平台架构以及代码分层结构

Version EIP7 Vue

广州宏天软件股份有限公司

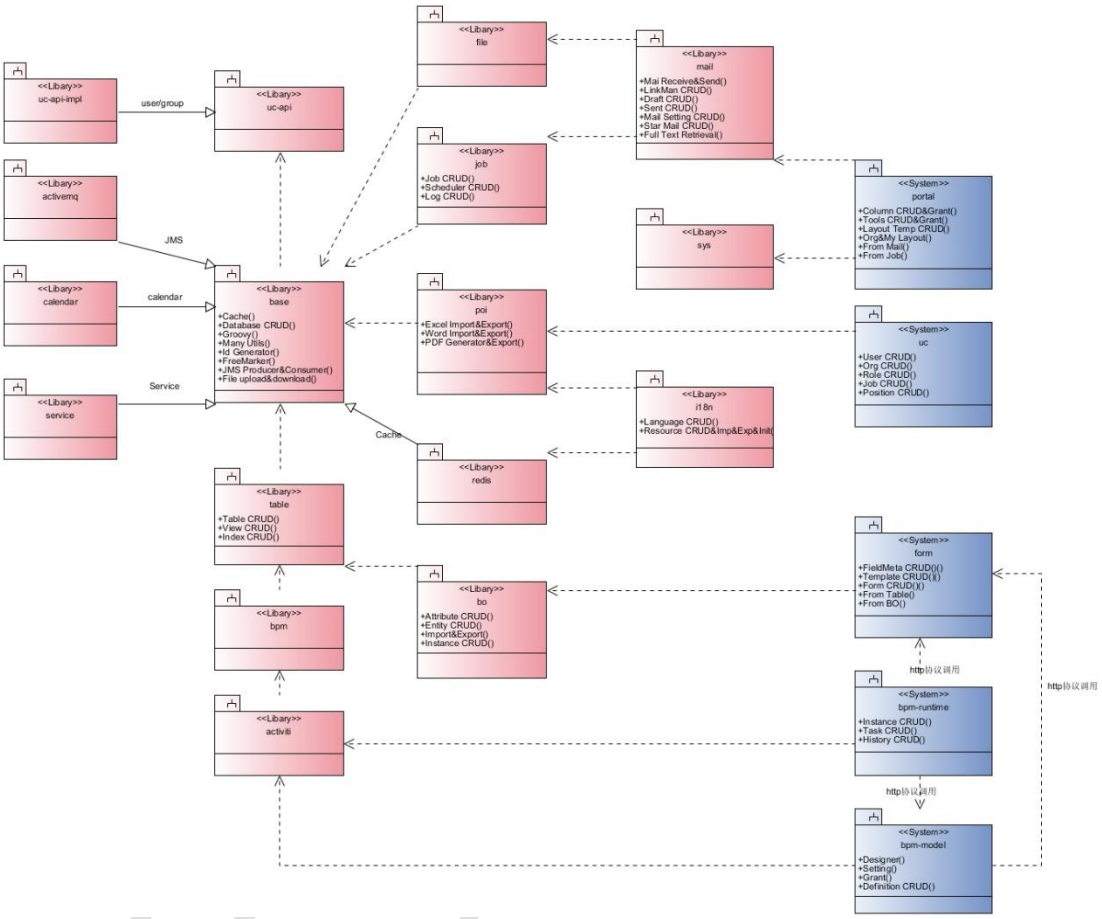
2020 年 03 月 20 日

版本管理

序号	版本	作者	日期	说明
1	V1.0	关建锋	2020 年 03 月 20 日	初始化版本

1 平台架构以及代码分层结构说明

1.1 模块划分



在上图中，红色模块为类库包，蓝色的为系统包。在下表中通过对比几个方面来描述两者之间的区别与联系。

	类库包（红色）	系统包（蓝色）
角色	不实现具体的业务功能，每个类库包只提供相对单一的通用功能，例如：缓存处理、附件处理、物理表操作等等。	提供具体的业务功能，比如设计一张表单、配置一个流程、添加一个用户等等。
标准	按照 Java interface 提供接	按照 http 标准提供接口服务

