**通知——系统原理设计**

2019-10-8

1. **系统框架设计**
2. **业务逻辑设计**

**①创建通知及查看已发通知模块图**



**②创建通知模块图**



**③收通知模块图**



**2.功能模块结构**



1. **模块之间关系**

**2.1系统内部模块之间关系**

**2.1.1系统内部模块之间关系介绍**

1. 新建通知时可从模板库导入模板至通知正文编辑框。
2. 新建通知时可将正在编辑的通知暂存到草稿箱中。
3. 新建通知模块在发送通知之后会将通知保存到已发通知库中，用户可以对已发通知进行管理。
4. 新建通知模块在发送通知之后向对应用户发送消息提醒。
5. 在管理已发通知中可以查看到已发通知的阅读详情统计。
6. 用户查收通知之后会产生已读记录，阅读详情模块从通知已读记录的数据中统计得出详情统计。
7. 根据阅读详情中的记录形成的未读名单，可以向这些未读的用户发送消息提醒。

**2.1系统外部模块之间关系**



**2.1.1系统外部模块之间关系介绍**

1. 通知模块中，在创建通知时选择对象界面构建发送者身份时需要由身份认证模块提供当前用户的身份信息。
2. 在创建通知的选择发送对象中，构建学科信息时由学科管理模块提供当前平台所需的学科信息。
3. 在创建通知的选择发送对象中，构建年级和行政班信息时由行政班管理模块提供发送对象的年级和行政班信息。
4. 在创建通知的选择发送对象中，构建教学班信息时由教学班管理模块提供发送对象的教学班信息。

5、通知发送之后拆分解析发送对象，调用用户档案管理接口，由其提供发送对象对应的用户信息。

6、通知模块在发送通知之后和发送阅读提醒都会调用消息中心的接口生成消息提醒。

# 系统重点模块设计及技术难点

**1、系统重点模块设计**

**1.1发送对象设计**

**(1)各身份对应的发布对象**

|  |  |
| --- | --- |
| **身份** | **发送对象** |
| **超级管理员** | →领导（→全部）  →普通管理员（→全部）  →老师（→全部  →某学科→某年级  →某年级→某学科）  →学生（→全部  →某年级）  →家长（→全部  →某年级） |
| **普通管理员** | →领导（→全部）  →老师（→全部  →某学科→某年级  →某年级→某学科）  →学生（→全部  →某年级）  →家长（→全部  →某年级） |
| **领导** | →教研组长（→全部）  →老师/任课老师（→全部  →某学科→某年级  →某年级→某学科）  →班主任（→全部  →某年级）  →学生（→全部  →某年级）  →家长（→全部  →某年级） |
| **教研组长** | →任课老师（所属学科）（→该学科全部老师  →某年级）  →学生（所属学科）（→该学科全部学生  →某年级）  →家长（所属学科）（→该学科全部学生对应的家长  →某年级） |
| **班主任** | →学生（→所带的全部行政班  →某行政班）  →家长（→所带的全部行政班学生对应的家长  →某行政班） |
| **任课老师** | →学生（→所教的全部教学班  →某教学班）  →家长（→所带的全部教学班学生对应的家长  →某教学班） |

表Ⅰ各身份对应的发布对象

|  |  |
| --- | --- |
| **发送对象ID** | **发送对象名称** |
| A0 | 超级管理员 |
| A1 | 普通管理员 |
| L1 | 领导 |
| T0 | 教研组长 |
| T1 | 教师/任课教师 |
| T2 | 班主任 |
| S0 | 学生 |
| P0 | 家长 |

