SpringMVC源码剖析（五)-消息转换器HttpMessageConverter - 相见欢

#概述 在SpringMVC中，可以使用@RequestBody和@ResponseBody两个注解，分别完成请求报文到对象和对象到响应报文的转换，底层这种灵活的消息转换机制，就是Spring3.x中新引入的HttpMessageConverter即消息转换器机制。

#Http请求的抽象 还是回到请求-响应，也就是解析请求体，然后返回响应报文这个最基本的Http请求过程中来。我们知道，在servlet标准中，可以用javax.servlet.ServletRequest接口中的以下方法：

```
public ServletInputStream getInputStream() throws IOException;
```

来得到一个ServletInputStream。这个ServletInputStream中，可以读取到一个原始请求报文的所有内容。同样的，在javax.servlet.ServletResponse接口中，可以用以下方法：

```
public ServletOutputStream getOutputStream() throws IOException;
```

来得到一个ServletOutputStream，这个ServletOutputSteam，继承自java中的OutputStream，可以让你输出Http的响应报文内容。

让我们尝试着像SpringMVC的设计者一样来思考一下。我们知道，Http请求和响应报文本质上都是一串字符串，当请求报文来到java世界，它会被封装成为一个ServletInputStream的输入流，供我们读取报文。响应报文则是通过一个ServletOutputStream的输出流，来输出响应报文。

我们从流中，只能读取到原始的字符串报文，同样，我们往输出流中，也只能写原始的字符。而在java世界中，处理业务逻辑，都是以一个个有业务意义的对象为处理维度的，那么在报文到达SpringMVC和从SpringMVC出去，都存在一个字符串到java对象的阻抗问题。这一过程，不可能由开发者手工转换。我们知道，在Struts2中，采用了OGNL来应对这个问题，而在SpringMVC中，它是HttpMessageConverter机制。我们先来看两个接口。

#HttpInputMessage 这个类是SpringMVC内部对一次Http请求报文的抽象，在HttpMessageConverter的read()方法中，有一个HttpInputMessage的形参，它正是SpringMVC的消息转换器所作用的受体“请求消息”的内部抽象，消息转换器从“请求消息”中按照规则提取消息，转换为方法形参中声明的对象。

```
package org.springframework.http;

import java.io.IOException;
import java.io.InputStream;

public interface HttpInputMessage extends HttpMessage {

        InputStream getBody() throws IOException;

}
```

#HttpOutputMessage 这个类是SpringMVC内部对一次Http响应报文的抽象，在HttpMessageConverter的write()方法中，有一个HttpOutputMessage的形参，它正是SpringMVC的消息转换器所作用的受体“响应消息”的内部抽象，消息转换器将“响应消息”按照一定的规则写到响应报文中。

```
package org.springframework.http;

import java.io.IOException;
import java.io.OutputStream;

public interface HttpOutputMessage extends HttpMessage {

        OutputStream getBody() throws IOException;

}
```

#HttpMessageConverter 对消息转换器最高层次的接口抽象，描述了一个消息转换器的一般特征，我们可以从这个接口中定义的方法，来领悟Spring3.x的设计者对这一机制的思考过程。

```
package org.springframework.http.converter;

import java.io.IOException;
import java.util.List;
```

```
import org.springframework.http.HttpInputMessage;
import org.springframework.http.HttpOutputMessage;
import org.springframework.http.MediaType;

public interface HttpMessageConverter<T> {

        boolean canRead(Class<?> clazz, MediaType mediaType);

        boolean canWrite(Class<?> clazz, MediaType mediaType);

        List<MediaType> getSupportedMediaTypes();

        T read(Class<? extends T> clazz, HttpInputMessage inputMessage)
                        throws IOException, HttpMessageNotReadableException

        void write(T t, MediaType contentType, HttpOutputMessage outputMess
                        throws IOException, HttpMessageNotWritableException

}
```

