**代码规范：**

**一、Java命名规范**

1、项目名全部小写

2、包名全部小写

3、类名首字母大写，如果类名由多个单词组成，每个单词的首字母都要大写。

如：public class MyFirstClass{}

4、变量名、方法名首字母小写，如果名称由多个单词组成，每个单词的首字母都要大写。

如：int index=0;

  public void toString(){} //即驼峰式命名方法

5、常量名全部大写

如：public static final String GAME\_COLOR=”RED”;

6、所有命名规则必须遵循以下规则：

1)、名称只能由字母、数字、下划线、$符号组成

2)、不能以数字开头

3)、名称不能使用JAVA中的关键字。

4)、坚决不允许出现中文及拼音命名。

**二、数据库命名规范**

1、数据库表命名规范:

1)表名前应该加上前缀，表的前缀一个用系统或模块的英文名称缩写，前缀全部大写或首字母大写，表名中包含的单词首字母大写。

2)数据库表名应该有意义，并且易于理解，最好使用可以表达功能的英文单词或缩写，如果用英文单词表示，建议使用完整的英文单词。

3)表名不可以太长，最好不要超过3个英文单词长度（22个字母）。

4)在数据库表命名时应该用英文单词的单数形式，如员工表命名：应该为Employee而不是Employees .

5)如果是后台表命名时应该在表名基础上加上后缀\_b(back 首字母 )

6)在表创建完成前，应该为表添加表的注释。

2.表字段命名规范：

1)数据库表字段应该是有意义而且易于理解的，最好是能够表达字段含义的英文字母

（有人认为如果用英文单词作为字段，因为翻译工具不同，而字段不统一建议使用汉语拼音首字母缩写；有人认为用汉语拼音缩写看起来不直观，老半天也不知道到底这个字段是干什么的）

2)系统中所有属于内码，即仅用于标识唯一性和程序内部用到的标识性字段，字段名称建议取为ID,采用类型为整型或长整型.

3)系统中属于是业务内的编号字段，代表一定业务信息，建议字段命名为

code ,如工作单编号wf\_code

4)不要在数据库表字段（列名）中包含数据类型，如：datetime

5)不要在数据库表字段（列名）命名时重复表名，可以使用表名首字母（不包含数据库表名前缀）

注意：

不要在数据库表字段（列名）命名时

不建议使用数据库关键字,如：name,time ,datetime password 等

3.表设计规范：

1)所有字段在设计时，除以下数据类型timestamp、image、datetime、smalldatetime、uniqueidentifier、binary、sql\_variant、binary、varbinary外，必须有默认值。字符型的默认值为一个空字符值串’’；数值型的默认值为数值0；逻辑型的默认值为数值0；其中：系统中所有逻辑型中数值0表示为“假”；数值1表示为“真”。

datetime、smalldatetime类型的字段没有默认值，必须为NULL。

2)当字段定义为字符串形时建议使用varchar而不用nvarchar。

注：在MySQL5.0以上的版本中，varchar数据类型的长度支持到了65535，也就是说可以存放65532个字节的数据，起始位和结束位占去了3个字节。

3)建议在大多数表中（如工作单），应都有以下字段：

字段名说明类型默认值

CreatorID创建者int 默认值为0

CreatedTime创建时间Datetime 默认值为NULL

4)字段的描述

a.字段必须填写描述信息（注释）

b.尽量遵守第三范式的标准（3NF）

表内的每一个值只能被表达一次（列名不重复）

表内的每一行都应当被唯一的标示（标识唯一性，如自动增长主键）

表内不应该存储依赖于其他键的非键信息

5)加索引规则

a.表建好后数据库自动为表生成一个索引（为自动增长的列生成唯一索引），如果在对这列添加索引，数据库会给一个警告,内容大概是，已经为这列添加了索引，建议修改索引名称和自动增长列名保持一致，为了方便使用。

b.如果在添加索引时，建议索引名称和数据库列名保持一致，为了方便使用

c.如果字段事实上是与其它表的关键字相关联而未设计为外键引用，需建索引。

d.如果字段与其它表的字段相关联，需建索引。

e.如果字段需做模糊查询之外的条件查询，需建索引。

f.除了主关键字允许建立簇索引外，其它字段所建索引必须为非簇索引。

4.存储过程命名规范

存贮过程的命名请遵循以下命名规范：USP\_ + 系统模块缩写（与表前缀类似）+\_ + 功能标识 + 代表存贮过程操作的主要表名（不带前缀）或功能的英文单词或英文单词缩写。

