前后端交互

1.BOMF.server.ajax

BOMF.server = {

/\*\*

\* ajax 操作 httpMethod 需要提交的方式，默认是get，如果是post请调用post方法 url 需要访问的地址，一般不传?后面的参数

\* params 需要提交的其他参数 callback 回调函数 有一个返回到前台的json数据类型的参数returndata

\*/

ajax: function (httpMethod, url, params, callback) {

if (httpMethod != "get" && params && typeof (params) == "object")

**//把js对象params转成JSON字符串params**

params = JSON.stringify(params);

$.ajax({

type: httpMethod,

url: BOMF.CONST.WEB\_APP\_NAME + url,

data: params,

async: true,

contentType: 'application/json',

dataType: 'json',

success: function (returnData) {

if (callback) {

callback(returnData);

} else {

return returnData;

}

}

});

},

…

如果一个service.js的函数是这样的：

XpFactoryTaskService.findUserTaskList=**function**(keyword,queryType,page,callback){

//该路径由SpringMVC框架的DispatcherServlet接受，由RequestMapping注解映射具体的url到具体的方法上，平台把DispatcherServlet隐藏了，因为平台能自动将服务写到其中，我们只要关注具体的业务逻辑怎么写（也就是serviceImpl）,而这里平台还提供了一个注解@MethodParameter用于参数数据传入具体业务方法，且其使

**var** url="/rest/factory/XpFactoryTaskService/findUserTaskList";

**var** data=**new** Object();

**var** httpMethod='get';

data.format='json';

data.keyword=keyword;

data.queryType=queryType;

data.page=page;

BOMF.server.ajax(httpMethod,url,data,callback);

}

那么这里传入的data是一个js对象，在BOMF.server.ajax中将其通过JSON.stringify转化成JSON字符串形式，然后下面又是jQuery的ajax方法，通过我们的url连接到数据源,data指定查询参数

**前端javascript调用示例**

这里json对象定义对照注解的参数

// 拼装主从表对象数据

        var jsonData = {

                       master : dataJson.master,

                       detail1 : dataJson.detail1,

                       detail2 : d3.hlx,

                       detail3 : d3.nbq,

                       detail4 : d3.zj,

                       detail5 : dataJson.detail2,

                       status : "update"

        };

    // js对象转换成字符串

        var jsonDataStr = JSON.stringify(jsonData);

// 直接调用平台自动生成的js方法**CscPmBaseService.saveMasterFiveDetails**

XpFactoryTaskService.findUserTaskList=**function**(keyword,queryType,page,callback){

//

**var** url="/rest/factory/XpFactoryTaskService/findUserTaskList";

**var** data=**new** Object();

**var** httpMethod='get';

data.format='json';

data.keyword=keyword;

data.queryType=queryType;

data.page=page;

**BOMF.server.ajax(httpMethod,url,data,callback);**

}

//该js与后台交互的方式是ajax，SpringMVC的**DispatcherServlet通过url找到自动写在其中的对应的service方法，**

更进一步地，我们知道**BOMF.server.ajax**内部大概是下面这样的

if (httpMethod != "get" && params && typeof (params) == "object")

params = JSON.stringify(params);

$.ajax({

type: httpMethod,

url: BOMF.CONST.WEB\_APP\_NAME + url,

data: params,

**把请求参数封装成JSON字符串**，为url添加项目名，然后$.ajax对data的处理是：**get请求中将附加在url后**。

**Post请求放在消息体中**

很合理。

于是，**找到对应service方法后，框架从http请求的流中获取这部分JSON数据传递给具体方法，而具体方法怎么使用这数据，就交给了MethodParameter来完成。**

**//这里function(data)是回调函数，用以额外操作，比如日志记录**

**CscPmBaseService.saveMasterFiveDetails**(jsonData,null,function(data){//以下部分是业务逻辑

               if (data.message == "ok") {

                       quickInfo("保存成功！", "success");

