# 软件编码规范说明书

# 教室借用系统

**版本：v1.0**

**团队：我们不一样**

**日期：2018.05.23**

目录

[引言 3](#_Toc5199)

[第一章Java代码规范 4](#_Toc18984)

[1 标识符命名规范 4](#_Toc21047)

[2 代码格式 9](#_Toc27677)

[3 注释规范 12](#_Toc30021)

[4 最佳实践和禁忌 14](#_Toc5849)

[第二章 前端编码规范 20](#_Toc8089)

[规范目的 20](#_Toc995)

[规范基本准则 20](#_Toc31014)

[1 文件规范 20](#_Toc9709)

[2 注释规范 22](#_Toc24855)

[3 命名规范 24](#_Toc30089)

[4书写规范 25](#_Toc1809)

[5 其他规范 28](#_Toc17357)

[6 自适应页面布局 29](#_Toc24525)

[7 帮助文档 29](#_Toc25249)

[第三章 数据库编码规范 31](#_Toc11614)

[1 规范目的、适用范围 31](#_Toc3178)

[2 SQL 语句可读性 32](#_Toc18090)

[3 注释 32](#_Toc4651)

[4 命名规范 33](#_Toc32666)

[5 异常处理 34](#_Toc19308)

[6 安全编码 35](#_Toc24519)

# 

# 引言

一、规范的代码可以促进团队合作。一个项目大多都是由一个团队来完成，如果没有统一的代码规范，那么每个人的代码必定会风格迥异。且不说会存在多个人同时开发同一模块的情况，即使是分工十分明晰的，等到要整合代码的时候也有够头疼的了。大多数情况下，并非程序中有复杂的算法或是复杂的逻辑，而是去读别人的代码实在是一件痛苦的事情。统一的风格使得代码可读性大大提高了，人们看到任何一段代码都会觉得异常熟悉。显然的，规范的代码在团队的合作开发中是非常有益而且必要的。

二、规范的代码可以减少bug处理。很多IT人士将程序员比做民工，这也的确非常的形象。就像刚才提到的，复杂的算法或逻辑只占项目中很小的比例，大多仅仅是垒代码的工作。可是越是简单，测试的bug反而是越多，而且是无穷无尽的bug。这里很大的程度上是由于代码不规范所致。 没有规范的对输入输出参数的规范，没有规范的异常处理，没有规范的日志处理等等，不但导致了我们总是出现类似空指针这样低级的bug而且还很难找到引起bug的原因。相反，在规范的开发中，bug不但可以有效减少，查找bug也变得轻而易举。 规范不是对开发的制约，而确实是有助于提高开发效率的。

三、规范的代码可以降低维护成本。随着我们项目经验的累积，会越来越重视后期维护的成本。而开发过程中的代码质量直接影响着维护的成本。因此，我们不得不从开发时便小心翼翼。 在第一点中曾提到，规范的代码大大提高了程序的可读性，几乎所有的程序员都曾做过维护的工作，不用多说，可读性高的代码维护成本必然会大大降低。 但是，维护工作不仅仅是读懂原有代码，而是需要在原有代码基础上作出修改。我们可以先想像没有统一风格的情况下，A完成开发以后，B进行维护加一段代码，过一段时间C又加一段代码。直到有一天X看到那一大堆乱码想死的心都有了，维护也就进行不下去了。因此，统一的风格有利于长期的维护。另外，好的代码规范会对方法的度量、类的度量以及程序耦合性作出约束。这样不会出现需要修改一个上千行的方法或者去扩展一个没有接口的类的情况。规范的代码对程序的扩展性提高，无疑也是对维护人员的一个奖励。

四、规范的代码有助于代码审查。我个人是比较赞同进行代码审查的，这样可以及时纠正一些错误，而且可以对开发人员的代码规范作出监督。团队的代码审查同时也是一个很好的学习机会，对成员的进步也是很有益的。但是，开发随意，加重的代码审查的工作量及难度，并且使得代码审查工作没有根据，浪费了大量的时间却收效甚微。代码规范不仅使得开发统一，减少审查拿督，而且让代码审查有据可查，大大提高了审查效率和效果，同时代码审查也有助于代码规范的实施。一举多得，何乐而不为呢。

五、养成代码规范的习惯，有助于程序员自身的成长 即使明白代码规范的好处，但是有的迫于项目压力，有的因为繁琐的规范作出很多额外的工作，更有的不重视维护的问题，而很难贯彻代码规范。那么，我们需要了解，规范开发最大的受益人其实是自己！你有没有花费很多的时候查找自己的代码呢？尤其是出现bug的时候需要逐行的debug？自己写的代码乱了头绪的确实也见了不少。我们应该做的就是规范开发，减少自己出现的错误。很多时候项目的压力一部分也是由于前期开发中遗留的众多的问题。还有的人觉得自己可以完成高难度的算法，就认为自己能力很强，不把规范放在眼里。很多人确实是这样，追求个性，大概让别人看他的代码一头雾水更觉得得意。殊不知复杂的算法确实可以体现你个人的逻辑能力，但是绝不代表你的开发水平。我们知道一些开源项目，一些大师级人物写得程序都是极其规范的。并非规范了就代表高水平，实际上是规范的代码更有利于帮助你理解开发语言理解模式理解架构，能够帮助你快速提升开发水平。要达到高水平的程序员，养成良好的开发习惯是绝对必需的。不要沉迷表面的得失，看似无用的东西要经过慢慢的累积由量变达到质变的时候，你才能感受到其价值所在。制定一个符合自己公司情况的开发规范是很简单的，重要的是我们能够认识到规范的重要性，并坚持规范的开发习惯。

