**Programming Criteria**

@author Count Monte Cristo

@create D20150703

@version 2017年3月12日 下午7:41:32

2016/11/10 15:55

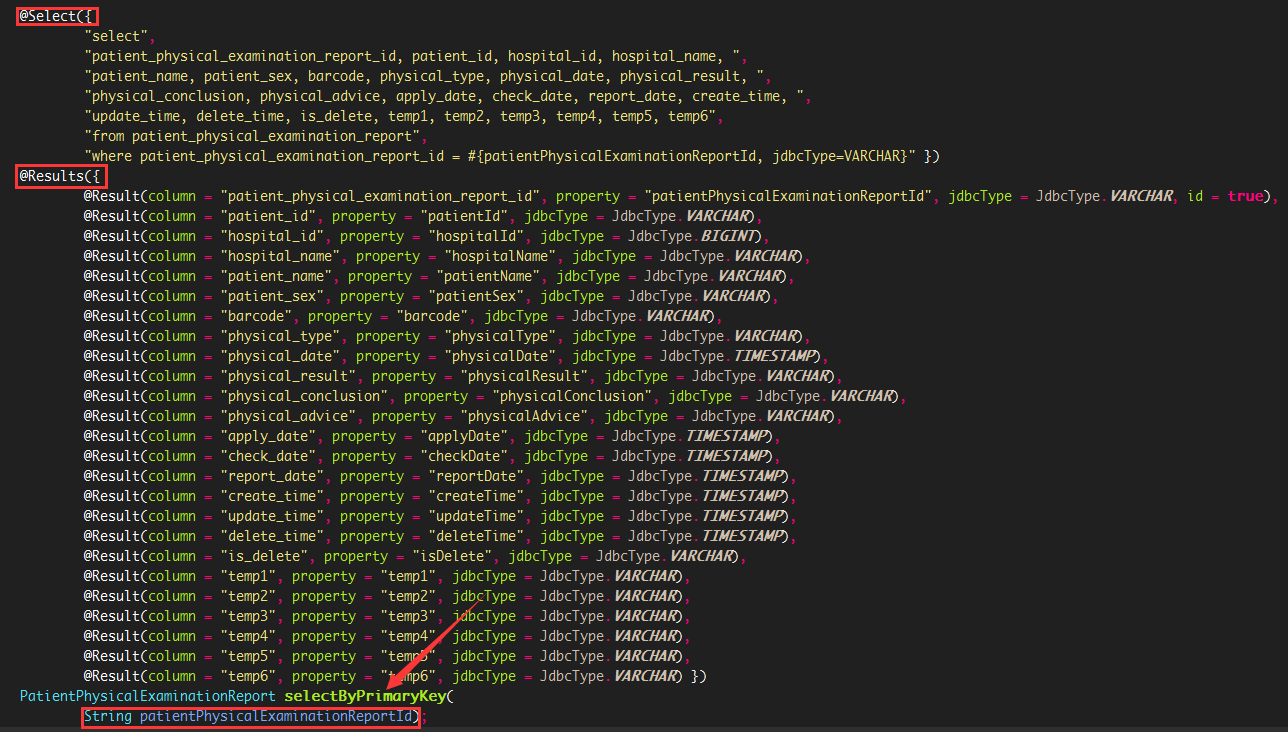
~~D20150813~~ ~~D20151022~~ ~~D20151110~~ ~~D20151120~~ ~~D20151124~~ ~~D20151204~~ D20151217