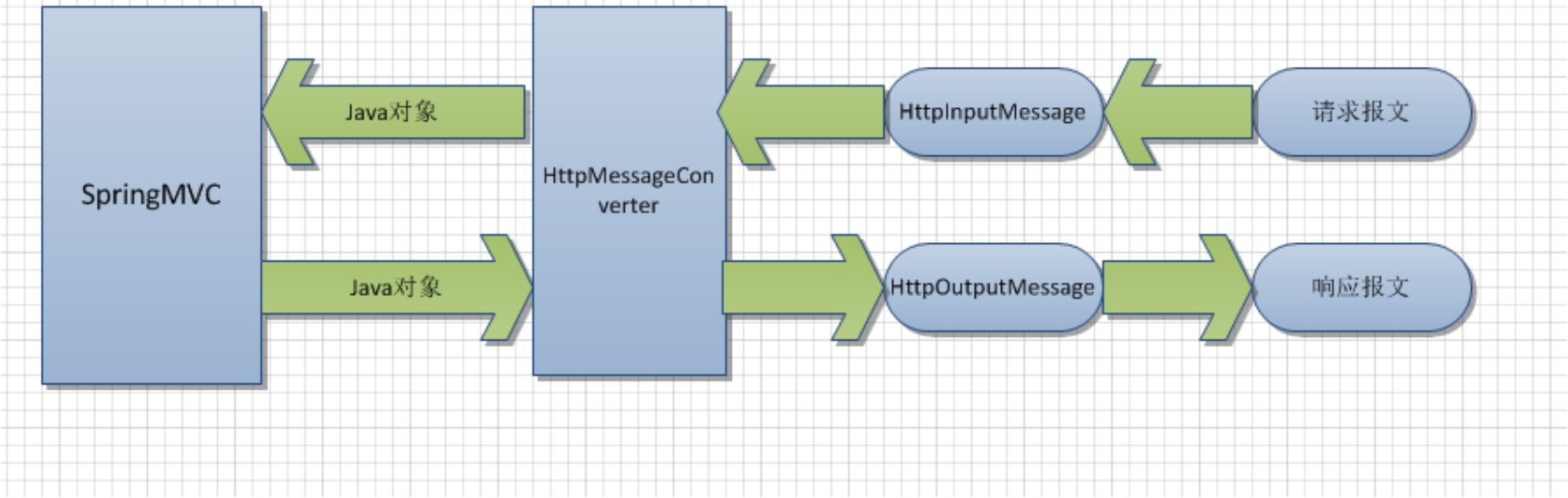
HttpMessageConverter接口的定义出现了成对的canRead()，read()和canWrite()，write()方法，MediaType是对请求的Media Type属性的封装。举个例子，当我们声明了下面这个处理方法。

```
@RequestMapping(value="/string", method=RequestMethod.POST)
public @ResponseBody String readString(@RequestBody String string) {
    return "Read string '" + string + "'";
}
```

在SpringMVC进入readString方法前，会根据@RequestBody注解选择适当的HttpMessageConverter实现类来将请求参数解析到string变量中，具体来说是使用了StringHttpMessageConverter类，它的canRead()方法返回true，然后它的read()方法会从请求中读出请求参数，绑定到readString()方法的string变量中。

当SpringMVC执行readString方法后，由于返回值标识了@ResponseBody，SpringMVC将使用StringHttpMessageConverter的write()方法，将结果作为String值写入响应报文，当然，此时canWrite()方法返回true。