表Ⅱ发送对象类型

|  |  |  |  |
| --- | --- | --- | --- |
| **发送对象类型**  **（第一层节点）** | **树层次类型** | **节点类型编号**  **(ObjType)** | **节点ID**  **(ObjID)** |
| **普通管理员** | **/** | A10 | **“AllGenAdmin”** |
| **领导** | **/** | L10 | **“AllLeader”** |
| **教研组长** | **/** | T00 | **“AllReseacher”** |
| **教师/任课教师** | **/** | T10 | **“AllTeacher”** |
| **学科** | T11 | **/** |
| **年级** | T12 | **/** |
| **学科下的年级** | T112 | **/** |
| **年级下的学科** | T121 | **/** |
| **/**  **（注：教研组长所管的全部学科）** | T13 | **“ReTeacher“** |
| **班主任** | **/** | T20 | **“ChargeTeacher”** |
| **年级** | T22 | **/** |
| **学生** | **/** | S00 | **“Student”** |
| **年级** | S01 | **/** |
| **/**  **（注：班主任所带的全部行政班）** | S02 | **“AdStudent”** |
| **/**  **（注：教师所带的全部教学班）** | S03 | **“TeStudent”** |
| **行政班** | S04 | **/** |
| **教学班** | S05 | **/** |
| **/**  **（注：教研组长所管的全部学科）** | S10 | **“ReStudent“** |
| **教研组长所管的学科** | S11 | **/** |
| **教研组长所管学科下的年级** | S112 | **/** |
| **家长** | **/** | P00 | **“Parent”** |
| **年级** | P01 | **/** |
| **/**  **（注：班主任所带的全部行政班）** | P02 | **“AdParent”** |
| **/**  **（注：教师所带的全部教学班）** | P03 | **“TeParent”** |
| **行政班** | P04 | **/** |
| **教学班** | P05 | **/** |
| **/**  **（注：教研组长所管的全部学科）** | P10 | **“ReParent“** |
| **教研组长所管的学科** | P11 | **/** |
| **教研组长所管学科下的年级** | P112 | **/** |

表Ⅲ选择对象树设计

**1.2构建节点数据（应后端需要的数据）**

创建通知的选择发布对象设计是先由通知模块识别当前登录用户的所有身份，根据用户所具有的身份来确定发布权限，再根据发布权限确定所需要的数据，调用学科管理、教学班管理或行政班管理的接口获取学科、年级、行政班、教学班的信息（包括id、名称），将获取的数据和身份数据等在后端拼接成约定好的格式，组成发布对象所需的数据，并发送给前端。

前端对获取到的数据进行解析，将其中各个层级的数据以树状节点的形式构建选择对象树。在每个节点都带有对应的数据，包括对象ID、对象类型、对象的名称、节点唯一ID、子节点数等（详情见选择对象相关类型说明），每个数据的作用如下表：

|  |  |
| --- | --- |
| 与节点对应的数据 | 说明 |
| ObjID | 相当于节点的ID，用来构建发布对象ID，提交给后台来拆解并在发布对象范围中以ID中的条件查询到对应的用户。 |
| ObjType | 每个节点所属的类型，用来构建发布对象的类型，发布对象的类型用来给后台快速判断发布对象的范围。 |
| ObjName | 节点的名称，用来显示在选择对象界面和构建发布对象的名称 |
| ObjUniqID | 节点在选择对象树中的路径，用来快速定位节点的地址，由ObjID和ObjUniqID组合而成，格式为ObjType+@+ObjUniqID |
| ObjParentUniqID | 父节点的ObjUniqID |
| ObjChildNum | 某节点的子节点个数 |



图Ⅰ选择对象(老师)树的JSON格式图Ⅱ选择对象(学生)树的JSON格式

**1.3节点选择**

在构建好选择对象之后，用户对节点（复选框）的选择情况会影响到其父节点以及子节点的选择状态。

具体流程：

①节点选中时向上影响父节点

首先在对某个节点勾选之后，获取所有与该节点的兄弟节点，判断它们的节点（包括该节点）是否勾选：

a:若判断得出所有都处于全选状态，则将父节点置为全选状态；

b:若判断得出存在一个处于未选或半选状态，则将父节点置为半选状态；

依照上述规则，依次向上判断，直到顶层节点。

②节点选中时向下影响子节点

当用户点击勾选某个节点之时，根据该节点的id递归判断是否有子节点存在，若存在，将子孙节点全部置为勾选状态。

③节点取消选中时向上影响父节点

若是取消勾选该节点，获取所有与该节点的兄弟节点，判断兄弟节点的选择状态：

a:若判断得出所有都处于未选状态，则将父节点置为未选状态；

b:若判断得出存在一个处于全选状态或半选状态，则将父节点置为半选状态

依照上述规则，依次向上判断，直到顶层节点。

④节点取消选中时向下影响子节点

当取消勾选某个节点之时，根据该复选框的id向下判断是否有有子节点存在，若存在，将子孙节点全部置为未勾选状态。

**1.4保存选择对象**

**1.4.1、构建结构化字符串**

构建结构化字符串：应后台数据处理需要，前端需构建四个结构化字符串：ObjTypeList、ObjIDList、ObjNameList、ObjUniqIDList。

