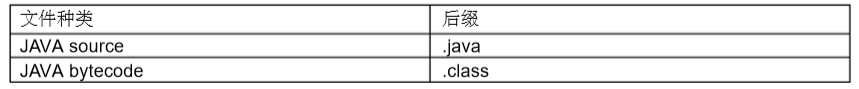
JAVA 编程语言的基本规范

1、文件名

这一节列举了常用的文件后缀与名称。

1．1文件后缀

1.2 普通文档名称 通常用的文档名称



2、文件的组织结构

一个文件应当由多个被分隔的段组成，并由空行和可选择注释来识别。

超过 2000 行的文件由于冗长而应当避免。

2.1 Java 源文件

每一个 Java 源文件包含一个单一的公共类或界面。当私有类或界面与一个公共类发生 联系时， 你可以将它们以一个公共类放入同一个源文件。 公共类应当是文件中的第一类或界 面。

Java 源文件有如下次序：

起始注释（参看“起始注释” ）

组件和导入语句 类和界面声明（参看“类和界面声明” ）

2.1.1 起始注释 所有源文件应当以 C格式注释开始，并列出类名，版本信息，时间和版权说明：

/\* \*类名

\* \*版本信息

\* \*时间

\* \*版权说明

\*/

2.1.2 组件和导入语句 对于大部分 Java 源文件而言， 第一个无注释行是组件语句。 之后，紧跟着是导入语句。

举例如下：

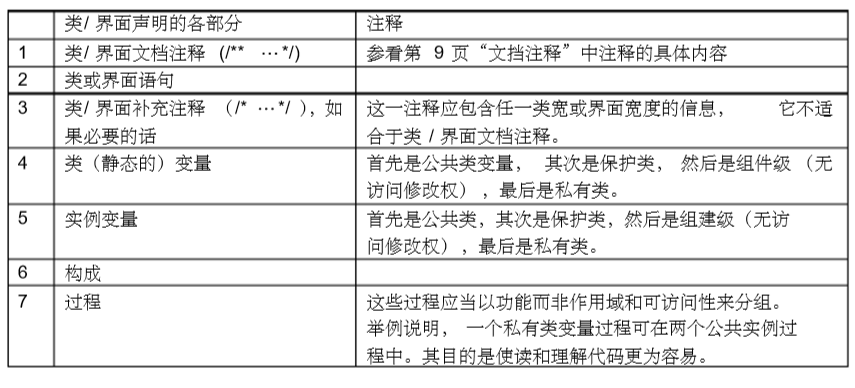
组件 java.awt;

导入 java.awt.peer.CanvasPeer

注释：对于唯一的组件名， 第一部分总是小写的 ASCII 码格式的文本文件， 并且是最高 级别的域名之一。目前可用 com,edu,gov,mil,net 或 ISO3166标准（1981）中规定的用于识 别国家的英文两个字母的模式。

2.1.3 类和界面声明

下表按出现的先后顺序描述了一个类或界面说明的各部分。



3、缩进格式

四个空格作为一个缩进单位。 确切的缩进格式结构 （空格符与制表符） 未被规范。 表格 的设置必须在每 8 个空格后（而非 4 个）。

3.1 行的长度

由于难于被大多数终端和工具进行处理，应当避免一行超过 80 个字符。

注释：在文档中所列举的范例其行的长度应稍短些，一般不超过 70 个字符。

3.2 绕回行

当表达式一行无法写完时，使用以下一般规则进行中断： 在逗号后中断。 在一个运算符前中断。优先选择高级中断指令。 新行起始表达式的位置应与旧行表达式的位置对齐。 如果上述规则导致代码混乱或编码时顶到了右边界，以8个空格代替。

以下是几个中断程序的调用范例：

someMethod(longExpression1, longExpression2, longExpression3,

longExpression4, longExpression5);

var = someMethod1(longExpression1,

someMethod2(longExpression2,

longExpression3);

