATCH。   
  
所有的请求默认都会是 HTTP GET 类型的。   
  
为了能降一个请求映射到一个特定的 HTTP 方法，你需要在 @RequestMapping 中使用 method 来声明 HTTP 请求所使用的方法类型，如下所示：

**代码**

1. @RestController
2. @RequestMapping("/home")
3. public class IndexController {
4. @RequestMapping(method = RequestMethod.GET)
5. String get() {
6. return "Hello from get";
7. }
8. @RequestMapping(method = RequestMethod.DELETE)
9. String delete() {
10. return "Hello from delete";
11. }
12. @RequestMapping(method = RequestMethod.POST)
13. String post() {
14. return "Hello from post";
15. }
16. @RequestMapping(method = RequestMethod.PUT)
17. String put() {
18. return "Hello from put";
19. }
20. @RequestMapping(method = RequestMethod.PATCH)
21. String patch() {
22. return "Hello from patch";
23. }
24. }

在上述这段代码中， @RequestMapping 注解中的 method 元素声明了 HTTP 请求的 HTTP 方法的类型。   
  
所有的处理处理方法会处理从这同一个 URL( /home)进来的请求, 但要看指定的 HTTP 方法是什么来决定用哪个方法来处理。   
  
例如，一个 POST 类型的请求 /home 会交给 post() 方法来处理，而一个 DELETE 类型的请求 /home 则会由 delete() 方法来处理。   
  
你会看到 Spring MVC 将使用这样相同的逻辑来映射其它的方法。   
  
**用 @RequestMapping 来处理生产和消费对象**   
  
可以使用 @RequestMapping 注解的 produces 和 consumes 这两个元素来缩小请求映射类型的范围。   
  
为了能用请求的媒体类型来产生对象, 你要用到 @RequestMapping 的 produces 元素再结合着 @ResponseBody 注解。   
  
你也可以利用 @RequestMapping 的 comsumes 元素再结合着 @RequestBody 注解用请求的媒体类型来消费对象。   
  
下面这段代码就用到的 @RequestMapping 的生产和消费对象元素：

**代码**

1. @RestController
2. @RequestMapping("/home")
3. public class IndexController {
4. @RequestMapping(value = "/prod", produces = {
5. "application/JSON"
6. })
7. @ResponseBody
8. String getProduces() {
9. return "Produces attribute";
10. }
12. @RequestMapping(value = "/cons", consumes = {
13. "application/JSON",
14. "application/XML"
15. })
16. String getConsumes() {
17. return "Consumes attribute";
18. }
19. }

在这段代码中， getProduces() 处理方法会产生一个 JSON 响应， getConsumes() 处理方法可以同时处理请求中的 JSON 和 XML 内容。   
  
使用 @RequestMapping 来处理消息头   
  
@RequestMapping 注解提供了一个 header 元素来根据请求中的消息头内容缩小请求映射的范围。   
  
在可以指定 header 元素的值，用 myHeader = myValue 这样的格式：

**代码**

1. @RestController
2. @RequestMapping("/home")
3. public class IndexController {
4. @RequestMapping(value = "/head", headers = {
5. "content-type=text/plain"
6. })
7. String post() {
8. return "Mapping applied along with headers";
9. }
10. }

在上面这段代码中， @RequestMapping 注解的 headers 属性将映射范围缩小到了 post() 方法。有了这个，post() 方法就只会处理到 /home/head 并且 content-typeheader 被指定为 text/plain 这个值的请求。   
  
你也可以像下面这样指定多个消息头：

**代码**

1. @RestController
2. @RequestMapping("/home")
3. public class IndexController {
4. @RequestMapping(value = "/head", headers = {
5. "content-type=text/plain",
6. "content-type=text/html"
7. }) String post() {
8. return "Mapping applied along with headers";
9. }
10. }

这样， post() 方法就能同时接受 text/plain 还有 text/html 的请求了。   
  
**使用 @RequestMapping 来处理请求参数**   
  
@RequestMapping 直接的 params 元素可以进一步帮助我们缩小请求映射的定位范围。使用 params 元素，你可以让多个处理方法处理到同一个URL 的请求, 而这些请求的参数是不一样的。   
  
你可以用 myParams = myValue 这种格式来定义参数，也可以使用通配符来指定特定的参数值在请求中是不受支持的。

**代码**

1. @RestController
2. @RequestMapping("/home")
3. public class IndexController {
4. @RequestMapping(value = "/fetch", params = {
5. "personId=10"
6. })
7. String getParams(@RequestParam("personId") String id) {
8. return "Fetched parameter using params attribute = " + id;
9. }
10. @RequestMapping(value = "/fetch", params = {
11. "personId=20"
12. })
13. String getParamsDifferent(@RequestParam("personId") String id) {
14. return "Fetched parameter using params attribute = " + id;
15. }
16. }

在这段代码中，getParams() 和 getParamsDifferent() 两个方法都能处理相同的一个 URL (/home/fetch) ，但是会根据 params 元素的配置不同而决定具体来执行哪一个方法。   
  