**Ⅰ、构建ObjTypeList**：在每一个节点上都有对应的ObjType，将处于全选状态的第一层所有节点上的ObjType作为发送对象的ObjType。

存在情况：

①只能全部发送的发送对象（如领导、教研组长），将发送对象节点的ObjType作为发送对象的ObjType。

②对于发送对象中分类只有一种（如领导发给学生，学生只有以年级分类等）的情况，这种情况将处于全选状态的第一层所有节点上的ObjType（包括发送对象节点）作为该发送对象（如学生）的ObjType（该层有多少个节点处于全选状态，拼接多少个节点的ObjType，每个节点ObjType之间用“；”隔开，共同组成该发送对象的ObjType）。

③对于选择对象分类存在多重的（如领导发给任课老师，任课老师有学科和年级分类），当发送对象节点处于全选状态时，将发送对象节点的ObjType作为发送对象的ObjType。

当发送对象节点处于半选状态时以多少个一级分类节点处于勾选（半选和全选）状态，就拼接多少个处于全选状态的第一层（非发送对象节点）节点上的ObjType，每个节点ObjType之间用“；”隔开。

依照上述规则，拼接所选的每个发送对象的ObjType，之间使用$隔开组成ObjTypeList。

**Ⅱ、构建ObjIDList**

1. 只能全部发送的发送对象构建ObjID为：**发送对象ObjID**。

②选择对象分类只有一种（如领导发给学生，学生只有年级分类），当发送对象节点处于全选状态下，构建结构化字符串的格式为：**发送对象ObjID(节点ObjID+节点ObjID+节点ObjID+拼接所有节点的ObjID))，**以逗号分隔。

当发送对象节点处于半选状态下，构建结构化字符串的格式为：**发送对象ObjID（如“学生”的ObjID）&节点ObjID（如：年级ObjID）**，该节点为处于全选状态的第一层子节点的ObjID，同样有多少个子节点处于全选状态，拼接多少个节点的**发送对象ObjID（如“学生”的ObjID）&节点ObjID（如：年级ObjID）**，之间用“;”隔开。

1. 对于选择对象分类存在多重的（如领导发给任课老师，任课老师有学科和年级分类），当发送对象节点处于全选状态时，构建ObjID为：**发送对象ObjID(一级分类节点ObjID+一级分类节点ObjID+（全部一级分类节点的ObjID））,**以”+”分隔。

当发送对象节点处于半选状态时，构建ObjID：**发送对象ObjID&一级分类节点ObjID:二级分类节点ObjID|二级分类节点ObjID|……，**有多少个一级分类节点处于勾选(全选或半选)状态，就拼接多少个ObjID，之间用“；”分隔。当一级分类节点处于全选状态时不拼接二级分类节点（如**发送对象ObjID&一级分类节点ObjID**）。

（举例：发送给任课老师，发送给语文（一年级、二年级）、数学（一年级、二年级）、英语，那么需拼接三次构建好的节点ObjID：“**任课老师的ObjID&语文的ObjID:一年级的ObjID|二年级ObjID；任课老师的ObjID&数学的ObjID:一年级的ObjID|二年级ObjID；任课老师的ObjID&英语的ObjID**”）