# 第一章Java代码规范

为什么要有编码规范？

编码规范对于程序员而言尤为重要，有以下几个原因：一个软件的生命周期中，80%的花费在于维护,几乎没有任何一个软件，在其整个生命周期中，均由最初的开发人员来维护；编码规范可以改善软件的可读性，可以让程序员尽快而彻底地理解新的代码；如果你将源码作为产品发布，就需要确任它是否被很好的打包并且清晰无误，一如你已构建的其它任何产品,为了执行规范，每个软件开发人员必须一致遵守编码规范。

### 1. 标识符命名规范

#### 1.1 概述

标识符的命名力求做到统一、达意和简洁。

#### 1.1.1 统一

统一是指，对于同一个概念，在程序中用同一种表示方法，比如对于供应商，既可以用supplier，也可以用provider，但是我们只能选定一个使用，至少在一个Java项目中保持统一。统一是作为重要的，如果对同一概念有不同的表示方法，会使代码混乱难以理解。即使不能取得好的名称，但是只要统一，阅读起来也不会太困难，因为阅读者只要理解一次。1.1.2 达意

达意是指，标识符能准确的表达出它所代表的意义，比如： newSupplier, OrderPaymentGatewayService等；而 supplier1, service2，idtts等则不是好的命名方式。准确有两成含义，一是正确，而是丰富。如果给一个代表供应商的变量起名是 order，显然没有正确表达。同样的，supplier1, 远没有targetSupplier意义丰富。

#### 1.1.3 简洁

简洁是指，在统一和达意的前提下，用尽量少的标识符。如果不能达意，宁愿不要简洁。比如：theOrderNameOfTheTargetSupplierWhichIsTransfered 太长， transferedTargetSupplierOrderName则较好，但是transTgtSplOrdNm就不好了。省略元音的缩写方式不要使用，我们的英语往往还没有好到看得懂奇怪的缩写。

#### 1.1.4 骆驼法则

Java中，除了包名，静态常量等特殊情况，大部分情况下标识符使用骆驼法则，即单词之间不使用特殊符号分割，而是通过首字母大写来分割。比如: SupplierName, addNewContract，而不是 supplier\_name, add\_new\_contract。

#### 1.1.5 英文 vs 拼音

尽量使用通俗易懂的英文单词，如果不会可以向队友求助，实在不行则使用汉语拼音，避免拼音与英文混用。比如表示归档，用archive比较好, 用pigeonhole则不好，用guiDang尚可接受。

### 1.2 包名

使用小写字母如 com.xxx.settlment，不要 com.xxx.Settlement。单词间不要用字符隔开，比如 com.xxx.settlment.jsfutil，而不要com.xxx.settlement.jsf\_util。

### 1.3 类名

#### 1.3.1 首字母大写

类名要首字母大写，比如 SupplierService, PaymentOrderAction；不要 supplierService, paymentOrderAction.

#### 1.3.2 后缀

类名往往用不同的后缀表达额外的意思，如下表：

|  |  |  |
| --- | --- | --- |
| 后缀名 | 意义 | 举例 |
| Service | 表明这个类是个服务类，里面包含了给其他类提同业务服务的方法 | PaymentOrderService |
| Impl | 这个类是一个实现类，而不是接口 | PaymentOrderServiceImpl |
| Inter | 这个类是一个接口 | LifeCycleInter |
| Dao | 这个类封装了数据访问方法 | PaymentOrderDao |
| Action | 直接处理页面请求，管理页面逻辑了类 | UpdateOrderListAction |
| Listener | 响应某种事件的类 | PaymentSuccessListener |
| Event | 这个类代表了某种事件 | PaymentSuccessEvent |
| Servlet | 一个Servlet | PaymentCallbackServlet |
| Factory | 生成某种对象工厂的类 | PaymentOrderFactory |
| Adapter | 用来连接某种以前不被支持的对象的类 | DatabaseLogAdapter |
| Job | 某种按时间运行的任务 | PaymentOrderCancelJob |
| Wrapper | 这是一个包装类，为了给某个类提供没有的能力 | SelectableOrderListWrapper |
| Bean | 这是一个POJO | MenuStateBean |

### 1.4 方法名

首字母小写，如 addOrder() 不要 AddOrder()  
动词在前，如 addOrder()，不要orderAdd()  
动词前缀往往表达特定的含义，如下表：

|  |  |  |
| --- | --- | --- |
| 前缀名 | 意义 | 举例 |
| create | 创建 | createOrder() |
| delete | 删除 | deleteOrder() |
| add | 创建，暗示新创建的对象属于某个集合 | addPaidOrder() |
| remove | 删除 | removeOrder() |
| init或则initialize | 初始化，暗示会做些诸如获取资源等特殊动作 | initializeObjectPool |
| destroy | 销毁，暗示会做些诸如释放资源的特殊动作 | destroyObjectPool |
| open | 打开 | openConnection() |
| close | 关闭 | closeConnection()< |
| read | 读取 | readUserName() |
| write | 写入 | writeUserName() |
| get | 获得 | getName() |
| set | 设置 | setName() |
| prepare | 准备 | prepareOrderList() |
| copy | 复制 | copyCustomerList() |
| modity | 修改 | modifyActualTotalAmount() |
| calculate | 数值计算 | calculateCommission() |
| do | 执行某个过程或流程 | doOrderCancelJob() |
| dispatch | 判断程序流程转向 | dispatchUserRequest() |
| start | 开始 | startOrderProcessing() |
| stop | 结束 | stopOrderProcessing() |
| send | 发送某个消息或事件 | sendOrderPaidMessage() |
| receive | 接受消息或时间 | receiveOrderPaidMessgae() |
| respond | 响应用户动作 | responseOrderListItemClicked() |
| find | 查找对象 | findNewSupplier() |
| update | 更新对象 | updateCommission() |