以下是两个关于中断算术表达式的例子。第一个例子由于中断发生在插入表达式以外， 因而选择了高级中断指令。

longName1 = longName2 \* (longName3 + longName4 – longName5)

+ 4 \* longname6; // PREFER

longName1 = longName2 \* (longName3 + longName4

longName5) + 4 \* longname6; // AVOID

以下是两个关于缩进程序的说明。第一个例子是常规情况。第二个例子如果采用常规 缩进方式，第二和第三行在换行时必然顶至最右，取而代之应空 8 格。 // 常规的缩进方式

someMethod(int anArg, Object anotherArg, String yetAnotherArg, Object andStillAnother) 。。。

// 缩 8 个空格以避免更深的锁进

private static synchronized horkingLongMethodName(int anArg,

Object anotherArg, String yetAnotherArg,

Object andStillAnother) 。。。

由于常规的 4 空格缩进方式使得体看上去过于复杂， 对于语句的绕回行我们一般采用 8 空格缩进方式。举例如下：

// 不要使用这种缩进方式

if ((condition1 && condition2)

(condition3 && condition4)

!(condition5 && condition6)) //BAD WRAPS

doSomethingAboutIt(); //MAKE THIS LINE EASY TO MISS

// 取而带之使用这种缩进方式

if ((condition1 && condition2)

(condition3 && condition4) !(condition5 &&condition6))

doSomethingAboutIt();

// 或使用这种方式 if ((condition1 && condition2) (condition3 && condition4)

!(condition5 &&condition6))

doSomethingAboutIt();

以下有三种可行的方式来格式化三元表达式：

alpha = (aLongBooleanExpression) ? beta : gamma;

alpha = (aLongbooleanExpression) ? beta

:gamma;

alpha = (aLongBooleanExpression)

beta

: gamma;

5 注释

JAVA 程序可以有二种类型的注释：执行注解和文件注释。执行注释是建立在 C++之上 的，以 /\* ⋯\*/为分隔符的注释；文件注释（即通常所说的 doc comment）是一种纯 JAVA 注 释，以/\*\* ⋯\*/ 为分隔符。Doc comment ( 文件注释 )能够通过 JAVADOCA 工具被摘录成 HEML 文件。

执行注释是指为代码注解释， 或为特别执行注解释。 Doc 注释旨在从一个自由执行的程

序描述代码的规范，对于软件开发者来说，手边不再需要源代码就可以阅读。

注释通常用于对代码做总的描述，同时提供附加的信息，这从代码本身来看是不容易得 到的。

其包含的信息只与阅读、理解该程序有关，例如：一个相应组件如何被建立的信息或驻 足在何地址录下的信息都将被包含在一个注解中。

对于重要的、非显而易见的设计决定的讨论是适当的， 但是应该避免重复信息在代码中出现。 而对于多余注释， 则很容易成为过时的。 总的说来， 应当避免把过时的注释做为编码的进展。

注意： 经常出现的注释有时反映编码质量的低下。 当你觉得不得不添加注释时， 建议你最 好重新编写，使得编码更清晰。 注释不能以星号或是其它字符为标识被附在一个大的逻辑单元内。 注释也不能包括特殊字符例如 form-feed 和 backspace。

4.1 执行注释的格式 程序可以有四种注释风格：块、单行、跟踪和行尾。

4.1.1 块注释 块注释用于提供文件的描述、 方法、数据结构和运算法则。 块注释用于每个文件的开始和方 法之前。 也可以用在其他地方， 比如用在方法中。 在一个函数或方法中的块注释， 应和他们 描述的代码排列到同一级别上。块注释应设置在一个空行的开始。

/\*

\* Here is a block comment.

