**专业方向综合项目 编程规范**

**张尧 陈东 姜其升**

**TEam 12**

**MEETING！**

基于微信小程序的协同日程安排应用

**目录**

[第一部分 Java 2](#_Toc44760333)

[变量和类命名 2](#_Toc44760334)

[方法命名 3](#_Toc44760335)

[获取get方法 3](#_Toc44760336)

[设置set方法 4](#_Toc44760337)

[普通注释 4](#_Toc44760338)

[类注释 5](#_Toc44760339)

[大括号 6](#_Toc44760340)

[非空判断 6](#_Toc44760341)

[用StringBuffer代替String 6](#_Toc44760342)

[尽量减少对变量的重复计算 7](#_Toc44760343)

[采用懒加载的策略 7](#_Toc44760344)

[移位操作代替 7](#_Toc44760345)

[循环创建对象引用外提 8](#_Toc44760346)

[数值和字符串转换 8](#_Toc44760347)

[时间计算 9](#_Toc44760348)

[数值转字符串 10](#_Toc44760349)

[第二部分 数据库 11](#_Toc44760350)

[命名 11](#_Toc44760351)

[数据库不做复杂操作 11](#_Toc44760352)

[第三部分 微信小程序 13](#_Toc44760353)

[WXML规范 13](#_Toc44760354)

[注释规范 13](#_Toc44760355)

[CSS规范 13](#_Toc44760356)

[注释规范 14](#_Toc44760357)

[JS规范 15](#_Toc44760358)

[命名规范 15](#_Toc44760359)

[回调函数规范 15](#_Toc44760360)

[数据绑定变量定义规范 16](#_Toc44760361)

[点击事件规范 17](#_Toc44760362)

[组件规范 17](#_Toc44760363)

[组件名命名规范 17](#_Toc44760364)

[触发事件规范 17](#_Toc44760365)

[externalClasses命名规范 17](#_Toc44760366)

[组件样式规范 17](#_Toc44760367)

[标点规范 18](#_Toc44760368)

[界面设计规范 18](#_Toc44760369)

[交互规范 19](#_Toc44760370)

第一部分 Java

# 变量和类命名

**变量**

命名时应始终采用完整的英文描述符。此外，一般应采用小写字母，但类名、接口名以及任何非初始单词的第一个字母要大写。

使用可以准确说明变量/字段/类的完整的英文描述符。例如，采用类似 firstName，grandTotal 或 CorporateCustomer 这样的名字。虽然象 x1，y1 或 fn 这样的名字很简短，输入起来容易，但是我们难以知道它们代表什么、结果是什么含义，因而使代码以理解、维护和改进。

采用该领域的术语。如果用户称他们的“客户” (clients) 为“顾客” mers)，那么就采用术语 Customer 来命名这个类，而不用 Client。许多程序开发者会犯的一个错误是，不去使用工业或领域里已经存在着很完美的术语时，却生造出一些普通词汇。

采用大小写混合，提高名字的可读性。一般应该采用小写字母，但是类和接口的名字的首字母，以及任何中间单词的首字母应该大写。

尽量少用缩写，但如果需要使用，可以选择单词的头4个字母作为缩写。如：DeptInfoValue，MapActi, MapActiInst 等等。

避免使用长名字（最好不超过 15 个字母）。 虽然 PhysicalOrVirtualProductOrService 看起来似乎是个不错的类名，但是这个名字太长了，应该考虑重新给它起个短一点的名字，比如象 Offering。

避免使用相似或者仅在大小写上有区别的名字。例如，不应同时使用变量名 persistentObject 和 persistentObjects，以及 anSqlDatabase 和 anSQLDatabase。

避免使用下划线作为名字的首末字母。以下划线为首末字母的名字通常为系统保留，除预处理定义之外，一般不用作用户命名。更重要的是，下划线经常造成麻烦而且难输入，所以尽量避免使用。