	类库包（红色）	系统包（蓝色）
	口或者直接提供实现类供其他包使用	
原则	1. 可以依赖另外一个类库包，不能依赖系统包； 2. 可以实现另外一个类库包中的接口； 3. 依赖关系只能单向，不能相互依赖，也不能循环依赖。	1. 只能依赖类库包，不能依赖另外的系统包； 2. 需要用到其他系统包的接口服务时，通过调用 http 服务实现。
建议	1. 每个包职责相对单一； 2. 一个通用功能有多种实现方式时，定义为接口，通过独立的包来实现接口。	有一些类库包的部分功能也需要发布为 http 接口服务，可以在类库包中实现功能（实现 rest controller），在系统包中装配（扫描对应的 controller）

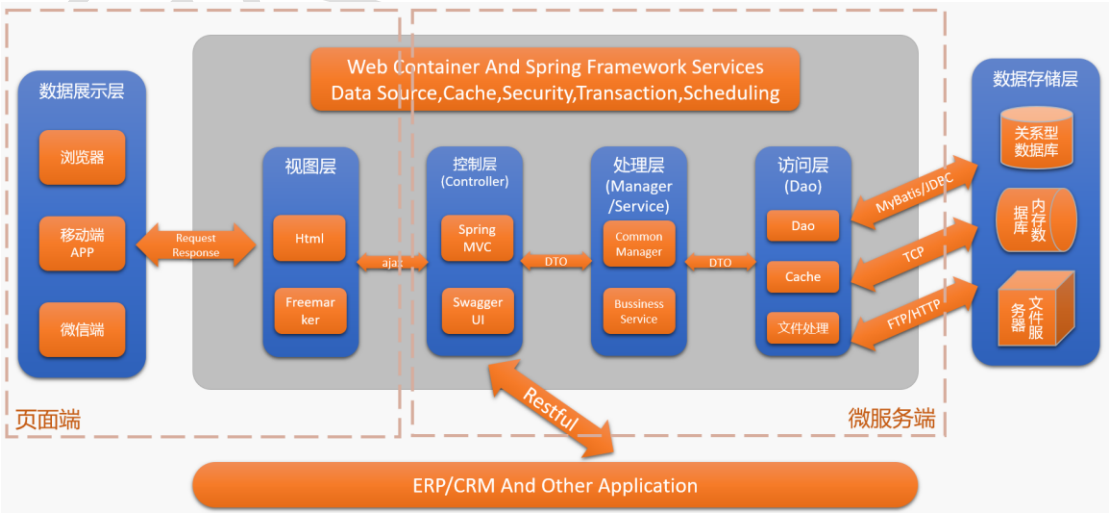
1.2 技术框架

整个平台采用微服务架构，使用 Spring Boot 2.X 作为基础框架，其他主要框架如下表所示：

框架	版本	说明
Spring Boot 系列	2.0.1.Release	微服务基础框架
Spring Cloud 系列	2.0.1.Release	微服务治理框架
Spring Web 系列	5.0.5.Release	Web 框架

框架	版本	说明
MyBatis	3.4.5	ORM 框架
PageHelper	5.1.3	分页框架
Druid	1.1.8	数据库连接池
Jackson	2.9.5	Json 框架
Groovy	2.1.6	动态脚本框架
Logback	1.2.3	日志框架
Freemarker	2.3.28	模板框架
Swagger	2.8.0	API 文档工具框架

1.3 分层结构



1.3.1 微服务端

微服务端采用了标准的 Spring MVC 三层结构，由 dao manager controller 标准的三层构成，另外在类库包中对外提供的接口封装了一层 service 方便其他类库包或系统包引用。