find方法在业务层尽量表达业务含义，比如 findUnsettledOrders()，查询未结算订单，而不要findOrdersByStatus()。 数据访问层，find,update等方法可以表达要执行的sql，比如findByStatusAndSupplierIdOrderByName(Status.PAID, 345)

### 1.5 域（field）名

#### 1.5.1 静态常量

全大写用下划线分割，如  
public static find String ORDER\_PAID\_EVENT = "ORDER\_PAID\_EVENT";

#### 1.5.2 枚举

全大写，用下划线分割，如  
public enum Events {  
ORDER\_PAID,  
ORDER\_CREATED  
}

#### 1.5.3 其他

首字母小写，骆驼法则，如：  
public String orderName;

### 1.6 局部变量名

参数和局部变量名首字母小写，骆驼法则。尽量不要和域冲突，尽量表达这个变量在方法中的意义。

### 2 代码格式

用空格字符缩进源代码，不要用tab，每个缩进4个空格。

#### 2.1 源文件编码

源文件使用utf-8编码，结尾用unix n 分格。

#### 2.2 行宽

行宽度不要超过130。

#### 2.3 包的导入

删除不用的导入，尽量不要使用整个包的导入。在eclipse下经常使用快捷键 ctrl+shift+o 修正导入。

#### 2.4 类格式

#### 2.5 域格式

每行只能声明一个域。域的声明用空行隔开。

#### 2.5 方法格式

#### 2.6 代码块格式

#### 2.6.1 缩进风格

大括号的开始在代码块开始的行尾，闭合在和代码块同一缩进的行首，例如：  
  
package com.test;

public class TestStyle extends SomeClass implements AppleInter, BananaInter {  
public static final String THIS\_IS\_CONST = "CONST VALUE";

private static void main(String[] args) {  
int localVariable = 0;  
}

public void compute(String arg) {  
if (arg.length() > 0) {  
System.out.println(arg);  
}

for (int i = 0; i < 10; i++) {  
System.out.println(arg);  
}

while (condition) {

}

do {  
otherMethod();  
} while (condition);

switch (i) {  
case 0:  
callFunction();  
break;  
case 1:  
callFunctionb();  
break;  
default:  
break;  
}  
}  
}

#### 2.6.2 空格的使用

###### **1） 表示分割时用一个空格**

不能这样：  
  
if ( a > b ) {  
//do something here  
};

###### **2）二元三元运算符两边用一个空格隔开**

如下：  
  
a + b = c;  
b - d = e;  
return a == b ? 1 : 0;

不能如下：

a+b=c;  
b-d=e;  
return a==b?1:0;

###### **3) 逗号语句后如不还行，紧跟一个空格**

如下：  
  
call(a, b, c);  
  
不能如下：  
  
call(a,b,c);

#### 2.6.3 空行的使用

空行可以表达代码在语义上的分割，注释的作用范围，等等。将类似操作，或一组操作放在一起不用空行隔开，而用空行隔开不同组的代码， 如图：  
  
order = orderDao.findOrderById(id);

//update properties  
order.setUserName(userName);  
order.setPrice(456);  
order.setStatus(PAID);

orderService.updateTotalAmount(order);

session.saveOrUpdate(order);  
  
上例中的空行，使注释的作用域很明显.

1. 连续两行的空行代表更大的语义分割。
2. 方法之间用空行分割
3. 域之间用空行分割
4. 超过十行的代码如果还不用空行分割，就会增加阅读困难

### 3 注释规范

#### 3.1注释 vs 代码

1. 注释宜少二精，不宜多而滥，更不能误导
2. 命名达意，结构清晰， 类和方法等责任明确，往往不需要，或者只需要很少注释，就可以让人读懂；相反，代码混乱，再多的注释都不能弥补。所以，应当先在代码本身下功夫。
3. 不能正确表达代码意义的注释，只会损害代码的可读性。
4. 过于详细的注释，对显而易见的代码添加的注释，罗嗦的注释，还不如不写。
5. 注释要和代码同步，过多的注释会成为开发的负担。
6. 注释不是用来管理代码版本的，如果有代码不要了，直接删除，svn会有记录的，不要注释掉，否则以后没人知道那段注释掉的代码该不该删除。

#### 3.2 Java Doc

表明类、域和方法等的意义和用法等的注释，要以javadoc的方式来写。Java Doc是个类的使用者来看的，主要介绍 是什么，怎么用等信息。凡是类的使用者需要知道，都要用Java Doc 来写。非Java Doc的注释，往往是个代码的维护者看的，着重告述读者为什么这样写，如何修改，注意什么问题等。 如下：  
  
/\*\*  
\* This is a class comment  
\*/  
public class TestClass {  
/\*\*  
\* This is a field comment  
\*/  
public String name;

