|  |  |  |  |
| --- | --- | --- | --- |
| **深圳国泰安教育技术有限公司** | **版本** | **密级** | **页数** |
| **V1.5** | **机密** | **共24页** |
|  | | |

**国泰安智慧教务管理平台V1.5\_架构设计说明书**

|  |  |  |  |
| --- | --- | --- | --- |
| **作者** | 张银 | **编制日期** | 2018-4-12 |
| **审核** |  | **审核日期** |  |
| **批准** |  | **批准日期** |  |



深圳国泰安教育技术有限公司

版权所有 侵权必究

**文档修改记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **版号** | **修改日期** | **修改内容** | **修改人** |
| V1.0 | 2017-10-09 | 初版 | 涂荣 |
| V1.5 | 2018-4-12 | 修改 | 张银 |
|  |  |  |  |

目 录

[1. 文档简介 4](#_Toc497404495)

[1.1. 文档目的 4](#_Toc497404496)

[1.2. 预期读者 4](#_Toc497404497)

[1.3. 文档范围 4](#_Toc497404498)

[1.4. 定义、缩写词和缩略语 4](#_Toc497404499)

[1.5. 开发平台 4](#_Toc497404500)

[2. 架构设计目标 5](#_Toc497404501)

[2.1. 关键功能 5](#_Toc497404502)

[2.2. 关键质量属性 5](#_Toc497404503)

[3. 架构设计原则 5](#_Toc497404504)

[3.1. 架构设计对后续工作的限制 5](#_Toc497404505)

[3.2. 架构设计原则 6](#_Toc497404506)

[4. 逻辑架构视图 6](#_Toc497404507)

[4.1. 职责划分与职责确定 6](#_Toc497404508)

[5. 开发架构视图 7](#_Toc497404509)

[5.1. 系统划分 7](#_Toc497404510)

[5.1.1. 工程继承关系 7](#_Toc497404511)

[5.1.2. 工程依赖关系 7](#_Toc497404512)

[5.2. 子系统结构 8](#_Toc497404513)

[5.2.1. 子系统目录结构指导 10](#_Toc497404514)

[6. 物理架构视图 10](#_Toc497404515)

[6.1. 物理拓扑 10](#_Toc497404516)

[6.2. 优化部署 11](#_Toc497404517)

[7. 公共设计 11](#_Toc497404518)

[7.1. 编码规范 11](#_Toc497404519)

[7.2. 命名规范 11](#_Toc497404520)

[7.2.1. Jar包 12](#_Toc497404521)

[7.2.2. 接口、类、javascript 12](#_Toc497404522)

[7.2.3. Package 12](#_Toc497404523)

[7.2.4. 页面命名 13](#_Toc497404524)

[7.2.5. 方法命名 13](#_Toc497404525)

[7.3. 数据库设计 15](#_Toc497404526)

[7.3.1. 整体设计原则 15](#_Toc497404527)

[7.3.2. 数据库规范 15](#_Toc497404528)

[7.3.3. 表设计规范 15](#_Toc497404529)

[7.3.4. 通用表字段参考 16](#_Toc497404530)

[7.4. 事务使用 16](#_Toc497404531)

[7.5. 异常处理 16](#_Toc497404532)

[7.6. 权限 17](#_Toc497404533)

[7.7. 配置项 17](#_Toc497404534)

[7.8. 持续集成 17](#_Toc497404535)

[7.9. 数据验证 18](#_Toc497404536)

[7.10. 数据字典 19](#_Toc497404537)

[1. 设计思路 19](#_Toc497404538)

[7.10.1. 开发步骤 19](#_Toc497404539)

[7.10.2. 数据字典与枚举的区别 19](#_Toc497404540)

[8. 关键质量属性的设计原理 19](#_Toc497404541)

[8.1. 学生选课 19](#_Toc497404542)

[8.2. 自动排课算法 20](#_Toc497404543)

# 文档简介

当前制定的统一开发框架更加适合于学校信息化项目，根据学校信息化系统的特点做了取舍，如果要适合于云技术，需要做一定的改变

## 文档目的

本文档主要从整个系统的概要层面，综合考虑技术和需求的相关问题，对系统做出整体的设计， 对系统采用的技术和方案做出选择，给出关键部件的详细设计。

## 预期读者

本文档面向的读者主要有：产品经理、项目经理、开发人员、测试人员、运维人员以及其他关系人。

## 文档范围

* 开发平台：采用的技术平台、工具包、版本约定等
* 命名规范等
* 分层设计规范，每层的设计、层与层的交互设计等
* 项目开发架构设计

## 定义、缩写词和缩略语

|  |  |
| --- | --- |
| **术语** | **定义和说明** |
| Em | 易管理 |
| Cu | 高校 |
| Edu | 教务 |
| Component | 公共组件 |
| UDF | 统一开发框架，Universal Develop Framework |