**类**

标准 Java类约定是使用完全的英文描述符，所有单词的第一个字母要大写，并且单词中大小写混合。类名应是单数形式。

示例：

Customer Employee Order OrderItem FileStream String

相关的类如果使用了某些已经约定的开发模式，应该使用相关约定的命名：如：LoggerFactory , UserDAO 等，DAO是我们数据库映射实体的类。

在同一个模块中的相关类，可以采用同样开头的名称以区分他们的相关性。如：UserValue, UserDAO 等。

# 方法命名

**成员方法**

成员函数的命名应采用完整的英文描述符，大小写混合使用：所有中间单词的第一个字母大写。成员函数名称的第一个单词常常采用一个有强烈动作色彩的动词。

示例：

openAccount() printMailingLabel() createUser() delete()

这种约定常常使人一看到成员函数的名称就能判断它的功能。虽然这种约定要使开发者多做一些输入的工作，因为函数名常常较长，但是回报是提高代码的可理解性。

# 获取get方法

获取函数作为一个成员函数，返回一个字段的值。除了布尔字段之外，应采用 get 作为字段的前缀；布尔字段采用 is 作为前缀。

示例：

public String getUserCode() {

return userCode;

}

public String getUserName() {

return userName;

}

getFirstName()

getAccountNumber()

布尔字段采用 is 作为前缀：

isPersistent()

isAtEnd()

遵循这个命名约定，显然，成员函数将返回对象的字段，布尔型的获取函数将返回布尔值“真”或者“假”。这个标准的另一个优点是：它遵循 beans development kit (BDK) 对获取成员函数采用的命名约定 [DES97]。它的一个主要的缺点是 get 是多余的，需要额外的录入工作。

# 设置set方法

设置函数，也叫变值函数，是可以修改一个字段值的成员函数。无论何种字段类型，都要在字段名的前面加上 set 前缀。

示例：

public void setUserCode(String userCode) {

this.userCode = userCode;

}

public void setUserName(String userName) {

this.userName = userName;

}

setFirstName(String name)

setAccountNumber(int accountNumber)

setReasonableGoals(Vector goals)

setPersistent(boolean isPersistent)

setAtEnd(boolean isAtEnd)

按照这种命名约定，显然是一个成员函数设定一个对象的字段值。这个标准的另一个优点是：它遵循 beans development kit (BDK) 对设置函数采用的命名约定 [DES97]。 它的一个主要的缺点是 set 是多余的，需要额外的录入。

# 普通注释

|  |  |  |
| --- | --- | --- |
| 注释语句类型 | 用法 | 示例 |
| 文档注释 | 在接口、类、成员函数和字段声明之前紧靠它们的位置用文档注释进行说明。文档注释由 javadoc 处理，为一个类生成外部注释文档，如下所示。 | /\*\*  User类时表示我们的用户信息的类  @author CD  \*/ |
| C语言风格注释 | 采用 C 语言风格的注释语句将无用的代码注释掉。保留这些代码是因为用户可能改变想法，或者只是想在调试中暂时不执行这些代码。 | /\*  如果这个xxx条件成立  \*/ |
| 单行注释 | 在成员函数内部采用单行注释语句对业务逻辑、代码片段和临时变量声明进行说明。 | //如果这个xxx条件成立  // 否则xxxx |

# 类注释

以下的信息应写在文档注释中紧靠类的定义的前面：

**类的目的和作用**。开发者需要了解一个类的一般目的，以判断这个类是否满足他们的需求。养成一个注释与类有关的任何好东西的习惯。

已知的问题。如果一个类有任何突出的问题，应说明出来，让其他的开发者了解这个类的缺点/难点。此外，还应注明为什么不解决问题的原因。

**类的开发/维护历史**。通常要包含一个历史记录表，列出日期、类的作者和修改概要。这样做的目的是让进行维护的程序员了解过去曾对一个类所做的修改，是谁做了什么样的修改。

版权信息。

对于以上几点：其中类的目的和作用这项是必须要填写的。采用Javadoc形式标志的一个示例如下：

/\*\*

\* <p>类说明: Event 事件类</p>

\* @author CD

\* @version 1.0

\* @since 2020-06-10

\* @modified fjy 2020-06-13 修改了XXXX

\* @modified fjy 2020-06-23 修改了XXXX

\*/

# 大括号

不管有几行，在分支或者循环体都要大括号。

# 非空判断

错误例子：

if(user.getUserName().equals("hollis")){

}

　　这段代码极有可能在实际运行的时候跑出NullPointerException。无论是user本身为空，还是user.getUserName()为空，都会抛出异常。 所以，在调用一个参数时要确保他是非空的。

