开发规范

1. 系统规范

1.1 系统命名

建议系统命名时遵循一定的命名规则，根据系统用途的不同，选择适当的后缀。例如，可以根据以下规则选择后缀：

- -web，页面展现类的系统，更多地是与用户交互
- -core，某个业务的核心处理系统
- -mng，管理类系统
- -gw，网关类系统

如有特殊命名要求，可以酌情放宽要求。

1.2 系统结构

系统必须由框架生成工具生成，以便自动获得模块化的多工程Maven项目、默认的配置文件以及编译部署脚本等内容。

为了避免将代码堆砌在一个模块内，建议将项目分为以下几个模块（生成工具默认会生成这些项目并建立依赖）：

1. base，存放最基本的全局配置和代码
2. dal，存放数据库操作相关的代码，myBatis映射、配置、DAO接口及实现等等
3. service
    1. service-api，存放发布的服务接口和模型类
    2. service-impl，存放服务的实现代码
    3. service-integration，集成外部服务
4. biz，存放业务逻辑
5. web，存放Web页面控制器
6. test，存放集成测试代码

1.3 部署环境

运行所需的基础软件（比如JDK、Tomcat、Maven等等），都统一安装在一个目录中，随后建立规范的link指向目标，以安装在~/soft为例：

1. ~/soft/java => ~/soft/jdk1.7.0_40
2. ~/soft/tomcat => ~/soft/apache-tomcat-7.0.42
3. ~/soft/maven => ~/soft/apache-maven-3.0.4

软件目录中仅放置未修改的“原始”版本，所有对配置的修改，环境变量的设置，都在项目的部署脚本中完成。（框架默认已支持）

2 数据库操作

以下涉及特定数据库特性时，如无说明均指MySQL数据库。

2.1 数据库连接池

项目中使用数据库时，严禁自行在代码中创建数据库连接，必须使用数据库连接池，可以选择在Spring中进行数据库连接池的配置，也可以选择JNDI方式，在应用服务器容器中进行配置。

以将数据库连接池配置在Spring容器中为例，选择的的数据库连接池为Druid。

有以下注意事项：

1. 连接数据库所用的配置信息必须剥离在对应的-db-config.properties中，不得硬编码在Spring配置文件中。
2. 连接使用的密码必须以密文形式存放，生产环境解密所用的公钥由运维管理。

2.2 Mapper与Dao的编写

myBatis的映射文件分为两部分——自动生成部分、手写部分

自动生成的内容包含Mapper接口、SQLMapping文件、DO对象，位于dal的目录中，SQLMapping放在mapping/auto中，严禁修改自动生成的内容。手工编写的映射文件放在mapping/manual中。

上层代码不直接使用Mapper，需要编写Dao接口及实现，由Dao进行统一的封装和调用。Dao必须位于指定的dao包下，Dao操作必须输出摘要日志。

2.3 敏感数据的处理

数据库中严禁存储以下内容：

- 信用卡有效期
- 信用卡CVV
- 用户的各种密码（应用本身的密码除外）

数据库中严禁以明文方式存储以下内容：

- 用户的各种密码（需加密存储于专门的库中）
- 信用卡卡号

信用卡的卡号不得在网络中明文传输，不得以明文形式出现在日志或标准输出中（可以出现前六后四的形式，或者索引卡号）；信用卡的有效期及CVV，不得出现在日志或标准输出中。关于日志的输出，详见日志规范。

2.4 表及字段命名

数据表名、字段名、索引名的命名中，使用_分隔单词，例如：

- 表名：user_info
- 字段名：id、gmt_create
- 索引名：pk_user_info、fk_ui_od、uniq_username

建议所有的数据表中都需要增加以下两个字段：

1. gmt_create，类型`datetime`，表示记录创建时间，一经创建不得修改
2. gmt_modified，类型`datetime`，表示记录最后修改时间，每次修改记录内容时，该字段的值必须改为`now()`