               } else {

                       quickInfo("保存失败！" + data.errorMsg, "fail");

               }

   }

);

对应的后端代码：

    @**Transactional**(rollbackFor = Exception.class)

        @**MethodParameter**(**desc** = "saveMasterFiveDetails", //desc描述

               input = "masterDetail5,params,user", //参数输入

               userParam = "user", //指定用户参数

               postType = { CscPmMasterDetails.class }, //

               httpMethod = "post",

               postName = "masterDetail5",

               queryString = "params")

        @Override

        public CscPmBase saveMasterFiveDetails(CscPmMasterDetails masterDetail5,

                       Map<String, Object> params, User user) throws Exception {

               CscPmBase cscPmBase = masterDetail5.getMaster();

               params = new HashMap<String, Object>();

               if (masterDetail5.getStatus().equals(BomfBeanConst.UPDATESTATUS)) {

                       // 更新主表

                      cscPmBase = super.saveUpdate(cscPmBase, params, user);

            // 更新从表

                       BeanUtilDao.saveMulti(masterDetail5.getDetail1(), params, user);

                       BeanUtilDao.saveMulti(masterDetail5.getDetail2(), params, user);

                       BeanUtilDao.saveMulti(masterDetail5.getDetail3(), params, user);

                       BeanUtilDao.saveMulti(masterDetail5.getDetail4(), params, user);

                       BeanUtilDao.saveMulti(masterDetail5.getDetail5(), params, user);

               }

               return cscPmBase;

        }

其中MethodParameter提供了注入的接口，前端的js通过service.js来调用，而serviceapi.xml是service.js的配置文件，所以我们把注解提供的接口在配置文件上注明，

那么，接口方法的前后端调用链就是：

service.java ---------🡪 service.js ----------🡪 view.js

| |

| MethodParameter |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_> serviceapi.xml

分析：

1）MethodParameter注解提供了注入的入口

2）service.js对应方法通过ajax异步地发出**http请求**，将参数传递，返回数据

3）view.js调用server.js的对应方法

4）记住一点，ajax发出http请求后只关心发回的http响应，至于http请求怎么处理（url怎么映射到处理器上）是服务器的事。因此，对于url的问题还是应该要到拦截url的地方去寻找。

下面我们就来合理猜测一下

（1）http请求关键部分就是**主体url**和**参数**，终究是封装成一个httprequest对象（这里是XMLHttpRequest），关键的还是谁来处理这个对象

（2）在简单的JSP&Servlet中，我们将url在web.xml配置，映射到servlet上，由容器解决这种映射关系

（3）而在SpringMVC框架中，通过一个servlet（DispatcherServlet）来拦截所有的请求，web.xml中的url-pattern配置为/或者/\*（连jsp也拦截）

然后再在控制器类中的bean调用中使用@RequestMapping注解来映射请求到具体的方法中

（4）然而具体到当前平台，

我们使用@MethodParameter来开放参数注入，

通过前端的js将请求url改装为json数据，

然后平台讲这些json数据注入，

处理后返回json数据给前端

再来一个具体的调用链：

客户端发送带参数的url-----🡪前端业务js为url添油加醋----🡪负责ajax的js 将url的参数打包成json，放在消息体或url中，然后发送XMLHttpRequeset-----🡪Web容器解析请求，通过url将其转发给DispatcherServlet----🡪DispatcherServlet根据url中指定的业务对象定位到那个位置-----🡪业务对象通过MethodParameter将http请求消息体中的json数据注入到业务方法中----🡪业务方法处理参数，通过MethodParameter返回JSON格式的数据----🡪DispatcherServlet输出http响应----🡪web容器转发响应到web服务器---🡪web服务器转发http响应给客户端

现在需要的就是：

1.@MethodParemeter的内部实现（势必包含JSON的转换处理）

2.@TargetMapping的内部实现（势必包含http请求参数的获取）

3.DispatcherServlet的内部实现

注解就是个简化版的DD，具体怎么解析还是靠外界来进行调用