**Java编码规范**

目录

[1、 介绍 3](#_Toc10210354)

[1.1 术语说明 4](#_Toc10210355)

[1.2 文档说明 4](#_Toc10210356)

[2、 源码文件基础 4](#_Toc10210357)

[2.1 文件名 4](#_Toc10210358)

[2.2 文件编码：UTF-8 5](#_Toc10210359)

[2.3 特殊字符 5](#_Toc10210360)

[3、 源码文件结构 6](#_Toc10210361)

[3.1 lincense 或者 copyright的声明信息 7](#_Toc10210362)

[3.2 包声明 7](#_Toc10210363)

[3.3 import语句 7](#_Toc10210364)

[3.4 类声明 8](#_Toc10210365)

[4、 格式 8](#_Toc10210366)

[4.1 花括号 9](#_Toc10210367)

[4.2 语句块的缩进：2空格 10](#_Toc10210368)

[4.3 一行最多只有一句代码 10](#_Toc10210369)

[4.4 行长度限制：80或100 11](#_Toc10210370)

[4.5 长行断行 11](#_Toc10210371)

[4.6 空白空间 12](#_Toc10210372)

[4.7 分组括号：建议使用 14](#_Toc10210373)

[4.8 特殊结构 14](#_Toc10210374)

[5、 命名 19](#_Toc10210375)

[5.1 适用于所有命名标识符的通用规范 19](#_Toc10210376)

[5.2 不同类型的标示符规范 19](#_Toc10210377)

[5.3 Camel case的定义 22](#_Toc10210378)

[6、 编程实践 23](#_Toc10210379)

[6.1 @override 都应该使用 23](#_Toc10210380)

[6.2 异常捕获 不应该被忽略 23](#_Toc10210381)

[6.3 静态成员的访问：应该通过类，而不是对象 24](#_Toc10210382)

[6.4 不使用Finalizers 方法 25](#_Toc10210383)

[7、 Javadoc 25](#_Toc10210384)

[7.1 格式规范 25](#_Toc10210385)

[7.2 摘要片段 26](#_Toc10210386)

[7.3 何处应该使用Javadoc 26](#_Toc10210387)

## 介绍

本文档为Google Java编程规范的完整定义。依照此规范编写的Java源码文件可以被称为Google Style。

和其他编程规范指南一样，规范不仅包括了代码的结构美学，也包括了其他一些业界约定俗成的公约和普遍采用的标准。本文档中的规范基本都是业界已经达成共识的标准，并且尽量避免去定义那些还存在争议的地方。

### 1.1 术语说明

本文档除非特殊说明，否则：

* class（类）统指普通的class类型、enum枚举类型、interface类型和annotation类型。
* comment（注释）总是指implementation comments（实现注释，/\* \*/）。我们不使用“文档注释”这样的说法，而会直接说javadoc。

其他术语说明，将在文档中需要说明的地方单独说明。

### 1.2 文档说明

本文档中的代码并不一定符合所有规范。即使这些代码遵循Google Style，但这不是唯一的代码规范。例子中可选的格式风格也不应该作为强制执行的规范。

## 源码文件基础

使用Android Java类库和工具存在一些惯例。在某些时候，这些惯例会作出重大变化，可之前的代码也许会用到过时的模板或类库。如果用到这部分过时的代码，沿用已有的风格就是了。当创建新的组件时，就不要再使用过时的类库了。

### 2.1 文件名

源码文件名由它所包含的顶级class的类名（大小写敏感），加上.java后缀组成。（除了package-info.java文件）。

### 2.2 文件编码：UTF-8

源码文件使用UTF-8编码。

### 2.3 特殊字符

#### 2.3.1 空格字符

除了换行符外，ASCII水平空白字符（0x20）是源码文件中唯一支持的空格字符。这意味着：

* 其他空白字符将被转义。
* Tab字符不被用作缩进控制。

#### 2.3.2 特殊转义字符串

任何需要转义字符串表示的字符（例如\b, \t, \n, \f, \r, \', \\等），采用这种转义字符串的方式表示，而不采用对应字符的八进制数（例如 \012）或Unicode码（例如 \u000a）表示。