1.3.1.1 实体层

如下所示，实体类均继承自 BaseModel，并指定泛型主键类型，实体类必须使用 swagger 的注解@ApiModel 说明该实体类的作用，所有的属性都用注解@ApiModelProperty 进行描述，对于必填的属性需要在注解中注明 required=true，对于有可选值得属性可以指定 allowableValues 进行值范围约束。

```java
/**
 * bo定义
 *
 * @company 广州宏天软件股份有限公司
 * @author heyifan
 * @email heyf@jee-soft.cn
 * @date 2018年4月12日
 */
@ApiModel(description="bo定义")
public class BoDef extends BaseModel<String> {
    private static final long serialVersionUID = 1L;

    @ApiModelProperty(name="id", notes="主键")
    protected String id="";

    @ApiModelProperty(name="alias", notes="别名", required=true)
    protected String alias="";

    @ApiModelProperty(name="description", notes="描述")
    protected String description="";

    @ApiModelProperty(name="boEnt", notes="业务实体")
    protected BoEnt boEnt = new BoEnt();

    @ApiModelProperty(name="supportDb", notes="是否支持数据库(0:不支持  1:支持)", allowableValues="0,1")
    protected Short supportDb = 0;

    @ApiModelProperty(name="categoryId", notes="所属分类ID", required=true)
    protected String categoryId = "";

    @ApiModelProperty(name="status", notes="状态(normal:正常 forbidden:禁用)", allowableValues="normal,forbidden")
    protected String status = "normal";

    @ApiModelProperty(name="deployed", notes="是否发布(0:未发布 1:已发布)", allowableValues="0,1")
    protected Short deployed = 0;
}
```

1.3.1.2 dao 层

mapper 文件的命名方式为 实体类名 Mapper.xml 的格式，统一放在每个包的 src/main/resources/mapper 目录下面。mapper 文件中 namespace 指向 dao 类，dao 的命名方式为实体类名 Dao.java，如下图所示:

dao 文件为一个接口，不是一个实现类，这个接口继承自 MyBatisDao，在 MyBatisDao 中提供了基础的增删改查方法。dao 有两个泛型需要指定，第一个泛型是主键的类型，第二个泛型是对应的实体类。

```java
/**
 * 流程定义处理接口
 *
 * @company 广州宏天软件股份有限公司
 * @author heyifan
 * @email heyf@jee-soft.cn
 * @date Apr 28, 2018
 */
public interface BoDefDao extends MyBatisDao<String, BoDef> {
    /**
     * 根据别名获取定义
     * @param alias     别名
     * @return          返回bo定义
     */
    BoDef getByAlias(String alias);

    /**
     * 通过别名删除定义
     * @param alias     别名
     * @return          返回删除了几个定义
     */
    int removeByAlias(String alias);
}
```

1.3.1.3 manager 层

manager 分为接口和实现类，manager 的命名方式为实体类名 Manager.java,

例如 BoDefManager 接口和 BoDefManagerImpl 实现类，其中接口扩展自

Manager，实现类继承自 AbstractManagerImpl 类并实现 BoDefManager 接口。在接口和实现类中均需指定两个泛型，第一个为主键的数据类型，第二个为对应的实体类。

```java
/**
 * 业务实体定义属性 处理接口
 *
 * @company 广州宏天软件股份有限公司
 * @author heyifan
 * @email heyf@jee-soft.cn
 * @date 2018年4月12日
 */
public interface BoAttributeManager extends Manager<String, BoAttribute> {

    /**
     * 根据实体ID获取BO属性
     * @param entId     实体ID
     * @return          返回bo属性列表
     */
    List<BoAttribute> getByEntId(String entId);

    /**
     * 根据实体ID删除属性。
     * @param entId     实体ID
     */
    void removeByEntId(String entId);
}
```

1.3.1.4 controller 层

controller 层的命名方式为实体类 Controller.java，继承自 BaseController，类上有三个注解：

- @RestController 表明其是一个 restful 控制类
- @RequestMapping("/bo/def/v1/") 标明了其映射的路径
- @Api(tags="BoDefController") 用以生成 swagger 在线接口文档

在每一个方法上需要添加两个注解:

- @RequestMapping(value="list", method=RequestMethod.POST, produces={"application/json; charset=utf-8" }) 该方法对应的地址，该方法对应的 restful 请求类型及支持的数据格式
- @ApiOperation(value = "获取 bo 定义列表（带分页信息）", httpMethod = "POST", notes = "获取 bo 定义列表" 用于生成 swagger 文档