/\*\*  
\* This is a method comment  
\*/  
public void call() {

}  
}

### 3.3 块级别注释

#### 3.3.1 块级别注释

单行时用 //, 多行时用 /\* .. \*/。

#### 3.3.2 较短的代码块用空行表示注释作用域

#### 3.3.3 较长的代码块要用 /\*------ start: ------\*/ 和 /\*-------- end: -------\*/ 包围 如： /\*----------start: 订单处理 ------- \*/ //取得dao OrderDao dao = Factory.getDao("OrderDao");

/\* 查询订单 \*/  
Order order = dao.findById(456);

//更新订单  
order.setUserName("uu");  
order.setPassword("pass");  
order.setPrice("ddd");

orderDao.save(order);  
/\*----------end: 订单处理 ------- \*/

#### 3.3.4 可以考虑使用大括号来表示注释范围

使用大括号表示注释作用范围的例子：  
  
/\*----------订单处理 ------- \*/  
{  
//取得dao  
OrderDao dao = Factory.getDao("OrderDao");

/\* 查询订单 \*/  
Order order = dao.findById(456);

//更新订单  
order.setUserName("uu");  
order.setPassword("pass");  
order.setPrice("ddd");

orderDao.save(order);  
}

### 3.4 行内注释

行内注释用 // 写在行尾

### 4 最佳实践和禁忌

1、每次保存的时候，都让你的代码是最美的。程序员都是懒惰的，不要想着等我完成了功能，再来优化代码的格式和结构，等真的把功能完成，很少有人会再愿意回头调整代码。

2、 使用log而不是System.out.println()，log可以设定级别，可以控制输出到哪里，容易区分是在代码的什么地方打印的，而System.out.print则不行。而且，System.out.print的速度很慢。所以，除非是有意的，否则，都要用log。至少在提交到svn之前把System.out.print换成log。

3、每个if while for等语句，都不要省略大括号{}

看下面的代码：  
  
if (a > b)  
a++;  
  
如果在以后维护的时候，需要在a > b 时，把b++，一步小心就会写成：  
  
if (a > b)  
a++;  
b++;  
  
这样就错了，因为无论a和b是什么关系，b++都会执行。 如果一开始就这样写：  
  
if (a > b) {  
a++;  
}  
  
相信没有哪个笨蛋会把b++添加错的。而且，这个大括号使作用范围更明显，尤其是后面那行很长要折行时。

4、 善用TODO:

在代码中加入 //TODO: ，大部分的ide都会帮你提示，让你知道你还有什么事没有做。比如：  
  
if (order.isPaid()) {  
//TODO: 更新订单  
}

5、在需要留空的地方放一个空语句或注释

比如：  
  
if (!exists(order)) {  
;  
}

或：

if (!exists(order)) {  
//nothing to do  
}

6、不要再对boolean值做true false判断

比如：  
  
if (order.isPaid() == true) {  
// Do something here  
}  
  
不如写成：  
  
if (order.isPaid()) {  
//Do something here  
}  
  
后者读起来就很是 if order is paid, .... 要比 if order's isPaid method returns true, … 更容易理解

7 、减少代码嵌套层次

代码嵌套层次达3层以上时，一般人理解起来都会困难。下面的代码是一个简单的例子：  
  
public void demo(int a, int b, int c) {  
if (a > b) {  
if (b > c) {  
doJobA();  
} else if (b < c) {  
doJobB()  
}  
} else {  
if (b > c) {  
if (a < c) {  
doJobC();  
}  
}  
}  
}  
  
减少嵌套的方法有很多：

合并条件

利用 return 以省略后面的else

利用子方法

比如上例，合并条件后成为：  
  
public void demo(int a, int b, int c) {  
if (a > b && b > c) {  
doJobA();  
}  
if (a > b && c > b) {  
doJobB();  
}  
if (a <= b && c < b && a < c) {  
doJobC();  
}  
}  
  
如果利用return 则成为：  
  
public void demo(int a, int b, int c) {  
if (a > b) {  
if (b > c) {  
doJobA();  
return;  
}  
doJobB()  
return;  
}

if (b > c) {  
if (a < c) {  
doJobC();  
}  
}  
}  
  
利用子方法，就是将嵌套的程序提取出来放到另外的方法里。

8、程序职责单一

关注点分离是软件开发的真理。人类自所以能够完成复杂的工作，就是因为人类能够将工作分解到较小级别的任务上，在做每个任务时关注更少的东西。让程序单元的职责单一，可以使你在编写这段程序时关注更少的东西，从而降低难度，减少出错。

9、变量的声明

初始化和被使用尽量放到一起。

比方说如下代码：  
  
int orderNum= getOrderNum();

//do something withou orderNum here

call(orderNum);

上例中的注释处代表了一段和orderNum不相关的代码。orderNum的声明和初始化离被使用的地方相隔了很多行的代码，这样做不好，不如这样：  
  
//do something withou orderNum here

int orderNum= getOrderNum();  
call(orderNum);

