javadoc 注释规范

javadoc 做注释
一. Java 文档

// 注释一行
/* ..... */ 注释若干行
/** ..... */ 注释若干行，并写入 javadoc 文档

通常这种注释的多行写法如下：

```
/**
* .........
* .........
*/
```

javadoc -d 文档存放目录 -author -version 源文件名.java
这条命令编译一个名为 “源文件名.java”的 java 源文件，并将生成的文档存放在“文档存放目录”指定的目录下，生成的文档中 index.html 就是文档的首页。-author 和 -version 两个选项可以省略。

二. 文档注释的格式

1. 文档和文档注释的格式化

生成的文档是 HTML 格式，而这些 HTML 格式的标识符并不是 javadoc 加的，而是我们在写注释的时候写上去的。
比如，需要换行时，不是敲入一个回车符，而是写入
，如果要分段，就应该在段前写入 <p>。
文档注释的正文并不是直接复制到输出文件（文档的 HTML 文件），而是读取每一行后，删掉前导的 * 号及 * 号以前的空格，再输入到文档的。如

```
/**
* This is first line. <br>
***** This is second line. <br>
This is third line.
*/
```

2. 文档注释的三部分
先举例如下

```
/**
```

```
* show 方法的简述.
* <p>show 方法的详细说明第一行<br>
* show 方法的详细说明第二行
* @param b true 表示显示，false 表示隐藏
* @return 没有返回值
*/
public void show(boolean b) {
frame.show(b);
}
```

第一部分是简述。文档中，对于属性和方法都是先有一个列表，然后才在后面一个一个的详细的说明
简述部分写在一段文档注释的最前面，第一个点号 (.) 之前 (包括点号)。换句话说，就是用第一个点号分隔文档注释，之前是简述，之后是第二部分和第三部分。

第二部分是详细说明部分。该部分对属性或者方法进行详细的说明，在格式上没有什么特殊的要求，可以包含若干个点号。

```
* show 方法的简述.
* <p>show 方法的详细说明第一行<br>
* show 方法的详细说明第二行
```

简述也在其中。这一点要记住了

第三部分是特殊说明部分。这部分包括版本说明、参数说明、返回值说明等。

```
* @param b true 表示显示，false 表示隐藏
* @return 没有返回值
```

三. 使用 javadoc 标记
javadoc 标记由“@”及其后所跟的标记类型和专用注释引用组成
javadoc 标记有如下一些：
@author 标明开发该类模块的作者
@version 标明该类模块的版本
@see 参考转向，也就是相关主题
@param 对方法中某参数的说明
@return 对方法返回值的说明
@exception 对方法可能抛出的异常进行说明

@author 作者名
@version 版本号
其中，@author 可以多次使用，以指明多个作者，生成的文档中每个作者之间使用逗号 (,) 隔开。@version 也可以使用多次，只有第一次有效

使用 @param、@return 和 @exception 说明方法

这三个标记都是只用于方法的。@param 描述方法的参数，@return 描述方法的返回值，@exception 描述方法可能抛出的异常。它们的句法如下：
@param 参数名 参数说明
@return 返回值说明
@exception 异常类名 说明

四. javadoc 命令
用法：
    javadoc [options] [packagenames] [sourcefiles]

选项：

-public 仅显示 public 类和成员
-protected 显示 protected/public 类和成员 (缺省)
-package 显示 package/protected/public 类和成员
-private 显示所有类和成员
-d <directory> 输出文件的目标目录
-version 包含 @version 段
-author 包含 @author 段
-splitindex 将索引分为每个字母对应一个文件
-windowtitle <text> 文档的浏览器窗口标题

javadoc 编译文档时可以给定包列表，也可以给出源程序文件列表。例如在 CLASSPATH 下有两个包若干类如下：

    fancy.Editor
    fancy.Test
    fancy.editor.ECommand
    fancy.editor.EDocument
    fancy.editor.EView

可以直接编译类：
javadoc fancy\Test.java fancy\Editor.java fancy\editor\ECommand.java fancy\editor\EDocument.java fancy\editor\EView.java

也可以是给出包名作为编译参数，如：javadoc fancy fancy.editor
可以自己看看这两种方法的区别

到此为止 javadoc 就简单介绍完了，想要用好她还是要多用，多参考标准 java 代码
Java 代码规范
--注释

@author LEI

@version 1.10 2005-09-01
1 注释文档的格式

注释文档将用来生成 HTML 格式的代码报告，所以注释文档必须书写在类、域、构造函数、方法、定义之前。注释文档由两部分组成——描述、块标记。

例如：

```
/**

* The doGet method of the servlet.

* This method is called when a form has its tag value method equals to get.

*

* @param request

* the request send by the client to the server

* @param response

* the response send by the server to the client

* @throws ServletException

* if an error occurred

* @throws IOException

* if an error occurred

*/

public void doGet (HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

doPost(request, response);
```

```
}
```