## 开发平台

|  |  |  |
| --- | --- | --- |
| 分类 | 版本 | 备注 |
| 数据库 | Oracle11gR2 |  |
| java | jdk1.7 |  |
| 应用服务器 | tomcat8 |  |
| 构建工具 | maven3 |  |
| 开发工具 | Eclipse Mars.2 Release (4.5.2) |  |
| Spring | Spring4.2.4RELEASE | 可小版本升级 |
| MVC | SpringMVC4.2.4RELEASE | 可小版本升级 |
| ORM | hibernate5.0.7.Final | 可小版本升级 |
| Javascript | jquery1.9.1 |  |
| 分页 | jqueryPagination1.1 |  |
| 树 | jquery.ztree3.5 |  |
| jqueryTreeTable |  |
| 弹出框 | artDialog |  |
| 模板渲染 | jquery.tmpl |  |
| 模块化 | Seajs2.2.0 |  |
| 富文本 | editor1.2.2 |  |
| 日期控件 | My97DatePicker |  |
| 上传插件 | webuploader |  |
| 图表插件 | echart |  |
| 前端校验 | jqueryValidate |  |

# 架构设计目标

以配合产品目标，本系统指定目标如下：

1. 提高产品的稳定性、可靠性及可扩展性
2. 支持多数据库
3. 支持可分布式部署

## 关键功能

1. 学生选课
2. 自动排课

## 关键质量属性

本系统关键质量属性有：

1. 学生选课功能满足500并发量
2. 自动排课算法

# 架构设计原则

## 架构设计对后续工作的限制

本系统采用的是jar包依赖的方式进行多子系统集成部署，当然也可以拆开进行单独部署，根据具体业务需求进行业务模块组合，考虑到性能需求时可将相应的模块独立集群部署。

使用缓存、队列、多线程时应征求架构团队的意见才能应用，避免乱用、不符合规范的使用影响系统稳定性和可扩展性。

任何复杂的业务逻辑必须有详细设计，如自动排课等。

## 架构设计原则

本系统采用4层架构，分别为视图层、服务层、业务层、数据访问层。视图层由纯前端技术通过rpc调用完成。服务层为向外提供服务接口，规定请求类型、请求参数及响应内容。业务层处理核心业务逻辑。数据访问层为业务层提供数据获取、保存、修改操作。

# 逻辑架构视图

## 职责划分与职责确定

系统采用前后端分离方式，视图层（Web）采用纯前端技术能单独部署在nginx、apache、tomcat中。后端采用springmvc + spring + hibernate框架，通过maven依赖根据具体的业务需求集成不同的业务功能部署在应用服务器中。数据库采用Oracle,但要考虑数据库切换。



系统采用四层架构，划分为四个分层，其中各层相关说明如下：

* 数据存储：底层的数据存储采用数据库和缓存技术实现,目前未配置缓存，如有需要可加入。
* 数据访问层（Data Access Object）：负责数据的存储和访问。如果为了提供数据访问的性能而使用缓存（Cache），对缓存的访问也需通过数据访问层进行。
* 业务逻辑层（Business）：负责系统业务逻辑的实现，供Service层调用。事务控制在该层进行。
* 服务层：使用Restful Service技术向其他系统暴露服务，通常仅包含JSON与VM或DTO的装换代码，业务逻辑直接调用业务逻辑层对象来实现。
* 前端展示层：使用纯前端的技术实现具体的业务呈现。
* PO (Persistant Object): 持久对象,可以看成是与数据库中的表相映射的java对象。最简单的PO就是对应数据库中某个表中的一条记录，多个记录可以用PO的集合。PO中应该不包含任何对数据库的操作，PO对象不用做业务数据交换，只能用于Business与DAO之间数据交换。
* DTO(Data Transfer Object):数据传输对象，controller中与Business的业务数据交换使用dto对象来操作，不可直接使用PO对象。DTO对象也可直接应用于DAO用于封装查询结果（绝不能用于持久化），并返回到Business及Service层，当DTO不能满足前端页面需求时，需要新建 VO。
* VO(View Object):视图对象，封装视图层需要的业务对象，用于controller与前端页面之间的数据交换。

# 开发架构视图

## 系统划分

应用构建框架（gtafe-udf）,工具包（gtafe-component），系统工程（cu-edu-master）用于集成各子系统的jar包用于部署和通用配置。

### 工程继承关系

下图说明了所有子工程程必须继承gtafe-em，em中对所用到的jar包做了统一的约束，所有子工程的pom包的parent节点继承tafe-em。



### 工程依赖关系

下图说明了所有子工程的依赖关系，其中cu-edu-master 是web工程，此工程根据自己的业务需求添加所需的业务jar包。所有的子工程（子系统）添加公共组件的依赖。



## 子系统结构

现系统包含后台管理、基础数据2个基础系统.