@description 此文件有且定义编程的标准

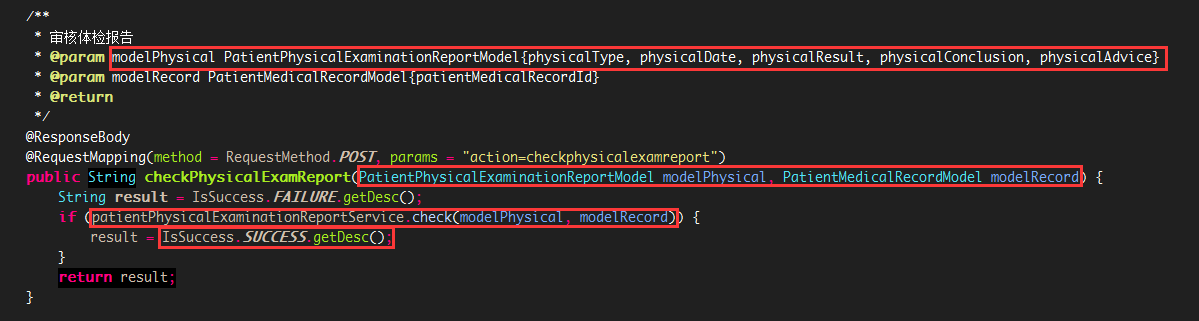
@motto VICTORY LOVES PREPARATION

@criteria 凯旋基诺

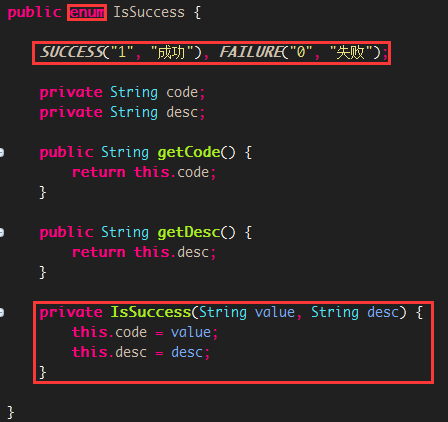
**目录**

1. **数据库的基本操作CRUD**
2. **Java Web普通业务标准处理流程**
3. **规范实例说明**
4. **程序copyright说明标准**
5. **计划VS项目**
6. **面向对象程序设计**
7. **页面跳转以及页面相关问题**
8. **服务层的基本操作CRUD**
   1. **原始含义【CRUD】**
      1. C：create
      2. R：retrieve
      3. U：update
      4. D：delete
   2. **实际使用【AGUD】（**@notice 不同的项目还是要入乡随俗 @version D20151217**）**
      1. **add**
      2. **get(list，多条数据)**
      3. **upt(update、form)**
      4. **del(del)**
9. **接口前缀**
   1. **set**
      1. 设置数据
      2. 单复数加上英文单词的正确形式
   2. **get**
      1. 获取数据
      2. 单复数加上英文单词的正确形式
   3. 注意事项
      1. 不要使用list当做前缀获取数据列表，使用get加上英文正确的单复数形式
10. **Java Web普通业务标准处理流程**
    1. Controller、Service和Dao中尽量采用面向对象程序设计思想，即参数尽量采用Domain Bean
    2. 具体的数据库操作则需要使用面向关系程序设计思想，即采用具体的字段
    3. **Java Web程序实例（基于SSM框架）**
       1. **Domain**
          1. **普通的POJO**
          2. **分类**
             1. Domain
             2. VO（或者DTO）
       2. **Service**
          1. **实例**
          2. **说明**
             1. 箭头所指地方是设计上的瑕疵，应该面向对象设计，直接传入Domain
             2. Service应面向接口编程
             3. 在Service层应该完成Domain到VO（或者DTO）的映射，应该直接使用VO（或者DTO）进行业务操作，到最后（调用Dao，交互数据库）把VO（或者DTO）转换为Domain
       3. **Dao**
          1. **实例**
          2. **说明**
             1. 箭头所指地方设计失误，应该直接传入Domain
             2. 此Dao使用了Mybatis ORM进行数据库操作，其他ORM框架亦是如此操作
             3. Dao层问题

数据的查询一定要保证依赖数据库做唯一查询，不能依赖后端中自定义的“唯一”标识，即数据的得到不依赖于代码而是数据库

* + 1. **Controller**
       1. **实例**
       2. **注意点**
          1. 传入的参数应该都是VO（或者DTO），并且直接传递给service
          2. 因为传入的是VO（或者DTO），所以在@param注释的时候要通过“数组”的形式对VO（或者DTO）的属性进行说明
          3. 要积极使用枚举（特别是枚举的构造方法要熟练掌握），提高代码的可扩展性和可修改性

枚举实例

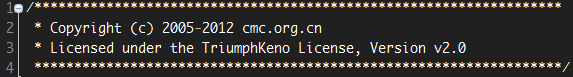


说明

因为是定义新的枚举，所以是Enum类型

枚举的构造方法返回类型不能是public，因为枚举的构造实例是写在枚举类型内部的

* + 1. **View**
       1. HTML、CSS、JS等组成
       2. 具体的页面应该只有HTML代码（可以使用velocity、freemarker等模板引擎），CSS和JS文件应该单独存放，通过外部引用的方式加载，要遵循单一职责原则
  1. **说明**
     1. 正式（平时代码可以没有）的代码都要注明作者和版权
        1. 实例



