链滴

浅析 SpringMVC 和 MyBatis 方法参数注入

作者：ccran

原文链接：https://ld246.com/article/1592649641503

来源网站：链滴

浅析SpringMVC和MyBatis方法参数注入

后端项目中，我们只需要关注Controller、Service、DAO三层，而其他层由于其通用性，框架已经我们做好了。

本篇博客浅析框架如何注入方法参数，主要内容如下：

- SpringMVC中Controller方法参数注入
- MyBatis中Mapper方法参数注入

一、实例

本篇博客以实现查找一个地区的同名用户接口为例，jdk版本1.8

浏览器发送HTTP GET请求，URL为**xxx/api/vi/nameSameArea?name=ccran&area=china**

1.1 Controller方法参数注入

Controller控制器代码可能如下所示：

```
@RestController
@RequestMapping("api/v1/")
public class MyController {
    @Autowired
    MyService myService;

    @GetMapping(path = "nameSameArea")
    public Response nameSameArea(String name,String area) {
        return Response.ok(myService.nameSameArea(name,area));
```

```
    }
}
```

1.2 Mapper方法参数注入

Service最终调用的Mapper代理，其接口代码可能如下所示：

```
public interface MyMapper{
    @Select("select count(*) from xxx where name=#{name} and area=#{area}")
    int countNameSameArea(@Param("name")String name,@Param("area")String area);
}
```

1.3 浅析

显然，参数的成功注入依赖于框架，而框架则依赖于反射完成方法参数的正确注入，我们只需要反射到方法的参数名称就行了。

● 对于Controller方法参数注入而言，SpringMVC会在HttpServletRequest中拿到对应参数名称作为ey的value即可正确注入方法参数。

```
    // 反射获取nameSameArea方法
    Method method = MyController.class.
            getDeclaredMethod("nameSameArea",
                    String.class,
                    String.class);
    // 获取参数名称构造入参
        Parameter[] parameters = method.getParameters();
        Object[] params = new Object[parameters.length];
        for (int i = 0; i < params.length; i++) {
            params[i] = request.getParameters(parameters[i].getName());
        }
    // 正确调用nameSameArea方法
        method.invoke(myController, params);
```

● 对于Mapper方法参数注入而言，MyBatis将入参封装成Map，并替换#{name},#{area}占位符为Mp中key为name以及area的值即可。

```
    // 由于创建的是代理对象，可以直接拿到method
    Map<String, Object> argsMap = new HashMap<>();
        Parameter[] parameters = method.getParameters();
        for (int i = 0; i < parameters.length; i++) {
            argsMap.put(parameters[i].getName(), args[i]);
        }
    // 根据Select注解的值以及参数Map生成sql
        String sql = generateSql(method.getAnnotation(Select.class).value(), argsMap);
    // 后续执行sql并获取结果封装成pojo返回
```

根据以上分析，可以通过**Parameter对象的getName方法**拿到参数名称。

但是，我们知道，Mapper接口中，不加入@Param注解，我们是无法正确注入参数的。而在Controllr中，即使不加@RequestParam注解，我们也能正确注入参数，这是为什么呢？是不是因为Controlle中的是非抽象方法，Mapper中的是抽象方法呢？

我们创建一个实例来看看怎么回事：

创建抽象类ReflectMethodParamDemo，其中method是非抽象方法，abstractMethod是抽象方，在main方法中打印两个方法的参数名称。

```
public abstract class ReflectMethodParamDemo {
    public abstract void abstractMethod(String name, String area);

    public void method(String name, String area) {
        System.out.println(name + ":" + area);
    }

    public static void main(String[] args){
        printParamName("abstractMethod",String.class,String.class);
        printParamName("method",String.class,String.class);
    }

    // 打印方法的参数名称
    public static void printParamName(String name, Class<?>... parameterTypes){
        Method method = null;
        try {
            method = ReflectMethodParamDemo.class.getDeclaredMethod(name,
                    parameterTypes);
        } catch (NoSuchMethodException e) {
            e.printStackTrace();
        }
        Parameter[] parameters = method.getParameters();
        for (Parameter parameter : parameters) {
            System.out.print(parameter.getName()+"\t");
        }
        System.out.println();
    }
}
```

输出如下：

```
arg0    arg1
arg0    arg1
```

可见，好像和方法是否抽象没什么关系，反射本质是去访问类的元信息，而类的元信息都在class文件面。

因此，我们用javap解析一下ReflectMethodParamDemo的字节码文件看看。

```
 public abstract void abstractMethod(java.lang.String, java.lang.String);
    descriptor: (Ljava/lang/String;Ljava/lang/String;)V
    flags: ACC_PUBLIC, ACC_ABSTRACT

  public void method(java.lang.String, java.lang.String);
    descriptor: (Ljava/lang/String;Ljava/lang/String;)V
    flags: ACC_PUBLIC
    Code:
      LineNumberTable:
      LocalVariableTable:
```

```
Start  Length  Slot  Name   Signature
    0      30     0  this   Lcom/ccran/jvm/ReflectMethodParamDemo;
    0      30     1  name   Ljava/lang/String;
    0      30     2  area   Ljava/lang/String;
```

我们可以看到，在非抽象方法的Code属性的子属性LocalVariableTable中有name、area的常量信。而在抽象方法中，因为它没有方法体，所以不会有Code属性，所以没有name、area的常量信息。

因此，我们大胆猜测，SpringMVC是通过局部变量表获取方法的参数名称。而在MyBatis中，因为apper文件中都是抽象方法，没有任何地方保存方法的参数名称，我们只能通过@Param注解来标志法的参数名称。

那Parameter对象的getName方法是如何获取方法的参数名称的呢。

查阅资料后，我们可以在用javac编译.java文件的时候加入 **-parameters** 参数，重新编译ReflectMetodParamDemo并执行

输出如下：

```
name    area
name    area
```

成功输出了方法的参数名称smile，通过java
解析class文件看看都发生了什么。

```
public abstract void abstractMethod(java.lang.String, java.lang.String);
  descriptor: (Ljava/lang/String;Ljava/lang/String;)V
  flags: ACC_PUBLIC, ACC_ABSTRACT
  MethodParameters:
    Name                           Flags
    name
    area
```

可以发现方法每个方法后面多了一个MethodParameters属性，看来Parameter对象的getName方就是访问MethodParameters属性中的信息，从而获取方法的参数名称，如果是空则给我们默认的arg,arg1这样的名称。

在IDEA中，可以进入File---Settings---Build,Execution,Deployment---Compiler---Java Compiler

在Additional command line parameters中加入 **-parameters** 参数

最后，我们做个测试，在Mybatis的Mapper文件中移除方法参数的@Param注解，加入 **-parameters** 编译参数，看看程序是否报错，所幸，一切正常。
miley

二、总结

1. 由于Controller中的方法是非抽象方法，SpringMVC可以通过局部变量表获取方法的参数名称；MBatis中Mapper接口的方法都是抽象方法没有方法体，所以没有Code属性，自然也没有局部变量表无法获取方法的参数名称，只能通过@Param来标志方法的参数名称。

2. 通过在javac编译时加入 **-parameters** 属性可以将方法参数名称这样的元信息加入到字节码文的**MethodParameters**属性中，从而保证Parameter对象的getName方法可以获取到方法的参数名。