10、缩小变量的作用域

能用局部变量的，不要使用实例变量，能用实例变量的，不要使用类变量。变量的生存期越短，以为着它被误用的机会越小，同一时刻程序员要关注的变量的状态越少。实例变量和类变量默认都不是线程安全的，局部变量是线程安全的。比如如下代码：  
  
public class OrderPayAction{  
private Order order;

public void doAction() {  
order = orderDao.findOrder();  
doJob1();  
doJob2();  
}

private void doJob1() {  
doSomething(order);  
}

private void doJob2() {  
doOtherThing(order);  
}  
}

上例中order只不过担当了在方法间传递参数之用，用下面的方法更好：  
  
public class OrderPayAction{

public void doAction() {  
order = orderDao.findOrder();  
doJob1(order);  
doJob2(order);  
}

private void doJob1(Order order) {  
doSomething(order);  
}

private void doJob2(Order order) {  
doOtherThing(order);  
}  
}

11、尽量不要用参数来带回方法运算结果

比如：  
  
public void calculate(Order order) {  
int result = 0;  
//do lots of computing and store it in the result

order.setResult(result);  
}

public void action() {  
order = orderDao.findOrder();  
calculate(order);

// do lots of things about order  
}

例子中calculate方法通过传入的order对象来存储结果， 不如如下写：  
  
public int calculate(Order order) {  
int result = 0;  
//do lots of computing and store it in the result

return result;  
}

public void action() {  
order = orderDao.findOrder();  
order.setResult(calculate(order));

// do lots of things about order  
}

# 第二章 前端编码规范

规范目的

为了提高工作效率，便于后台人员添加功能及前端后期优化维护，输出高质量的文档，在网站建设中，使结构更加清晰，代码简明有序，有一个更好的前端架构。

规范基本准则

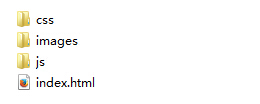
符合web标准，使用具有语义的标签，使结构、表现、行为分离，兼容性优良。页面性能优化，代码简洁、明了、有序，尽可能的减少服务器的负载，保证最快的解析速度。

### 1 文件规范

#### 1.1 HTML部分

　　1.1.1 建包问题

　　　　文件均归档至约定的目录中，建包格式如下：



　　　　注意：所有的css文件放在css文件夹中，image放在images文件夹中，js放在js文件夹中

　　1.1.2 HTML头部编写

　　　　(1) 编码：所有编码均采用xhtml/html，标签必须闭合，编码统一为UTF-8，在多语言的网站建议添加<html lang="zh-CN">，说明内容是以中文显示和阅读为基础的

　　　　(2) 语义化：正确使用标签，充分利用无兼容性问题的html自身标签

　　　　(3) 文件头部head内容：

　　　　　　• title: 需要添加标题

　　　　　　• 编码: charset=UTF-8

　　　　　　• meta: 可以添加description、keywords内容

#### 1.2 CSS部分

　　1.2.1 CSS种类及其命名

　　　　可以将CSS样式表分为三类：全局样式表、模块通用样式表和独立样式表

　　　　　　• 全局样式表常用命名：public.css

　　　　　　• 模块通用样式表命名：模块名\_basic.css

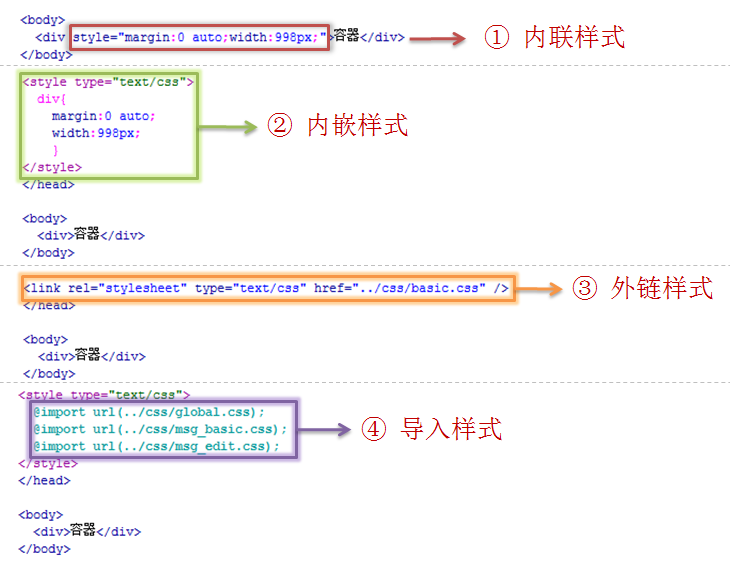
　　　　　　• 独立样式表：模块名\_页面名.css

　　1.2.2 CSS引入

　　　　CSS文件引入可通过外联或者内联方式引入

　　• 外联方式

　　• 内联方式



　　　　注意：link和style标签都应该放入head中，原则上，不允许在html上直接写样式。避免在CSS中使用@import，嵌套不要超过一层。

### 2 注释规范

2.1 顶部文档注释（推荐使用）

1 /\*

2 \* @description: 中文说明

3 \* @author: name

4 \* @update: name(xxxx-xx-xx xx:xx)

5 \*/

2.2 属性注释

1 /\*Header\*/

2 /\*Nav\*/

3 /\*Container\*/

4 ...

5 /\*Footer\*/

2.3 功能模块注释

1 /\* module: module1 by 小王 \*/

2 ...