* + 1. 包的命名要符合规范
    2. 导入的包要分门别类
    3. **任何类以及方法之前要有合适的文档注释**
       1. **类和方法采用文档注释**
       2. **具体代码块采用块注释**
       3. **具体行代码采用行注释**
    4. 类内的开头和结尾以及方法与方法之间要空一行
    5. 尽量不要折行
    6. 文档注释中参数和说明要在同一行
    7. **不要定义不需要复用使用的变量**

1. **计划VS项目**
   1. **计划**
      1. 具有两重含义，其一是计划工作，是指根据对组织外部环境与内部条件的分析，提出在未来一定时期内要达到的组织目标以及实现目标的方案途径。其二是计划形式，是指用文字和指标等形式所表述的组织以及组织内不同部门和不同成员，在未来一定时期内关于行动方向、内容和方式安排的管理事件。
   2. **项目**
      1. 指一系列独特的、复杂的并相互关联的活动，这些活动有着一个明确的目标或目的，必须在特定的时间、预算、资源限定内，依据规范完成。
   3. **Notes**
      1. 计划和项目在实际生活中的使用是有一定的区别的，前者更为广泛应用，后者多见于工程化方面。但是后者更符合现代化的高效、规范等特点，采用工程化的方法更容易达到目的，**所以以后生活中建议使用“项目”字段取代“计划”**。

**5. 面向对象程序设计**

* 1. 对象
     1. 不仅能够表示具体的事物，还能表示抽象的规则、计划或者事件（具体的事情，含事情的**域和动作**）
  2. 具体
     1. 一个方法代表一个动作（动词）
     2. 一个修饰词代表一个条件（形容词）
  3. 实例
     1. 用户进行提现操作实例
        1. 因为此事件功能较多并不是简单的提现操作，所以此时的“对象”则是“事件”，此时类的名称可以是MoneyDrawingController（**尽量不要使用动宾结构**），而具体的提现操作可以是draw(Object obj)、或者drawMoney(Ojbect obj)，建议使用前者，**因为后者存在在意思上和所处类存在重复性。**

1. **页面问题**
   1. 跳转到页面的时候不能直接进入页面，而是需要先进入对应该页面的Controller进行页面的渲染操作
   2. 功能处理
      1. 能在前端进行判断的应该都在前端进行判断，需要后台（数据库）支持的应该提交到后端进行处理
      2. 到后端的处理过程建议使用（JQuery）AJAX进行异步刷新处理，不建议使用Form表单进行提交操作
         1. 实例
      3. 后端返回数据需要进一步使用的话建议把后端原始数据赋值到HTML中（通过hidden类型的<input>标签），然后JS引用对应<input>的值
      4. JS文件应该放在页面的底部，因为JS是解释型语言，一旦解释错误就不会再往下执行
      5. JS文件建议单独放在一个文件中，页面进行外部引用，不建议直接放在页面上

