

springMVC的消息转换器（Message Converter）

简介

请求和响应都有对应的body，而这个body就是需要关注的主要数据。

请求体与请求的查询参数或者表单参数是不同的，请求体的表述一般就是一段字符串，而查询参数可以看作url的一部分，这两个是位于请求报文的不同地方。表单参数可以按照一定格式放在请求体中，也可以放在url上作为查询参数。总之可以把请求体看作客户端通过请求报文捎带的字符串。

响应体则是浏览器渲染页面的依据，对于一个普通html页面得响应，响应体就是这个html页面的源代码。

请求体和响应体都是需要配合Content-Type头部使用的，这个头部主要用于说明body中得字符串是什么格式的，比如：text，json，xml等。对于请求报文，只有通过此头部，服务器才能知道怎么解析请求体中的字符串，对于响应报文，浏览器通过此头部才知道应该怎么渲染响应结果，是直接打印字符串还是根据代码渲染为一个网页。

还有一个与body有关的头部是Accept，这个头部标识了客户端期望得到什么格式的响应体。服务器可根据此字段选择合适的结果表述。

对于HttpServletRequest和HttpServletResponse，可以分别调用getInputStream和getOutputStream来直接获取body。但是获取到的仅仅只是一段字符串，而对于java来说，处理一个对象肯定比处理一个字符串要方便得多，也好理解得多。所以根据Content-Type头部，将body字符串转换为java对象是常有的事。反过来，根据Accept头部，将java对象转换客户端期望格式的字符串也是必不可少的工作。

spring消息转换器源码简要分析

而springMVC为我们提供了一系列默认的消息转换器。

▼ **messageConverters** = {HttpMessageConverters@3975}

▼ **converters** = {Collections\$UnmodifiableRandomAccessList@3988} size = 9

- ▶ 0 = {ByteArrayHttpMessageConverter@3990}
- ▶ 1 = {StringHttpMessageConverter@3991}
- ▶ 2 = {StringHttpMessageConverter@3992}
- ▶ 3 = {ResourceHttpMessageConverter@3993}
- ▶ 4 = {SourceHttpMessageConverter@3994}
- ▶ 5 = {AllEncompassingFormHttpMessageConverter@3995}
- ▶ 6 = {MappingJackson2HttpMessageConverter@3996}
- ▶ 7 = {MappingJackson2HttpMessageConverter@3997}
- ▶ 8 = {Jaxb2RootElementHttpMessageConverter@3998}

对于消息转换器的调用，都是在RequestResponseBodyMethodProcessor类中完成的。它实现了HandlerMethodArgumentResolver和HandlerMethodReturnValueHandler两个接口，分别实现了处理参数和处理返回值的方法。

而要动用这些消息转换器，需要在特定的位置加上@RequestBody和@ResponseBody。

```
/**
 * RequestResponseBodyMethodProcessor.class
 */
...
@Override
public boolean supportsParameter(MethodParameter parameter) {
    return parameter.hasParameterAnnotation(RequestBody.class);
}

public boolean supportsReturnType(MethodParameter returnType) {
    return (AnnotatedElementUtils.hasAnnotation(returnType.getContainingClass(), ResponseBody.class) ||
            returnType.hasMethodAnnotation(ResponseBody.class));
}
...
```

对返回值的消息转换来说：

```
/**
 * RequestResponseBodyMethodProcessor.class
 */
...
public void handleReturnValue(Object returnValue, MethodParameter returnType,
                             ModelAndViewContainer mavContainer, NativeWebRequest webRequest,
```

throws IOException, HttpMediaTypeNotAcceptableException, HttpMe

大致流程为：

1.根据返回值获取其类型，其中MethodParameter封装了方法对象，可获去方法返回值类型。

```
/**
 * AbstractMessageConverterMethodProcessor.class
 */
...

        outputValue = value;
        valueType = getReturnValueType(outputValue, returnType);
        declaredType = getGenericType(returnType);
```

2.根据request的Accept和HandlerMapping的produces属性经过比对、排序从而得到最应该转换的消息格式（MediaType）。

```
/**
 * AbstractMessageConverterMethodProcessor.class
 */
...

        List<MediaType> requestedMediaTypes = getAcceptableMediaTypes(request);
        List<MediaType> producibleMediaTypes = getProducibleMediaTypes(request);
        Set<MediaType> compatibleMediaTypes = new LinkedHashSet<MediaType>();
        ...
//匹配
        ...

        List<MediaType> mediaTypes = new ArrayList<MediaType>(compatibleMediaTypes);
        MediaType.sortBySpecificityAndQuality(mediaTypes);

        MediaType selectedMediaType = null;
        ...
//选择最具体的MediaType
        ...
```

3.遍历所有已配置的消息转换器，调用其canWrite方法，根据返回值类型（valueType）和消息格式（MediaType）来检测是否可以转换。

```
/**
 * AbstractMessageConverterMethodProcessor.class
 */
```

```

...
for (HttpMessageConverter<?> messageConverter : this.messageConverters) {
...
    if (messageConverter.canWrite(valueType, selectedMediaType)){
...
        ((HttpMessageConverter) messageConverter).write(outputValue, select
...
    }
}
}