#### 2.3.3 非ASCII字符

对于其余非ASCII字符，直接使用Unicode字符（例如 ∞），或者使用对应的Unicode码（例如 \u221e）转义，都是允许的。唯一需要考虑的是，何种方式更能使代码容易阅读和理解。

注意：在使用unicode码转义，或者甚至是有时直接使用unicode字符的时候，添加一点说明注释将对别人读懂代码很有帮助。

例如：

|  |  |
| --- | --- |
| **Example** | **Discussion** |
| String unitAbbrev = "μs"; | Best: perfectly clear even without a comment. |
| String unitAbbrev = "\u03bcs"; // "μs" | Allowed, but there's no reason to do this. |
| String unitAbbrev = "\u03bcs"; // Greek letter mu, "s" | Allowed, but awkward and prone to mistakes. |
| String unitAbbrev = "\u03bcs"; | Poor: the reader has no idea what this is. |
| return '\ufeff' + content; // byte order mark | Good: use escapes for non-printable characters, and comment if necessary. |

注意：不要因为担心一些程序无法正常处理费ASCII字符而不使用它，从而导致代码易读性变差。如果出现这样的问题，应该由出现问题的程序去解决。

## 源码文件结构

源码文件按照先后顺序，由以下几部分组成：

* License或者copyright声明信息。（如果需要声明）
* 包声明语句
* import语句
* class类声明（每个源码文件只能有唯一一个顶级class）

每个部分之间应该只有一行空行作为间隔。

### 3.1 lincense 或者 copyright的声明信息

如果需要声明lincense或copyright信息，应该在文件开始时声明。

### 3.2 包声明

包声明的行，没有行长度的限制。单行长度限制（4.4部分有详细说明，80或100）不适用于包声明。

### 3.3 import语句

字段应该定义在文件开头，或者紧挨着使用这些字段的方法之前。

#### 3.3.1 不使用通配符import

不应该使用通配符import，不管是否是静态导入。

#### 3.3.2 没有行长度限制

import语句的行，没有行长度的限制。单行长度限制（4.4部分有详细说明，80或100）不适用于import语句所在行。

#### 3.3.3 顺序和空行

import语句应该被分为几个组，每个组之间由单行的空行隔开。分组的顺序如下：

* 所有的static import为归为一组。
* com.google 包的import归为一组。
* 使用的第三方包的引用。每个顶级第三方包归为一组。第三方包之间按ASCII码排序。例如：android, com, junit, org, sun
* java包归为一组。
* javax包归为一组。

同一组内的import语句之间不应用空行隔开。同一组中的import语句按ASCII码排序。

### 3.4 类声明

#### 3.4.1 只声明唯一一个顶级class

每个源码文件中只能有一个顶级class（package-info.java文件除外）。

#### 3.4.2 类成员顺序

类成员的顺序对代码的易读性有很大影响，但是没有一个统一正确的标准。不同的类可能有不同的排序方式。

重要的是，每个class都要按照一定的逻辑规律排序。当被问及时，能够解释清楚为什么这样排序。例如，新增加的成员方法，不是简单地放在class代码最后面，按日期排序不是按逻辑排序。

##### 3.4.2.1 重载方法：不应该分开

当一个类有多个构造函数，或者多个同名成员方法时，这些函数应该写在一起，不应该被其他成员分开。

## 格式

术语说明：块状结构（block-like construct）指类、成员函数和构造函数的实现部分（花括号中间部分）。注意，在后面的4.8.3.1节中讲到数组初始化，所有的数组初始化都可以被认为是一个块状结构（非强制）。