共9个子系统：

选课管理(gtafe-cu-edu-choicecourse）

排课管理(gtafe-cu-edu-courseplan)

考务管理(gtafe-cu-edu-examplan)

毕业管理(gtafe-cu-edu-graduation)

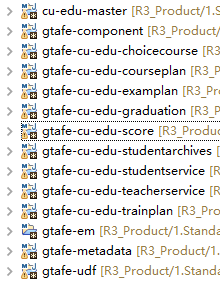
学籍管理(gtafe-cu-edu-studentarchives)

成绩管理(gtafe-cu-edu-score)

学生服务(gtafe-cu-edu-studentservice)

老师服务(gtafe-cu-edu-teacherservice)

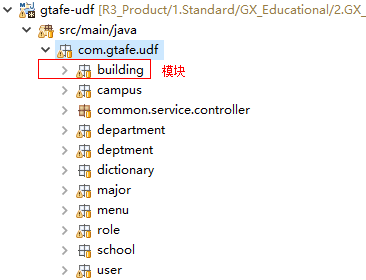
培养方案(gtafe-cu-edu-trainplan)





视图层使用ajax调用服务层提供的接口。

### 子系统目录结构指导



# 物理架构视图

## 物理拓扑

数据库服务应该单独部署，应用服务器现考虑部署在同一台服务器并且其中包含web应用。



当单机不能满足性能需求情况向下，可进行集群部署。

* 所有的应用系统依然共享同一个数据库，当一台数据库服务器依然不能满足可考虑数据库集群。
* 应用系统可根据系统访问频次进行拆分并根据性能需要进行部分系统的集群部署。
* Web服务器一般只需要一台即可，部署时建议使用nginx。
* 当需要负载时，软件负载可采用nginx，硬负载可采用F5，采用cdn加速和dns负载等，根据具体的业务场景采取别要的负载方式。



## 优化部署

基于产品需求、网络情况和成本等因素的权衡考虑，系统上线部署时需要两台服务器。

1. 将数据库部署到单独服务器。
2. 另一台服务器部署应用系统、web程序、缓存服务及文件服务。
3. Web程序建议使用 nginx部署并对静态文件做缓存。
4. 两台服务器需部署在同一局域网。

# 公共设计

## 编码规范

严格遵循公司Java及前端编码规范。

## 命名规范

约定：为叙述方便，我们以em代表易管理产品，cu代表高校，edu代表教务，course代表排课管理系统，plan代表教学计划功能为例说明。**命名规范的整体原则是命名能准确地表达功能意义，杜绝使用简写（除非是世界公认的简写）及词不达意的命名。涉及到Id、Name、Code的命名需要准确地将意义表达出来，比如命名为department则有问题（到底是单位id，单位代码还是单位名称？），命名为departmentId，departmentName则非常清晰明确。**

### Jar包

Jar包命名规范：使用对应的Java Project中包含的最顶层的Package名，省略掉域名前缀，使用“-”替换对应的“.”。

我们规定一个子系统就是一个Java Project，发布时生成一个独立的jar包，供业务系统集成部署。

应用支撑框架是一个jar包，作为一个业务系统的基础项目，其他的子系统与应用支撑框架相同的方式供业务系统通过maven依赖进行集成部署。业务系统通过自由组合依赖子系统的jar包的方式满足不同业务场景的需求。

以应用支撑框架UDF为例举例：

* Java Project名字：gtafe-udf
* 顶层Package名字：com.gtafe.udf
* Jar包名字：gtafe-udf.jar

以排课系统为例：

* Java Project名字：gtafe-em-cu-edu-course
* 顶层Package名字：com.gtafe.em.cu.edu.course
* Jar包名字：gtafe-em-cu-edu-course.jar

### 接口、类、javascript

接口以I作为前缀，对应的实现类以Impl作为后缀。再和具体的功能名称进行组合（例子：IPlanService/PlanServiceImpl）,所有接口名、类名使用全称。Javascript文件命名除配置相关文件其他文件以小写英文字母为前缀。

* 接口命名规范：IXxxDao（I功能名称+分层名称），
* 对应的实现类命名规范：XxxDaoImpl（功能名称+分层名称+Impl）。

|  |  |  |
| --- | --- | --- |
| **接口/类名** | **例子** | **定义和说明** |
| IXxxDao | IPlanDao | 数据访问类的接口 |
| XxxDaoImpl | PlanDaoImpl | 数据访问类 |
| IXxxBusiness | IPlanBusiness | 业务逻辑类的接口 |
| XxxBusinessImpl | PlanBusinessImpl | 业务逻辑类 |
| XxxDto | PlanDto | 业务数据传输类 |
| XxxController | PlanController | 控制器类 |
| Xxx | Plan | 持久化对象，对应分成中的po层 |
| XxxVo | PlanVo | 视图模型类 |
| xxx.js | plan.js | js文件 |