列表方法使用 QueryFilter 作为入参，该参数封装了通用查询和分页信息；返回值为 PageList<?>，该参数的泛型为对应的实体类，在该返回值中包含了分页信息和列表数据。
方法可以直接 throws 异常信息，异常消息的处理和记录会统一在 BaseController 中进行。

```java
/**
 * bo定义控制器
 *
 * @company 广州宏天软件股份有限公司
 * @author heyifan
 * @email heyf@jee-soft.cn
 * @date 2018年4月18日
 */
@RestController
@RequestMapping("/bo/def/v1/")
@Api(tags="BoDefController")
public class BoDefController extends BaseController {
    @Resource
    BoDefManager boDefManager;
    @Resource
    BoEntRelManager boEntRelManager;
    @Resource
    BoEntManager boEntManager;
    @Resource
    BoDataHandler boDataHandler;

    @RequestMapping(value="list", method=RequestMethod.POST, produces={"application/json; charset=utf-8" })
    @ApiOperation(value = "获取bo定义列表（带分页信息）", httpMethod = "POST", notes = "获取bo定义列表")
    public PageList<BoDef> listJson(@ApiParam(name="queryFilter",value="通用查询对象")@RequestBody QueryFilter queryFilter) throws Exception {
        return boDefManager.query(queryFilter);
    }

    @RequestMapping(value="detail",method=RequestMethod.GET, produces = { "application/json; charset=utf-8" })
    @ApiOperation(value = "获取bo定义详情", httpMethod = "GET", notes = "获取bo定义详情")
    public Object detail(@ApiParam(name="alias",value="bo定义别名", required = true) @RequestParam String alias) throws Exception{
        return boDefManager.getPureByAlias(alias);
    }

    @RequestMapping(value="save",method=RequestMethod.POST, produces = { "application/json; charset=utf-8" })
    @ApiOperation(value = "添加bo定义", httpMethod = "POST", notes = "添加bo定义")
    public Object save(@ApiParam(name="json",value="bo定义的json数据", required = true) @RequestBody String json) throws Exception{
        boDefManager.save(json);
        return modelMap("添加成功");
    }

    @RequestMapping(value="remove",method=RequestMethod.DELETE, produces = { "application/json; charset=utf-8" })
    @ApiOperation(value = "删除bo定义", httpMethod = "DELETE", notes = "删除bo定义")
    public Object remove(@ApiParam(name="alias",value="bo定义别名", required = true) @RequestParam String alias) throws Exception{
        int removeItems = boDefManager.removeByAlias(alias);
        Map<String, Object> map = modelMap("删除成功");
        map.put("count", removeItems);
        return map;
    }
}
```

1.3.1.5 特别说明

1. QueryFilter 和 PageBean 等辅助查询对象只到 manager 层，不允许在 dao 层中使用；
2. PageBean 和 Page 都是分页辅助类，前者用作分页查询使用，后者为返回分页信息使用，构建分页查询条件时使用 PageBean，不允许使用 Page;

3. 实体类需要转为 json 数据，json 数据也需要实例化为实体类，整个项目使用 jackson 作为 json 处理框架，不允许再引入 gson 或 fastjson;
4. 实体类与 json 的转换过程中，有属性不参与转换时在该属性的 get 方法上添加@JsonIgnore 注解;

```
@JsonIgnore
public BoEnt getBoEnt() {
 return boEnt;
}
```