如果一个存贮过程只对一个表进行操作，建议存贮过程的名称就用存贮过程所操作的表的表名（不带前缀）。这样有利于根据表名找到相应的存贮过程。例如：

用于新增的存贮过程USP\_MESSAGE\_Add\_Model

用于修改的存贮过程USP\_ MESSAGE\_Upt\_Model

用于删除的存贮过程USP\_ MESSAGE\_Del\_ Modele

注：USP是user stored procedure

缩写

5.存储过程设计规范

在存贮过程中必须说明以下内容：

1)目的：说明此存贮过程的作用。

2)作者：首次创建此存贮过程的人的姓名。在此请使用中文全名，不允许使用英文简称。

3)创建日期：创建存贮过程时的日期。

4)修改记录：

修改记录需包含修改顺序号、修改者、修改日期、修改原因，修改时不能直接在原来的代码上修改，也不能删除原来的代码，只能先将原来的代码注释掉，再重新增加正确的代码。修改顺序号的形式为：log1，log2，log3。。。，根据修改次数顺序增加，同时在注释掉的原来的代码块和新增的正确代码块前后注明修改顺序号。

5)对存贮过程各参数及变量的中文注解。

建议：在数据库中创建一个文本文件保存创建脚本

6.视图命名规范

视图的命名请遵循以下命名规范：UV \_ + 系统模块缩写（与表前缀类似）+\_ + 功能标识 + 代表视图查询的主要表名（不带前缀）或功能的英文单词或英文单词缩写。

如果一个视图只对一个表进行查询，建议视图的名称就用视图所查询的表的表名（不带前缀）。这样有利于根据表名找到相应的视图。

注：UV是userView缩写

7.视图设计规范

在视图中必须说明以下内容：

1)目的：说明此视图的作用。

2)创建者：首次创建此视图的人的姓名。在此请使用中文全名，不允许使用英文简称。

3)修改者、修改日期、修改原因：如果有人对此视图进行了修改，则必须在此视图的前面加注修改者姓名、修改日期及修改原因。

4)对视图各参数及变量的中文注解

建议：在数据库中创建一个文本文件保存创建脚本

8.触发器命名规范

Insert触发器加'\_i'，Delete触发器加'\_d'，Update触发器加'\_u'

9.触发器设计规范

在视图中必须说明以下内容：

1)目的：说明此触发器的作用。

2)创建者：首次创建此触发器的人的姓名。在此请使用中文全名，不允许使用英文简称。

3)修改者、修改日期、修改原因：如果有人对此触发器进行了修改，则必须在此触发器的前面加注修改者姓名、修改日期及修改原因。

4)对触发器各参数及变量的中文注解

建议：在数据库中创建一个文本文件保存创建脚本

1. **注释规范**

**1、类注释**

在每个类前面必须加上类注释，注释模板如下：

/\*\*

\* Copyright (C), 2006-2010, ChengDu Lovo info. Co., Ltd.

\* FileName: Test.java

\* 类的详细说明

\*

\* @author 类创建者姓名  
\* @Date    创建日期

\* @version 1.00

\*/

**2、属性注释**

在每个属性前面必须加上属性注释，注释模板如下：

/\*\* 提示信息 \*/

private String strMsg = null;

**3、方法注释**

在每个方法前面必须加上方法注释，注释模板如下：

/\*\*

\* 类方法的详细使用说明

\*

\* @param 参数1 参数1的使用说明

\* @return 返回结果的说明

\* @throws 异常类型.错误代码 注明从此类方法中抛出异常的说明

\*/

**4、构造方法注释**

在每个构造方法前面必须加上注释，注释模板如下：

/\*\*

\* 构造方法的详细使用说明

\*

\* @param 参数1 参数1的使用说明

\* @throws 异常类型.错误代码 注明从此类方法中抛出异常的说明

\*/

1. **函数使用**
2. 接口函数使用

在同一项目组应明确规定对接口函数参数的合法性检查应由函数的调用者负责还是由接口函数本身负责，缺省是由函数调用者负责。

说明：对于模块间接口函数的参数的合法性检查这一问题，往往有两个极端现象，即：要么是调用者和被调用者对参数均不作合法性检查，结果就遗漏了合法性检查这一必要的处理过程，造成问题隐患；要么就是调用者和被调用者均对参数进行合法性检查，这种情况虽不会造成问题，但产生了冗余代码，降低了效率。