### 4.1 花括号

#### 4.1.1 花括号在需要的地方使用

花括号一般用在if, else, for, do, 和 while等语句。甚至当它的实现为空或者只有一句话时，也需要使用。

#### 4.1.2 非空语句块采用K&R风格

对于非空语句块，花括号遵循K&R风格：

* 左括号前不换行。
* 左括号后换行。
* 右括号前换行。
* 如果右括号结束一个语句块或者函数体、构造函数体或者有命名的类体，则需要换行。例如，当右括号后面接else或者逗号时，不应该换行。

例子：

return new MyClass() {

@Override public void method() {

if (condition()) {

try {

something();

} catch (ProblemException e) {

recover();

}

}

}

};

一些例外的情况，将在4.8.1节讲枚举类型的时候讲到。

#### 4.1.3 空语句块：使代码更简洁

一个空的语句块，可以在左花括号之后直接接右花括号，中间不需要空格或换行。但是当一个由几个语句块联合组成的语句块时，则需要换行。（例如：if/else-if/else try/catch/finally）.

例子：

void doNothing() {}

### 4.2 语句块的缩进：2空格

每当一个新的语句块产生，缩进就增加两个空格。当这个语句块结束时，缩进恢复到上一层级的缩进格数。缩进要求对整个语句块中的代码和注释都适用。（例子可参考之前4.1.2节中的例子）。

### 4.3 一行最多只有一句代码

每句代码的结束都需要换行。

### 4.4 行长度限制：80或100

不同的项目可以选择采用80个字符或者100个字符作为限制。除了以下几个特殊情况外，其他代码内容都需要遵守这个长度限制。这在4.5节会有详细解释。

例外：

* 按照行长度限制，无法实现地方（例如：javadoc中超长的URL地址， 或者一个超长的JSNI方法的引用）；
* package和import语句不受长度限制。（见3.2、3.3节）；
* 注释中的命令行指令行，将被直接复制到shell中执行的。

### 4.5 长行断行

术语说明：当一行代码按照其他规范都合法，只是为了避免超出行长度限制而换行时，称为长行断行。

长行断行，没有一个适合所有场景的全面、确定的规范。但很多相同的情况，我们经常使用一些行之有效的断行方法。

注意：将长行封装为函数，或者使用局部变量的方法，也可以解决一些超出行长度限制的情况。并非一定要断行。

#### 4.5.1 在何处断行

断行的主要原则是：选择在更高一级的语法逻辑的地方断行。其他一些原则如下：

* 当一个非赋值运算的语句断行时，在运算符号之前断行。（这与Google的C++规范和JavaScrip规范等其他规范不同）。
* 当一个赋值运算语句断行时，一般在赋值符号之后断行。但是也可以在之前断行。
* 在调用函数或者构造函数需要断行时，与函数名相连的左括号要在一行。也就是在左括号之后断行。
* 逗号断行时，要和逗号隔开的前面的语句断行。也就是在逗号之后断行。

#### 4.5.2 断行的缩进：至少4个字符

当断行之后，在第一行之后的行，我们叫做延续行。每一个延续行在第一行的基础上至少缩进四个字符。

当原行之后有多个延续行的情况，缩进可以大于4个字符。如果多个延续行之间由同样的语法元素断行，它们可以采用相同的缩进。

4.6.3节介绍水平对齐中，解决了使用多个空格与之前行缩进对齐的问题。

### 4.6 空白空间

#### 4.6.1 垂直空白

单行空行在以下情况使用：

* 类成员间需要空行隔开：例如成员变量、构造函数、成员函数、内部类、静态初始化语句块（static initializers）、实例初始化语句块（instance initializers）。例外：成员变量之间的空白行不是必需的。一般多个成员变量中间的空行，是为了对成员变量做逻辑上的分组。
* 在函数内部，根据代码逻辑分组的需要，设置空白行作为间隔。
* 类的第一个成员之前，或者最后一个成员结束之后，用空行间隔。（可选）
* 本文档中其他部分介绍的需要空行的情况。（例如 3.3节中的import语句）