\*/ 块注释以 /\*开始，并且单独领导一行，缩进一格作为块注释的开始，这已经是约定俗成的， 不需要另外重新定义格式。例如：

/\*-

\* Here is a block comment with some very special

\* formatting that I want indent(1) to ignore. \*

\* one

\* two

\* three

\*/

Note: If you don't use indent(1), you don't have to use /\*- in your code or make any other

concessions to the possibility that someone else might run indent(1) on your code.

4.1.2 单行注释 简短的注释可以出现在单行上， 和其描述的代码在同一级别上。 单行注释应遵循块注释的格 式。单行注释应单起一行。例如：

if (condition) {

/\* Handle the condition. \*/

...

}

4.1.3 跟踪注释

简短的注释可以和其所描述的代码放到同一行上。 但应和代码保持足够远的空间。 如果不只 一个简短的注释出现在大块的代码段中，它们应有同样的 tab 设置。例如：

if (a == 2) {

return TRUE; /\* special case \*/

} else {

return isPrime(a); /\* works only for odd a \*/

}

4.1.4 行尾注释 //注释界定符能注释一整行或行的一部分，它不应该用在连续的多行文本注释中。然而，它 可以用在连续的多行代码段中。例如：

if (foo > 1) {

// Do a double-flip.

...

}

else {

return false; // Explain why here.

} //if (bar > 1) {

//

// // Do a triple-flip.

// ...

//}

//else {

// return false;

//}

4.2 文件注释

文件注释描述 Java 类、界面、容器、方法和域，每一个文件注释都放在注释界定符 /\*\*...\*/ 之间。注释应在声明之前。

/\*\*

\* The Example class provides ...

\*/

public class Example { ...

注意顶级类和界面不缩进， 子类和界面则要缩进。 类和界面的第一行文件注释不缩进， 后面 的注释行有有一个空格的缩进。子类，包括容器，有4个空格和五个空格的缩进。如果你需要给出关于类、 界面, 变量, 或方法的信息， 这些信息以文件的方式给出是不适当 的，那麽你可以直接的在声明之后使用执行块注释或单行注释。 例如，类执行的细节信息应 放进执行块注释中，并跟在类语句之后，而不是放在类的文件注释中。文件注释不应该放到方法或容器的定义块中，因为 Java 将文件注释和第一个注释之后的声明发生联系。

5、声明

5.1 每一行的数量 每一行中都应有相应的注释，换句话说

int level; // indentation level

int size; // size of table

这种形式是首选的，而不是 int level, size; 不要将不同类型的放入同一行中，例如： int foo, fooarray[]; //WRONG! 注意：上面的例子中在类型和标识之间有一个空格，另一个可接受的方法是用 tab，例如：

int level; // indentation level

int size; // size of table

Object currentEntry; // currently selected table entry

5.2 初始化 被声明的局部变量，应设法将其初始化。

5.3 布置 声明应放在块的开始。 不要一直到用到第一个变量的时候才开始声明， 那将使粗心的程序员 迷惑，并且妨碍代码的轻便性。

void myMethod() {

int int1 = 0; // beginning of method block

if (condition) {

int int2 = 0; // beginning of "if" block ...

}

}

在 Java 中，for 循环中的指针可以在 for 语句中声明，例如

for (int i = 0; i < maxLoops; i++) { ... } 避免在更高层的声明当中隐含局部声明， 例如，不要在一个内部块中声明有相同名字的变量。

int count;

...

myMethod() { if (condition) {

int count = 0; // AVOID!

...

}

...

}

5.4 声明类和界面

当在译码 java 的类和界面的时候，应遵循以下的格式化规则： 列出参数时，在方法名和圆括号之间不应该有空格。 "{" 出现在声明语句同一行的结尾。 "}" 单起一行，但要和最近的“ { ”匹配。但如果是空语句则例外。

class Sample extends Object {

int ivar1;

int ivar2;

Sample(int i, int j) {

ivar1 = i;

ivar2 = j;

}

int emptyMethod() {}

...

