原 1.盘点springmvc的常用接口之HandlerMethodArgumentResolver

_Core 发表于 2年前 阅读 1359 收藏 2 点赞 1 评论 1

HOT

摘要: 盘点springmvc的常用接口之HandlerMethodArgumentResolver

1. 盘点springmvc的常用接口之HandlerMethodArgumentResolver###

前言

在初学springmvc框架时，我就一直有一个疑问，为什么controller方法上竟然可以放这么多的参数，而且都能得到想要的对象，比如HttpServletRequest或HttpServletResponse，各种注解`@RequestParam`、`@RequestHeader`、`@RequestBody`、`@PathVariable`、`@ModelAttribute`等。相信很多初学者都曾经感慨过。

这一章就是讲解处理这方面工作的

`org.springframework.web.method.support.HandlerMethodArgumentResolver`接口。

springmvc自带的一些实现：

- `ServletRequestMethodArgumentResolver`和`ServletResponseMethodArgumentResolver`处理了自动绑定HttpServletRequest和HttpServletResponse
- `RequestParamMapMethodArgumentResolver`处理了`@RequestParam`
- `RequestHeaderMapMethodArgumentResolver`处理了`@RequestHeader`
- `PathVariableMapMethodArgumentResolver`处理了`@PathVariable`
- `ModelAttributeMethodProcessor`处理了`@ModelAttribute`
- `RequestResponseBodyMethodProcessor`处理了`@RequestBody`
- ……

我们可以模仿springmvc的源码，实现一些我们自己的实现类，而方便我们的代码开发。

接口说明

```
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 {
        //用于判定是否需要处理该参数分解，返回true为需要，并会去调用下面的方法resolveArgument。
        boolean supportsParameter(MethodParameter parameter);
        //真正用于处理参数分解的方法，返回的Object就是controller方法上的形参对象。
        Object resolveArgument(MethodParameter parameter, ModelAndViewContainer mavContainer,
                        NativeWebRequest webRequest, WebDataBinderFactory binderFactory) throws Excep

}
```

示例1

1.盘点springmvc的常用接口之HandlerMethodArgumentResol…

_Core 发表于2年前

实现访问POST http://localhost:8080/demo1

post 数据

first_name = Bill

last_name = Gates

初学者一般喜欢类似下面的代码：

```
package com.demo.controller;

import javax.servlet.http.HttpServletRequest;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.demo.domain.Person;
import com.demo.mvc.annotation.MultiPerson;

import lombok.extern.slf4j.Slf4j;

@Slf4j
@Controller
@RequestMapping("demo1")
public class HandlerMethodArgumentResolverDemoController {

        @ResponseBody
        @RequestMapping(method = RequestMethod.POST)
        public String addPerson(HttpServletRequest request) {
                String firstName = request.getParameter("first_name");
                String lastName = request.getParameter("last_name");
                Person person = new Person(firstName, lastName);
                log.info(person.toString());
                return person.toString();
        }
}
```

这样的代码强依赖了javax.servlet-api的HttpServletRequest对象，并且把初始化Person对象这“活儿”加塞给了controller。代码显得累赘不优雅。在controller里我只想使用person而不想组装person，想要类似下面的代码：

```
@RequestMapping(method = RequestMethod.POST)
public String addPerson(Person person) {
  log.info(person.toString());
  return person.toString();
}
```

直接在形参列表中获得person。那么这该如实现呢?