2、非调度函数应减少或防止控制参数，尽量只使用数据参数。

说明：本建议目的是防止函数间的控制耦合。调度函数是指根据输入的消息类型或控制命令，来启动相应的功能实体（即函数或过程），而本身并不完成具体功能。控制参数是指改变函数功能行为的参数，即函数要根据此参数来决定具体怎样工作。非调度函数的控制参数增加了函数间的控制耦合，很可能使函数间的耦合度增大，并使函数的功能不唯一。

示例：如下函数构造不太合理。

int add\_sub( int a, int b, unsigned char add\_sub\_flg )

{

    if (add\_sub\_flg == INTEGER\_ADD)

    {

        return (a + b);

    }

    else

    {

        return (a - b);

    }

}

不如分为如下两个函数清晰。

int add( int a, int b )

{

    return (a + b);

}

int sub( int a, int b )

{

    return (a - b);

}

3、除非必要，最好不要把与函数返回值类型不同的变量，以编译系统默认的转换方式或强制的转换方式作为返回值返回。

4、在调用函数填写参数时，应尽量减少没有必要的默认数据类型转换或强制数据类型转换。

5、设计高扇入、合理扇出（小于7）的函数。

说明：扇出是指一个函数直接调用（控制）其它函数的数目，而扇入是指有多少上级函数调用它。

扇出过大，表明函数过分复杂，需要控制和协调过多的下级函数；而扇出过小，如总是1，表明函数的调用层次可能过多，这样不利程序阅读和函数结构的分析，并且程序运行时会对系统资源如堆栈空间等造成压力。函数较合理的扇出（调度函数除外）通常是3-5。扇出太大，一般是由于缺乏中间层次，可适当增加中间层次的函数。扇出太小，可把下级函数进一步分解多个函数，或合并到上级函数中。当然分解或合并函数时，不能改变要实现的功能，也不能违背函数间的独立性。

扇入越大，表明使用此函数的上级函数越多，这样的函数使用效率高，但不能违背函数间的独立性而单纯地追求高扇入。公共模块中的函数及底层函数应该有较高的扇入。

较良好的软件结构通常是顶层函数的扇出较高，中层函数的扇出较少，而底层函数则扇入到公共模块中。

6、避免使用BOOL参数。

说明：原因有二，其一是BOOL参数值无意义，TURE/FALSE的含义是非常模糊的，在调用时很难知道该参数到底传达的是什么意思；其二是BOOL参数值不利于扩充。还有NULL也是一个无意义的单词。

7、对于提供了返回值的函数，在引用时最好使用其返回值。

**五、代码测试与维护**

1、单元测试要求至少达到语句覆盖

2、单元测试开始要跟踪每一条语句，并观察数据流及变量的变化

3、清理、整理或优化后的代码要经过审查及测试。

4、代码版本升级要经过严格测试

5、使用工具软件对代码版本进行维护

6、正式版本上软件的任何修改都应有详细的文档记录

7、其他

1)发现错误立即修改，并且要记录下来。

2)关键的代码在汇编级跟踪。

3)仔细设计并分析测试用例，使测试用例覆盖尽可能多的情况，以提高测试用例的效率。

4)尽可能模拟出程序的各种出错情况，对出错处理代码进行充分的测试。

5)仔细测试代码处理数据、变量的边界情况。

6)保留测试信息，以便分析、总结经验及进行更充分的测试。

7)不应通过“试”来解决问题，应寻找问题的根本原因。

8)对自动消失的错误进行分析，搞清楚错误是如何消失的。

9)修改错误不仅要治表，更要治本。

10)测试时应设法使很少发生的事件经常发生。

11)明确模块或函数处理哪些事件，并使它们经常发生。

12)坚持在编码阶段就对代码进行彻底的单元测试，不要等以后的测试工作来发现问题。

13)去除代码运行的随机性（如去掉无用的数据、代码及尽可能防止并注意函数中的“内部寄存器”等），让函数运行的结果可预测，并使出现的错误可再现。