上面的代码可以改为：

if(user!=null&&"hollis".equals(user.getUserName())){

}

# 用StringBuffer代替String

　在循环中构建一个String对象时从性能上讲使用StringBuffer来代替String对象 例如：

String s = "";

for (int i = 0; i < field.length; ++i) {

s = s + field[i]；

}

应该改为StringBuffer，使用append方法：

StringBuffer buf = new StringBuffer();

for (int i = 0; i < field.length; ++i) {

buf.append(field[i]);

}

String s = buf.toString();

# **尽量减少对变量的重复计算**

明确一个概念，对方法的调用，即使方法中只有一句语句，也是有消耗的，包括创建栈帧、调用方法时保护现场、调用方法完毕时恢复现场等。所以例如下面的操作：

for (int i = 0; i < list.size(); i++){

　　...

}

建议替换为：

for (int i = 0, length = list.size(); i < length; i++){

　　...

}

# 采用懒加载的策略

String str = "aaa";

if (i == 1)

{

　　list.add(str);

}

建议替换为：

if (i == 1)

{

　　String str = "aaa";

　　list.add(str);

}

# 移位操作代替

用移位操作可以极大地提高性能，因为在计算机底层，对位的操作是最方便、最快的，因此建议修改为：

for (val = 0; val < 100000; val += 5)

{

　　a = val << 3;

　　b = val >> 1;

}

相当于\*8和/2，但是由于可读性变差，需要明确注释信息

# 循环创建对象引用外提

for (int i = 1; i <= count; i++)

{

Object obj = new Object();

}

这种做法会导致内存中有count份Object对象引用存在，count很大的话，就耗费内存了，建议为改为：

Object obj = null;

for (int i = 0; i <= count; i++)

{

obj = new Object();

}

这样的话，内存中只有一份Object对象引用，每次new Object()的时候，Object对象引用指向不同的Object罢了，但是内存中只有一份，这样就大大节省了内存空间了。

# 数值和字符串转换

基本数据类型.toString()是最快的方式、String.valueOf(数据)次之、数据+”"最慢

把一个基本数据类型转为一般有三种方式，我有一个Integer型数据i，可以使用i.toString()、String.valueOf(i)、i+”"三种方式，三种方式的效率如何，看一个测试：

复制代码

public static void main(String[] args)

{

int loopTime = 50000;

Integer i = 0;

long startTime = System.currentTimeMillis();

for (int j = 0; j < loopTime; j++)

{

String str = String.valueOf(i);

}

System.out.println("String.valueOf()：" + (System.currentTimeMillis() - startTime) + "ms");

startTime = System.currentTimeMillis();

for (int j = 0; j < loopTime; j++)

{

String str = i.toString();

}

System.out.println("Integer.toString()：" + (System.currentTimeMillis() - startTime) + "ms");

startTime = System.currentTimeMillis();

for (int j = 0; j < loopTime; j++)

{

String str = i + "";

}

System.out.println("i + ""：" + (System.currentTimeMillis() - startTime) + "ms");

}

复制代码

运行结果为：

String.valueOf()：11ms

Integer.toString()：5ms

i + "" ：25ms

所以以后遇到把一个基本数据类型转为String的时候，优先考虑使用toString()方法。至于为什么，很简单：

　　String.valueOf()方法底层调用了Integer.toString()方法，但是会在调用前做空判断

　　Integer.toString()方法就不说了，直接调用了

　　i + “”底层使用了StringBuilder实现，先用append方法拼接，再用toString()方法获取字符串

三者对比下来，明显是2最快、1次之、3最慢。

# 时间计算

计算两段代码时间差，很多人是采用以下这种方式。

long startTime = System.currentTimeMillis();

// 执行代码

long endTime = System.currentTimeMillis();