单空行时使用多行空行是允许的，但是不要求也不鼓励。

#### 4.6.2 水平空白

除了语法、其他规则、词语分隔、注释和javadoc外，水平的ASCII空格只在以下情况出现：

* 所有保留的关键字与紧接它之后的位于同一行的左括号之间需要用空格隔开。（例如if、for、catch）
* 所有保留的关键字与在它之前的右花括号之间需要空格隔开。（例如else、catch）
* 在左花括号之前都需要空格隔开。只有两种例外：

@SomeAnnotation({a, b})

String[][] x = {{"foo"}};

* 所有的二元运算符和三元运算符的两边，都需要空格隔开。
* 逗号、冒号、分号和右括号之后，需要空格隔开。
* // 双斜线开始一行注释时。双斜线两边都应该用空格隔开。并且可使用多个空格，但是不做强制要求。
* 变量声明时，变量类型和变量名之间需要用空格隔开。
* 初始化一个数组时，花括号之间可以用空格隔开，也可以不使用。（例如：new int[] {5, 6} 和 new int[] { 5, 6 } 都可以）

注意：这一原则不影响一行开始或者结束时的空格。只针对行内部字符之间的隔开。

#### 4.6.3 水平对齐：不做强制要求

术语说明：水平对齐，是指通过添加多个空格，使本行的某一符号与上一行的某一符号上下对齐。

这种对齐是被允许的，但是不会做强制要求。

以下是没有水平对齐和水平对齐的例子;

private int x; // this is fine

private Color color; // this too

private int x; // permitted, but future edits

private Color color; // may leave it unaligned

注意：水平对齐能够增加代码的可读性，但是增加了未来维护代码的难度。考虑到维护时只需要改变一行代码，之前的对齐可以不需要改动。为了对齐，你更有可能改了一行代码，同时需要更改附近的好几行代码，而这几行代码的改动，可能又会引起一些为了保持对齐的代码改动。那原本这行改动，我们称之为“爆炸半径”。这种改动，在最坏的情况下可能会导致大量的无意义的工作，即使在最好的情况下，也会影响版本历史信息，减慢代码review的速度，引起更多merge代码冲突的情况。

### 4.7 分组括号：建议使用

非必须的分组括号只有在编写代码者和代码审核者都认为大家不会因为没有它而导致代码理解错误的时候，或者它不会使代码更易理解的时候才能省略。没有理由认为所有阅读代码的人都能记住所有java运算符的优先级。

### 4.8 特殊结构

#### 4.8.1 枚举类型

每个逗号后接一个枚举变量，不要求换行。

枚举类型，如果没有函数和javadoc，处理格式是可以按照数组初始化来处理。

例子：

private enum Suit { CLUBS, HEARTS, SPADES, DIAMONDS }

枚举类型也是一种类（Class），因此Class类的其他格式要求，也适用于枚举类型。

#### 4.8.2 变量声明

##### 4.8.2.1 每次声明一个变量

不要采用一个声明，声明多个变量。例如 int a, b;

##### 4.8.2.2 当需要时才声明，尽快完成初始化

局部变量不应该习惯性地放在语句块的开始处声明，而应该尽量离它第一次使用的地方最近的地方声明，以减小它们的使用范围。

局部变量应该在声明的时候就进行初始化。如果不能在声明时初始化，也应该尽快完成初始化。

#### 4.8.3 数组

##### 4.8.3.1 数组初始化：可以类似块代码处理

所有数组的初始化，都可以采用和块代码相同的格式处理。例如以下格式都是允许的：

new int[] { new int[] {

0, 1, 2, 3 0,

} 1,

2,

new int[] { 3,

0, 1, }

2, 3

} new int[]

{0, 1, 2, 3}

##### 4.8.3.2 不能像C风格一样声明数组

方括号应该是变量类型的一部分，因此不应该和变量名放在一起。例如：应该是String[] args，而不是 String args[] 。

#### 4.8.4 switch语句

术语说明：switch语句是指在switch花括号中，包含了一组或多组语句块。每组语句块都由一个或多个switch标签（例如case FOO：或者 default：）打头。