}

Methods

6、语句

6.1 简单语句 每行最多只能包括一个语句。比如：

argv++;// Correct

argc--;//Correct

argv++;//argc--; //AVOID

6.2 复合语句 复合语句是包括一系列包含在括号 {statement} 中的语句。请看以下例子： 被包含的语句必须比复合语句缩进一格。 开始的括号必须在开始复合语句的那一行的末尾； 关闭的括号应该重起一行并相对复合语句 的开头来说缩进。当括号是控制结构的一部分时，它们在所有语句中使用，像 if-elsea 或 for 语句即便是简要 语句。

6.3 return 语句 在某种程度上，如果一个 return 语句的返回值并不非常明显，那么不应该加圆括号。例如：

return;

return myDisk.size();

return (size ? size : defaultSize); 7.4 if, if-else, if else-if else 语句 if-else 语句可以分为以下几类：

if (condition) {

statements;

}

if (condition) {

statements;

} else {

statements;

}

if (condition) {

statements;

} else if (condition) {

statements;

} else{

statements;

}

Note: if statements always use braces {}. A void the following error-prone form:

if (condition) //A VOID! THIS OMITS THE BRACES {}!

statement;

6.5 for 语句 For 语句的形式如下所示：

for (initialization; condition; update) {

statements;

}

一条空的 for 语句的形式如下所示：

for (initialization; condition; update); 当在初始化或更新子句的时，避免使用超过三个以上的变量，如果有必要的话，可以在 for 循环开始之前或结束后使用个别的语句。

6.6 while 语句 while 语句的形式如下所示：

while (condition) {

statements;

}

一个空的 while 语句的形式如下：

while (condition);

6.7 do-while 语句 do-while 语句的形式如下：

do {

statements;

} while (condition);

6.8 switch 语句

switch (condition) {

case ABC:

statements;

/\* falls through \*/

case DEF:

statements;

break;

case XYZ:

statements;

break;

default:

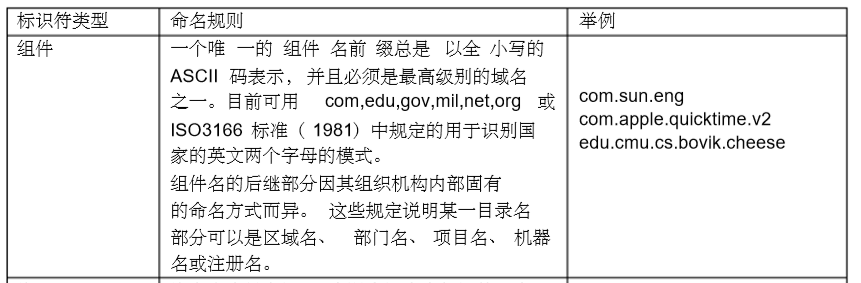
statements;