例如，当 URL 是 /home/fetch?id=10 的时候, getParams() 会执行，因为 id 的值是10,。对于 localhost:8080/home/fetch?personId=20 这个URL, getParamsDifferent() 处理方法会得到执行，因为 id 值是 20。   
  
  
**使用 @RequestMapping 处理动态 URI**   
  
@RequestMapping 注解可以同 @PathVaraible 注解一起使用，用来处理动态的 URI，URI 的值可以作为控制器中处理方法的参数。你也可以使用正则表达式来只处理可以匹配到正则表达式的动态 URI。

**代码**

1. @RestController
2. @RequestMapping("/home")
3. public class IndexController {
4. @RequestMapping(value = "/fetch/{id}", method = RequestMethod.GET)
5. String getDynamicUriValue(@PathVariable String id) {
6. System.out.println("ID is " + id);
7. return "Dynamic URI parameter fetched";
8. }
9. @RequestMapping(value = "/fetch/{id:[a-z]+}/{name}", method = RequestMethod.GET)
10. String getDynamicUriValueRegex(@PathVariable("name") String name) {
11. System.out.println("Name is " + name);
12. return "Dynamic URI parameter fetched using regex";
13. }
14. }

在这段代码中，方法 getDynamicUriValue() 会在发起到 localhost:8080/home/fetch/10 的请求时执行。这里 getDynamicUriValue() 方法 id 参数也会动态地被填充为 10 这个值。   
  
方法 getDynamicUriValueRegex() 会在发起到 localhost:8080/home/fetch/category/shirt 的请求时执行。不过，如果发起的请求是 /home/fetch/10/shirt 的话，会抛出异常，因为这个URI并不能匹配正则表达式。   
  
@PathVariable 同 @RequestParam的运行方式不同。你使用 @PathVariable 是为了从 URI 里取到查询参数值。换言之，你使用 @RequestParam 是为了从 URI 模板中获取参数值。   
  
**@RequestMapping 默认的处理方法**   
  
在控制器类中，你可以有一个默认的处理方法，它可以在有一个向默认 URI 发起的请求时被执行。   
  
下面是默认处理方法的示例：

**代码**

1. @RestController
2. @RequestMapping("/home")
3. public class IndexController {
4. @RequestMapping()
5. String
6. default () {
7. return "This is a default method for the class";
8. }
9. }

在这段代码中，向 /home 发起的一个请求将会由 default() 来处理，因为注解并没有指定任何值。   
  
**@RequestMapping 快捷方式**   
  
Spring 4.3 引入了方法级注解的变体，也被叫做 @RequestMapping 的组合注解。组合注解可以更好的表达被注解方法的语义。它们所扮演的角色就是针对 @RequestMapping 的封装，而且成了定义端点的标准方法。   
  
例如，@GetMapping 是一个组合注解，它所扮演的是 @RequestMapping(method =RequestMethod.GET) 的一个快捷方式。   
方法级别的注解变体有如下几个： 

* @GetMapping
* @PostMapping
* @PutMapping
* @DeleteMapping
* @PatchMapping

如下代码展示了如何使用组合注解：

**代码**

1. @RestController
2. @RequestMapping("/home")
3. public class IndexController {
4. @GetMapping("/person")
5. public @ResponseBody ResponseEntity < String > getPerson() {
6. return new ResponseEntity < String > ("Response from GET", HttpStatus.OK);
7. }
8. @GetMapping("/person/{id}")
9. public @ResponseBody ResponseEntity < String > getPersonById(@PathVariable String id) {
10. return new ResponseEntity < String > ("Response from GET with id " + id, HttpStatus.OK);
11. }
12. @PostMapping("/person")
13. public @ResponseBody ResponseEntity < String > postPerson() {
14. return new ResponseEntity < String > ("Response from POST method", HttpStatus.OK);
15. }
16. @PutMapping("/person")
17. public @ResponseBody ResponseEntity < String > putPerson() {
18. return new ResponseEntity < String > ("Response from PUT method", HttpStatus.OK);
19. }
20. @DeleteMapping("/person")
21. public @ResponseBody ResponseEntity < String > deletePerson() {
22. return new ResponseEntity < String > ("Response from DELETE method", HttpStatus.OK);
23. }
24. @PatchMapping("/person")
25. public @ResponseBody ResponseEntity < String > patchPerson() {
26. return new ResponseEntity < String > ("Response from PATCH method", HttpStatus.OK);
27. }
28. }

在这段代码中，每一个处理方法都使用 @RequestMapping 的组合变体进行了注解。尽管每个变体都可以使用带有方法属性的 @RequestMapping 注解来互换实现, 但组合变体仍然是一种最佳的实践 — 这主要是因为组合注解减少了在应用程序上要配置的元数据，并且代码也更易读。   
  
**@RequestMapping 总结**   
  
如你在本文中所看到的，@RequestMapping 注解是非常灵活的。你可以使用该注解配置 Spring MVC 来处理大量的场景用例。它可以被用来在 Spring MVC 中配置传统的网页请求，也可以是 REST 风格的 Web 服务。