目录

SDMC

文件状态

密级	P
文件状态	[ ] 草稿 [ ] 正在修改 [ ] 经过评审［√］正式发布

版本控制

版本号	日期	修改人	说明
V1.0	2011-07-05	梁石磷	创建
V1.0	2011-07-06		审核
V1.0	2011-07-06		发布

修订说明：

1. 文件修订时，应将关键的修订内容填入本表
2. 对于文件整章节的大规模变动为大版本号变更，否则为小版本的增加
3. 版本编号为 V1.0>V1.1>…>V2.0>…

SDMC

1. Java 的命名规范

1.1 类（Class）和接口（Interface）命名

Java 的类命名按照 Sun 提出的 Java 的标准命名方式命名：The standard Java convention is to use a full English descriptor starting with the first letter capitalized using mixed case for the rest of the name。即：使用完整的英文单词（首字母大写），若有多个单词组成，则每个单词的首字母均需大写。需要注意的是，类或接口的名称的长度以不超过 25 个字符为准，若超过 25 个字符，则使用英文缩写，例如：PlayStationPortableService，这样的类（接口）的名称太长，可以改为：PSPService。

Examples:

```
Customer
Employee
FileStream
OrderItem
AbstractList
```

1.2 方法（functions）的命名

Java 的方法命名按照 Sun 提出的 Java 的标准命名方式命名：Member Functions should be named using a full English description, using mixed case with the first letter of any non-initial word capitalized. It is also common practice for the first word of a member function name to be a strong, active verb。

使用完整的英文单词（首字母小写），若有多个单词组成，则接着的单词的首字母大写。方法通常是执行某一个或一组动作，方法名的长度以不超过 25 个字符为宜，若英文太长，则使用约定俗成的英文缩写词代替。方法名第一个单词通常都是表示动作，即动词。

Examples:

```
openAccount()
createAccount()
getDate()
setName()
save()
delete()
```

1.2.1 返回 boolean 值的方法命名

是否：Examples:

```
isArray() 是否是数组
isEmpty() 是否为空
isVisible() 是否显示
```

有没有：Examples:

```
hasNext() 有没有下一个
hasData() 有没有数据
```

包含？：Examples:

```
containsKey() 包含 Key?
containsValue() 包含 Value?
```

```
containsObject() 包含 Object?
```

1.2.2 常见的动作或指令的方法命名

```
设置对象或值（set-） >>> Examples：
        setFirstname()
        setCharacter()
删除对象或值（delete-） >>> Examples：
        deleteRow()
        deleteFile()
保存对象或值或文件等（save-） >>> Examples：
        saveFile()
        saveUserSetting()
创建对象或文件等（create-） >>> Examples：
        createObject()
        createCache()
获取对象或值等（get-） >>> Examples：
        getObject()
        getClass()
打印对象或值等（print- 或 echo-） >>> Examples：
        printTableData()
        echoDataList()
载入对象或值等（load-） >>> Examples：
        loadFactory()
        loadParser()
添加对象或值等（add- 或 append） >>> Examples：
        addTime()
        appendManagement()
计算（calculate-） >>> Examples：
        calculateSum()
```

1.3 变量（variables）的命名

Java 的变量命名按照 Sun 提出的 Java 的标准命名方式命名：You should use a full English descriptor to name your fields (Gosling, Joy, Steele, 1996; Ambler 1997) to make it obvious what the field represents. Fields that are collections, such as arrays or vectors, should be given names that are plural to indicate that they represent multiple values。

使用完全的英文描述变量。如果变量表示集合，例如数组或者 Vector，应该用名词复数形式表示。变量名开头以小写开始。长度以不超过 25 个字符为宜，下面的例子是 Sun 的变量命名标准规范。

```
Examples:
        firstName
        huangyuanFactory
        unitPrice
        queryAdministrator
        (List)lstOrderItems
```

必须使用的变量命名方法。如果变量开头是个只取开头字母的缩写词，那么该词需全部小写，例如：

sqlDatabase，而不是 sQLDatabase。

循环迭代中的变量命名，约定用 i、j、k、l、m、n……

```
Examples:
for(int i = test.length(); --i >= 0;) {
    // Do Something……
    for(int j = 0; j < some.length; j++) {
        // Do Something……
    }
}
```

类成员变量在变量名前加"m"作为标记，类的静态变量即 static 变量加"s"作为标记，后继的单词首字母大写。

```
Examples:
class ClassName{
        private int mMember;
        private static int sStaticMember;
}
```