**7. 关于对象和变量使用问题**

* 1. 使用之前一定要判断其值是否符合程序基本要求，如不能为空值

**8. 关于SQL复杂度和Service复杂度的均衡** @version D2015117 @status 有待解决

a) 建议单一SQL功能，在Service中进行重组

**9. 条件的判断处理**

* 1. 凡是可能出现的情况都要进行处理
  2. 处理的结果按照前后端分别处理

**10. “就近原则”**

* 1. 数据的获取需要从最近的**请求**中获取，不应该依赖其他请求

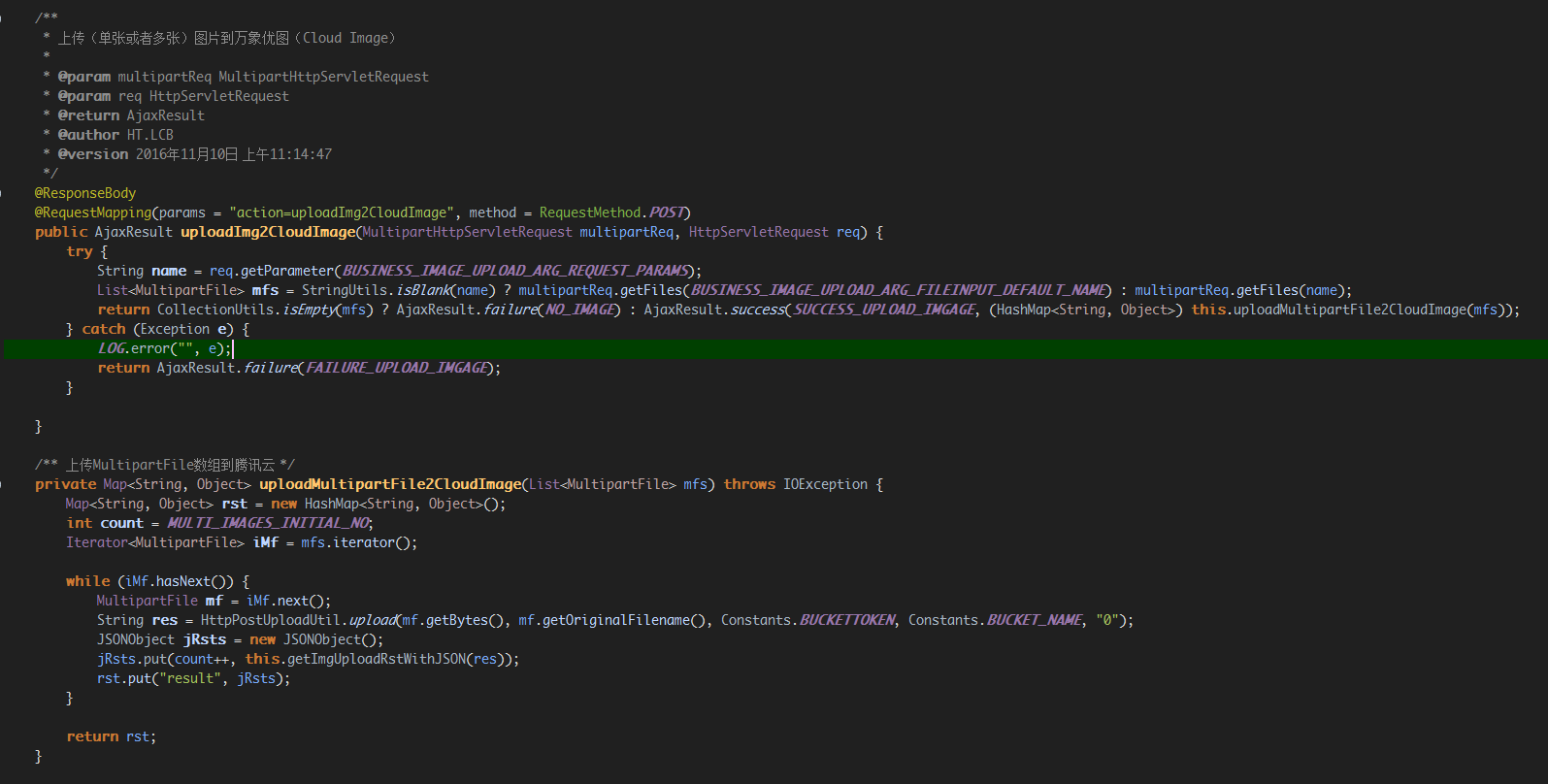
**11. 命名规则**

* 1. 驼峰命名法 + 结合数据库表格

**12. 流程相似业务处理**

* 1. 放在一起处理，通过flag“分流”

1. **异常处理 @version 2016/11/10 15:55**
   1. **统一交给最外层调用者处理**
2. **If else 要考虑是否能够使用三元运算符**
3. **代码块前后无论有没有注释都要空一行，代码行则不需要**
4. **先处理异常情况，满足则直接返回，然后再往下走，这样可以降低代码层级关系**
5. **实例**





1. **代码格式思想**
   1. **易懂**
   2. **简短**
   3. **格式整齐**
2. **属于哪个service的业务就要放到哪个service中，从而service可以调用service**
3. **数据校验**
   1. **<http://www.cnblogs.com/pixy/p/5306567.html>**
   2. **对于后端处理程序，虽然业务上前端做了数据校验，但是传入到后端的数据可能经过http请求相关工具的处理，导致前端校验失效，所以后端也许要进行数据校验（语义上、业务上的数据校验）。**
   3. **数据校验也是有程度的，有的数据（不符合业务，但是基本上不会出错，出错了影响程度基本为零）不需要进行校验。**