##### 4.8.4.1 缩进

和其他语句块一样，switch花括号之后缩进两个字符。

每个switch标签之后，后面紧接的非标签的新行，按照花括号相同的处理方式缩进两个字符。在标签结束后，恢复到之前的缩进，类似花括号结束。

##### 4.8.4.2 继续向下执行的注释

在switch语句中，每个标签对应的代码执行完后，都应该通过语句结束（例如：break、continue、return 或抛出异常），否则应该通过注释说明，代码需要继续向下执行下一个标签的代码。注释说明文字只要能说明代码需要继续往下执行都可以（通常是 //fall through）。这个注释在最后一个标签之后不需要注释。例如：

switch (input) {

case 1:

case 2:

prepareOneOrTwo();

// fall through

case 3:

handleOneTwoOrThree();

break;

default:

handleLargeNumber(input);

}

##### 4.8.4.3 default标签需要显式声明

每个switch语句中，都需要显式声明default标签。即使没有任何代码也需要显示声明。

#### 4.8.5 Annotations

Annotations应用到类、函数或者构造函数时，应紧接javadoc之后。每一行只有一个Annotations。

Annotations所在行不受行长度限制，也不需要增加缩进。例如：

@Override

@Nullable

public String getNameIfPresent() { ... }

例外情况：如果Annotations只有一个，并且不带参数。则它可以和类或方法名放在同一行，例如：

@Override public int hashCode() { ... }

Annotations应用到成员变量时，也是紧接javadoc之后。不同的是，多个annotations可以放在同一行。例如：

@Partial @Mock DataLoader loader;

对于参数或者局部变量使用Annotations的情况，没有特定的规范。

#### 4.8.6 注释

##### 4.8.6.1 语句块的注释风格

注释的缩进与它所注释的代码缩进相同。可以采用 /\* \*/ 进行注释，也可以用 // 进行注释。当使用 /\*\*/ 进行多行注释时，每一行都应该以 \* 开始， 并且 \* 应该上下对齐。

例如：

/\*

\* This is // And so /\* Or you can

\* okay. // is this. \* even do this. \*/

\*/

多行注释时，如果你希望集成开发环境能自动对齐注释，你应该使用 /\*\*/， //一般不会自动对齐。

#### 4.8.7 修饰符

多个类和成员变量的修饰符，按Java Lauguage Specification中介绍的先后顺序排序。具体是：

public protected private abstract static final transient volatile synchronized native strictfp

## 命名

### 5.1 适用于所有命名标识符的通用规范

### 5.2 不同类型的标示符规范

#### 5.2.1 包名

包名全部用小写字母，通过 . 将各级连在一起。不应该使用下划线。

#### 5.2.2 类名

类型的命名，采用以大写字母开头的大小写字符间隔的方式（UpperCamelCase）。

class命名一般使用名词或名词短语。interface的命名有时也可以使用形容词或形容词短语。annotation没有明确固定的规范。

测试类的命名，应该以它所测试的类的名字为开头，并在最后加上Test结尾。例如：HashTest 、 HashIntegrationTest。

#### 5.2.3 方法名

方法命名，采用以小写字母开头的大小写字符间隔的方式（lowerCamelCase）。

方法命名一般使用动词或者动词短语。

在JUnit的测试方法中，可以使用下划线，用来区分测试逻辑的名字，经常使用如下的结构：test<MethodUnderTest>\_<state> 。例如：testPop\_emptyStack 。

测试方法也可以用其他方式进行命名。

#### 5.2.4 常量名

常量命名，全部使用大写字符，词与词之间用下划线隔开。（CONSTANCE\_CASE）。

常量是一个静态成员变量，但不是所有的静态成员变量都是常量。在选择使用常量命名规则给变量命名时，你需要明确这个变量是否是常量。例如，如果这个变量的状态可以发生改变，那么这个变量几乎可以肯定不是常量。只是计划不会发生改变的变量不足以成为一个常量。下面是常量和非常量的例子：