System.out.println(endTime - startTime);



我们来看另外一种方式。

这样可以避免因为系统不同导致的不一致。

long startTime = System.nanoTime();

// 执行代码

long endTime = System.nanoTime();

System.out.println(endTime - startTime);

# 数值转字符串

当要把其它对象或类型转化为字符串时，使用 String.valueOf(value) 比""+value 的效率更高。

反例：

int i = 1;

String s = "" + i;

正例：

int i = 1;

String s = String.valueOf(i);

第二部分 数据库

# 命名

**数据库命名**

采用26个英文字母(区分大小写)和0-9的自然数(经常不需要)加上下划线'\_'组成;

命名简洁明确(长度不能超过30个字符);

例如：user, stat, log, 也可以wifi\_user, wifi\_stat, wifi\_log给数据库加个前缀;

除非是备份数据库可以加0-9的自然数：user\_db\_20151210;

表**命名**

采用26个英文字母(区分大小写)和0-9的自然数(经常不需要)加上下划线'\_'组成;

命名简洁明确,多个单词用下划线'\_'分隔;

例如：user\_login, user\_profile, user\_detail, user\_role, user\_role\_relation,

user\_role\_right, user\_role\_right\_relation

表前缀'user\_'可以有效的把相同关系的表显示在一起;

**字段命名**

采用26个英文字母(区分大小写)和0-9的自然数(经常不需要)加上下划线'\_'组成;

命名简洁明确,多个单词用下划线'\_'分隔;

例如：user\_login表字段 user\_id, user\_name, pass\_word, eamil, tickit, status, mobile, add\_time;

每个表中必须有自增主键,add\_time(默认系统时间)

表与表之间的相关联字段名称要求尽可能的相同;

**字段类型命名**

用尽量少的存储空间来存数一个字段的数据;

例如：能使用int就不要使用varchar、char,能用varchar(16)就不要使用varchar(256);

IP地址最好使用int类型;

固定长度的类型最好使用char,例如：邮编;

能使用tinyint就不要使用smallint,int;

最好给每个字段一个默认值,最好不能为null;

索引

命名简洁明确,例如：user\_login表user\_name字段的索引应为user\_name\_index唯一索引;

为每个表创建一个主键索引;

为每个表创建合理的索引;

建立复合索引请慎重;

# 数据库不做复杂操作

不在数据库做运算，也就是不用SQL执行复杂的运算，或者复杂的查询; 计算务必移至Java业务层，复杂的多表查询也可以在业务层分批查出来Java中筛选连接。

第三部分 微信小程序

# WXML规范

wxml标签可以单独出现的情况，尽量单独出现，如<input />。

<input />

控制每行HTML的代码数量在50个字符以内，方便阅读浏览，多余的代码进行换行处理，标签所带属性每个属性间进行换行。

自动检测

<c-music

wx:if="{{classic.type===200}}"

img="{{classic.img}}"

content="{{classic.content}}"

>

</c-music>

合理展现分离内容，不要使用内联样式。

自动检测

//推荐使用

<image class="tag"></image>

## 注释规范

除组件外的其他块级元素，均需注释出其功能，并在其上下空出一行与其他代码进行区分。

自动检测

<view>...</view>

//导航栏

<view>...</view>

<view>...</view>

# CSS规范

在开发过程中rpx和px均可能用到，如通常情况下间距使用rpx，字体大小和边框等使用px，开发者根据实际情况而定。

自动检测

width: 100rpx;

font-size: 14px;

CSS代码需有明显的代码缩进。每一个样式类之间空出一行。

自动检测

.c-tag{

width: 100%;

}

.c-container{

width: 100%;

}

尽量使用简写属性，并且同一属性放置在一起，避免散乱。

自动检测

/\*\*使用简写属性\*\*/

.c-image{

margin: 0 auto;

}

/\*\*同一属性放在一块\*\*/

.c-tag{

margin-left: 10rpx;

margin-right: 10rpx

}

采用flex进行布局，禁止使用float以及vertical-align。

自动检测

.container{

disaplay: flex;

flex-dirextion: row

}

## 注释规范

成组的wxss规则之间用块状注释。请勿在代码后面直接注释。

自动检测

/\*\* 修改button默认的点击态样式类\*\*/

.button-hover {

background-color: red;

