冰叔博客

spring mvc常用注解的说明

最近一段时间学习了springboot,所以熟悉一下mvc中常用的注解，这样可以方便开发

简介：

@RequestMapping

RequestMapping是一个用来处理请求地址映射的注解，可用于类或方法上。用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。

RequestMapping注解有六个属性，下面我们把她分成三类进行说明。

1、 value， method；

value： 指定请求的实际地址，指定的地址可以是URI Template 模式（后面将会说明）；

method： 指定请求的method类型， GET、POST、PUT、DELETE等；

2、 consumes，produces；

consumes： 指定处理请求的提交内容类型（Content-Type），例如application/json, text/html;

produces: 指定返回的内容类型，仅当request请求头中的(Accept)类型中包含该指定类型才返回；

3、 params，headers；

params： 指定request中必须包含某些参数值是，才让该方法处理。

headers： 指定request中必须包含某些指定的header值，才能让该方法处理请求。

示例：

1、value / method 示例

默认RequestMapping("....str...")即为value的值；

```
@Controller
@RequestMapping("/appointments")
public class AppointmentsController {

    private AppointmentBook appointmentBook;

    @Autowired
    public AppointmentsController(AppointmentBook appointmentBook) {
        this.appointmentBook = appointmentBook;
    }

    @RequestMapping(method = RequestMethod.GET)
    public Map<String, Appointment> get() {
        return appointmentBook.getAppointmentsForToday();
    }
```

SpringSecurity
Struts(3)
测试(1)
设计模式(1)
数据结构(2)

随笔档案

2017年11月 (1)
2017年10月 (3)
2017年9月 (3)
2017年8月 (3)
2017年7月 (7)
2017年6月 (4)
2017年5月 (6)
2016年12月 (4)

最新评论

1. Re:spring boot 之热部署(三)
@huanglei2010能实现就好，自己感觉因为版本的不同，有的能用有的不能用。欢迎随时讨论问题。...
--冰叔博客
2. Re:spring boot 之热部署(三)
thank you，
确实实现了热部署
而且是从main函数启动的热部署，而不是mvn spring-boot:run启动的热部署
--huanglei2010
3. Re:spring boot入门
@luoshupeng对本来想的是下次再说的。添加引用包就可以，也可以添加微信。...
--冰叔博客
4. Re:spring boot入门
@冰叔博客好 了解！...
--Thieves
5. Re:spring boot入门
@Thieves忘记说了，引入fastjson需要添加jar包...
--冰叔博客

阅读排行榜

1. spring boot 之热部署(三)(3431)
2. spring boot 之fastJson的使用（二）(1836)
3. spring boot入门(500)
4. 图片上传(237)
5. java图片上传（mvc）(180)

评论排行榜

1. 图片上传(8)
2. spring boot入门(6)
3. spring boot 之热部署(三)(2)

推荐排行榜

1. spring boot 之热部署(三)(2)

```

    @RequestMapping(value="/{day}", method = RequestMethod.GET)
    public Map<String, Appointment> getForDay(@PathVariable @DateTimeFormat(iso=ISO.DATE) Date day, Model model) {
        return appointmentBook.getAppointmentsForDay(day);
    }

    @RequestMapping(value="/new", method = RequestMethod.GET)
    public AppointmentForm getNewForm() {
        return new AppointmentForm();
    }

    @RequestMapping(method = RequestMethod.POST)
    public String add(@Valid AppointmentForm appointment, BindingResult result) {
        if (result.hasErrors()) {
            return "appointments/new";
        }
        appointmentBook.addAppointment(appointment);
        return "redirect:/appointments";
    }
}
```

value的uri值为以下三类：

A ）可以指定为普通的具体值；

B）可以指定为含有某变量的一类值(URI Template Patterns with Path Variables)；

C) 可以指定为含正则表达式的一类值(URI Template Patterns with Regular Expressions);

example B)

```
@RequestMapping(value="/owners/{ownerId}", method=RequestMethod.GET)
public String findOwner(@PathVariable String ownerId, Model model) {
  Owner owner = ownerService.findOwner(ownerId);
  model.addAttribute("owner", owner);
  return "displayOwner";
}
```

example C)

```
@RequestMapping("/spring-web/{symbolicName:[a-z-]+}-{version:\d\.\d\.\d}.{extension:\.[a-z]}")
  public void handle(@PathVariable String version, @PathVariable String extension) {
    // ...
  }
}
```