### Package

Package命名规范：域名后缀.公司域名.业务线.业务方向.系统名.子系统名.模块名.分层名.实现名。

* 域名后缀、公司域名：必须的，固定为：com.gtafe。
* 业务线：除公共组件（component）与应用支撑框架(udf)外，其他必须声明，如易管理是em。
* 业务方向：必须的，比如，高校产品就是cu。
* 系统名：必须的，比如，教务就是edu。
* 子系统名：必须的，比如，排课管理就是course。
* 模块名：必须的，比如，排课管理中的教学计划就是plan。
* 分层名：可选的，如果有分层就是必须的。对于一些工具包没有分层，就不需要。分层名固定为四个：dao、service、business、dto
* 子分层名：dao与business层的业务实现类统一命名为当前层下的impl，dao层下建立po，Service层下建立vm。

我们约定系统的分层名是固定的，不能随意修改删除增加

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **域名** | **业务线** | **业务方向** | **系统名** | **子系统名** | **模块名** | **分层名** | **定义说明** |
| com.gtafe | em | cu | edu | course | plan | service | 服务提供层（View Object类在该层新增vo包） |
|  |  |  |  |  |  | business | 业务接口及实现（实现类新增包名为impl） |
|  |  |  |  |  |  | dao | 数据操作接口及实现（实现类新增包名为impl，持久化对象在该层下建立po包） |
|  |  |  |  |  |  | dto | 业务数据传输对象层 |

### 页面命名

|  |  |  |  |
| --- | --- | --- | --- |
| **类型** | **命名** | **例子** | **说明** |
| 主页面 | xxx\xxx\index | course\plan\index | 主页面，可能是一个框架页，通常用不到 |
| 列表 | xxx\ xxx\list | course\plan\list | 列表页面 |
| 编辑 | xxx\ xxx\edit | course\plan\edit | 修改、新增页面，通常这两个页面可以使用一个页面来实现，如果分开，修改页面使用这个名字 |
| 新增 | xxx\ xxx\add | course\plan\add | 新增页面 |
| 查看 | xxx\ xxx\view | course\plan\view | 浏览查看页面 |

页面命名规范：子系统名称\功能名称\后缀。子系统名称、功能名称作为一个目录，相同功能的一组页面放在同一个目录下面。

### 方法命名

方法命名必须使用动词，为某个动作。

1. service层的controller中的方法

|  |  |  |
| --- | --- | --- |
| **方法名** | **定义** | **和说明** |
| getAll | ResultResponse getAll () | 查询所有 |
| getItem | ResultResponse getItem(int id) | 获取一个实体对象 |
| getList | ResultResponse getList(XxxQueryVo queryVo) | 获取多个实体对象 |
| getPagedList | ResultResponse getPagedList(XxxQueryVo queryVo, int pageIndex, int pageSize) | 分页获取多个实体对象 |
| add | ResultResponse add(XxxVo xxx) | 新增 |
| update | ResultResponse update(XxxVo xxx) | 修改 |
| delete | ResultResponse delete(String[] ids) 或ResultResponse delete(String id) | 删除/单个删除（只在不支持多个删除的情况下存在） |
| importFile | ResultResponse importFile() | 导入 |
| exportFile | void exportFile() | 导出 |

1. business类中的方法

|  |  |  |
| --- | --- | --- |
| **方法名** | **定义** | **和说明** |
| getAll | List<XxxDto> getAll () | 查询所有 |
| getItem | XxxDto getItem(int id) | 获取一个实体对象 |
| getList | List<XxxDto> getList(XxxQueryDto queryDto) | 获取多个实体对象 |
| getPagedList | Page<XxxDto> getPagedList(XxxQueryDto queryDto, int pageIndex, int pageSize) | 分页获取多个实体对象 |
| add | void add(XxxDto xxx) | 新增 |
| update | void update(XxxDto xxx) | 修改 |
| delete | void delete(String[] ids) 或void delete(String id) | 删除/单个删除（只在不支持多个删除的情况下存在） |
| importFile | void importFile(FileInputStream fileInputStream) | 导入 |
| exportFile | FileOutputStream exportFile(XxxDto xxx) | 导出 |

1. dao类的方法

|  |  |  |
| --- | --- | --- |
| **方法名** | **定义** | **和说明** |
| getAll | List<Xxx> getAll () | 查询所有 |
| getItem | Xxx getItem(int id) | 获取一个实体对象 |
| getList | List<Xxx> getList(XxxQueryDto queryDto) | 获取多个实体对象 |
| getPagedList | Page <Xxx> getPagedList(XxxQueryDto queryDto, int pageIndex, int pageSize) | 分页获取多个实体对象 |
| add | void add(Xxx xxx) | 新增 |
| update | void update(Xxx xxx) | 修改 |
| delete | void delete(String[] ids) 或void delete(String id) | 删除/单个删除（只在不支持多个删除的情况下存在） |