2.5 SQL编写注意事项

2.5.1 不得省略或者使用*代替字段列表

自己手工编写的SQL语句中，需要明确标明数据表字段，不得省略或者使用`*`代替，例如：

1. 在`select`语句中，需要固定返回字段列表
2. 在`insert`语句中，需要指定插入字段，不能仅有`value`

2.5.2 锁定记录的方法

如果是在使用Oracle数据库，锁定记录必须使用`for update nowait`。

2.5.3 大数据量查询

对于返回值很多的查询，如无特殊情况，尽量采用分页查询的方式，逐批返回查询结果。

可以采用如下步骤查询数据：

1. 通过`select count(*)`查询符合条件的记录的总条数，此时不用带`order by`子句
2. 通过`select limit`查询指定范围内的记录，此时需要指定排序顺序，如果使用日期类型的字段排序，要注意相同时间的记录的顺序，建议再指定一个能标明顺序的字段

2.6 事务

2.6.1 事务的配置

在需要事务保障的地方，必须使用框架支持的方式启用事务，严禁自行调用数据库连接开始、提交或者回滚事务。在Spring中提供了以下声明事务的方式：

1. 编程式（如无必要，不建议使用）
2. 声明式
    1. XML配置
    2. 注解（推荐）

在能够明确是只读操作的情况下，优先选择只读事务，即：

```
    @Transactional(readOnly = true)
```

2.6.2 大数据量操作

在大批量数据操作时，如无特殊要求，必须使用批量操作，不能变更一条记录提交一次。

对大量数据进行更新和删除时，必须拆分进行，不能一次处理。

2.6.3 其他

在事务中，不得出现以下情况：

1. 主动调用`Thread.sleep`，并应尽量避免等待各种资源
2. 进行远程调用（特定场景除外，比如分布式事务的情况）

3 并发操作

3.1 线程池

在系统中如无特殊情况，不得自行创建线程，需要配置线程池来管理线程。

在Spring的应用程序中，可以使用`ThreadPoolTaskExecutor`来配置线程池。对于不需要返回值的情况，直接使用`execute`方法，需要返回值的情况使用`submit`方法。

3.2 java.util.concurrent的使用

在系统中，如果可能出现并发的场景，需要考虑用`java.util.concurrent`中的辅助类，替换常用的一些类，比如：

- `ConcurrentHashMap`
- `CopyOnWriteArrayList`
- `LinkedBlockingQueue`

一些原子的数据类型，可以使用`java.util.concurrent.atomic`中的对应类。

3.3 ThreadLocal的使用

使用`ThreadLocal`时，必须自己手动调用`remove`方法进行清理，避免多线程操作时产生影响，比如线程池线程复用时存在遗留数据。

4 Web与服务

4.1 Web控制器的声明

TBD

4.2 REST服务的声明

TBD

4.3 REST服务的调用

TBD

4.4 服务结果的返回

所有服务的响应中建议包含以下两个内容：

1. `resultCode`，结果码，0表示系统执行成功且业务成功，其他结果下文详细描述
2. `resultMessage`，结果描述，以文字形式描述返回内容

结果码非成功状态构成规范：

```
4位系统标识符_一级结果码_二级结果码
```

- 4位系统标识符，表示结果码的来源，0000-0999为保留段，1000-9999可用，每个系统的标示符另行约定
- 一级结果码，0开头的为保留段，系统的结果码首位从1开始
- 二级结果码，一般为外部系统传入的结果码，可一同返回

4.5 Web安全

TBD

5 代码风格

团队中的所有成员需要使用一样的代码模板、遵循一致的代码风格，如果使用Eclipse，可以导入统一的CodeTemplate、CleanUp和Formatter配置。