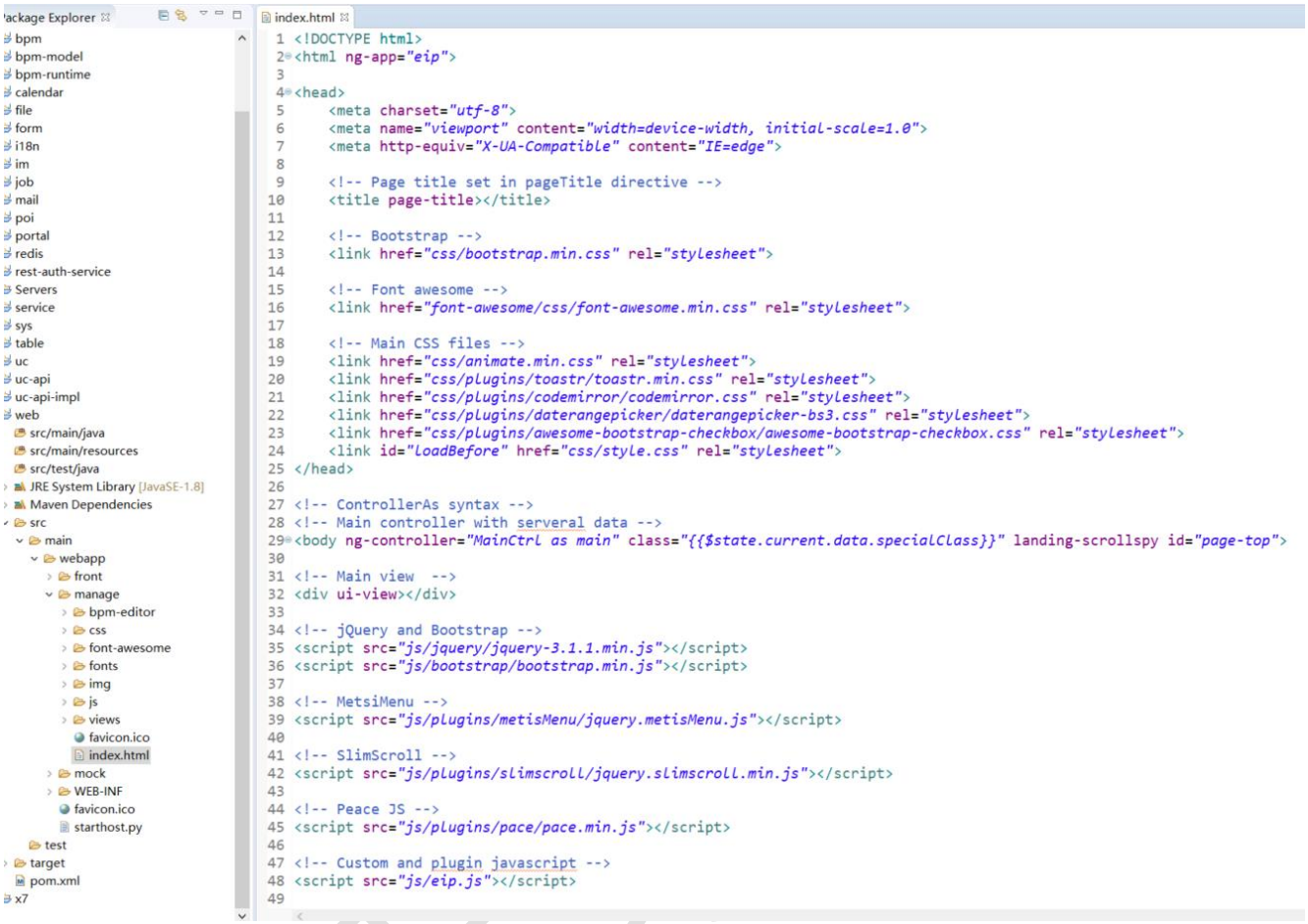
5. 实体类与 json 转换中，有的属性需要另外命名为其他名称时使用 @JsonProperty 注解。

```
@JsonProperty("attrs")
public List<BoAttribute> getBoAttrList() {
 return boAttrList;
}
```