## 数据库设计

### 整体设计原则

1. 跨数据库设计：尽量使用跨数据库标准进行数据库设计，为将来跨数据库提供更多便利性，设计必须满足该设计原则。
2. 范式与数据冗余：尽可能满足范式标准及第三范式，但是为了提高数据库性能可以适当使用数据冗余
3. 命名标准化：使用统一的表、存储过程、函数等数据库对象的命名规则，并提供详细注释
4. 设计工具：使用Powser Desginer进行设计

### 数据库规范

1. 编码使用utf-8（默认设置）
2. 数据库命名使用“项目名称简写\_版本号\_用途”，如EDU\_BETA1.0\_DEV，用途包括开发版DEV，测试版TEST，发布版RTM

### 表设计规范

1. 命名使用全大写
2. 数据库对象使用“一对象类型前缀字符+系统或模块缩写字符\_非复数对象名”
3. 表类型前缀：T，如TSYS\_USER, TSYS\_ROLE\_USER，其中SYS为某子应用系统缩写，如果是多个应用系统共享同一数据库，建议使用应用系统缩写进行分隔，如果是一个应用系统，建议使用模块名称缩写进行分隔
4. 存储过程前缀：P
5. 视图前缀：V
6. 函数前缀：F
7. 表字段命名
8. 除世界公认的缩写，表设计命名不使用缩写。
9. 除主键外，字段命名不加表前缀，当字段名为关键字时添加表前缀。
10. 表关联数据字典时，关联字段加特殊后缀\_CODE并且使用数据字典编号。
11. 主键使用32位uuid而不是自增长，交由程序控制主键生成（方便数据初始化、导入导出、迁移等操作），根据表明在加“ID”来命名(USER\_ID)，当做外键关联时能快速辨析。
12. 字符型命名规则：根据字段长度需要固定采用50的倍数长度。
13. 所有业务主表加上创建时间、创建人、更新时间、更新人字段。
14. 所有表避免使用复合主键。
15. 外键
16. 表设计中不设计表的关联关系，有程序控制。
17. 外键命名统一使用对应表的主键名，方便识别外键表及统一数据权限过滤。

### 通用表字段参考

以下是通用表字段设计规范及其说明。

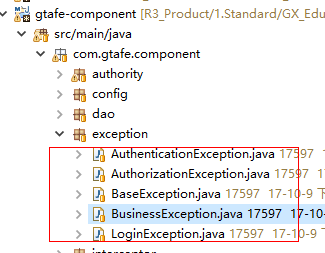
|  |  |  |
| --- | --- | --- |
| **字段名** | **类型** | **描述** |
| XXXX\_ID | NVARCHAR2 (50) | 表主键字段 |
| XXXX\_NAME | NVARCHAR2 (50) | 表中的关键字，加表明 |
| XXXXXX\_ID | NVARCHAR2 (50) | 外键表中的主键 |
| XXXXX\_CODE | NVARCHAR2 (50) | 关联数据字典编码的外键字段 |
| IS\_DELETED | NUMBER | 是否删除 |
| IS\_ENABLED | NUMBER | 是否可用 |
| CREATE\_USER\_ID | NVARCHAR2 (50) | 业务主表键入创建人ID |
| CREATE\_TIME | TIMESTAMP | 业务主标键入创建时间，精确到秒 |
| UPDATE\_USER\_ID | NVARCHAR2 (50) | 业务主表键入修改人ID |
| UPDATE\_TIME | TIMESTAMP | 业务主标键入修改时间，精确到秒 |

## 事务使用

事务统一使用注解方式添加，各模块根据具体的需求在Business层添加@Transactional注解，并慎重考虑事务传播机制等细节问题。

## 异常处理

框架中已包含统一的异常处理类，在代码中统一将异常往外抛，框架将会拦截异常进行统一的格式化输出。



## 权限设计

### 系统授权原则

1. 系统菜单、菜单按钮的显示和访问通过角色进行授权。
2. 业务数据的显示通过用户进行授权。
3. 超级管理员创建的角色为共享角色，子系统管理员创建的角色为私有角色，共享角色可以被所有子系统管理员进行授权操作，私有角色只能被创建者进行授权操作。
4. 超级管理员和子系统管理员的菜单权限初始化到权限表中，管理员只能把自己拥有的权限授权给其他用户。

### 系统内置角色

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **角色名称** | **角色类型** | **系统内置** | **默认菜单权限** | **说明** |
| 超级管理员 | 总管理员角色 | 是 | 全部菜单权限。 | 角色列表中不显示该角色。 |
| 教务系统管理员 | 子系统管理员角色 | 是 | 教务系统全部菜单权限 | 超级管理员的角色列表中显示子系统管理员，可以对子系统管理员进行授权。子系统管理员的角色列表中只显示自身和超级管理员创建的业务角色（不显示管理员角色）。 |
| 教职工 | 业务角色 | 是 | 无 | 教职工账号默认加入该角色 |
| 学生 | 业务角色 | 是 | 无 | 学生账号默认加入该角色 |