// Constants

static final int NUMBER = 5;

static final ImmutableList<String> NAMES = ImmutableList.of("Ed", "Ann");

static final Joiner COMMA\_JOINER = Joiner.on(','); // because Joiner is immutable

static final SomeMutableType[] EMPTY\_ARRAY = {};

enum SomeEnum { ENUM\_CONSTANT }

// Not constants

static String nonFinal = "non-final";

final String nonStatic = "non-static";

static final Set<String> mutableCollection = new HashSet<String>();

static final ImmutableSet<SomeMutableType> mutableElements = ImmutableSet.of(mutable);

static final Logger logger = Logger.getLogger(MyClass.getName());

static final String[] nonEmptyArray = {"these", "can", "change"};

常量一般使用名词或者名词短语命名。

#### 5.2.5 非常量的成员变量名

非常量的成员变量命名（包括静态变量和非静态变量），采用lowerCamelCase命名。

一般使用名词或名词短语。

#### 5.2.6 参数名

参数命名采用lowerCamelCase命名。

应该避免使用一个字符作为参数的命名方式。

#### 5.2.7 局部变量名

局部变量采用lowerCamelCase命名。它相对于其他类型的命名，可以采用更简短宽松的方式。

但即使如此，也应该尽量避免采用单个字母进行命名的情况，除了在循环体内使用的临时变量。

即使局部变量是final、不可改变的，它也不能被认为是常量，也不应该采用常量的命名方式去命名。

#### 5.2.8 类型名

类型名有两种命名方式：

* 单独一个大写字母，有时后面再跟一个数字。（例如，E、T、X、T2）。
* 像一般的class命名一样（见5.2.2节），再在最后接一个大写字母。（例如，RequestT、FooBarT）。

### 5.3 Camel case的定义

有时一些短语被写成Camel case的时候可以有多种写法。例如一些缩写词汇，或者一些组合词：IPv6 或者 iOS 等。

为了统一写法，Google style给出了一种几乎可以确定为一种的写法。

* 将字符全部转换为ASCII字符，并且去掉 ' 等符号。例如，"Müller's algorithm" 被转换为 "Muellers algorithm" 。
* 将上一步转换的结果拆分成一个一个的词语。从空格处和从其他剩下的标点符号处划分。

注意：一些已经是Camel case的词语，也应该在这个时候被拆分。（例如 AdWords 被拆分为 ad words）。但是例如iOS之类的词语，它其实不是一个Camel case的词语，而是人们惯例使用的一个词语，因此不用做拆分。

* 经过上面两部后，先将所有的字母转换为小写，再把每个词语的第一个字母转换为大写。
* 最后，将所有词语连在一起，形成一个标示符。

注意：词语原来的大小写规则，应该被完全忽略。以下是一些例子：

|  |  |  |
| --- | --- | --- |
| **Prose form** | **Correct** | **Incorrect** |
| "XML HTTP request" | XmlHttpRequest | XMLHTTPRequest |
| "new customer ID" | newCustomerId | newCustomerID |
| "inner stopwatch" | innerStopwatch | innerStopWatch |
| "supports IPv6 on iOS?" | supportsIpv6OnIos | supportsIPv6OnIOS |
| "YouTube importer" | YouTubeImporter YoutubeImporter\* |  |

\* 号表示可以接受，但是不建议使用。

注意，有些词语在英文中，可以用 - 连接使用，也可以不使用 - 直接使用。例如 “nonempty”和 “non-empty”都是可以的。因此，方法名字为checkNonempty 或者checkNonEmpty 都是可以的。