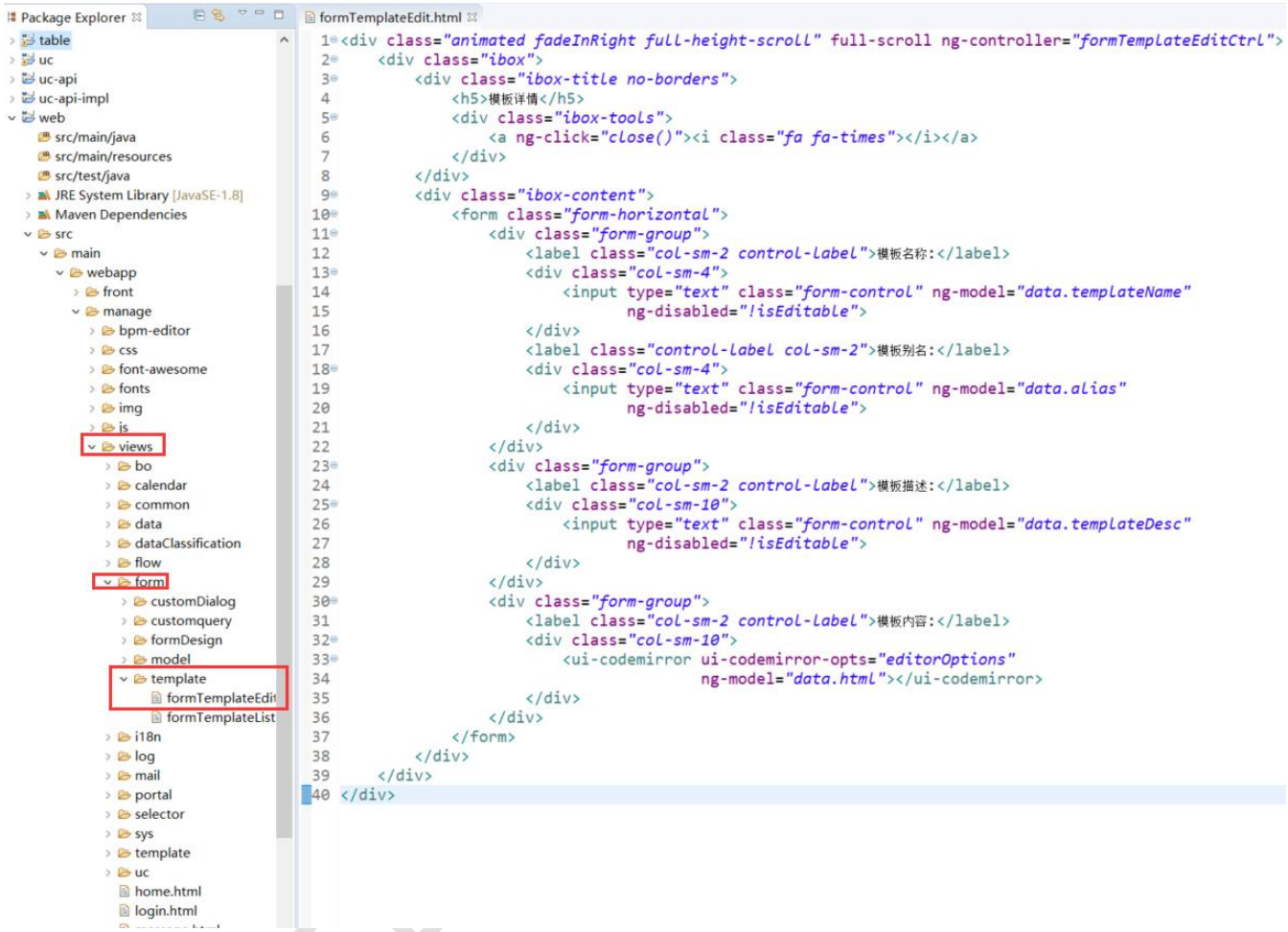
1.3.2 页面端

前端页面采用单页面应用的结构，样式框架采用 Bootstrap3 + Font-awesome 图标库。JavaScript 框架采用了 AngularJs 1.5 + jQuery 3.1.1。

整个系统只有一个 index.html 是完整的页面，通用的 css 文件和 js 文件在该页面引入，如下图所示：



其他的功能页面只有 html 片段，通过 ui-router 实现前端路由。功能页面的 html 代码如下图所示。



在 ui-router 中注册路由时，可以指定页面需要加载的 js、css 资源，这里指定的资源采用懒加载的方式，只有访问到该页面的时候才会加载这些资源。

```
.state('form.templateEdit', {
    url: "/templateEdit",
    templateUrl: "views/form/template/formTemplateEdit.html",
    data: { pageTitle: '表单模板编辑', specialClass: 'fixed-sidebar' },
    resolve: {
        loadPlugin: function ($ocLazyLoad) {
            return $ocLazyLoad.load([
                {
                    name: 'form',
                    files: ['js/form/formControllers.js']
                },
                {
                    name: 'ui.codemirror',
                    files: ['js/plugins/ui-codemirror/ui-codemirror.min.js']
                }
            ]);
        }
    }
})
```