# 大医疗项目代码编写及审查标准

1. 命名检查

* 包名必须以com.aspirecn.msp开头；包名必须全小写。
* 变量采用Java标准的驼峰命名方法。
* 包、类、方法、参数、命名是否能够见文知意。

示例：

|  |  |
| --- | --- |
| **包名** | **说明** |
| com.aspirecn.msp.audit.model | 智能审核业务模型对象 |
| com.aspirecn.msp.audit.service | 智能审核业务逻辑接口 |
| com.aspirecn.msp.audit.service.impl | 智能审核业务逻辑实现接口 |
| com.aspirecn.msp.audit.dao | 智能审核数据操作接口 |
| com.aspirecn.msp.audit.web.controller | 智能审核控制器 |
| com.aspirecn.msp.portal.model | 基础门户业务模型对象 |
| com.aspirecn.msp.portal.service | 基础门户业务逻辑接口 |
| com.aspirecn.msp.portal.service.impl | 基础门户业务逻辑实现接口 |
| com.aspirecn.msp.portal.dao | 基础门户数据操作接口 |
| com.aspirecn.msp.portal.web.controller | 基础门户业务控制器 |
| com.aspirecn.msp.dict.model | 基础字典业务模型对象 |
| com.aspirecn.msp.dict.service | 基础字典业务逻辑接口 |
| com.aspirecn.msp.dict.service.impl | 基础字典业务逻辑实现接口 |
| com.aspirecn.msp.dict.dao | 基础字典数据操作接口 |
| com.aspirecn.msp. dict.web.controller | 基础字典业务控制器 |
| 其它依此类推 | |

注：红色部分根据业务变化

各包下可根据情况建立子包

* Controller的URL规则为必须以该业务域开头

示例如下：

* 基础门户的所有URL都以/portal开头
* 基础字典的所有URL都以/dict开头
* 智能审核的所有URL都以/audit开头
* 智能审核接口的所有URL都以/auditPort开头
* 其它依此类推。
* Web资源目录标准

|  |  |
| --- | --- |
| **目录** | **说明** |
| content | 静态资源及js根目录 |
| dynamic | 动态（jsp）资源目录 |
| content/common/js | 公共js目录；  自开发公共js文件  第三方webui组件等 |
| content/common/images | 公共图片等其他资源 |
| content/common/css | 公共样式 |
| dynamic/common | 公共JSP文件 |
| content/portal/js | 基础门户js文件 |
| content/portal/images | 基础门户图片等其他资源 |
| content/portal/css | 基础门户样式 |
| dynamic/portal | 基础门户的动态jsp文件 |
| content/audit/js | 智能审核js文件 |
| content/audit/images | 智能审核图片等其他资源 |
| content/audit/css | 智能审核样式 |
| dynamic/audit | 智能审核的动态jsp文件 |
| content/dict/js | 基础字典js文件 |
| content/dict/images | 基础字典图片等其他资源 |
| content/dict/css | 基础字典css |
| dynamic/dict | 基础字典的动态jsp文件 |
| 其他依此类推 | |

备注：

* + - * 不允许同时引入多个第三方组件，如：引入jquery多个版本的组件包是不允许的。
      * 不允许引入多个同样功能的组件，如：引入两个同样的富文本编辑器是不允许的。
      * 第三方包的引入要有严格控制。
      * 各业务点根据需要可以在规定目录下根据情况建立子目录。

1. 同一业务模型同一层次不允许有多个对象

* 不允许对同一数据字典的数据库操作存在两个dao对象
* 不允许对同一数据字典的逻辑操作存在两个service对象
* 不允许对同一数据字典操作存在两个控制器

1. 引用检查

* 不允许出现类似于import java.util.\*的全部引用。
* 去除无效引用。
* 不允许同时引入同名对象,如: java.sql.Date，java.util.Date。

1. 逻辑检查

* 函数复杂度是否合理(代码行不超过10行).
* 函数是否只完成单一职责.
* 是否使用本地缓存(在集群环境中可能会出现问题).
* 是否存在重复代码块，如果存在需要做抽象后拆分。
* 是否有更好的实现方式。
* 函数应该避免存在多个出口。
* 实体对象是否都实现了系列化接口。
* 是否能够减少不必要的操作。
* 子类中部允许定义父类中已经存在的属性[分布式调用时反序列化时会存在问题]。
* 是否存在业务逻辑中提取了过多的数据，例如：
  + 加载列表时提取了无须展现的属性。
  + 只需要其中一个或者几个属性但是查找了提取了整个对象，如：提取分屏基础业务时加载了所有属性，应该只需要基础业务编码就可以了。
  + 判断是否存在提取了数据列表：如判断是否存在的操作，只需要提取记录条数则可。
* 循环语句是否可能存在死循环。
* Switch是否有default默认动作。
* 尽量用返回值来标识数据与业务检查，尽量避免使用Exception来处理业务检查错误。
* 不允许存在e.printStackTrace()；要用logger.error来输出异常。
* 输出日志的级别是否合理，必须采用slf4j来输出日志，并且避免采用拼接字符串的形式。

建议方式：logger.info("用户[{}]登录失败,没有找到对应的用户。",name);

不建议的方式：logger.info("用户[" + name + "]登录失败,没有找到对应的用户。");

1. SQLMap检查

* 过多的if else if分支是否能够转换成switch。
* 必须存在的条件在SQLMap中不允许加<if test></if>判断
* 加载列表时尽量按分页加载，如：用户登录日志等
* SQLMap中参数除分区关键字外采用#{}来包装,避免采用${}来包装.
* SQLMAP中不允许加上数据库的schema引用。

1. 应统一集中处理的部分

* 除必要的业务提示外异常统一处理。
* 对于函数的调用及调用最好统一处理。

1. SQL语句检查

* 不允许存在is null和is not null
* 避免使用not取反或者<>，!=

低效：select name from employee where salary<>3000;

高效：select name from employee where salary<3000 or salary>3000;

* 避免使用select \*
* 用EXISTS替代IN、用NOT EXISTS替代NOT IN，尽量避免使用Not取反
* 避免在条件列上使用计算

低效：SELECT … FROM DEPT WHERE SAL \* 12 > 25000;

高效: SELECT … FROM DEPT WHERE SAL > 25000/12;

* 尽量用>=替代>
* 用IN来替换OR

低效: SELECT…. FROM LOCATION WHERE LOC\_ID = 10 OR LOC\_ID = 20 OR LOC\_ID = 30

高效：SELECT… FROM LOCATION WHERE LOC\_ID IN (10,20,30);

* 尽量使用union all 替代union
* 表关联尽量以小表做基表，查询条件尽量把过滤数据量大的过滤条件放到最后面（SQL99标准）（右）。

1. 其他

* 用findbug进行检查重要等级及以上的问题必须修复.