# APM PLUS后端开发

**版本记录**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本编号 | 版本日期 | 说 明 | 编制人 |
| V1.0.0 | 2019/7/19 | 文档初始化 | 陈永鹏 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### 设备管理

* 1. 设备浏览

对设备、网关、传感器的增删改查，身份、铭牌、参数属性、物理链接的增删改查。设备表现在是abb\_equipment变为equipment；设备参数属性表abb\_eq\_attr变为equipment\_attributes。

* 1. 设备类型

设备类型分为一级、二级，一级设备的类型目前在数据库维护，这个后期要有前端页面管理员去配置，一级类型用来给前端判断跳转不同设备类型的页面使用，目前前端设备类型中能看到一级类型，这个考虑不显示，只有左边的设备类型树可以看到。点击左边的树节点，右边列表刷新显示对应树节点下面的设备类型；在右上角的搜索框中搜索，模糊匹配名称和描述，列表刷新显示，可能匹配到多个不同的设备类型，左边的树自动打开对应所在的节点。



设备类型在数据库中有单独的表去存放，目前是abb\_equipmenttype，为了区别不同的表，从表名中可以看出存放的是什么，修改为equipment\_type设备类型表。

增加设备的时候，添加属性，属性是从全局的属性中拷贝到设备类型的属性表中，我看现在的表abb\_eqtype\_attr里面有很多关联不到对应的设备类型，估计删除的时候没删属性，在删除一个设备类型的时候，我查看下面是否挂载了设备，如果没有才能删除，删除的时候也要清空对应的设备类型属性。表名修改为equipment\_type\_attributes设备类型属性表。

设备类型需要equipment\_type和equipment\_type\_attributes这两个表应该就够了。

### DATABUS

之前叫service-bus，这里在aplus中是接收数据的单独模块，现在改为data-bus。和service-bus做一个区分。

data-bus模块要接受的数据有：

* + - * 1. Apm变压器和断路器模型计算输出的诊断和分数结果json数据（webservice方式）
        2. 轴承的设备 -> 测点 -> 测量数据 -> 趋势数据 -> 频谱数据 -> 测点总值 -> 报警数据（http）
        3. 电机的参数数据（http）
        4. 压缩机的参数数据
        5. 腐蚀的参数数据
        6. 建筑结构件的参数数据

根据接受数据的类型，data-bus按类型分controller和service，存放对应的数据库表。

现在轴承的测点表abb\_eq\_point、测量数据表abb\_point\_data、趋势数据表abb\_point\_trend、频谱数据表abb\_point\_frequency、测点总值表abb\_point\_vaue、报警数据表abb\_eq\_maintain。我们依次改为bearing\_point、bearing\_measure、bearing\_trend、bearing\_spectrum。

报警的数据要在监视中心进行统一管理，所以都存放在一张标准表中，现在是abb\_eq\_maintain，改为problem\_handle。这个表中目前记录了问题HeaderText和建议RecommendationText，还少个原因ConditionText，和报警数据来源的issueId。

变压器、断路器的模型输出数据示例如下，报警我们只是用了Messages中的数据，Score数据就是每一项的分值数据，这个合并到安军的表里，所有的设备打分使用一张表，在接口层面判断设备类型从而调用不同的设备类型实现，后期数据库设计思路就是复用，不同设备类型能用一张表实现的就一张表实现。

{

"AssetId": "3070000203547",

"OutputPushTime": "2019-07-15 11:04:19",

"ModelOutput": {

"Score": {

"Value": 86,

"RiskFlag": 2,

"SubScores": {

"Compressor": {

"Value": 25,

"SubScores": {}

},

"Driver": {

"Value": 40,

"SubScores": {}

},

"Lubrication": {

"Value": 10,

"SubScores": {}

},

"cooling": {

"Value": 10,

"SubScores": {}

}

}

},

"Messages": [

{

"AlarmTime": "2019-07-15 10:05:19",

"issueId": "141400BFCF3014CF8E404595D4791BF6401EF7",

"HeaderText": {

"TextId": "041400BFCF3014CF8E404595D4791BF6401EF8",

"TextValues": "检测到曲轴箱主轴承问题"

},

"ConditionText": {

"TextId": "061400BFCF3014CF8E404595D4791BF6401AS1",

"TextValues": "可能存在润滑不足或润滑油质量存在问题，可能轴承及相关部件磨损或松动"

},

"RecommendationText": {

"TextId": "071400BFCF3014CF8E404595D4791BF6401AS2",

"TextValues": "建议措施包括检查润滑油供应压力、换油或者检查轴承。"

},

"TimeframeText": {

"TextId": "911400BFCF3014CF8E404595D4791BF6401DS9",

"TextValues": "应尽快采取建议措施。"

}

}

]

}

}

### 数据库

MYSQL-5.7版本，考虑是否能升级5.8，5.8版本的性能更好，数据库引擎统一是InnoDB，现在有的表是InnoDB，有的是MyISAM；字符统一utf8mb4，优于utf8在docker中的使用，排序规则统一使用utf8mb4\_general\_ci。

字段和表名都是小写，单词之间下划线连接，SQL语句尽量不要写MYSQL专有的SQL，后续也许考虑使用sqlserver数据库，所以尽量写通用的SQL语句，字段的定义少用datetime这种MYSQL数据库自带的或者其他函数，后续切换数据库的时候不好匹配。SQL语句千万不要写几十行，几十行的时候说明数据库设计已经不合理；同一个模块之间不同的表可以做冗余，不同模块之间表如果有关联关系，尽量使用关联表。

每个表的注释，表中每个字段的注释都的加上，枚举的写明每个值代表的意思。比如现在的设备是否报废，数据库中的记录有的是0，有的是false。

表名一般按模块去划分开头，便于区分。

### 后端代码

后端JAVA代码使用JDK1.8版本开发，springboot我们统一升级到2.1.6，目前好像是2.1.3或者更低。GITLAB代码库，IDEA开发工具，SpringCloud版本Greenwich SR2，目前是Greenwich，我们都向前升级一下，应该没有大问题。

代码规范遵循阿里巴巴编码规约插件，在IDEA中可以下载；IDEA中配置copyright和类注解模版。

/\*

\* Copyright (c) 2019, ABB and/or its affiliates. All rights reserved.

\* ABB PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.

\*/

/\*\*

\* <类名>

\*

\* @author xxxxx

\* @date 2019-03-28

\*/

配置文件共用部分写在application.yml文件中，开发环境写application-dev.yml，测试环境写application-test.yml。提交代码之前使用快捷键去掉无效的import引用，使用快捷键格式化代码，所有文件默认UTF-8编码；Code Style -> JAVA下面去掉使用tab字符。记得安装lombok插件，所有实体类用注解实现get和set方法。方法和接口都的加注解，整个文件在IDEA编辑区的右上角地方最好显示一个绿色的对勾。使用@ Autowired注解的报黄色的除外，可以不修改。

包名都是com.abb开始，后面跟自定义，访问层controller，服务接口层service，接口实现在service下面新建impl，接口实现类类名在接口类后面加Impl字段表明实现哪个接口。

### 国际化

报警故障、单位这种的需要后台返回数据的国际化，通过自定义AOP切面注解实现，这些数据在数据库中有一张标准字典表，使用spring cache快速加载。使用这些字段的地方都用code代替，国际化这个可以单独做一个服务，根据code和语言，自动返回所对应的语言。后期对这一个表开发前端维护界面，翻译的这些工作由使用的管理员自动增加和修改更新。