2 consumes、produces 示例

cousumes的样例：

```
@Controller
@RequestMapping(value = "/pets", method = RequestMethod.POST, consumes="application/json")
public void addPet(@RequestBody Pet pet, Model model) {
    // implementation omitted
}
```

方法仅处理request Content-Type为“application/json”类型的请求。

produces的样例：

```
@Controller
@RequestMapping(value = "/pets/{petId}", method = RequestMethod.GET,
produces="application/json")
@ResponseBody
public Pet getPet(@PathVariable String petId, Model model) {
    // implementation omitted
}
```

方法仅处理request请求中Accept头中包含了"application/json"的请求，同时暗示了返回的内容类型为application/json;

3 params、headers 示例

params的样例：

```
@Controller
@RequestMapping("/owners/{ownerId}")
public class RelativePathUriTemplateController {

  @RequestMapping(value = "/pets/{petId}", method = RequestMethod.GET,
params="myParam=myValue")
  public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {
    // implementation omitted
  }
}
```

仅处理请求中包含了名为“myParam”，值为“myValue”的请求；

headers的样例：

```
@Controller
@RequestMapping("/owners/{ownerId}")
public class RelativePathUriTemplateController {

@RequestMapping(value = "/pets", method = RequestMethod.GET,
headers="Referer=http://www.ifeng.com/")
  public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {
    // implementation omitted
  }
}
```

@RequestBody

作用：

i) 该注解用于读取Request请求的body部分数据，使用系统默认配置的HttpMessageConverter进行解析，然后把相应的数据绑定到要返回的对象上；

ii) 再把HttpMessageConverter返回的对象数据绑定到 controller中方法的参数上。

使用时机：

A) GET、POST方式提时， 根据request header Content-Type的值来判断:

- application/x-www-form-urlencoded， 可选（即非必须，因为这种情况的数据@RequestParam, @ModelAttribute也可以处理，当然@RequestBody也能处理）；
- multipart/form-data, 不能处理（即使用@RequestBody不能处理这种格式的数据）；
- 其他格式， 必须（其他格式包括application/json, application/xml等。这些格式的数据，必须使用@RequestBody来处理）；

B) PUT方式提交时， 根据request header Content-Type的值来判断:

- application/x-www-form-urlencoded， 必须；
- multipart/form-data, 不能处理；
- 其他格式， 必须；

说明：request的body部分的数据编码格式由header部分的Content-Type指定；

@ResponseBody

作用：

该注解用于将Controller的方法返回的对象，通过适当的HttpMessageConverter转换为指定格式后，写入到Response对象的body数据区。