## 编程实践

### 6.1 @override 都应该使用

@override annotations只要是符合语法的，都应该使用。

### 6.2 异常捕获 不应该被忽略

一般情况下，catch住的异常不应该被忽略，而是都需要做适当的处理。例如将错误日志打印出来，或者如果认为这种异常不会发生，则应该作为断言异常重新抛出。

如果这个catch住的异常确实不需要任何处理，也应该通过注释做出说明。例如：

try {

int i = Integer.parseInt(response);

return handleNumericResponse(i);

} catch (NumberFormatException ok) {

// it's not numeric; that's fine, just continue

}

return handleTextResponse(response);

例外：在测试类里，有时会针对方法是否会抛出指定的异常，这样的异常是可以被忽略的。但是这个异常通常需要命名为： expected。例如：

try {

emptyStack.pop();

fail();

} catch (NoSuchElementException expected) {

}

### 6.3 静态成员的访问：应该通过类，而不是对象

当一个静态成员被访问时，应该通过class名去访问，而不应该使用这个class的具体实例对象。例如：

Foo aFoo = ...;

Foo.aStaticMethod(); // good

aFoo.aStaticMethod(); // bad

somethingThatYieldsAFoo().aStaticMethod(); // very bad

### 6.4 不使用Finalizers 方法

重载Object的finalize方法是非常非常罕见的。

注意：不应该使用这以方法。如果你认为你必须使用，请先仔细阅读并理解 Effective Java 第七条 “Avoid Finalizers”。然后不要使用它。

## Javadoc

### 7.1 格式规范

#### 7.1.1 通用格式

最基本的javadoc的通用格式如下例：

/\*\*

\* Multiple lines of Javadoc text are written here,

\* wrapped normally...

\*/

public int method(String p1) { ... }

或者为单行格式：

/\*\* An especially short bit of Javadoc. \*/

通用格式在任何时候使用都是可以的。当javadoc块只有一行时，可以使用单行格式来替代通用格式。

#### 7.1.2 段落

空白行：是指javadoc中，上下两个段落之间只有上下对齐的 \* 字符的行。每个段落的第一行在第一个字符之前，有一个<p>标签，并且之后不要有任何空格。

#### 7.1.3 @从句

所有标准的@从句，应该按照如下的顺序添加：@param、@return、@throws、@deprecated。并且这四种@从句，不应该出现在一个没有描述的Javadoc块中。

当@从句无法在一行写完时，应该断行。延续行在第一行的@字符的位置，缩进至少4个字符单位。

### 7.2 摘要片段

每个类或者成员的javadoc，都是由一个摘要片段开始的。这个片段非常重要。因为它是类或者方法在使用时唯一能看到的文本说明。

主要摘要只是一个片段，应该是一个名词短语或者动词短语，而不应该是一个完整的句子。但是它应该像一个完整的句子一样使用标点符号。

注意：一种常见的错误是以这种形式使用javadoc：/\*\* @return the customer ID \*/.这是不对的。应该改为：/\*\* Returns the customer ID. \*/.

### 7.3 何处应该使用Javadoc

至少，Javadoc应该应用于所有的public类、public和protected的成员变量和方法。和少量例外的情况。例外情况如下。

#### 7.3.1 例外：方法本身已经足够说明的情况

当方法本身很显而易见时，可以不需要javadoc。例如：getFoo。没有必要加上javadoc说明“Returns the foo”。

单元测试中的方法基本都能通过方法名，显而易见地知道方法的作用。因此不需要增加javadoc。

注意：有时候不应该引用此例外，来省略一些用户需要知道的信息。例如：getCannicalName 。当大部分代码阅读者不知道canonical name是什么意思时，不应该省略Javadoc,认为只能写/\*\* Returns the canonical name. \*/ 。

#### 7.3.2 例外：重载方法

重载方法有时不需要再写Javadoc。

#### 7.3.3 例外：可选的javadoc

一些在包外不可见的class和成员变量或方法，根据需要，也可以使用javadoc。当一个注释用以说明这个类、变量或者方法的总体目标或行为时，可以使用Javadoc。