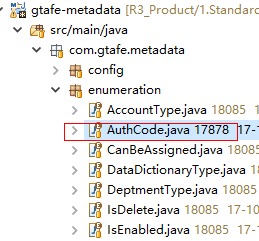
### 系统内置用户

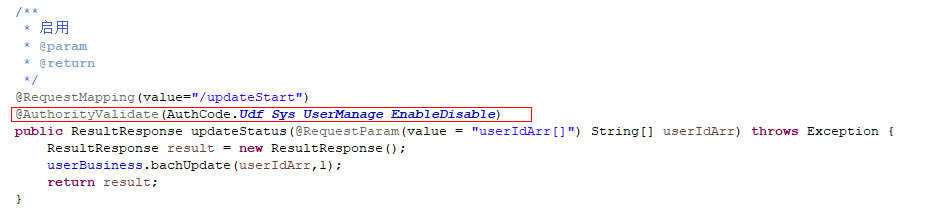
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **姓名** | **用户账号** | **所属角色** | **系统内置** | **默认数据权限** | **权限说明** |
| 超级管理员 | super | 超级管理员 | 是 | 全部数据权限。 | 用户列表中不允许被再次授数据权限。 |
| 教务系统管理员 | jwmaster | 教务系统管理员 | 是 | 全部数据权限 | 用户列表中不允许被再次授数据权限。 |

### 共享对象

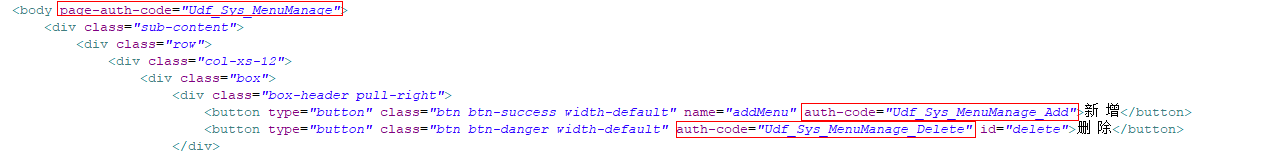
1、权限实现和校验

系统中Web接口、HTML静态页面、菜单的使用权限通过权限码实现，编码前应该为当前系统定义所有的权限码，controller的方法中为当前的方法配置对应的权限码。





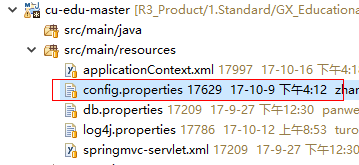
前端对应的功能上需要加上权限码，浏览器加载后框架根据按钮配置的权限码与当前登录的用户进行鉴权，没有权限的按钮将被移除。示例如下：



### 数据权限

本系统中存在数据权限控制，在支撑平台中针对每个用户可以设置对应的院系、校区、开课单位的数据权限。在各自的业务代码中获取session中用户所对应的数据权限内容，在实现具体的业务逻辑时加入对应的数据权限进行过滤（超级管理员和子系统管理员拥有所有的数据权限）。数据权限帮助类为：DataAuthorityUtils。

## 配置项



系统配置信息统一存放到master工程的config.properties中。

## 持续集成

本系统使用持续集成部署到开发测试环境进行准出测试及功能验证。

持续集成地址为：<http://192.168.21.204:8888/jenkins>

打开页面有没有账号的同事可以先注册账号，登录后可看到如下界面：



根据项目开发的工程会建立对应的任务，全局进行打包时可直接点击gtafe-em行最后构建按钮（红色框出来的图标）。也可以根据开发改动的工程进行单独打包，最后jenkins会将后台程序打包到服务器。前端程序需要单独打包，点击edu任务的构建按钮即可。

构建后访问地址为：<http://192.168.21.204:81/edu/index.html>。

## 数据验证

从架构上来说，数据验证包括前端及后端验证，从验证内容上来说，数据验证包括输入数据格式的验证及业务逻辑验证。

1. 前端验证

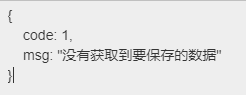
使用jQuery validation插件进行输入数据格式验证，如必填、某种固定格式的验证。对于单行文本框的输入长度限制，务必使用单行文本框的maxlength属性进行限制，不需要使用jQuery validation。

1. 后端验证

所有business中需要对接口传入的参数进行验证，确保数据有效性。校验失败的参数统一使用BusinessException异常向外抛出，框架将根据抛出的内容封装成统一的格式。**杜绝在前端及Controller中进行业务逻辑验证。**



校验失败后框架封装的数据格式如下：



## 数据字典

### 设计思路

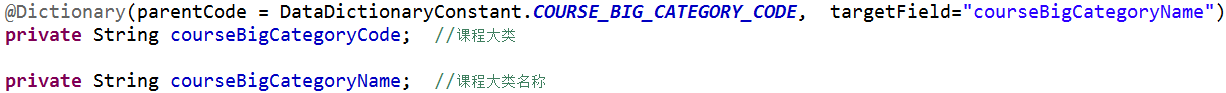
数据字典用于存储国家标准及各种需要业务扩展而一般又不需要去改动程序的配置数据。数据字典采用树形结构，其数据约束规则如下：