```

4.若有对应的转换器，则执行消息转换，即write方法。在write方法中，将返回值被转换后得到的字符串写在Response的输出流中。若找不到对应的转换器，则抛出HttpMediaTypeNotAcceptableException异常，浏览器会收到一个406 Not Acceptable状态码。

自定义消息转换器

除了spring提供的9个默认的消息转换器，还可以添加自定义的消息转换器，或者更换消息转换器的实现。

一个自定义消息转换器的例子：

该例子旨在将json转换器替换为fastjson实现，xml转换器替换为jackson-dataformat-xml实现。

首先添加依赖：

```

<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.17</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.8.7</version>
</dependency>

```

配置类：

```

@Configuration
public class Cfg_Web {

```

```

//message converter
@Bean
public HttpMessageConverters messageConverters(){
    //json
    FastJsonHttpMessageConverter jsonMessageConverter = new FastJsonHttpMessageConverter();
    FastJsonConfig fastJsonConfig = new FastJsonConfig();
    fastJsonConfig.setCharset(Charset.forName("utf-8"));
    jsonMessageConverter.setFastJsonConfig(fastJsonConfig);
    List<MediaType> jsonMediaTypes = new ArrayList<>();
    jsonMediaTypes.add(MediaType.APPLICATION_JSON);
    jsonMediaTypes.add(MediaType.APPLICATION_JSON_UTF8);
    jsonMessageConverter.setSupportedMediaTypes(jsonMediaTypes);

    //xml
    MappingJackson2XmlHttpMessageConverter xmlMessageConverter = new MappingJackson2XmlHttpMessageConverter();
    xmlMessageConverter.setObjectMapper(new XmlMapper());
    xmlMessageConverter.setDefaultCharset(Charset.forName("utf-8"));
    List<MediaType> xmlMediaTypes = new ArrayList<>();
    xmlMediaTypes.add(MediaType.APPLICATION_XML);
    xmlMediaTypes.add(MediaType.TEXT_XML);
    xmlMessageConverter.setSupportedMediaTypes(xmlMediaTypes);

    return new HttpMessageConverters(Arrays.asList(jsonMessageConverter, xmlMessageConverter));
}
}

```

测试使用：

```

public class Student {
    private String code;
    private String name;
    ...//省略set和get方法
    @Override
    public String toString() {
        return "Student{" +
            "code='" + code + '\'' +
            ", name='" + name + '\'' +
            '}';
    }
}

```

@Controller

```
public class TestController {
```

```

    @RequestMapping(value = "json", produces = MediaType.APPLICATION_XML_VALUE)
    @ResponseBody

```

```

public Object jsonTest(@RequestBody Student student){
    System.out.println(student);

    return student;
}

@RequestMapping(value = "xml", produces = MediaType.APPLICATION_JSON_UTF
@ResponseBody
public Object xmlTest(@RequestBody Student student){
    System.out.println(student);

    return student;
}
}

```

这个测试主要是将json格式和xml格式请求响应互转。

REQUEST
URL
POST GET POST PUT
New request Paste Request Authentication...
Content to Send Headers Parameters
Content Type:
Content Options: Base64 Parameter Body
☒ Content ☐ File
{"code": "123", "name": "JerryL"}

RESPONSE
POST on http://localhost:8080/json
Status: 200

该 XML 文件并未包含任何关联的样式信息

- <Student>
<code>123</code>
<name>JerryL</name>
</Student>

json -> xml

REQUEST
URL
POST GET POST PUT
New request Paste Request Authentication...
Content to Send Headers Parameters
Content Type:
Content Options: Base64 Parameter Body
☒ Content ☐ File
<student><code>123</code><name>JerryL</name></student>

RESPONSE
POST on http://localhost:8080/xml
Status: 200
JSON 原始数据 头
保存 复制
code: "123"
name: "JerryL"

xml -> json

一些小细节

1.如果一个Controller类里面所有方法的返回值都需要经过消息转换器，那么

可以在类上面加上@ResponseBody注解或者将@Controller注解修改为@RestController注解，这样做就相当于在每个方法都加上了@ResponseBody注解了。

2.@ResponseBody和@RequestBody都可以处理Map类型的对象。如果不确定参数的具体字段，可以用Map接收。@RequestBody同样适用。

3.方法上的和类上的@ResponseBody都可以被继承。

4.默认的xml转换器Jaxb2RootElementHttpMessageConverter需要类上有@XmlRootElement注解才能被转换。

```
/**
 * Jaxb2RootElementHttpMessageConverter.class
 */
@Override
public boolean canWrite(Class<?> clazz, MediaType mediaType) {
    return (AnnotationUtils.findAnnotation(clazz, XmlRootElement.class))
}
```

5.返回值类型可声明为基类的类型，不影响转换，但参数的类型必需为特定的类型。这是显而易见的。