3 /\* module: module2 by 小李 \*/

2.4 特殊注释

按 Ctrl+C 复制代码



按 Ctrl+C 复制代码



　注意：

1. 每一个文档对应一个文档注释（主要注释内容包括：文档创建人、创建时间、主要内容描述等）
2. 属性注释说明：可以分CSS属性来进行命名（如：margin/padding值、CSS Hack、全局Hover等）
3. 功能模块注释说明：分模块来编写CSS样式（如：头部、导航、按钮、页脚等等）

### 3 命名规范

#### 3.1 如何命名

1）css最好用class来命名，js用id来命名，已做区分

2）id和class的命名应反映该元素的功能或使用通用名称，而不要用抽象的晦涩的命名

#### 命名示例

1. div1{} /\* 不推荐；无意义 \*/
2. a\_green{} /\* 不推荐；无意义 \*/
3. menu{} /\* 推荐；特殊性\*/
4. header{} /\* 推荐；通用性\*/

#### 命名精简

id和class命名越精简越好，只要足够表达意思，这样有助于理解，同时也能提高代码效率

1. navigation{} /\* 不推荐 \*/
2. login\_box\_inside\_con{} /\* 不推荐 \*/
3. nav{} /\* 推荐 \*/

#### 3.4 命名嵌套问题

　　书写css要注意先后顺序和嵌套问题，从性能上考虑尽量减少选择器的层级

1. .nav ul.list{} /\* 不推荐 \*/
2. .nav .list{} /\* 推荐 \*/

#### 3.5 注意事项

1. 规则命名中，一律采用小写加下划线的方式
2. 命名中尽量避免使用中文拼音，应该采用更简明有语义的英文单词进行组合
3. 命名注意缩写，但是不能盲目缩写
4. 不允许通过1、2、3等序号进行命名
5. 避免class与id重名
6. id注意用于标识模块或页面的某一个父容器区域，名称必须唯一，不要随意新建id
7. class用于标识某一个类型的对象，命名必须言简意赅
8. 尽可能提高代码模块的复用，样式尽量用组合的方式
9. 规则名称中不应该包含颜色、定位等与具体显示效果相关的信息，应该用意义命名，而不是结果名称

### 4书写规范

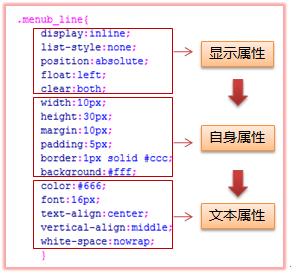
#### 4.1 排版规范

1. 使用4tab来缩进
2. 规则可以写成单行或者多行，但是整个文件内的规则排版必须统一
3. 书写风格每一个属性值必须添加分号
4. 如果多个属性公用一个样式集，则多个属性必须写成多行形式

4.2 属性编写顺序

（一般遵循显示属性 -> 自身属性 -> 文本属性 -> 其他属性的书写格式）

1. 显示属性：display/list-style/position/float/clear...
2. 自身属性（盒模型）：width/height/margin/padding/border
3. 背景：background
4. 行高：line-height
5. 文本属性：color/font/text-decoration/text-align/text-indent/vertical-align/white-space/content...
6. 其他：cursor/z-index/zoom/overflow...
7. CSS3属性：transform/transition/animation/box-shadow/border-radius
8. 如果使用CSS3的属性，如果有必要加入浏览器前缀，则按照-webkit-/-moz-/-ms-/-o-/std的顺序进行添加，标准属性写在最后
9. 链接的样式请严格按照如下顺序添加：a:link -> a:visited -> a:hover -> a:active



#### 4.3 代码性能优化

1. 合并margin、padding、border的-top/-right/-bottom/-left的设置，尽量使用短名称
2. 选择器应该在满足功能的基础上尽量简短，减少选择器嵌套，查询消耗。但是一定要避免覆盖全局样式设置
3. 禁止在css中使用\*选择符
4. 0后面不需要单独，比如0px可以省略成0，0.8px可以省略成.8px
5. 如果可以颜色尽量用三位字符表示，比如#ccc
6. 如果没有边框时，不要写成border:0;应该写成border:none
7. 在保存代码解耦的前提下，尽量合并重复的样式
8. background、font等可以缩写的属性，尽量使用缩写形式
9. 能以背景形式呈现的图片，尽量都写入CSS样式中

#### 4.4 CSS Hack的使用

　　尽量少使用浏览器检测和CSS Hacks，先试试别的解决办法。考虑到代码高效率和易管理，虽然这两种办法能快速解决浏览器解析差异，但应被视为最后的手段。在长期的项目中，允许使用hack只会带来更多的hack，所以尽量少用

1. IE6: \_property:value
2. IE6/7: \*property:value
3. IE6/7/8/9: property:value\9

#### 4.5 IE兼容性

　　IE支持通过特定的<meta>标签来确定绘制当前页面所应该采用的IE版本，除非有强烈的特殊需求，否则最好是设置edge mode，从而通知IE采用其所支持的最新的模式

　　　　<!-- IE8及以上的版本都会以最高版本IE来渲染页面 -->  
　　　　<meta http-equiv="X-UA-Compatible" content="IE=edge">

　　注意：X-UA-Compatible这个是IE8的专用标记，用来指定IE8浏览器去模拟某个特定版本的IE浏览器的渲染方式

　　　　   <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">中的chrome=1效果是如果安装了GCF，则使用GCF来渲染页面，如果未安装GCF，则使用最高版本的IE内核进行渲染

#### 4.6 字体规则

1. 为了防止文件合并及编码转换时造成问题，建议将样式中文字体名字改成对应的英文名字，如：黑体(SimHei)、宋体(SimSun)、微软雅黑(Microsoft Yahei)
2. 字体粗细采用具体数值，粗体bold写成700，正常normal写成400
3. font-size必须以px为单位
4. 为了对font-family取值进行统一，更好的支持各个操作系统上各个浏览器的兼容性，font-family不允许在业务代码中随意设置