1. 数据字典存储对象为数据字典（如性别）及其下数据字典项（如男、女）
2. 数据字典Id使用程序自动生成的UUID
3. 数据字典Code全局唯一，数据字典项Code在父数据字典下唯一
4. 系统数据字典不允许修改

数据字典使用注解配置的方式自动根据注解的属性值设置数据字典名称。

### 开发步骤

1. 确定数据字典属性及命名：
2. 国标字典初始化时，字典Code参考国家标准，字典和字典项为：系统级别。
3. 非国标字典初始化时，字典Code命名规则为：“GTA\_” + “大写以\_分割的英文名称”，字典和字典项为：用户级别。
4. 在DataDictionaryConstant中定义数据字典编码，编码常量的命名规则为大写字母以下划线分割，并以CODE结尾，如课程大类则命名为COURSE\_BIG\_CATEGORY\_CODE
5. 根据数据字典项Code获取数据字典项Name：根据数据字典项Code以及所属数据字典Code获取数据字典项Name。在需要根据数据字典项Code获取数据字典项Name的DTO对象中配置注解



### 数据字典与枚举的区别

什么时候该使用数据字典，什么时候该使用枚举？从程序设计及业务角度按如下规则来考虑：

1. 如果数据字典Code或Name需要在程序中硬编码并且一般不会变动，则使用枚举，如是否删除（0：否、1：是）、审核状态（0：待审核、1：已审核通过、2：审核不通过）。
2. 如果数据字典Code及Name在业务上需要扩展（比如以后可能需要新增数据字典项）并且数据字典Code不需要在程序中硬编码，则使用数据字典，如环节类别、课程大类。

特殊情况处理：

1. 部分数据字典需要在程序中硬编码怎么办？对需要使用硬编码的数据字典项Code建立枚举即可，特别注意不需要硬编码的数据字典项不应该出现在枚举值中。
2. 前端程序该怎么处理枚举及数据字典？
3. 枚举：前端程序建立与后端对应的枚举，与后端约定规则即可。
4. 数据字典：前端通过后端提供的数据接口使用枚举，不需要自行处理。

## 引用

本项目中数据库未设置表关联关系，业务表之间的关联关系通过程序维护，因此会出现主表数据删除，但业务表中对主表的数据是强依赖关系。

**@StateField**  是否删除的字段，标记了该注解的，将会根据枚举类进行判断是否删除的属性，获取值时会通过是否删除来获取未删除的字段，当实体类未标识删除字段，则表示数据表中的字段是物理删除，将不进行过滤。

**@QuoteField** 默认获取ID作为被引用字段，但是有当前QuoteField注解，则按照配置QuoteField注解的字段

**@Quote** 当一个持久化对象引用了另一个持久化对象时，使用此注解当前项目中主要用到做删除时判断当前数据有没有被使用

**@DeletedField** 状态字段，是否被引用时将设置到对应的字段中。

**QuoteExistUtils** 是否被引用帮助类，程序通过帮助类中的方法进行判断当前对象或集合中的数据是否要有被引用

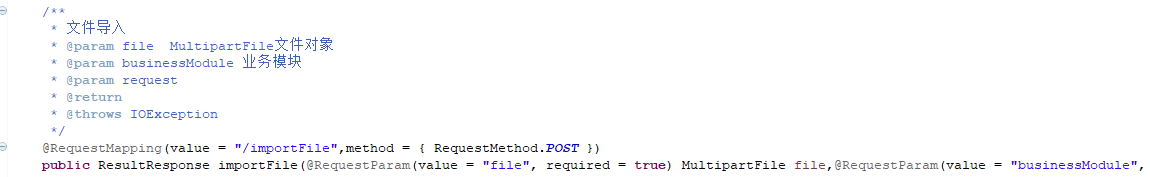
**isExist(List<T> list, Class<T> clazz)**向集合中设置是否被引用的状态，其中clazz对象必须标记StateField注解，并且字段类型为Integer/int，只有当被引用的对象才会设置StateField字段为Disable,使用枚举IsEnabled

**isExist(T obj, Class<T> clazz)**判断obj对象中应用的字段类型是否被其他数据存储对象使用

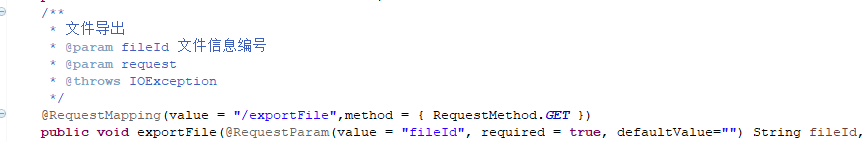
## 文件处理

本系统提供文件统一处理服务（gtafe-filesystem），所有业务系统文件存储全使用当前服务进行文件的存储与读取。系统中文件导入与导出操作不需要对文件进行存储，因此各自业务系统使用poi的帮助类完成。文件服务提供一下方法：