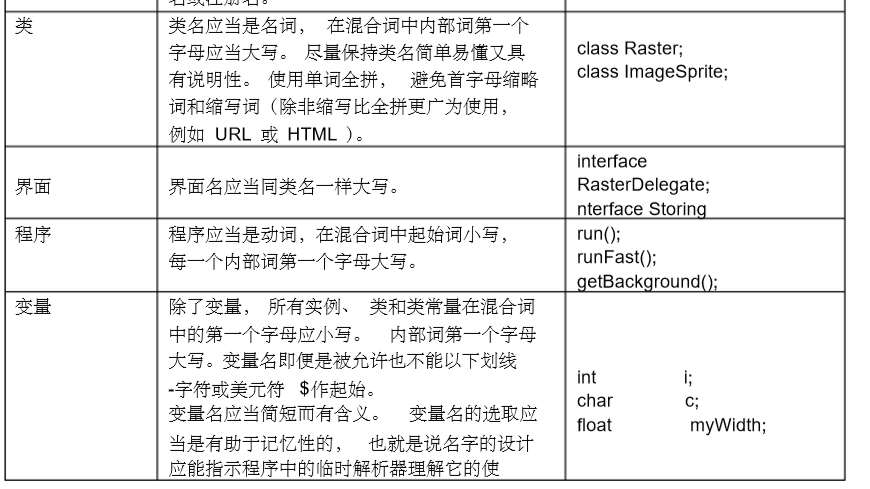
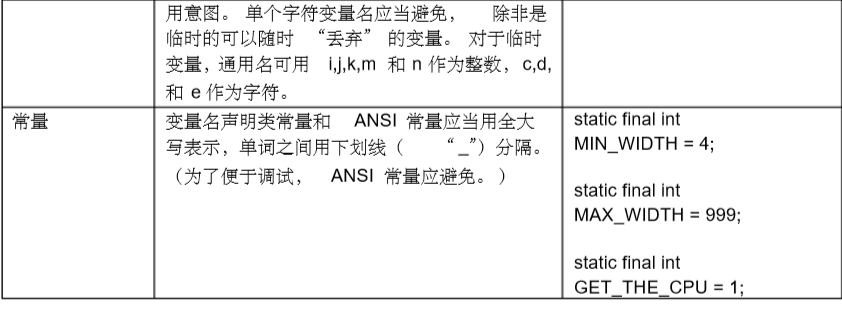
break;

}

每一个 switch 语句应该包括一个默认的 case，在默认的 case 之中的 break 是多余的，但是 这样可以防止以后另外的 case加入而引起的错误。

7、命名规范

命名规范使得程序更易读易懂，同时它们又给出了标识符的功能信息，例如是否是常 量、组件或类。这对理解代码是极其有用的。



8、程序设计惯例

8.1 使用实例和类变量 没有非常好的理由， 请不要声明任何实例或类变量， 实例变量经常会在调用方法上起副作用。 适当的公用实例变量的例子是类实质上是一个数据结构， 而不是行为的情况下使用的。 换句 话说，如果你情愿用 struct 而不用 class，那麽使用类的实例变量是恰当的。

8.2 改进类变量和方法 避免用一个对象访问类（静态）变量或方法，用一个类名来代替。 例如：

classMethod(); //OK

AClass.classMethod(); //OK

anObject.classMethod(); //AVOID!

8.3 常量 数字常量不应该直接被编码，除了 1、0、-1，这几个数字常量可以出现在一个 for 循环中， 作为计数值。

8.4 变量分配 在一个单一语句中，避免给一个变量分配同一个值，那会没有可读性。 例如：

fooBar.fChar = barFoo.lchar = 'c'; // A VOID! 在容易误解的地方，不要使用任务操作符。

if (c++ = d++) { // AVOID! (Java disallows)

...

} 应该写成：

if ((c++ = d++) != 0) {

...

} 如果试图提高运行速度，请不要使用包含式的分配，因为这是编译器的工作。 例如：

d = (a = b + c) + r; // AVOID! 应该写成：

a = b + c;

d = a + r;

8.5 混合惯例

8.5.1 圆括号 为了避免操作符的优先问题，通常情况下不加限制的使用圆括号（包括混合操作符的情况）是个好办法，即使操作符的优先性对你来说看起来很清晰，因为你们不能猜测， 其他的程序员会和你们一样清楚操作符的优先性。

if (a == b && c == d) // AVOID!

if ((a == b) && (c == d)) // RIGHT

8.5.2 返回值 尽量使你的程序的结构达到相称的目的，例如：

if (booleanExpression) {

return true;

} else {

return false;

} 不应该写成：

return booleanExpression; 同样的

if (condition) {

return x;

}

return y; 不要写成：

return (condition ? x : y);

8.5.3 条件表达式之前的？

如果一个含有二进制操作符的表达式出现在？之前， 并且为三重的？： 操作符， 应该加上括号，例如：

(x >= 0) ? x : -x; 10.5.4 特殊注释 用 XXX 作为注释信息来试图标记一些事情，那是不切实际的。