1.3.1 变量 - 匈牙利命名法

The "Hungarian Notation" (McConnell, 1993) is based on the principle that a field should be named using the following approach: xEeeeeeEeeeee where x indicates the component type and EeeeeEeeeee is the full English descriptor.

```
Examples:
        sFirstName (String)
        iZipCode (int)
        fDepositRate (float)
        ICollection (interface)
        cOrderItems (Collection)
```

这种命名方法通常用在 C++编程中，此外，可以让程序员一眼就知道变量的类型，也就知道如何处理，但是这种命名方法太过于笨拙，不允许在 Java 中使用该变量命名方法。推荐在 Javascript 编码中使用该变量命名方法。

1.3.2 变量 – 下划线打头或结尾的方法

C++中常用的变量命名方法。用下划线来指示它为一个变量。

```
Examples:
        _firstName
        firstName_
```

不允许在 Java 编程中使用该变量命名方法。

1.4 常量的命名

In Java, constants, values that do not change, are typically implemented as *static final* fields of classes. The recognized convention is to use full English words, all in uppercase, with underscores between the words (Gosling, Joy, Steele, 1996; Sandvik, 1996; NPS, 1996).

在 Java 编程中，常量用完整的英文描述，并且常量全部用大写字母表示，每个单词之间用下划线（_）

连接。必须使用该常量命名方法，常量一律采用该方法命名。

Examples:

```
protected static final String STRING_ENCRYPT_KEY
protected static final String HUANGYUAN_USER_COOKIE
protected static final String SYSTEM_CONFIG_FILE
```

1.5 组件（Component）的命名

Java 编程中，特定的前缀名称和完整的英文描述后缀构成对组件的命名。xxxEeeeeEeeee，xxx 是特定组件的小写缩写名，EeeeeEeeee 是对该组件的完整描述。

Examples:

```
ButtonGroup >>> 前缀 btnGroup –
JApplet >>> 前缀 apt –
JButton >>> 前缀 btn -
JComboBox >>> 前缀 cmb –
JCheckBox >>> 前缀 chk –
JCheckBoxMenuItem >>> 前缀 chkMenu –
JDesktopPane >>> 前缀 desktop –
JDialog >>> 前缀 dlg –
JFrame >>> 前缀 frm –
JInternalFrame >>> 前缀 innerFrm –
JLabel >>> 前缀 lbl –
JList >>> 前缀 lst –
JMenu >>> 前缀 menu –
JMenuItem >>> 前缀 mItem –
JPanel >>> 前缀 pnl –
JRadioButton >>> 前缀 rb –
JTable >>> 前缀 tab –
JTextArea JTextField >>> 前缀 txt -
JToolBar JToolTip >>> 前缀 tool –
JTree >>> 前缀 tree –
JWindow >>> 前缀 win –
```

SDMC

2. Java 编码规范

2.1 注释规范

开发人员必须遵守的编码规范。

2.1.1 文件头部注释

每个新创建的类文件或者配置文件都需要头部注释，注释格式：

```
Examples:
/*
 * Copyright (C) 2008-2011 Shenzhen SDMC Technology CO.LTD, Inc. All rights
reserved.
 *
 * @(#)ClassName.java  1.12(版本号) 2011-04-07
 */
```

第一行：版权声明

第二行：@(#)类名.java 版本号 yyyy-MM-dd

2.1.2 类（Class、Interface）注释

类注释遵照 Sun 的 Javadoc 注释标准（包括内部类），当有版本更新时，添加版本更新描述。

```
格式:
/**
 * 类的（使用）说明。
 * <pre> [类的使用举例] </pre>
 *
 * @author  类的作者 [可写多个作者，也可以直接写多个@author]
 * @version  版本描述，日期（yyyy/MM/dd）
 * [@since   这个类或接口加入应用的版本]
 * [@see    相关联的类或接口（完整的名称，包括包名，若在同一个包内，则可省去包名）]
 */
```

```
Examples:
/**
 * <code>UserService</code>接口的默认实现类。
 *
 * @author Liangshilin
 * @version 1.0, 2011/07/02 <br />
 *          1.1, 2011/07/03 Liangshilin 添加 getUserByEmail 方法 <br />
 *          1.2, 2011/07/05 Liangshilin 添加 getUserByName 方法 <br />
 *          1.3, 2011/07/06 Liangshilin 调整 findUserList 方法的算法
 * @see com.ztgame.service.UserService
 */
```

2.1.3 方法（Functions）注释

方法注释遵照 Sun 的 Javadoc 注释标准：

```
Examples:
/**
 * 方法说明。
 * <pre> [方法使用举例]
 * </pre>
 * @param 参数名称 参数说明
 * [@param 参数名称 参数说明 多个参数 ]
 * [@return 返回类型 返回值说明 ]
 * [@see  查看类拟的方法或者类]
 */
[@deprecated]
```