依照上述规则，拼接所选的每个发送对象的ObjID，之间使用$隔开组成ObjIDList。

**Ⅲ、构建ObjNameList**

①只能全部发送的发送对象，构建ObjName为：全部+**发送对象节点的ObjName**。

②选择对象分类只有一种（如领导发给学生，学生只有年级分类），判断ObjChildNum：

第一，当判断得出子节点的全选数量和ObjChildNum相等，（即发送对象节点（如学生）处于全选情况下）,那么构建ObjName的格式为：**全部**+**发送对象节点的ObjName:子节点ObjName;子节点ObjName……（全部子节点）；**

1. 当判断得出子节点的全选数量和ObjChildNum不相等（发送对象节点（如学生）处于半选情况下），构建ObjName的格式为：**发送对象节点ObjName:处于勾选状态子节点的ObjName1;处于勾选状态子节点的ObjName2……，**子节点之间用“；”隔开。

③对于选择对象分类存在多重的（如领导发给任课老师，任课老师有学科和年级分类），判断ObjChildNum：

第一，当判断得出子节点的全选数量和ObjChildNum相等(即发送对象节点（如任课老师）处于全选情况下)，构建ObjName的格式为：全部+**发送对象节点的ObjName:一级分类节点ObjName；一级分类子节点ObjName……（全部一级子节点）；**在选择对象框中只显示**全部**+**发送对象节点的ObjName。**

第二，当判断得出子节点的全选数量和ObjChildNum不相等（即发送对象节点（如任课老师）处于半选情况下），构建ObjName为：**发送对象节点ObjName:一级分类节点ObjName(二级分类节点ObjName|二级分类节点ObjName|……)**，有多少个一级分类节点处于勾选（全选或半选）状态，拼接多少个“一级分类节点ObjName(二级分类节点ObjName|二级分类节点ObjName|……)”，之间用;分隔开来。

依照上述规则，拼接所选的每个发送对象的ObjName，之间使用$隔开组成ObjNameList。

**Ⅳ、构建已选对象名称（显示在新建通知界面的已选对象）**

在**Ⅲ**中将构建好的**ObjNameList**结构化字符串进行分隔处理

①将结构化字符串中的$和&替换为分号“;”。

②以“;”为分隔条件，拆解**ObjNameList**多个名称字符串，对字符串中符号文字进行处理。

③首先判断每个字符串的前两个字符是否为“全部”，若是，对符合该条件的字符串再进行以“（”作为起始位置进行切割，获取前半部分，最后获得的字符串作为**显示在新建通知界面的已选对象名称。如:全部领导、全部任课老师**

④若判断出字符串的前两个字符不是为“全部”，那么再判断字符串中是否存在“:”字符，

第一，若不存在“:”

再判断是否存在“;”：

1. 若存在“;”将字符串中的“;”替换为“、”，所得字符串为**显示在新建通知界面的已选对象名称。如:学生(一年级、二年级)**
2. 若不存在“;”则以“(”切割字符串，将获得字符串数组去除“)”，字符串数组的两个元素切换顺序拼接成一个新的字符串为**显示在新建通知界面的已选对象名称。如：一年级学生**

**第二，**若存在“:”，将字符串中的“(”替换为“[”，“:”替换为“(”、“|”替换为“、”，“)”替换为“)]”，最后获得的字符串为**显示在新建通知界面的已选对象名称。如：任课老师[一年级(语文、数学)]**

**Ⅴ、构建ObjUniqID，依照构建ObjNameList中处理逻辑**：

1. 选择对象分类只存在一种的情况时：

第一，当发送对象节点（如学生）处于全选情况下，构建Objuniqid的格式为：**第一层全选状态节点objuniqid（子节点objuniqid+子节点objuniqid+……（所有子节点的objuniqid））**

第二，当发送对象节点（如学生）处于半选情况下，将某个发送对象中处于全选状态的第一层所有节点的objuniqid，每个ObjUniqID之间以“;”分隔。

②当选择对象分类存在一种以上（如领导发给任课老师，任课老师有学科和年级分类，可假设一级分类为学科，二级分类为年级，可互换）时，

第一，当发送对象节点（如任课老师）处于全选情况下，构建ObjUniqID的格式为：**第一层全选状态节点objuniqid（一级分类子节点objuniqid+一级分类子节点objuniqid+……（所有一级分类子节点的objuniqid））**