使用时机：

返回的数据不是html标签的页面，而是其他某种格式的数据时（如json、xml等）使用；

HttpMessageConverter

```
<span style="font-family:Microsoft YaHei;">/**
 * Strategy interface that specifies a converter that can convert from and to HTTP requests
and responses.
 *
 * @author Arjen Poutsma
 * @author Juergen Hoeller
 * @since 3.0
 */
public interface HttpMessageConverter<T> {

    /**
     * Indicates whether the given class can be read by this converter.
     * @param clazz the class to test for readability
     * @param mediaType the media type to read, can be {@code null} if not specified.
     * Typically the value of a {@code Content-Type} header.
     * @return {@code true} if readable; {@code false} otherwise
     */
    boolean canRead(Class<?> clazz, MediaType mediaType);

    /**
     * Indicates whether the given class can be written by this converter.
     * @param clazz the class to test for writability
     * @param mediaType the media type to write, can be {@code null} if not specified.
     * Typically the value of an {@code Accept} header.
     * @return {@code true} if writable; {@code false} otherwise
     */
    boolean canWrite(Class<?> clazz, MediaType mediaType);

    /**
     * Return the list of {@link MediaType} objects supported by this converter.
     * @return the list of supported media types
     */
    List<MediaType> getSupportedMediaTypes();

    /**
     * Read an object of the given type form the given input message, and returns it.
```

```
     * @param clazz the type of object to return. This type must have previously been passed
to the
     * {@link #canRead canRead} method of this interface, which must have returned {@code
true}.
     * @param inputMessage the HTTP input message to read from
     * @return the converted object
     * @throws IOException in case of I/O errors
     * @throws HttpMessageNotReadableException in case of conversion errors
     */
    T read(Class<? extends T> clazz, HttpInputMessage inputMessage)
            throws IOException, HttpMessageNotReadableException;

    /**
     * Write an given object to the given output message.
     * @param t the object to write to the output message. The type of this object must have
previously been
     * passed to the {@link #canWrite canWrite} method of this interface, which must have
returned {@code true}.
     * @param contentType the content type to use when writing. May be {@code null} to
indicate that the
     * default content type of the converter must be used. If not {@code null}, this media
type must have
     * previously been passed to the {@link #canWrite canWrite} method of this interface,
which must have
     * returned {@code true}.
     * @param outputMessage the message to write to
     * @throws IOException in case of I/O errors
     * @throws HttpMessageNotWritableException in case of conversion errors
     */
    void write(T t, MediaType contentType, HttpOutputMessage outputMessage)
            throws IOException, HttpMessageNotWritableException;

}
</span>
```

该接口定义了四个方法，分别是读取数据时的 canRead(), read() 和 写入数据时的canWrite(), write()方法。

在使用 <mvc:annotation-driven />标签配置时，默认配置了RequestMappingHandlerAdapter（注意是RequestMappingHandlerAdapter不是AnnotationMethodHandlerAdapter,详情查看Spring 3.1 document “16.14 Configuring Spring MVC”章节），并为他配置了一下默认的HttpMessageConverter:

```
    ByteArrayHttpMessageConverter converts byte arrays.

    StringHttpMessageConverter converts strings.

    ResourceHttpMessageConverter converts to/from org.springframework.core.io.Resource for all
media types.

    SourceHttpMessageConverter converts to/from a javax.xml.transform.Source.

    FormHttpMessageConverter converts form data to/from a MultiValueMap<String, String>.

    Jaxb2RootElementHttpMessageConverter converts Java objects to/from XML — added if JAXB2 is
present on the classpath.

    MappingJacksonHttpMessageConverter converts to/from JSON — added if Jackson is present on
the classpath.

    AtomFeedHttpMessageConverter converts Atom feeds — added if Rome is present on the
classpath.

    RssChannelHttpMessageConverter converts RSS feeds — added if Rome is present on the
classpath.
```

ByteArrayHttpMessageConverter: 负责读取二进制格式的数据和写出二进制格式的数据；

StringHttpMessageConverter: 负责读取字符串格式的数据和写出二进制格式的数据；

ResourceHttpMessageConverter：负责读取资源文件和写出资源文件数据；

FormHttpMessageConverter： 负责读取form提交的数据（能读取的数据格式为 application/x-www-form-urlencoded，不能读取multipart/form-data格式数据）；负责写入application/x-www-from-urlencoded和multipart/form-data格式的数据；

MappingJacksonHttpMessageConverter: 负责读取和写入json格式的数据；

SouceHttpMessageConverter： 负责读取和写入 xml 中javax.xml.transform.Source定义的数据；

Jaxb2RootElementHttpMessageConverter: 负责读取和写入xml 标签格式的数据；

AtomFeedHttpMessageConverter: 负责读取和写入Atom格式的数据；

RssChannelHttpMessageConverter: 负责读取和写入RSS格式的数据；

当使用@RequestBody和@ResponseBody注解时，RequestMappingHandlerAdapter就使用它们来进行读取或者写入相应格式的数据。