}

# JS规范

## 命名规范

变量名以及函数名统一采用驼峰命名法，正常情况下函数名前缀需加上清晰的动词表示函数功能，私有函数或者属性以下划线开头表明。常量需用const 声明。

类的命名首字母需大写。

采用ES6 关键字let定义变量，尽量不使用var

自动检测

//定义常量

const a = 1

//定义变量

let imageContent = res.data

//函数命名

getInfo:function(){

return '';

}

//私有函数

\_getInfo:function(){

return '';

}

## 回调函数规范

回调函数统一使用Promise函数的方式进行编写，回调成功的参数统一为res，错误参数为err。

自动检测

// promise 处理回调

let back = new Promise((resolve, reject) => {

if (/\* 异步操作成功 \*/){

resolve(value);

} else {

reject(error);

}

});

back.then((res) => {

console.log('成功回调！', res);

}).catch((err) => {

console.log('失败回调！', error);

});

私有函数以及回调函数统一放置在生命周期函数后。

删除js文件中未用到的生命周期函数，保持代码的整洁。

自动检测

Pages({

data:{

},

onLoad:function(event){

},

\_self:function(){

}

})

每个函数之间用一个空行分离结构。

## 数据绑定变量定义规范

所有涉及到数据绑定的变量均需在data中初始化。禁止在不定义的情况下直接setData。

自动检测

Pages({

data:{

id : null

},

onLoad:function(event){

let id = event.target.dataset.id

this.data.id = id

}

})

## 点击事件规范

点击事件函数命名方式为 on + 事件名 或者业务名。

自动检测

onLike: function(event){

}

# 组件规范

## 组件名命名规范

组件在使用时命名以 “c-”为开头的组件名，若组件名称为多个单词名拼接而成，采用 ' - ' 连接。

**c来源于单词 ‘**components**’**

自动检测

<c-movies />

## 触发事件规范

组件点击触发事件建议用冒号分隔开

自动检测

<c-component-tag-name bind:myevent="onMyEvent" />

## externalClasses命名规范

命名格式采用如下形式：c-class-{name}，name可自行定义

c-class-icon

## 组件样式规范

命名必须以 c- 开头，不允许使用内联样式以及id样式

自动检测

.c-container{

disaplay: flex;

flex-dirextion: row

}

## 标点规范

JS语句需以分号结束

JS中一致使用单引号' ' , 不使用双引号。

WXML、CSS、JSON中均应使用双引号。

CSS属性中冒号中后面用一个空格分隔开。

执行一致的缩进（4个空格）

# 界面设计规范

小程序的“所有页面”右上角位置，都固定放置了小程序的菜单，在设计界面时需预留出该区域空间。



**小程序菜单固定样式**

微信提供了深浅两种配色样式，以便更好的融合到各种风格的页面中，需注意保持小程序菜单清晰的辨识度。

功能导航栏目

微信有提供小程序的底部标签样式，建议标签数量在2-4个适宜。



加载样式规范

全局加载是小程序名称左侧的加载图标。模态的加载样式将覆盖整个页面的，由于无法明确告知具体加载的位置或内容将可能引起用户的焦虑感，建议谨慎使用。



# 交互规范

友好礼貌

为了避免用户在微信中使用小程序服务时，注意力被周围复杂环境干扰，小程序在设计时应该注意减少无关的设计元素对用户目标的干扰，礼貌地向用户展示程序提供的服务，友好地引导用户进行操作。

重点突出

每个页面都应有明确的重点，以便于用户每进入一个新页面的时候都能快速地理解页面内容，在确定了重点的前提下，应尽量避免页面上出现其他干扰项影响用户的决策和操作。

清晰明确

一旦用户进入我们的小程序页面，我们就有责任和义务清晰明确地告知用户身在何处、又可以往何处去，确保用户在页面中游刃有余地穿梭而不迷路，这样才能为用户提供安全的愉悦的使用体验。