5.1 命名

5.1.1 包名

Java代码中包名使用全小写字母，使用完整拼写的单词，或者有含义的缩写。

5.1.2 接口与类

接口名和类名遵循驼峰规则，首字母大写，需要是个名词的完整形式，尽量避免不常见的缩写，接口名无需以`I`开头，接口的实现类可以使用`-Impl`后缀。

5.1.3 常量、变量与枚举

常量名必须使用全大写，以`_`分隔单词，需要是个名词，仅日志的`Logger`定义除外，可以用以下形式：

```
    private static final Logger logger = LoggerFactory.getLogger(xxx.class);
```

变量名需要遵循驼峰规则，首字母小写，需要是个名词。

枚举的属性命名规则同常量。

一行仅一个声明，避免在一行内为多个变量赋值。仅在代码块的开头处定义变量。

内外层的变量尽量不要重名，以免产生误解和分歧。

Java接口中的常量默认是`public static final`的，所以定义时无需添加这三个修饰符。

5.1.4 方法

方法名需要遵循驼峰规则，首字母小写，必须以动词开头。

类成员属性的`getter`和`setter`方法使用IDE自动生成的功能，遵循JavaBean的命名规范。

Java接口中的方法默认是`public abstract`的，所以定义时无需添加这两个修饰符。

5.2 注释

一个标准的Java文件中，需要包含以下注释：

1. 文件版权信息，使用Javadoc格式
2. 类型说明，使用Javadoc格式，需要包含简要说明和作者信息
3. 成员属性注释，使用Javadoc格式（可选）
4. 构造方法注释，使用Javadoc格式
5. 所有public和protected的方法需要注释，使用Javadoc格式，需要包含简要说明、参数、返回值和异常信息

IDE自动生成的`getter`和`setter`方法无需注释。

5.3 缩进

禁止使用Tab进行缩进，所有的缩进均用Soft Tab，用4个空格代替Tab进行缩进。

一行代码仅包含一条语句。同级的代码无需缩进。

5.4 空格

1. 关键字和随后的括号`(`间应该有空格分隔
2. 括号`)`和随后的花括号`{`间应该由空格分隔
3. 参数列表定义中的`,`后应该带有空格
4. 所有操作符前后应该有空格（`.`除外）
5. 强制类型转换的类型后面应该有个空格
6. `for`语句中的表达式应该用空格分割

5.5 括号

所有的`{`都无需另起一行，紧跟之前的内容，之间用空格分隔。所有的`}`都单独一行。

所有的分支、循环语句，必须用`{}`包裹，即使只有一行内容也是如此。如果分支或循环语句的执行体较长，在`}`后建议添加注释说明是哪个的结尾。

在复杂的表达式中，必须用`()`来确定计算优先级。

5.6 长度

1. 单行的长度建议控制在100个字符以内，最多不要超过120个字符，如语句过长可以换行并缩进
2. 方法的长度建议控制在200行以内，最好是一屏以内
3. 类的长度建议不要超过2000行
4. 方法的参数建议不要超过5个，如果参数较多可以封装成参数类，REST接口的参数无论多少都建议封装成参数类
5. 方法内的嵌套建议不要超过3层

6 其他

6.1 字符串处理

对于字符串的处理，尽可能使用`StringUtils`等辅助类进行。

比如，可以使用使用Apache Common Langs 3.1，StringUtils的具体方法，详见官方JavaDocs。

6.1.1 字符串判空

严禁使用下列的方式进行判断：

```
str1 == null || str1.length == 0
```

可以根据情况使用下面的方式：

```
StringUtils.isBlank(str1)
StringUtils.isEmpty(str1)
```

6.1.2 字符串比较

严禁使用下列方法进行判断：

```
str1 == str2
str1.equals(str2)
```

必须使用`StringUtils`的相关方法：

```
StringUtils.equals(str1, str2)
StringUtils.equalsIgnoreCase(str1, str2)
```

