|  |
| --- |
| Microsoft |
| 先知蓝创编码规范 |
| 编码规范第一版 |
|  |
| **gycx** |
| **2017/4/17** |

|  |
| --- |
| 为了今后的项目中编码更清晰的编写与维护，特编写此编码规范，具体细节随时间更新完善 |

目录

[1 编程风格 2](#_Toc480379721)

[1.1 统一编程风格的意义 2](#_Toc480379722)

[1.2 命名的规则 2](#_Toc480379723)

[1.3 注释规范 2](#_Toc480379724)

[1.4 规范要求示例 3](#_Toc480379725)

[2 代码优化 3](#_Toc480379726)

[2.1 代码优化的意义 4](#_Toc480379727)

[2.2 代码优化的方法 4](#_Toc480379728)

[3 细节要求 4](#_Toc480379729)

[3.1 关于Controller 4](#_Toc480379730)

[3.2 关于Mapping/Mapper 5](#_Toc480379731)

[3.3 前台表单验证要求使用spirngMVC内置标签 5](#_Toc480379732)

[3.4 Service方法命名 5](#_Toc480379733)

# 编程风格

## 统一编程风格的意义

* 增加开发过程代码的强壮性、可读性、易维护性
* 减少有经验和无经验开发人员编程所需的脑力工作 
* 为软件的良好维护性打下好的基础 
* 在项目范围内统一代码风格 
* 通过人为以及自动的方式对最终软件应用质量标准 
* 使新的开发人员快速适应项目氛围 
* 支持项目资源的复用：允许开发人员从一个项目区域（或子项目团队）移动到另一个，而不需要重新适应新的子项目团队的氛围
* 一个优秀而且职业化的开发团队所必需的素质

## 命名的规则

1. 驼峰式命名
2. 要达到“见名知义”
3. 类名首字母须大写
4. 属性名和方法名首字母小写，之后遵循第1条和第2条
5. Controller中前台方法加后缀 .do，例：login.do；后天方法加后缀 .emp，例：login.emp

## 注释规范

1. 类的注释

从“功能”、“作者”、“时间”三个方面用如下格式注释：

/\*\*

\* 用户登录controller

\* Create by gycx 2017/4/17.

\*/

1. 方法的注释

从“功能”、“参数”、“返回值”三个方面用如下格式注释：

/\*\*

\* 输出字符name

\* **@param** name

\* **@param** age

\* **@return**

\*/

**public** String demo1(String name,**int** age){

System.***out***.println(name);

**return** "规范演示";

}

1. 变量的注释

/\*\* 用户账号 \*/

**private** String userAccount;

## 规范要求示例

**package** com.easy.emp.bean;

**public** **class** XianzhiDemo {

/\*\* 用户账号 \*/

**private** String userAccount;

**private** **int** age;

/\*\*

\* 方法的命名、参数名和变量名要做到见名知义并采用驼峰式命名

\*/

**public** **void** showCodeDemo(String name,**int** age){

//方法体内的代码块要缩进一个制表符（tab）

**if**(0!=age&&**null**!=name||""!=name){//为避免出现空指针异常，异常的判断，把已知的值或常量值放在表达式左侧

name = "规范演示";

age = 1+1;

}

}//方法体的结束右括号要与方法开始的第一行在同一制表位

/\*\*

\* 输出字符name

\* **@param** name

\* **@param** age

\* **@return**

\*/

**public** String demo1(String name,**int** age){

System.***out***.println(name);

**return** "规范演示";

}

}

# 代码优化

要求方法内的代码不得超过100行，如果超过，须进行代码拆分。

拆分代码的2种思路：

第一种：按照编程思路拆分

第二种：按照业务逻辑拆分

## 代码优化的意义

* 仅仅对符合功能说明书的要求、能正确运行的代码进行优化是有意义的 
* 代码优化能减少冗余代码的数量，用更少的代码来实现同样的功能 
* 提高代码的内聚程度，减少耦合程度 对代码的抽象能提高代码的重用度，对今后其他项目的进度有非常重要的意义

## 代码优化的方法

1. 选择一个更好的算法

应该熟悉算法语言,知道各种算法的优缺点,一般很多计算机资料文本上有介绍,应该能够看得懂算法描述。

还要选择一种合适的数据结构(记着,你的程序所干的唯一一件事就是在计算机里搬数,把一堆数从一个地方提出来,处理一下,甩到另一个地方,那么按什么方式搬数有多重要你应该知道了),比如你在一堆随机存放的数中使用了大量的插入和删除指令,那使用链表要快得多.如果你要做二分法查找,那提前排下序非常重要。

1. 写一些清晰,可读性好并且简单的代码。
2. 透视程序

一个程序写出来,凭直觉就应该感觉出哪些地方快,哪些地方慢,一般说来,最快的运算就是分配一块内存,给指针赋值,还有就是两个整数的加法运算,别的都有点慢,最慢的就要数打开文件,打开新的进程,读写一大块内存,寻找,排序等等。

# 细节要求

## 关于Controller

编写代码时，尽可能的使用SpringMVC提供的注解达到解耦合的目的，比如session的取值与赋值。

|  |  |  |  |
| --- | --- | --- | --- |
| 关键字 | 说明 | 参数 | 返回值 |
| SessionAttributes | SessionAttributes({"userLogin"})获取session中的userLogin，使用此注解后，可在controller中直接使用userLogin。 |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 关键字 | 说明 | 参数 | 返回值 |
| ModelAttribute | ModelAttribute("userLogin")User userLogin 获取session中的userLogin并赋值给User。 |  |  |

## 关于Mapping/Mapper

因前台判断默认将空值视为0 所以Int型数据的值不要使用0来表示含义

## 前台表单验证要求使用spirngMVC内置标签

<sf:form class="test" action="test" modelAttribute="test" id="saveForm">

</sf:form>

## Service方法命名

事务性的方法：执行后将会使数据库中的数据发生变化 如：增、删、改

要对应使用以下形式作为方法名的开头：

<tx:method name="save\*" propagation="REQUIRED"/>

<tx:method name="delete\*" propagation="REQUIRED" />

<tx:method name="update\*" propagation="REQUIRED" />

<tx:method name="insert\*" propagation="REQUIRED" />

例：增加一个用户的方法

InsertUser（）；

非事务性的方法：如查询方法使用以下形式作为方法名的开头：

<tx:method name="find\*" propagation="SUPPORTS" read-only="true" />

<tx:method name="get\*" propagation="SUPPORTS" read-only="true" />

<tx:method name="select\*" propagation="SUPPORTS" read-only="true" />

例：通过ID查询用户

getUserById（）；

findUserById（）；

selectUserById（）；