* 1. 文件组织cs

一个文件由被空行分割而成的段落以及标识每个段落的可选注释共同组成。超过3000行的程序难以阅读，应该尽量避免。而方法长度应控制在500行以内。

一个Java 源文件应该包含如下的元素，并按照以下顺序书写：

* 版本信息和版权声明
* 包的声明
* 引用声明
* 类或接口的声明
  1. 文件encoding

默认统一将源文件的encoding设置为UTF-8。

* 1. 起始注释tm

每一个源文件应该以一个包含版权声明和版本信息的注释块为起始。

示例如下：

/\*\*

\* Copyright *®*2011 GuoHuaLife Co. Ltd.<br/> 版权

\* All rights reserved.<br/>版权所有

\* Package: ${enclosing\_package}<br/>

\* FileName: ${file}<br/>

\* Description: 系统名称-子系统名称-组件名称<br/>描述

\*/

* 1. 类和接口注释cs/tm

类和接口注释通常放在类和接口定义的前面。通常介绍其功能，作者，版本和修改履历等信息。

示例如下：

/\*\*

\*【类或接口功能描述】

\*

\* @author 创建人

\* @date 创建日期

\* @version 1.0

\*

\*【修改描述】

\* @author 修改人

\* @date 修改日期

\*/

* 1. 方法注释cs/tm

方法注释的内容有方法功能描述，参数说明，返回类型说明等信息。

示例如下：

/\*\*

\* 方法功能描述

\*

\* @param 参数名称 参数说明

\* @return 返回类型说明

\*/

public User getUserById(int piId){

…

return oUser;

}

* 1. 属性注释cs

所有成员变量都需要加注属性描述注释。

示例如下：

/\*\*

\* 属性描述

\*/

private int iUserId;

* 1. 方法内注释

原则上在代码行上加注行注释，如加注不便时，可以使用右侧加注。

示例如下：

public User getUserById(int piId){

//调用UserDao查询用户

User oUser = this.userDAO.getById(

piId, // 用户ID

sName // 用户名称

);

//返回用户实体

return oUser；

}

* 1. 修改注释tm

原则上针对以下几种情况请使用修改注释：

* 严重bug 的修改
* 版本相对稳定后代码的修改
* 修改别人的代码

示例如下：

**A．添加**

// add by 修改人名称 修改日期yyyy-mm-dd begin

// 修改功能描述

......

具体添加的程序代码

......

// add by 修改人名称 修改日期yyyy-mm-dd end

**B．修改**

// modify by 修改人名称 修改日期yyyy-mm-dd begin

// 修改功能描述

/\*

......

修改的原始程序代码

......

\*/

......

具体修改的程序代码

......

// modify by 修改人名称 修改日期yyyy-mm-dd end

**C．删除**

// del by 修改人名称 修改日期yyyy-mm-dd begin

//删除功能描述

/\*

......

具体删除的程序代码

......

\*/

// del by 修改人名称 修改日期yyyy-mm-dd end

* 1. SVN提交注释

团队协作开发代码提交时，请使用SVN提交注释：

**需求号、BUG号或运维号：xxxx**

**提交人：xxxx**

**提交原因：xxxx**

* 1. 包的命名

包的名字应该都是由小写单词构成，使用规则如下：

<com.guohualife> + <.子系统> + [.组件]

* 1. 类的命名 首字母大写

类的名字必须由大写字母开头而其他字母都小写的单词连接组成，只用字母，不要用数字、下划线等符号，用字母大小写来分割各单词。cs

**规则如下：**

<名称>+<功能(类别)>

DTO类必须以DTO作为后缀 实体类

Form类必须以Form作为后缀

Controller类必须以Controller作为后缀

DAO类必须以DAO作为后缀

Service类必须以Service作为后缀

DAO和Service的实现类必须以Impl作为后缀

**示例如下：**

LoginDTO、LoginForm、LoginController、LoginServiceImpl、LoginDAO等

说明：以下123是自查小知识，与规范无关

1、entity里的每一个字段，与数据库相对应，

2、dto里的每一个字段，是和你前台页面相对应，

3、VO，这是用来转换从entity到dto，或者从dto到entity的中间的东西。

* 1. 变量的命名

变量的命名采用大小写混合的形式。以小写字母开头，名字中其他单词或者只取首字母的缩写单词以大写字母开头，所有其它的单词都为小写字母，不要使用下划线分隔单词。名字应为有意义的名词或者名词短语。