### 5 其他规范

1. 不要轻易改动全站级CSS和通用CSS库，改动后，要经过全面测试
2. 避免使用filter
3. 避免在CSS中使用expression
4. 避免过小的背景图片平铺
5. 尽量不要在CSS中使用!important
6. 绝对不要在CSS中使用"\*"选择符
7. 层级(z-index)必须清晰明确，页面弹窗、气泡为最高级(最高级为999)，不同弹窗气泡之间可在三位数之间调整，普通区块为10-90内10的倍数；区块展开、弹出为当前父层级上个位增加，禁止层级间盲目攀比
8. 背景图片在情况允许，尽可能使用sprite技术，减小http请求，考虑到多人协作开发，sprite按照模块、业务、页面来划分
9. 页面内部尽量避免使用style属性，CSS放在head标签中，由link标签引入，使页面的结构与表现分离
10. 尽量减少使用float、position等影响性能的属性，这样可以避免新手在布局时出现的混乱
11. 尽量少使用<br />来断行
12. 不要连续出现多个 (空格)，也尽量少使用全角空格（英文字符集下，全角空格会变成乱码），空白应该尽量使用text-indent、maring/padding等方法来实现
13. 排版如果遇到需要首行缩进的处理，可以使用text-indent:2em;
14. 图片如果需要加载就在页面上用img标签写出，并指明宽高，重要的图片必须加上alt属性，给重要的元素和截断的元素上加上title
15. 如果有跳转的地方，统一使用a标签，<a href="#"></a>，需要跳转到新页面，则还需要加上targent="\_blank"属性，如果点击的是空链接(#)，则会自动将当前页面重置到首端，可以使用"javascript:void()"来替代原来的"#"
16. 分清楚什么情况下jpg/gif/png图片

6 自适应页面布局**（响应式布局，暂不考虑低版本IE兼容性）**

#### 6.1 布局细节

1. 首先头部head中加入meta新标签

　　　　<!-- 让viewport的宽度等于物理设备上的真实分辨率，不允许用户缩放，一般主流的web app都是这样设置 -->  
　　　　<meta name="viewport" content="width=device-width, initial-scale=1">

1. position：不能使用绝对定位
2. width/height/margin/padding: 不能使用px，应该使用百分比、auto或em
3. font: 不能使用绝对大小，应使用em

#### 6.2 CSS3引入的Media Query模块，可自动探测屏幕宽度

1. 加载相应的CSS文件，建议根据不同的屏幕分辨率，选择应用不同的CSS规则，如：@media screen and (max-width:799px) {...}
2. 图片自适应：img{max-width:100%;}

### 7 帮助文档

7.1 全局样式写法（reset css）



#### 7.2 清浮动

　　需要清除浮动的地方有：

1. 若子元素浮动，而父元素内容塌陷（也就是没有包住）
2. 布局出现混乱，譬如下一层的跑到上一层去了

　　解决办法（四种方法）

1. 给父元素同样适用浮动，保证子元素与父元素浮动后还是在同一层
2. 正确使用overflow:hidden;总所周知，overflow:hidden主要意思是溢出隐藏的意思，但是同样有清浮动的效果
3. 在需要的元素下面添加<div class="clear"></div>，CSS中：clear{clear:both;}（不推荐，增加代码，冗余）
4. 使用clearfix来清除浮动（推荐），相当于创建一个隐形的内容为空的块的目标元素来清除浮动

　　　　.clearfix{\*zoom:1;} /\* 针对IE7 hack，触发IE7的haslayout，以清除浮动 \*/  
　　　　.clearfix:before,.clearfix:after{display:table;content:"";line-height:0;}  
　　　　.clearfix:after{clear:both;}

#### 7.3 各大网站的字体样式：

1. Google: font-family:arial, sans-serif;
2. Yahoo: font:13px/1.25 "Helvetica Neue",Helvetica,Arial;
3. Apple: font:12px/18px "Lucida Grande", "Lucida Sans Unicode",Helvetica,Arial,Verdana,sans-serif;
4. Baidu: font:12px arial;
5. taobao: font:12px/1.5 tahoma,arial,'Hiragino Sans GB',\5b8b\4f53,sans-serif;
6. Weibo: font:12px/1.125 Arial,Helvetica,sans-serif;
7. Tencent: font:12px "宋体","Arial Narrow",HELVETICA;
8. Sina Weibo: font:12px/1.3 "Arial","Microsoft YaHei";
9. Sina: font:12px/20px "SimSun","宋体","Arial Narrow",HELVETICA;
10. JD: font:12px/150% Arial,Verdana,"宋体";
11. zhihu: font-family:'Helvetica Neue', Helvetiva,Arial,Sans-serif;

　　默认字体样式：

1. 兼容性好：font-family:Helvetica,Tahoma,Arial,Sans-serif;
2. 更接近设计：font-family:Geogia,"Times New Roman",Times,serif;

7.4 文本多行显示添加省略号（文本溢出省略）

　　给需要的标签添加具体宽度限制、white-space:nowrap;text-overflow:ellipsis;overflow:hidden;