第二，当发送对象节点（如任课老师）处于半选情况下，构建ObjUniqID的格式为：**第一层处于全选状态节点objuniqid（如语文节点处于全选）；第一层全选状态节点objuniqid（如数学节点处于半选，下层一年级处于全选）|第一层全选状态节点objuniqid（如数学节点处于半选，下层二年级处于全选）|第一层全选状态节点objuniqid（如数学节点处于半选，下层三年级处于全选）；第一层处于全选状态节点objuniqid（如英语节点处于全选）**

一级分类（如学科）内部的二级分类节点（如年级）的objuniqid之间用“|”分隔，一级分类与一级分类之间使用“;”分隔。

依照上述规则，拼接所选的每个发送对象的objuniqid，之间使用$隔开组成ObjUniqIDList。

**1.4.2、拆解结构化字符串**

前端拼接好上述四个结构化字符串之后，发送给后台会直接存储在通知数据表中，后台通过ObjTypeList、ObjIDList两个数据调用接口获取每个接收用户。

①首先在后台代码中从ObjTypeList获取ObjType；

②根据《选择对象相关类型说明》中的对应类型编号的分类，遍历识别ObjType对应的用户类型（如管理员、领导、任课老师等）；

③截取ObjIDList种对应顺序的ObjID，判别ObjID中发送对象ObjID对应的用户类型是否与ObjType对应的用户类型相同；

④根据《选择对象相关类型说明》中对应的处理条件去ObjID中截取所需条件信息（如类型为任课老师，截取一级分类ObjID，再截取二级分类ObjID）；

⑤根据用户档案管理接口或学科管理接口等接口的规范将获取的查询条件拼接成json格式数据，发送请求并接收对应用户信息（初定获取的用户信息需要：用户UserID）；

⑥将ObjUniqIDList中对应顺序的ObjUniqID拆解出来作为通知接收用户信息表的ObjUniqID；

⑤将UserID和ObjUniqID，以及从前端传输过来的已知的noticeID插入到通知接收用户信息表中。由于通过ObjType和ObjID查询出的用户可以是个体也可以是集合，因此在由同一顺序的ObjType和ObjID查询出来并插入数据表的UserID都是插入相同的ObjUniqID。

**1.5已选对象回显**

用户从草稿箱中引用草稿数据，后端返回给前端草稿箱中某个通知的所有数据，若通知数据中有已选择对象，需要回显发送对象在选择对象框，并且在用户点击“修改”之后在选择对象界面回显已选对象节点的选择状态。

方案一：由后端直接发送ObjTypeList、ObjIDList、ObjNameList、ObjUniqIDList给前端，由前端解析并构建选择对象框中已选发送对象，并由后端拆解ObjUniqIDList中ObjUniqID，得出所有已选对象树中的第一层全选节点，在需发送给前端的选择对象树的Json数据中增加表示节点选择状态的属性，根据获取的ObjUniqID在Json格式数据封装前将对应在选择对象树中的节点的选择状态属性置为全选状态，再封装为Json格式数据发送给前端，由前端解析并构建选择对象树。

方案二：由后端直接将存储在数据表中ObjTypeList、ObjIDList、ObjNameList、ObjUniqIDList发送给前端，由前端拆解四个结构化字符串，获取已选对象的节点数据，分别构建选择对象框中已选发送对象和中已选对象的选择对象树。

两个方案由前端与后台经过比较论证，并进行实际性能测试之后最终再确立实施性能的较优的方案。

**1.6 阅读详情名单展示**

|  |  |
| --- | --- |
| 发送身份 | 阅读详情需展现名单的发送对象 |
| 超级管理员 | 普通管理员（展现全部）  领导（展现全部） |
| 普通管理员 | 领导（展现全部） |
| 领导 | 教研组长（展现全部）  班主任（分年级展现） |
| 教研组长 | 任课老师（展现全部） |
| 班主任 | 学生（分行政班展现）  家长（分行政班展现） |
| 任课老师 | 学生（分教学班展现）  家长（分教学班展现） |