注：变量命名规则自动生成代码除外、POJO类属性除外。

**后缀命名规则如下：**

**Array**

数组用途描述 + <Arr>

**List**

用途描述 + <Lst>

**Hashtable、xxxMap**

用途描述 + <Map>

**xxxSet**

用途描述 + <Set>

* 1. 常量的命名cs

常量的命名都为大写字母，并用下划线分隔单词。

**示例如下：**

MIN\_VALUE，MAX\_BUFFER\_SIZE，OPTION\_FIEL\_NAME等

* 1. 方法的命名cs

方法命名采用大小写混合的形式。以小写字母开头，名字中其他单词或者只取首字母的缩写单词以大写字母开头，所有其它的单词都为小写字母，不要使用下划线分隔单词。方法的命名应该能描绘出方法的作用和功能，方法的名字建议使用祈使动词或者动词短语。

获取设置类属性的get/set方法，应该统一使用getProperty()/setProperty()，其中property 是指类属性的名字。

判断类属性的get/set方法，应该统一使用isProperty()/setProperty()，其中property 是指类属性的名字。

**示例如下：**

showStatus()、getName()/setName()、isVisible()/setVisible()等

* 1. 包的引用cs

import 语句应该从第一列开始，并用一个空格分隔import 关键字和import type name。引用时应保证所有引入类都被使用，并避免\*的引入，因为如果引入不确切的类将很难理解当前类的上下文关系及相关性。

**示例如下：**

import com.guohualife.platform.web.bookview // 推荐

import com.guohualife.plateform.\* // 不推荐

包的引入中标准的包名要在本地的包名之前，而且按照字母顺序排列。

**示例如下：**

import java.io.\*;

import java.util.\*;

import com.guohualife.plateform...; // 自定义的包放在后面

注：Eclipse中可以使用 【Ctrl】 + 【Shift】+ 【o】 来自动排序。

* 1. 属性/方法修饰符

要求统一书写顺序，在同一行内写完。cs

**规则如下：**

[synchronized]<[<public|private|protected>

[static][final]]|[abstract]><变量类型>{<变量名>|<方法名>…}

如果方法需要抛出异常，”throws Exception”应该写在下一行，并且必须缩进 8个空白。

**示例如下：**

public String setName(String psValue)

throws Exception

{

…

}

* 1. 空行

空行的使用有益于将代码按照逻辑分段，以提高代码的可读性。在下列情况下建议使用一个空行：

* 在版权声明块、包声明块、引用声明块之后
* 在类的声明之间
* 在方法的声明之间
* 在类中声明最后一个属性之后，声明第一个方法之前
* 在方法内部主要逻辑块之间

**示例如下：**

package com.guohualife.view;

<-空行

import java.util.Date;

<-空行

public class BookView

{

private String sBookID;

private String sBookName;

private String sBookAuthor;

<-空行

public BookView()

{

sBbookID = “0001”;

sBookName = “Time Machine”;

sBookAuthor = “Richard”;

}

public void book()

{

逻辑块1……

<-空行

逻辑块2……

<-空行

逻辑块3……

}

}

* 1. 空格

使用半角空格符，不要使用TAB符。以下情况下建议插入空格符：

* 在一个关键字和它所修饰的块之间。可用于如下的关键字：catch，for，if，switch，synchronized，while 。下列关键字的后面请不要跟随空格符：super ，this
* 在任何关键字和它所用的参数之间。例如：return true；
* 在赋值号，比较符号的前后。例如：

i = 5;// 推荐

i=5;// 不推荐

i >= 5;// 推荐

i>=5;// 不推荐

* 在一个列表中逗号之后。例如：

foo(long\_expression1, long\_expression2, long\_expression3);// 推荐

foo(long\_expression1,long\_expression2,long\_expression3);// 不推荐

* 注释行起始位置。例如：

//注释 <- 不推荐

// 注释 <- 推荐

* 1. 缩进

缩进应该是每行4个空格. 不要在源文件中保存Tab字符. 因为在使用不同的源代码管理工具时 Tab 字符将因为用户设置的不同而扩展为不同的宽度。

* 1. 页宽cs

页宽应该设置为 120 字符. 源代码一般不会超过这个宽度, 并导致无法完整显示, 但这一设置也可以灵活调整. 在任何情况下, 超长的语句应该在一个逗号或者一个操作符后折行. 一条语句折行后, 应该比原来的语句再缩进 4 个字符。