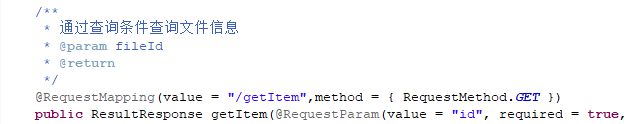
单文件导入：



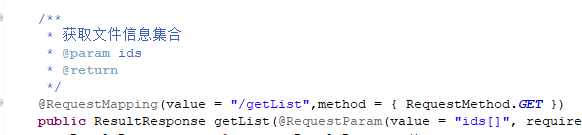
文件导出：



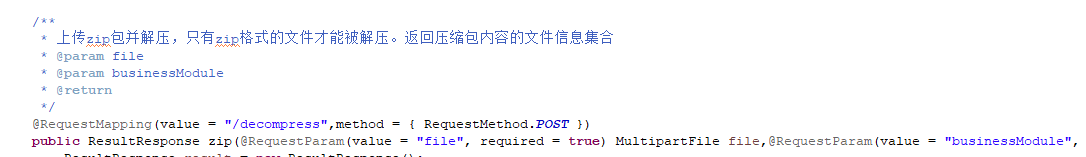
查询文件信息



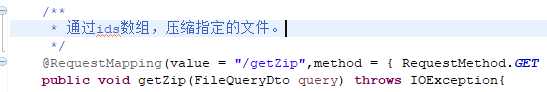
获取文件信息集合



上传压缩包并解压：

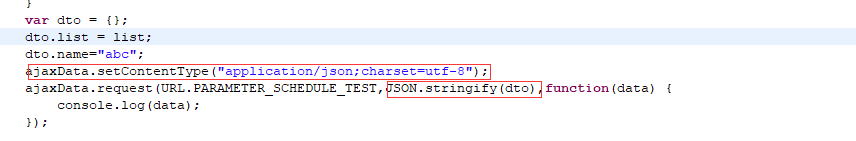


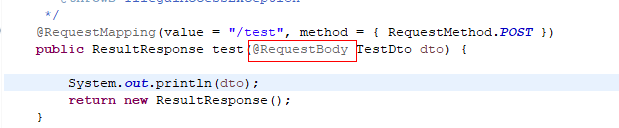
下载压缩包：



## 前端复合对象传参

前端向后台提交参数往往是一个复杂的对象，其中会包含集合等多层结构。现总结以下方式可正确的获取前端传入的值。（红色框中是与其他请求有区别的地方，请特别关注）





## 前端左右列表添加与移除

系统中部分需求是通过左右列表进行选择操作，现提供公共处理方法，如下图所示，公共方法负责创建左右列表及左右列表的交互。参考功能模块：排课系统->实践任务->环节老师设置中的选择老师页面。

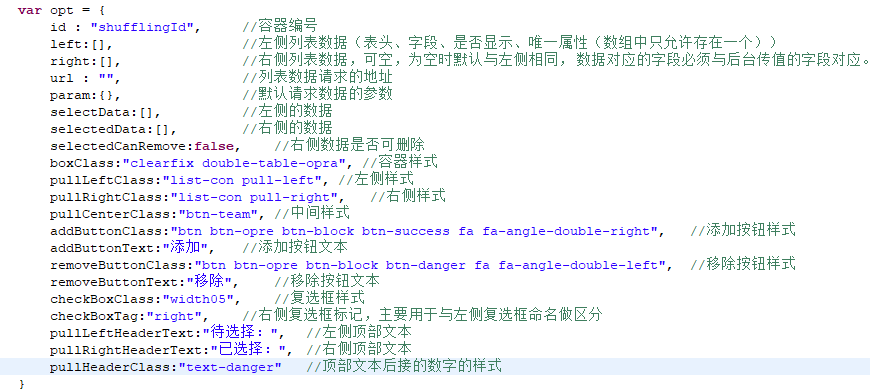


**使用方法如下：**

页面中引用shuffling.js，并通过对象初始化将相关参数传入。



**参数描述：**



**获取右侧的值：**

通过shuffling.getData()方法获取已选择的数据集合

# 关键质量属性的设计原理

## 学生选课

**目标：**满足性能需求500并发。

**措施：**将选课子系统拆分成3个子系统分别独立部署在3个服务器，通过nignx负载均衡配置选课系统的请求由这3个系统来处理。加入缓存、队列，学生选课的数据操作分批次操作数据库，减少数据库压力。

## 自动排课算法

**目标：**提高自动排课执行时间，降低排课失败频次。

**措施：**

1. 执行排课算法前将排课需要的数据预先加载到内存中。
2. 根据需求的梳理在排课前预先设定好排课条件。
3. 采用模块化课插拔方式校验排课规则。
4. 多线程处理排课逻辑。