前两行为描述，描述完毕后，由@符号起头为块标记注视。

2 注释的种类

2.1 文件头注释

文件头注释以 /*开始，以*/结束，需要注明该文件创建时间，文件名，命名空间信息。

例如：

```
/*

* Created on 2005-7-2

* /
```

2.2 类、接口注释

类、接口的注释采用 /** … */，描述部分用来书写该类的作用或者相关信息，块标记部分必须注明作者和版本。

例如：

```
/**Title: XXXX DRIVER 3.0
*Description: XXXX DRIVER 3.0
*Copyright: Copyright (c) 2003
*Company:XXXX 有限公司
*
* @author Java Development Group
* @version 3.0
*/
```

例如：

```
/**
* A class representing a window on the screen.
* For example:
*
* Window win = new Window(parent);
* win.show();
*
*
* @author Sami Shaio
* @version %I%, %G%
```

```
* @see java.awt.BaseWindow
* @see java.awt.Button
*/

class Window extends BaseWindow {

...

}
```

2.3 构造函数注释

构造函数注释采用 /** … */，描述部分注明构造函数的作用，不一定有块标记部分。

例如：

```
/**

* 默认构造函数

*/
```

有例如：

```
/**

* 带参数构造函数,初始化模式名,名称和数据源类型

*

* @param schema

* Ref 模式名

* @param name

* Ref 名称

* @param type

* byVal 数据源类型

*/
```

2.4 域注释

域注释可以出现在注释文档里面，也可以不出现在注释文档里面。用/** … */的域注释将会被认为是注释文档热出现在最终生成的 HTML 报告里面，而使用/* … */的注释会被忽略。

例如：

```
/* 由于 triger 和表用一个 DMSource，所以要区分和表的迁移成功标记 */

boolean isTrigerSuccess = false;
```

又例如：

```
/** 由于 triger 和表用一个 DMSource，所以要区分和表的迁移成功标记 */

boolean isTrigerSuccess = false;
```

再例如：

```
/**

* The X-coordinate of the component.

*

* @see #getLocation()

*/

int x = 1263732;
```

2.5 方法注释

方法注释采用 /** … */，描述部分注明方法的功能，块标记部分注明方法的参数，返回值，异常等信息。例如：

```
/**

* 设置是否有外码约束

*

* @param conn
```

* Connection 与数据库的连接

*/
2.6 定义注释

规则同域注释。
3 注释块标记
3.1 标记的顺序

块标记将采用如下顺序：

…

*

* @param (classes, interfaces, methods and constructors only)

* @return (methods only)

* @exception (@throws is a synonym added in Javadoc 1.2)

* @author (classes and interfaces only, required)

* @version (classes and interfaces only, required. See footnote 1)

* @see

* @since

* @serial (or @serialField or @serialData)

* @deprecated (see How and When To Deprecate APIs)

* …

一个块标记可以根据需要重复出现多次，多次出现的标记按照如下顺序：

@author 按照时间先后顺序（chronological）

@param 按照参数定义顺序（declaration）

@throws 按照异常名字的字母顺序（alphabetically）

@see 按照如下顺序：

@see #field

@see #Constructor(Type, Type...)

@see #Constructor(Type id, Type id...)

@see #method(Type, Type,...)

@see #method(Type id, Type, id...)

@see Class

@see Class#field

@see Class#Constructor(Type, Type...)

@see Class#Constructor(Type id, Type id)

@see Class#method(Type, Type,...)

@see Class#method(Type id, Type id,...)

@see package.Class

@see package.Class#field

@see package.Class#Constructor(Type, Type...)

@see package.Class#Constructor(Type id, Type id)

@see package.Class#method(Type, Type,...)

@see package.Class#method(Type id, Type, id)