2.1.4 变量注释

变量注释遵照 Sun 的 Javadoc 注释标准。私有变量（容易产生歧义的除外）不需要注释，静态不可变常量需要 Javadoc 注释说明，受保护的和公有的变量需要 Javadoc 注释说明。

```
Examples:
/**
 * DSA_PUBLIC_KEY 的说明
 */
protected static final String DSA_PUBLIC_KEY = “11223344556677889900”;
```

2.1.5 Javadoc 注释标识列表

标 记	使用范围	说 明
@author name	类、接口	指明该代码的作者，如果有多个作者，每个作者都要写一行@author
@deprecated	类、接口、方法	指明对于这个 API 中的类、接口是方法，成员变量已经过时，因此，在任何的时候都不要使用它
@exception name 描述	方法	描述成员方法运行时可能抛出的异常，每个异常都需要写一行@exception 来描述，并且应该给出这个异常类的全称
@param name 描述	方法	描述成员方法的参数，每个参数需要写一行@param 来描述，并且指定这个参数的数据类型
@return 描述	方法	在方法有返回值是，用此标记描述返回值的类型，以及返回值的描述
@since	类、接口、方法	用此标记说明类、接口或是方法是哪个版本时开始存在（创建）的
@see ClassName	类、接口、方法、变量	在 Javadoc 中的 Documents 中生成一个指向 ClassName 的超链接
@see ClassName#方法名	类、接口、方法、变量	在 Javadoc 中的 Documents 中生成一个指向 ClassName 的方法[方法名]的超链接

@version text	类、接口	指定的类或者接口的版本标识

2.1.5.1 @author

@author 标签指明了类或接口的作者：

```
@author Liangshilin
@author Kevin
@author Frank
```

我们可以根据需要使用任意多个@author 段落，而每个@author 段落应该只列出一位作者，以保证在所有情况下都会得到一致的输出。

2.1.5.2 @version

@version 标签使我们可以为类或接口指定任意版本的规范：

```
@version 1.1
```

2.1.5.3 @since

@since 标签使我们可以指定任意版本的规范，以表示由该标签注释的实体是何时添加到系统中的。

```
@since 2.1
```

2.1.5.4 {@ literal}和{@ code}

行间标签 {@literal text}可以让 text 按照原样被打印，不会被解释成 HTML 源码。我们可以用&、<和>，而不必使用&、<和>。

行间标签 {@code text}的行为与{@literal text}很相似，差别只是它的 text 会以代码字体打印。使用 HTML 标签<code>和</code>来包装{@literal text}标签可以达到相同的效果。

2.1.5.5 {@ value}

{@value static-field-name} 标签会被替换指定的静态常量字段的实际值。这个标签使用的成员语法与@see 标签相同。

```
The valid range is 0 to {@value java.lang.Short#MAX_VALUE}
// 输出：The valid range is 0 to 32767
```

2.2 代码格式规范

2.2.1 代码缩进

代码缩进为一个 tab（需全部转换为空格 — 4 个空格）。

2.2.2 代码块格式

Java 中的代码块是以{}分隔的，左大括号需和语句在同一行上：

```
Examples:
public class SimpleClass {
    // Coding……
}
public void setValue(String key, String value) {
    // Coding……
}
for(int i = 0; i < count; i++) {
    // Coding……
}
```

If…Else…（try…catch…）语句

```
Examples:
if(isEmpty()) {
    // Coding……
} else {
    // Coding……
}
try {
    // Coding……
} catch (Exception e) {
    // Coding……
} finally {
    // Coding……
}
```

```
Examples:
SimpleComponent component = new ConcreteComponent(
        new OtherConcreteComponent(new OtherConcrete()));
/* 一个特例：当方法的参数有很多个时，左大括号可以换行 */
// 但是方法中的参数不要超过 8 个，如果参数有很多，可以考虑用 Hashtable 或者
// HashMap 进行封装
public static void getUser(String name, String password, int gender,
        String email, String trueName, Timestamp registerDate)
{
    // Coding……
}
```

2.3 编码规范

2.3.1 接口（Interface）

Java 编程中，接口的方法默认的访问域就是 public，编码时，常用的方式是不书写访问域声明。

例如：

```
public interface UserService {
    /**
     * 用给定的必须的用户属性，创建一个新的用户。
     *
     * @param fields 给定的用户属性。
     * @return 新的用户。
     * @throws UserAlreadyExistsException 当给定的用户已经存在时，抛出此异常。
     */
    User createUser(String…fields) throws UserAlreadyExistsException;
}
```