我们需要定义如下的一个参数分解器：

```
package com.demo.mvc.component;

import org.springframework.core.MethodParameter;
import org.springframework.web.bind.support.WebDataBinderFactory;
import org.springframework.web.context.request.NativeWebRequest;
import org.springframework.web.method.support.HandlerMethodArgumentResolver;
import org.springframework.web.method.support.ModelAndViewContainer;

import com.demo.domain.Person;

public class PersonArgumentResolver implements HandlerMethodArgumentResolver {

        @Override
        public boolean supportsParameter(MethodParameter parameter) {
                return parameter.getParameterType().equals(Person.class);
        }

        @Override
        public Object resolveArgument(MethodParameter parameter, ModelAndViewContainer mavContainer,

                String firstName = webRequest.getParameter("first_name");
                String lastName = webRequest.getParameter("last_name");
                return new Person(firstName, lastName);
        }
```

在`supportsParameter`中判断是否需要启用分解功能，这里判断形参类型是否为Person类，也就是说当形参遇到Person类时始终会执行该分解流程`resolveArgument`。

在`resolveArgument`中处理person的初始化工作。

注册自定义分解器：

spring-boot方式

```
package com.demo;

import java.util.List;

import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.method.support.HandlerMethodArgumentResolver;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurationSupport;

import com.demo.mvc.component.MultiPersonArgumentResolver;
import com.demo.mvc.component.PersonArgumentResolver;

@SpringBootApplication
public class WebMvcConfiguration extends WebMvcConfigurationSupport {

        @Override
        protected void addArgumentResolvers(List<HandlerMethodArgumentResolver> argumentResolvers) {

                // 注册Person的参数分解器
                argumentResolvers.add(new PersonArgumentResolver());
        }
}
```

或者传统XML配置：

```
<mvc:annotation-driven>
        <mvc:argument-resolvers>
        <bean class="com.demo.mvc.component.PersonArgumentResolver"/>
        </mvc:argument-resolvers>
</mvc:annotation-driven>
```

或

```
<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">
        <property name="customArgumentResolvers">
        <bean class="com.demo.mvc.component.PersonArgumentResolver"/>
    </property>
</bean>
```

####示例2

加强版Person分解器，支持多个person对象。

比如 POST http://localhost:8080/demo1

post 数据

person1.first_name = Bill

person1.last_name = Gates

person2.first_name = Steve

person2.last_name = Jobs

用前缀区分属于哪个person对象。

定义一个注解用于设定前缀：

```
package com.demo.mvc.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
```

```
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
public @interface MultiPerson {

        public String value();
}
```

参数分解器：

```
package com.demo.mvc.component;

import org.springframework.core.MethodParameter;
import org.springframework.web.bind.support.WebDataBinderFactory;
import org.springframework.web.context.request.NativeWebRequest;
import org.springframework.web.method.support.HandlerMethodArgumentResolver;
import org.springframework.web.method.support.ModelAndViewContainer;

import com.demo.domain.Person;
import com.demo.mvc.annotation.MultiPerson;

public class MultiPersonArgumentResolver implements HandlerMethodArgumentResolver {

        @Override
        public boolean supportsParameter(MethodParameter parameter) {
                return parameter.hasParameterAnnotation(MultiPerson.class) && parameter.getParameter
        }

        @Override
        public Object resolveArgument(MethodParameter parameter, ModelAndViewContainer mavContainer,
                MultiPerson annotation = parameter.getParameterAnnotation(MultiPerson.class);
                String firstName = webRequest.getParameter(annotation.value() + ".first_name");
                String lastName = webRequest.getParameter(annotation.value() + ".last_name");
                return new Person(firstName, lastName);
        }

}
```

controller：

```
@ResponseBody
@RequestMapping(value = "multi", method = RequestMethod.POST)
public String addPerson(@MultiPerson("person1") Person person1, @MultiPerson("person2") Person perso
  log.info(person1.toString());
  log.info(person2.toString());
  return person1.toString() + "\n" + person2.toString();
}
```

友情链接：

盘点springmvc的常用接口目录

分类：SpringMVC 字数：917

标签： springmvc java

程序员 嘉兴

粉丝 22 | 博文 12 | 码字总数 14027

+ 关注

相关博客

springMVC的使用
BobwithB
17 0

java cms系统 springmvc mybatis
木穑
11 0

java cms系统 springmvc mybatis
迟欹
5 1

评论 (1)

Ctrl+Enter 发表评论

开源中国首席脑科主任
1楼 2016/07/18 16:55
牛逼!