代码规范及实践

代码中必须有unittest覆盖必要的逻辑

1. 熟悉junit常用功能（写unittest的基本工具）
2. 在业务逻辑中，多使用接口进行解耦，以方便进行unittest （但你觉得逻辑写unittest太难时，这意味着你的业务设计本身有问题，各个模块耦合度太高，导致功能划分不清楚，这就要求你需要对现有代码进行重构）。

（**需要组织一个unittest的实践分享**）

熟悉第三方常用jar

在自己要实现一些业务无关的通用逻辑时，先搜索下是否已经有相关通用lib已经提供了这种功能，要避免自己写一些low-level且容易出错的逻辑，比如目录递归删除，文件拷贝，对set进行union等等。总之，要避免自己闭门造车。

1. 熟悉guava常用的utilities，如Preconditions，Throwables，Immutable collections，Caches等 ，参考https://github.com/google/guava/wiki

   maven dependency

   ```
   <!-- https://mvnrepository.com/artifact/com.google.guava/guava -->
   <dependency>
       <groupId>com.google.guava</groupId>
       <artifactId>guava</artifactId>
       <version>29.0-jre</version>
   </dependency>
   ```

2. 熟悉apache commons-lang常用类，如MutableXXX(MutableObject，MutableInteger，...)，XXXUtils(ArrayUtils，CollectionUtils，...)，Pair，Tripple等等（参考http://commons.apache.org/proper/commons-lang/javadocs/api-release/index.html）。

   maven dependency

   ```
   <dependency>
     <groupId>org.apache.commons</groupId>
     <artifactId>commons-lang3</artifactId>
     <version>3.10</version>
   </dependency>
   ```

3. 熟悉apache commons常用类（https://commons.apache.org/proper/commons-collections/javadocs/api-4.4/index.html），如CollectionUtils，ArrayUtils，StringUtils等。

   maven dependency

   ```
   <dependency>
       <groupId>org.apache.commons</groupId>
       <artifactId>commons-collections4</artifactId>
       <version>4.1</version>
   </dependency>
   ```

4. 熟悉apache commons-io常用类(参考http://commons.apache.org/proper/commons-io/javadocs/api-release/index.html)，如常用的文件操作FileUtils，CopyUtils等。

   maven dependency

   ```
   <dependency>
       <groupId>commons-io</groupId>
       <artifactId>commons-io</artifactId>
       <version>2.7</version>
   </dependency>
   ```

5. 熟练使用lombok，将自己从手动编写java的constructor，getter/setter等中解放出来，同时习惯使用其builder，了解其基本原理

maven dependency

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.12</version>
    <scope>provided</scope>
</dependency>
```

代码规范：

1. 熟悉阿里的代码规范https://www.cnblogs.com/han-1034683568/p/7680354.html
2. 深入理解并实践常用的设计模式，如template，singleton，adapter，facade等，这有助于改进我们代码整体设计

接口设计规范

1. 创建接口：单个创建接口：返回为EnosCommonRsp<T>，成功的话data字段应为对象的id，或者为该对象最简洁的结构；
   批量创建接口：返回为EnosBatchRsp<T>，返回结果包含每个实例的操作结果信息。某一个实例的返回成功的话，data字段应为对象的id，或者为该对象最简洁的结构；失败的话显示具体错误信息；
2. 删除接口：单个删除接口：返回为EnosCommonRsp<Void>，data字段设置为null，成功的话issuccess==true，失败的话issuccess==false；如果待删除的对象不存在，错误码为公共错误码404；
   批量删除接口：返回为EnosBatchRsp<T>，返回结果包含每个实例的操作结果信息。某一个实例的返回成功的话，data字段应为对象的id，或者表示实例的结构体；失败的话显示具体错误信息；
3. 更新接口：单个更新接口：返回为EnosCommonRsp<Void>，data字段设置为null，成功的话issuccess==true，失败的话issuccess==false；即使对象内容在执行前后没有发生改变，也返回成功。如果待更新的对象不存在，错误码为公共错误码404；
   批量更新接口：返回为EnosBatchRsp<T>，返回结果包含每个实例的操作结果信息。某一个实例的返回成功的话，data字段应为对象的id，或者表示实例的结构体；失败的话显示具体错误信息；
4. 查询接口：单个查询接口：返回为EnosCommonRsp<T>，成功的话data字段应为对象结构；
   批量分页接口：返回为EnosPageRsp<T>，成功的话data字段应为对象结构列表，最大每页1000条，该接口最大查询到的记录数为10000。默认分页大小为10条，分页大小最好保持跟前端一致，根据实际情况可自定义。
   翻页接口：　　当需要数据量超过1万的查询场景时，例如数据聚合计算，制作报表等。推荐增加该查询接口，EnosScrollRsp<XXX>接口。
   批量不分页接口：（例如ListDevices）：返回为EnosBatchRsp<T>，返回结果包含所有的结果信息。每一个结果的返回成功的话，data字段应为对象；失败的话显示具体错误信息；

代码实践和用例

1. 申明变量时，能用接口就用接口，避免使用实现类做申明。

变量声明

```
public void declareVariable() {
    //
    HashMap<String, HashSet<String>> coll1 = new HashMap<>();

    //
    Map<String, Set<String>> coll2 = new HashMap<>();
}
```

2. 方法返回集合类时，避免返回mutable的集合（除非你清楚你在做什么），如果类中的集合本身不会变，则使用guava的immutable集合；否则，使用Collections的unmodifiable方法进行wrap。
（补充实例）
3. 判断集合或者数组为空时，采用可读性更高的CollectionUtils.isEmpty, ArrayUtils.isEmpy, StringUtils.isBlank等
4. 返回空集合时，避免new中一个空集合，而应该采用Collections.emptyMap(), Collections.emptySet()的形式 （思考下，我们为什么需要这么做？）。
5. 不会变动的变量加上final申明（包括类中和方法中的变量），让编译器帮助我们避免因为bug修改一个本应不应该修改的值
6. 字符串操作统一使用com.envision.eos.commons.utils.StringUtil
7. 一定要逻辑复用，避免重复代码 （否则，容易不一致，难以维护）
8. 避免传入不必要或者重复的参数
9. 非法参数校验要尽可能前置
10. 能用接口的地方尽量用接口，不用要具体类（这样的代码更易复用，而不容因为具体类变更而修改已有逻辑）
11. 用面向对象的逻辑去思考，将复杂的逻辑抽象出来，if-else情况多时，考虑用模板抽象
12. 暴露的接口和预期不相符(暴露Map<String, Object> measurepoints，但只接受size为1)
13. 接口改造后，我们这边的client（比如dm-bff）也要相应改过来使用新的接口
14. 不要过渡lambda使用，简化应该是以优雅明确易懂的方式进行
15. 如果是模板，以template进行编码，面向接口编程和面向对象编程，而不是lambda进行行为流程化。
16. 对于本地缓存，需要 ” 超时时间可配置 “，"缓存刷新异步化"，"缓存命中率统计日志"，"缓存击穿、缓存雪崩等边界条件"。