#### 7.5 input框的具体样式

　　input框行高问题，一般不需要写line-height，但是在IE8/IE7需要写hack

# 第三章 数据库编码规范

### 1 规范目的、适用范围

规范的目的是保证代码的稳定性、可读性、一致性以及效率。规范涵盖SQL 语句编写和各种数据库对象的命名，适用于新创建或修改的数据库对象。

### 2 SQL 语句可读性

1、SQL 语句关键词全部用大写，除declare 和类型名称。

示例：下面的代码是正确的。

SELECT \* FROM surveyEntity

2、复杂的多表JOIN 查询语句，为阅读方便，对表使用别名。

3、较长的SQL（超过80 字必须换行），使用多行书写，JOIN、WHERE、括号、等结构性语句放在新

行之首。

4、对于存储过程、函数、视图和触发器中的批量SQL 语句：

使用缩进策略，缩进单位为4 个空格(注意不要使用TAB 键)；

程序块的分界符（BEGIN 和END）应各独占一行并且位于同一列，同时与引用它们的语句左对齐；

IF, BEGIN ,END,WHILE 等结构性语句自占一行

示例：下面的代码是正确的。

IF exists(SELECT \* FROM surveyEntity WHERE surveyid=@surveyid)

BEGIN

SELECT @status=status FROM surveyEntity WHERE surveyid=@surveyid

IF(@status=0) --还未发布

BEGIN

SET @@result=-1

RETURN

END

IF(@status=2) --已关闭

BEGIN

SET @@result=-2

RETURN

END

END

### 3 注释

#### 3.1 SQL 语句

在存储过程、函数、视图和触发器中的批量SQL 语句中，对关键代码、关键算法使用注释，注，释应与其描述的代码相近，对代码的注释应放在其上方或右方（对单条语句的注释）相邻位置，不可放在下面，如果放于上方则需与其上面的代码用空行隔开。变量如果其命名不是充分自注释的，或者特别重要的，在声明时都必须加以注释，分支语句（条件分支、循环语句等）除非逻辑特别简单，都要编写注释在代码交付前，应删除无关注释

#### 3.2 存储过程注释

存储过程定义开始之前必须加注释，注释中描述存储过程的功能，参数，作者，修改者，脚

本日期。

示例：下面的代码是正确的。

/\*

功能：新增一个用户

参数：@UserName 用户名

@@UserId 用户id

作者：james

创建时间：2007-11-15

\*/

CREATE PROCEDURE [dbo].[p\_User\_Add]

… …

#### 3.3 视图注释：

视图定义开始之前必须加注释，注释中描述视图程的功能，作者，脚本日期；

示例：下面的代码是正确的。

/\*

功能：查询所有用户

作者：james

创建时间：2018-11-15

\*/

CREATE View [dbo].[v\_VoteAllList]

… …

#### 3.4 函数注释：

示例：下面的代码是正确的。

/\*

功能：根据开始日期，有效周期计算有效的结束日期

参数:@start 开始日期, @period 有效的周期@isBefore 是否为有效期内的

作者：XX

创建时间：2018-11-15

\*/

CREATE FUNCTION [dbo].[getValidDate](@start datetime, @period int =0,

@isAfter bit = 1)

RETURNS datetime

AS

… …

#### 3.5 其他对象注释：

定义其他对象时，也要添加注释，标明功能，作者，创建时间。

### 4 命名规范

1）总则

命名使用有意义的英文名称，除数据库名称长度为1－15 个字符，其余为1－30 个字符，Database link名称也不要超过30 个字符，避免使用关键字，如type,name,id 等，使用Pascal 命名方法。

2）数据库名

规则：项目的英文名称

示例：BitAuto,UcarMis

3）表名

规则：同一项目下的表名，用模块名缩写(或者Schema 名称)+实际表名(一律单数)

注意：表名不要加下划线。

示例： CarBrand ,CarEngine

4）存储过程

规则：p\_表名\_动作名

示例：p\_Student\_Insert

5）视图

规则：v\_单表名+查询类型

v\_有意义的名称或者v\_单表名\_单表名\_...

示例：v\_CarBrandList v\_CarBrandBeijing

6）索引

规则：ix\_表名\_字段名；

多字段：ix\_表名\_字段名1\_字段名2

复合索引可适当简写

7）字段命名

规则：

1. 主键：表名或表名缩写+ID

2. 其他字段：Pascal 式命名，即每个单词的首字母均大写

3. 字符类型：中文(nvarchar)

4. 注意事项：列名不要使用关键字，比如id，type 等。

无须在字段名中加类型标识作为前后缀，如*intMonth* 是不允许的。

字段不要加下划线。

8）主，外键

规则：pk\_表名\_主键标识，如pk\_Vendor\_id

fk\_表名\_字段名\_主键标识，如fk\_VendorCarList\_Vendor\_id

9）触发器

tr\_表名\_i,u,d 的任意组合

10）函数

func\_功能名称

11）规则

Default：df\_标识名

Rule： ru\_标识名

### 5 异常处理

在SQL2005 中使用TRY…CATCH 捕获错误，进行处理。

举例：

BEGIN TRY

SELECT \* FROM sys.messages WHERE message\_id = 21

END TRY

BEGIN CATCH

SELECT ERROR\_NUMBER() AS ErrorNumber

END CATCH

### 6 安全编码

1. 数据库用户

数据库用户权限使用最小化原则，严禁使用sa 用户，使用系统维护人员分配的用户。

2. 防SQL 注入

网站用户输入的字符串，在传入业务层之前要进行安全操作：针对字符串要进行防SQL 注入过滤；如果是数值类型，要把输入的字符串转换成数值。如果使用SqlParameter 传递参数，则不用上述安全操作。