我们可以用下面的图，简单描述一下这个过程。



#RequestResponseBodyMethodProcessor 将上述过程集中描述的一个类是org.springframework.web.servlet.mvc.method.annotation.RequestResponseBodyMethodProcessor，这个类同时实现了HandlerMethodArgumentResolver和HandlerMethodReturnValueHandler两个接口。前者是将请求报文绑定到处理方法形参的策略接口，后者则是对处理方法返回值进行处理的策略接口。两个接口的源码如下：

```
package org.springframework.web.method.support;

import org.springframework.core.MethodParameter;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.support.WebDataBinderFactory;
import org.springframework.web.context.request.NativeWebRequest;

public interface HandlerMethodArgumentResolver {

        boolean supportsParameter(MethodParameter parameter);

        Object resolveArgument(MethodParameter parameter,
                                                ModelAndViewContainer ma
                                                NativeWebRequest webRequ
                                                WebDataBinderFactory bin

}

package org.springframework.web.method.support;

import org.springframework.core.MethodParameter;
import org.springframework.web.context.request.NativeWebRequest;

public interface HandlerMethodReturnValueHandler {

        boolean supportsReturnType(MethodParameter returnType);
```

```
        void handleReturnValue(Object returnValue,
                                                        MethodParameter returnTy
                                                        ModelAndViewContainer ma
                                                        NativeWebRequest webRequ

}
```

RequestResponseBodyMethodProcessor这个类，同时充当了方法参数解析和返回值处理两种角色。我们从它的源码中，可以找到上面两个接口的方法实现。

对HandlerMethodArgumentResolver接口的实现：

```
public boolean supportsParameter(MethodParameter parameter) {
        return parameter.hasParameterAnnotation(RequestBody.class);
}

public Object resolveArgument(MethodParameter parameter, ModelAndViewContai
                NativeWebRequest webRequest, WebDataBinderFactory binderFac
        Object argument = readWithMessageConverters(webRequest, parameter,

        String name = Conventions.getVariableNameForParameter(parameter);
        WebDataBinder binder = binderFactory.createBinder(webRequest, argum

        if (argument != null) {
                validate(binder, parameter);
        }

        mavContainer.addAttribute(BindingResult.MODEL_KEY_PREFIX + name, bi

        return argument;
}
```

对HandlerMethodReturnValueHandler接口的实现

```
public boolean supportsReturnType(MethodParameter returnType) {
        return returnType.getMethodAnnotation(ResponseBody.class) != null;
}

        public void handleReturnValue(Object returnValue, MethodParameter r
                ModelAndViewContainer mavContainer, NativeWebRequest webReq
                throws IOException, HttpMediaTypeNotAcceptableException {
```

```
        mavContainer.setRequestHandled(true);
        if (returnValue != null) {
                writeWithMessageConverters(returnValue, returnType, webRequ
        }
}
```

看完上面的代码，整个HttpMessageConverter消息转换的脉络已经非常清晰。因为两个接口的实现，分别是以是否有@RequestBody和@ResponseBody为条件，然后分别调用HttpMessageConverter来进行消息的读写。

如果你想问，怎么样跟踪到RequestResponseBodyMethodProcessor中，请你按照前面几篇博文的思路，然后到这里spring-mvc-showcase下载源码回来，对其中HttpMessageConverter相关的例子进行debug，只要你肯下功夫，相信你一定会有属于自己的收获的。

#思考 张小龙在谈微信的本质时候说：“微信只是个平台，消息在其中流转”。在我们对SpringMVC源码分析的过程中，我们可以从HttpMessageConverter机制中领悟到类似的道理。在SpringMVC的设计者眼中，一次请求报文和一次响应报文，分别被抽象为一个请求消息HttpInputMessage和一个响应消息HttpOutputMessage。

处理请求时，由合适的消息转换器将请求报文绑定为方法中的形参对象，在这里，同一个对象就有可能出现多种不同的消息形式，比如json和xml。同样，当响应请求时，方法的返回值也同样可能被返回为不同的消息形式，比如json和xml。

在SpringMVC中，针对不同的消息形式，我们有不同的HttpMessageConverter实现类来处理各种消息形式。但是，只要这些消息所蕴含的“有效信息”是一致的，那么各种不同的消息转换器，都会生成同样的转换结果。至于各种消息间解析细节的不同，就被屏蔽在不同的HttpMessageConverter实现类中了。

标签： SpringMVC