**2、系统主要技术难点**

2.1通知的发送方式

通知的发送方式有两种，一种是立即发送，另一种是定时发送。其中定时发送是根据用户设定的时间进行发送，并且用户设定的时间是存储在数据库表中。

难点：由于发送时间存储在数据库中，需要定时访问数据库，查询定时发送的时间，再通过定时器来发送，做不到完全的定时发送效果，通过java定时器定时访问数据库会占用很大的数据库连接池资源，并且这种轮询数据库的方式存在一定的延时，因此就需要找到一种更加高效方式。

解决方案一：初定解决方案为使用Redis 2.8.0版本之后及之后版本的键空间消息（Redis Keyspace Notifications），由于redis中存在数据过期机制，将定时发送的通知备份存储在redis中，给这个通知的key设置过期事件，使其键一旦过期就会往特定频道推消息，我在自己的客户端这边就一直消费这个频道就好了。

以后一来一条定时发送，就将其压缩成一个键，并且过期时间为距这个任务执行的时间差。那么当键一旦到期，就到了任务该执行的时间，Redis 自然会把过期消息推去，后台就能接收到，再将这个通知发送出去，这样一来就起到了定时任务的作用。

解决方案二：采用特殊数据结构：（1）环形队列，例如可以创建一个包含3600个slot的环形队列（本质是个数组），（2）任务集合，环上每一个slot是一个Set<Task>。同时，启动一个timer，这个timer每隔1s，在上述环形队列中移动一格，有一个Current Index指针来标识正在检测的slot。

Task结构中有两个很重要的属性：

（1）Cycle-Num：当Current Index第几圈扫描到这个Slot时，执行任务

（2）Task-Function：需要执行的任务指针

假设当前Current Index指向第一格，当有延时消息到达之后，例如希望3610秒之后，触发一个延时消息任务，只需：

（1）计算这个Task应该放在哪一个slot，现在指向1，3610秒之后，应该是第11格，所以这个Task应该放在第11个slot的Set<Task>中

（2）计算这个Task的Cycle-Num，由于环形队列是3600格（每秒移动一格，正好1小时），这个任务是3610秒后执行，所以应该绕3610/3600=1圈之后再执行，于是Cycle-Num=1

Current Index不停的移动，每秒移动到一个新slot，这个slot中对应的Set<Task>，每个Task看Cycle-Num是不是0：

①如果不是0，说明还需要多移动几圈，将Cycle-Num减1

②如果是0，说明马上要执行这个Task了，取出Task-Funciton执行（可以用单独的线程来执行Task），并把这个Task从Set<Task>中删除

# 开发工具及运行环境

**1、技术框架：**

①后台：Springboot

②数据库ORM框架：mybatis

**2、开发工具**

①编程工具：Springtools

②接口测试工具：postman

③数据库可视化工具：navicat12

④数据库：mysql5.7

⑤接口调用工具：restTemplate

⑥jdk版本：1.8

**3、运行环境：**

1）服务器运行环境要求：

①操作系统：windows server 2008或windows server 2012

②数据库：mysql5.7

③ web服务器：tomcat 9

2）客户端运行环境要求：

①操作系统：windows XP及以上

②浏览器内核：Trident（IE浏览器）：IE10.0至11；

Gecko（Firefox）：最新版本及主流版本；

Chrome（Chrome浏览器）：最新版本及主流版本；

**五、系统主要指标**

**1、性能指标**

1）正常网络下，页面加载时间应在2-3秒内全部完成；

2）正常网络下，页面操作应在2秒以内进行响应，最多不超过5秒；

3）服务器正常的情况下，系统可以支持24\*30小时的不间断正常运行；

**2、安全指标**

1）系统可阻止SQL注入攻击，即输入敏感T-Sql字符不会破坏数据或非法获取数据；

2）页面可防止XSS攻击，即输入“</>”等标签不会导致界面排版错乱，不会受JS脚本攻击；

3）数据接口安全，前后端接口进行加密，由前端将请求参数、token、时间戳等数据封装，使用AES加密之后异步发送给后台进行时间戳的过时判断等，并解密验证，后台在返回数据时使用对称算法进行数据加密之后再发送给前端，由前端解密数据，将其呈现再界面。