实在无法使用`StringUtils`的时候，必须将明确不为`null`的对象放在前面：

```
"abc".equals(str1)
```

6.1.3 字符串拼接

在有为数众多（大于10个）的字符串需要拼接时，避免使用`+`，应该使用`StringBuffer`（线程安全）或者`StringBuilder`（非线程安全）的`append`方法进行拼接。或者直接使用`StringUtils`的`join`方法进行拼接。

6.1 数字处理

在进行字符串到数字类型的转换时，建议使用Apache Commons Langs中的`NumberUtils`辅助类，例如：

```
NumberUtils.toInt("123", 0);
```

判断字符串是否为数字时，可以考虑使用其中的：

```
NumberUtils.isDigits("123");
NumberUtils.isNumber("123L");
```

6.1 金额处理

表示金额时，不得使用原子类型，比如double、float、long。在系统中，必须使用Joda Money中的`Money`类来表示金额。

在自动生成的代码中（仅限DAL中），如果无法使用`Money`，则必须使用`BigDecimal`来表示金额。且使用范围局限在DAL中，必须将整个Entity转换成业务的对象后在外层使用。

在数据库中存储金额时，以分为单位。

6.2 日期操作

在系统中表示日期时间时，需要使用`java.util.Date`，而不是`java.sql.Date`！

在进行日期操作时，建议采用Joda Time。比如，格式化时间对象时：

- 不得使用`SimpleDateFormat`，该类不是线程安全的
- 建议使用Joda Time的`DateTimeFormatter`辅助类。

```
    DateTimeFormatter fmt = DateTimeFormat.forPattern("yyyyMMdd");
    DateTime dt = fmt.parseDateTime(strInputDateTime);
```

6.3 资源操作

所有的资源操作，在完成后必须按顺序关闭资源，并且注意捕获异常（即使是在`finally`段中）！

如果不关注关闭资源时的异常，可以使用Apache Commons IO中的`IOUtils`的`closeQuietly`方法，例如：

```
    IOUtils.closeQuietly(someFileReader);
```

6.4 异常处理

捕获到的异常，必须按如下格式输出到错误日志中：

```
    logger.error("相关描述", e);
```

不得使用

```
    logger.error("相关描述" + e);
```

这样会丢失异常堆栈。

对于已知的可接受的业务异常，可以不用ERROR级别输出，使用WARN级别。详见日志规范。

6.5 超时处理

数据库操作和远程调用都要设置合理的超时时间。

6.5.1 数据库超时

数据库的超时均设置在连接池配置中，使用Druid连接池连接MySQL数据库时需要注意的超时，包括但不局限于：

1. `connectTimeout`，建立Socket连接的超时，默认0无超时
2. `socketTimeout`，网络Socket操作的超时，默认0无超时
3. `queryTimeout`，执行SQL语句的超时
4. `maxWait`，从连接池获取连接的超时

要注意`socketTimeout`和`queryTimeout`的设置，前者必须大于后者，不然后者设置无效。

6.5.2 远程调用超时

以REST风格的远程调用为例，同样需要设置超时：

1. `connectTimeout`，建立Socket连接的超时，默认0无超时
2. `readTimeout`，从Socket读取数据的超时，默认0无超时

在服务端如果使用了Nginx来做负载，需要注意`proxy_read_timeout`的设置，后端的操作时间较长时，避免Nginx提前超时，而后端仍在执行的情况。

6.6 参数处理

禁止在方法内对参数进行赋值，对于引用类型的参数，在方法内用`=`改变引用，对方法外部是没有影响的。

在方法中，起码是`public`的方法中，建议在开头就对关键参数进行校验，比如是否为`null`。如果使用Spring Framework，可以考虑用`Assert`辅助类进行断言判断。

6.7 Maven配置

多模块Maven项目中仅能在顶层的DependencyMangement中指定依赖Artifact的版本号，子模块中不得添加version版本号。

7. 日志规范

详见日志规范文档。

8. 修订记录

2013-12-17: Version 0.2

> 移除日志规范部分，增加修订大量内容

2013-10-24: Version 0.1

> 撰写初稿