* 1. 括号

左右括号与前后字符之间不应该出现空格。

**示例如下：**

CallProc( parameter ); // 不推荐

CallProc(parameter); // 推荐

不要在语句中使用无意义的括号. 括号只应为达到某种目的而出现在代码中。

**示例如下：**

if ((i) == 42) {... // 不推荐 – 括号毫无意义

条件中不论各运算符的优先级, 必须强制加上括号来保证逻辑运算的次序

if ((i == 42)||(j == 42)){... // 推荐

if (i == 42||j == 42){... // 不推荐

* 1. 赋值和表达式

每行只包含一条语句。

**示例如下：**

a = b + c; // 推荐

count++; // 推荐

a = b + c; count++; // 不推荐

* 1. 局部变量声明

局部变量的声明应在不同的行中。

**示例如下：**

int j = 4; // 推荐

int i, j = 4, k; // 不推荐

* 1. 数组声明

数组的[]应该紧跟类型名，而不是数组名。在“[”之前不能有空格。

**示例如下：**

char buf[]; // 不推荐

char[] buf; // 推荐

* 1. return语句

return 语句后面的返回值不可以用括号包起来，除非是后面所跟为一个表达式。

在方法内部原则上最多出现三个return语句，否则将使程序的可读性变差。cs

**示例如下：**

return (true); // 不推荐

return true; // 推荐

return (s.length() + s.offset); // 推荐

* 1. if语句tm

if语句原则上最多允许嵌套三层，过多的嵌套将导致程序可读性变差。cs

**示例如下：**

if (condition) {

statements;

}

// if注释行

if (condition) {

statements;

}

// else if注释行

else If (condition) {

statements;

}

// else注释行

else {

statements;

}

* 1. for语句tm

**示例如下：**

for (initialization; condition; update) {

statements;

}

* 1. while语句tm

**示例如下：**

while (condition) {

statements;

}

如使用无限循环，那么建议使用：

while (true) {

statements;

}

* 1. do-while语句tm

**示例如下：**

do {

statements;

} while (condition);

* 1. switch语句tm

**示例如下：**

switch (condition) {

case 1:

case 2:

statements;

break;

default:

statements;

break;

}

* 1. try语句tm

try语句原则上只允许嵌套一层，过多的嵌套将导致程序可读性变差。cs

**示例如下：**

try {

statements;

}

// 注释行

catch (exception-declaration) {

statements;

}

try {

statements;

}

// 注释行

finally {

statements;

}

try {

statements;

}

// 注释行

catch ( exception-declaration) {

statements;

}

// 注释行

finally {

statements;

}

* 1. synchronized语句tm

**示例如下：**

synchronized (expression) {

statements;

}

最佳实践

* 1. final

不要因为性能的原因将类定义为 final 的（除非程序的框架要求），如果一个类还没有准备好被继承，最好在类文档中注明，而不要将它定义为 final 的。这是因为没有人可以保证会不会由于什么原因需要继承它。

* 1. 调试

调试在软件开发中是一个很重要的部分，存在于软件生命周期的各个部分中。请统一使用logger.debug()方法输出调试信息，具体请参考“日志编码规范”章节。不要使用System.out.pringln输出调试信息。cs

* 1. 可移植性

不要破坏程序的移植性，在一开始就注意与平台相关的细节，如路径的写法，文件名的大小写(以 unix 环境为标准，例如路径：/opt/data)。

* 1. 使用 StringBuffer对象

在处理 String 的时候要尽量使用 StringBuffer 类，StringBuffer 类是构成 String 类的基础。String 类将 StringBuffer 类封装了起来，（以花费更多时间为代价）为开发人员提供了一个安全的接口。当我们在构造字符串的时候，我们应该用 StringBuffer 来实现大部分的工作，当工作完成后将 StringBuffer 对象再转换为需要的 String 对象。

比如：如果有一个字符串必须不断地在其后添加许多字符来完成构造，那么我们应该使用 StringBuffer 对象和它的 append() 方法。如果我们用 String 对象代替 StringBuffer 对象的话，会花费许多不必要的创建和释放对象的 CPU 时间。

* 1. synchronized 关键字

谨慎使用不必要的关键字 synchronized，这是一个避免死锁的好方法。

* 1. 异常的捕获

如需异常捕获处理，应尽量在方法中只使用一个try 配合一个或多个catch块捕获异常，并尽量将代码都放到try块中。应尽量不要在循环中使用try catch。

* 1. @SuppressWarning抑制警告

谨慎使用不必要的注解@SuppressWarning，以免一些必要信息被隐含，原则上只允许在DAO层次内部使用压制警告。

* 1. DTO的使用

DTO（数据传输对象）用于将前台页面与后台逻辑间数据的传输，请不要使用自动生成的Model对象，也不允许继承自动生成Model对象定义DTO对象，以降低层间耦合性。