@see package

3.2 标记介绍

3.2.1 @param 标记

@param 后面空格后跟着参数的变量名字（不是类型），空格后跟着对该参数的描述。

在描述中第一个名字为该变量的数据类型，表示数据类型的名次前面可以有一个冠词如：a,an,the。如果是 int 类型的参数则不需要注明数据类型。例如：

…

* @param ch the char 用用来……

* @param _image the image 用来……

* @param _num 一个数字……

…

对于参数的描述如果只是一短语，最好不要首字母大写，结尾也不要句号。

对于参数的描述是一个句子，最好不要首字母大写，如果出现了句号这说明你的描述不止一句话。如果非要首字母大写的话，必须用句号来结束句子。（英文的句号）

公司内部添加 ByRef 和 ByVal 两个标记，例如：

* @param _image the image ByRef 用来……

说明该参数是引用传递（指针），ByVal 可以省略，表示是值传递。

3.2.2 @return 标记

返回为空（void）的构造函数或者函数，@return 可以省略。

如果返回值就是输入参数，必须用与输入参数的@param 相同的描述信息。

必要的时候注明特殊条件写的返回值。

3.2.3 @throws 标记

@throws 以前使用的是@exception。

@throws 的内容必须在函数的 throws 部分定义。

3.2.4 @author 标记

类注释标记。

函数注释里面可以不出现@author。

3.2.5 @version

类注释标记。

函数注释里面可以不出现@version

3.2.6 @since

类注释标记。

标明该类可以运行的 JDK 版本

例如：

@since JDK1.2
3.2.7 @deprecated

由于某种原因而被宣布将要被废弃的方法。

/**

* @deprecated As of JDK 1.1, replaced by

* setBounds

* @see #setBounds(int,int,int,int)

*/
3.2.8 @link 标记

语法：{@link package.class#member label}

Label 为链接文字。

package.class#member 将被自动转换成指向 package.class 的 member 文件的 URL。
4 HTML 代码的使用

在注释描述部分可以使用 HTML 代码。

…
表示段落

* ….

表示自动标号
5 注释示例

/**

* Graphics is the abstract base class for all graphics contexts
* which allow an application to draw onto components realized on
* various devices or onto off-screen images.
* A Graphics object encapsulates the state information needed
* for the various rendering operations that Java supports. This
* state information includes:
*
* The Component to draw on
* A translation origin for rendering and clipping coordinates
* The current clip
* The current color
* The current font
* The current logical pixel operation function (XOR or Paint)
* The current XOR alternation color
* (see setXORMode)
*
*
* Coordinates are infinitely thin and lie between the pixels of the
* output device.
* Operations which draw the outline of a figure operate by traversing
* along the infinitely thin path with a pixel-sized pen that hangs
* down and to the right of the anchor point on the path.

```
* Operations which fill a figure operate by filling the interior
* of the infinitely thin path.
* Operations which render horizontal text render the ascending
* portion of the characters entirely above the baseline coordinate.
*
* Some important points to consider are that drawing a figure that
* covers a given rectangle will occupy one extra row of pixels on
* the right and bottom edges compared to filling a figure that is
* bounded by that same rectangle.
* Also, drawing a horizontal line along the same y coordinate as
* the baseline of a line of text will draw the line entirely below
* the text except for any descenders.
* Both of these properties are due to the pen hanging down and to
* the right from the path that it traverses.
*
* All coordinates which appear as arguments to the methods of this
* Graphics object are considered relative to the translation origin
* of this Graphics object prior to the invocation of the method.
* All rendering operations modify only pixels which lie within the
* area bounded by both the current clip of the graphics context
* and the extents of the Component used to create the Graphics object.
*
```

```
* @author Sami Shaio
* @author Arthur van Hoff
* @version %I%, %G%
* @since 1.0
*/
public abstract class Graphics {
/**
* Draws as much of the specified image as is currently available
* with its northwest corner at the specified coordinate (x, y).
* This method will return immediately in all cases, even if the
* entire image has not yet been scaled, dithered and converted
* for the current output device.
*
* If the current output representation is not yet complete then
* the method will return false and the indicated
* {@link ImageObserver} object will be notified as the
* conversion process progresses.
*
* @param img the image to be drawn
* @param x the x-coordinate of the northwest corner
* of the destination rectangle in pixels
* @param y the y-coordinate of the northwest corner
```

```
* of the destination rectangle in pixels
* @param observer the image observer to be notified as more
* of the image is converted. May be
* null
* @return true if the image is completely
* loaded and was painted successfully;
* false otherwise.
* @see Image
* @see ImageObserver
* @since 1.0
*/
public abstract boolean drawImage(Image img, int x, int y,
ImageObserver observer);
/**
* Dispose of the system resources used by this graphics context.
* The Graphics context cannot be used after being disposed of.
* While the finalization process of the garbage collector will
* also dispose of the same system resources, due to the number
* of Graphics objects that can be created in short time frames
* it is preferable to manually free the associated resources
* using this method rather than to rely on a finalization
* process which may not happen for a long period of time.
```

```
*
* Graphics objects which are provided as arguments to the paint
* and update methods of Components are automatically disposed
* by the system when those methods return. Programmers should,
* for efficiency, call the dispose method when finished using
* a Graphics object only if it was created directly from a
* Component or another Graphics object.
*
* @see #create(int, int, int, int)
* @see #finalize()
* @see Component#getGraphics()
* @see Component#paint(Graphics)
* @see Component#update(Graphics)
* @since 1.0
*/
public abstract void dispose();
/**
* Disposes of this graphics context once it is no longer
* referenced.
*
* @see #dispose()
* @since 1.0
```

```
*/
public void finalize() {
dispose();
}
}
```