HttpMessageConverter匹配过程：

@RequestBody注解时： 根据Request对象header部分的Content-Type类型，逐一匹配合适的HttpMessageConverter来读取数据；

spring 3.1源代码如下：

```
private Object readWithMessageConverters(MethodParameter methodParam, HttpInputMessage
inputMessage, Class paramType)
            throws Exception {

        MediaType contentType = inputMessage.getHeaders().getContentType();
        if (contentType == null) {
            StringBuilder builder = new
StringBuilder(ClassUtils.getShortName(methodParam.getParameterType()));
            String paramName = methodParam.getParameterName();
            if (paramName != null) {
                builder.append(' ');
                builder.append(paramName);
            }
            throw new HttpMediaTypeNotSupportedException(
                    "Cannot extract parameter (" + builder.toString() + "): no Content-Type
found");
        }

        List<MediaType> allSupportedMediaTypes = new ArrayList<MediaType>();
        if (this.messageConverters != null) {
            for (HttpMessageConverter<?> messageConverter : this.messageConverters) {
                allSupportedMediaTypes.addAll(messageConverter.getSupportedMediaTypes());
                if (messageConverter.canRead(paramType, contentType)) {
                    if (logger.isDebugEnabled()) {
                        logger.debug("Reading [" + paramType.getName() + "] as \"" +
contentType
                                +"\" using [" + messageConverter + "]");
                    }
                    return messageConverter.read(paramType, inputMessage);
```

```
                }
            }
        }
        throw new HttpMediaTypeNotSupportedException(contentType, allSupportedMediaTypes);
    }
```

@ResponseBody注解时： 根据Request对象header部分的Accept属性（逗号分隔），逐一按accept中的类型，去遍历找到能处理的HttpMessageConverter；

源代码如下：

```
private void writeWithMessageConverters(Object returnValue,
                HttpInputMessage inputMessage, HttpOutputMessage outputMessage)
                throws IOException, HttpMediaTypeNotAcceptableException {
            List<MediaType> acceptedMediaTypes = inputMessage.getHeaders().getAccept();
            if (acceptedMediaTypes.isEmpty()) {
                acceptedMediaTypes = Collections.singletonList(MediaType.ALL);
            }
            MediaType.sortByQualityValue(acceptedMediaTypes);
            Class<?> returnValueType = returnValue.getClass();
            List<MediaType> allSupportedMediaTypes = new ArrayList<MediaType>();
            if (getMessageConverters() != null) {
                for (MediaType acceptedMediaType : acceptedMediaTypes) {
                    for (HttpMessageConverter messageConverter : getMessageConverters()) {
                        if (messageConverter.canWrite(returnValueType, acceptedMediaType)) {
                            messageConverter.write(returnValue, acceptedMediaType,
outputMessage);
                            if (logger.isDebugEnabled()) {
                                MediaType contentType =
outputMessage.getHeaders().getContentType();
                                if (contentType == null) {
                                    contentType = acceptedMediaType;
                                }
                                logger.debug("Written [" + returnValue + "] as \"" +
contentType +
                                        "\" using [" + messageConverter + "]");
                            }
                            this.responseArgumentUsed = true;
                            return;
                        }
                    }
                }
                for (HttpMessageConverter messageConverter : messageConverters) {
                    allSupportedMediaTypes.addAll(messageConverter.getSupportedMediaTypes());
                }
            }
            throw new HttpMediaTypeNotAcceptableException(allSupportedMediaTypes);
        }
```

补充：

MappingJacksonHttpMessageConverter 调用了 objectMapper.writeValue(OutputStream stream, Object)方法，使用@ResponseBody注解返回的对象就传入Object参数内。若返回的对象为已经格式化好的json串时，不使用@RequestBody注解，而应该这样处理：
1、response.setContentType("application/json; charset=UTF-8");
2、response.getWriter().print(jsonStr);
直接输出到body区，然后的视图为void。

分类: MVC

标签: MVC, springmvc

« 上一篇：idea创建Maven多模块项目

posted @ 2017-11-04 17:57 冰叔博客 阅读(30) 评论(0) 编辑 收藏