* 1. goto语句的使用

谨慎使用goto语句，除非在很深的嵌套层次进行跳转，否则将破坏程序的可读性。

日志编码规范

* 1. 基本原则

日志是项目开发中重要的细节，良好的日志记录方式可以加快开发速度、提高测试时Bug 查找的效率，并且在测试后维护阶段容易跟踪程序的状态。其实施基本原则如下：

* 机能模块的输入/输出应该进行日志输出，例如：文件、网络、数据库，用户等I/O端
* 对于重要机能模块的修改，要进行日志输出，记录修改前和修改后的状态信息
* 可恢复的业务级错误要进行日志输出
* 不可恢复的系统级错误要进行日志输出
  1. 实施要点
* 在机能模块初始化运行时，要进行日志输出记录开始运行的信息
  + LOG级别：INFO
* 在机能模块进行初始化或结束时
  + 如果出现不影响机能运行的错误出现时，要进行日志输出记录出现的错误
    - LOG级别：WARN
  + 如果出现影响机能运行的错误出现时，要进行日志输出记录出现的错误
    - LOG级别：ERROR
* 机能模块初始化结束时，要记录初始化完成的信息
  + LOG级别：INFO
* 在方法的入口，如有参数验证的情况，则需进行日志输出相应参数的信息
  + LOG级别：DEBUG
  + 如果参数验证不正确，要进行日志输出相应信息
    - LOG级别：WARN
* 在调用重要的API时，如果失败要进行日志输出相应错误信息
  + LOG级别：ERROR
* 在调用没有经过严格测试的API时，要进行日志输出记录返回的结果
  + LOG级别：DEBUG
* 在捕获异常时，要进行日志输出相应错误信息
  + LOG级别：ERROR
* 在数据库操作进行时，要进行日志输出执行的SQL语句，对于返回的结果，最好输出完整的对象信息
  + LOG级别：DEBUG
* 在与其它外围系统进行通信时，要进行日志记录下通信过程中的重要信息
  + LOG级别：DEBUG
* 对于多线程运行的程序，它们的初始化与完成也要进行日志输出记录
  + LOG级别：INFO
* 在需要检查运行性能的地方，要进行日志输出记录运行耗时
  + LOG级别：DEBUG

**实施一览表**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 初始化  操作 | 参数验证 | API调用 | 捕获异常 | 数据库  存取 | 外围通信 | 多线程 | 性能验证 |
| DEBUG |  | ○ | ○ |  | ○ | ○ |  | ○ |
| INFO | ○ |  |  |  |  |  | ○ |  |
| WARN | ○ | ○ |  |  |  |  |  |  |
| ERROR | ○ |  | ○ | ○ |  |  |  |  |

* 1. 格式

DEBUGtm

**格式如下：**

if (logger.isDebugEnabled()) {

logger.debug(“方法名(参数列表) – 变量类型 变量名=“ + 变量);

}

\*注：按照编码最佳实践，字符串的连接可考虑使用StringBuffer构成。

**示例如下：**

if (logger.isDebugEnabled()) {

logger.debug(“selectByName(String) – String name=“ + name);

}

INFOtm

**格式如下：**

logger.info(“方法名(参数列表) – 日志信息");

**示例如下：**

logger.info(“getPerson(String) – This is a test info");

WARNtm

**格式如下：**

logger.warn(“方法名(参数列表) – 日志信息”);

**示例如下：**

if(name.isBlank()) {

xxx

}

else {

logger.warn("configure(File) – count error");

xxx

}

ERRORtm

**格式如下：**

logger.error(“方法名(参数列表) – 日志信息”);

logger.error(“方法名(参数列表) “, 异常对象);

**示例如下：**

try{

xxx

}

catch(IOException e){

logger.error("configure(File)", e);

xxx

}

数据库事务处理规范

* 1. 事务配置

<tx:attributes>  
             <tx:method name="save\*" propagation="REQUIRED"/>  
             <tx:method name="delete\*" propagation="REQUIRED"/>  
             <tx:method name="update\*" propagation="REQUIRED"/>  
             <tx:method name="find\*" read-only="true"/>  
             <tx:method name="get\*" read-only="true"/>  
             <tx:method name="select\*" read-only="true"/>  
             <tx:method name="\*" propagation="REQUIRED" />  
         </tx:attributes>

* 1. 处理说明
* 在service层 save、delete、update作为前缀的方法会开启事务管理
* 在service层 find、get、select作为前缀的方法会声明为read only
* 在service层 qry、query、que、load作为前缀的方法会声明为read only

为了避免开发人员方法名并非以save、delete、update为前缀但是需要事务管理时的失误，其它名称前缀的方法都将开启事务管理