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.dass#member label}
Label 为链接文字。
package.dass#member 将被自动转换成指向 package.dass 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 dass 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 dip
*
- 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
```

Eclipse 快捷键大全(转载)

Ctrl+1 快速修复(最经典的快捷键,就不用多说了)

Ctrl+D: 删除当前行

Ctrl+Alt+↓ 复制当前行到下一行(复制增加)

Ctrl+Alt+↑ 复制当前行到上一行(复制增加)

Alt+↓ 当前行和下面一行交互位置(特别实用,可以省去先剪切,再粘贴了)

Alt+↑ 当前行和上面一行交互位置(同上)

```
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
```

Alt+← 前一个编辑的页面

Alt+→ 下一个编辑的页面(当然是针对上面那条来说了)

Alt+Enter 显示当前选择资源(工程,or 文件 or 文件)的属性

Shift+Enter 在当前行的下一行插入空行(这时鼠标可以在当前行的任一位置,不一定是最后)

Shift+Ctrl+Enter 在当前行插入空行(原理同上条)

Ctrl+Q 定位到最后编辑的地方

```
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();
}
}
```

Ctrl+L 定位在某行 (对于程序超过 100 的人就有福音了)

Ctrl+M 最大化当前的 Edit 或 View (再按则反之)

Ctrl+/ 注释当前行,再按则取消注释

Ctrl+O 快速显示 OutLine

Ctrl+T 快速显示当前类的继承结构

Ctrl+W 关闭当前 Editer

Ctrl+K 参照选中的 Word 快速定位到下一个

Ctrl+E 快速显示当前 Editer 的下拉列表

(如果当前页面没有显示的用黑体表示)
Ctrl+/(小键盘) 折叠当前类中的所有代码
Ctrl+×(小键盘) 展开当前类中的所有代码
Ctrl+Space 代码助手完成一些代码的插入(但一般和输入法有冲突,可以修改输入法的热键,也可以暂用 Alt+/来代替)
Ctrl+Shift+E 显示管理当前打开的所有的 View 的管理器(可以选择关闭,激活等操作)
Ctrl+J 正向增量查找(按下 Ctrl+J 后,你所输入的每个字母编辑器都提供快速匹配定位到某个单词,如果没有,则在 stutes line 中显示没有找到了,查一个单词时,特别实用,这个功能 Idea 两年前就有了)
Ctrl+Shift+J 反向增量查找(和上条相同,只不过是从后往前查)
Ctrl+Shift+F4 关闭所有打开的 Editer
Ctrl+Shift+X 把当前选中的文本全部变味小写
Ctrl+Shift+Y 把当前选中的文本全部变为小写
Ctrl+Shift+F 格式化当前代码
Ctrl+Shift+P 定位到对于的匹配符(譬如{}) (从前面定位后面时,光标要在匹配符里面,后面到前面,则反之)

下面的快捷键是重构里面常用的,本人就自己喜欢且常用的整理一下(注:一般重构的快捷键都是 Alt+Shift 开头的了)
Alt+Shift+R 重命名 (是我自己最爱用的一个了,尤其是变量和类的 Rename,比手工方法能节省很多劳动力)
Alt+Shift+M 抽取方法 (这是重构里面最常用的方法之一了,尤其是对一大堆泥团代码有用)
Alt+Shift+C 修改函数结构(比较实用,有 N 个函数调用了这个方法,修改一次搞定)
Alt+Shift+L 抽取本地变量(可以直接把一些魔法数字和字符串抽取成一个变量,尤其是多处调用的时候)
Alt+Shift+F 把 Class 中的 local 变量变为 field 变量(比较实用的功能)
Alt+Shift+I 合并变量(可能这样说有点不妥 Inline)
Alt+Shift+V 移动函数和变量(不怎么常用)
Alt+Shift+Z 重构的后悔药(Undo)

编辑
作用域 功能 快捷键
全局 查找并替换 Ctrl+F
文本编辑器 查找上一个 Ctrl+Shift+K
文本编辑器 查找下一个 Ctrl+K
全局 撤销 Ctrl+Z
全局 复制 Ctrl+C
全局 恢复上一个选择 Alt+Shift+↓
全局 剪切 Ctrl+X
全局 快速修正 Ctrl1+1
全局 内容辅助 Alt+/
全局 全部选中 Ctrl+A
全局 删除 Delete
全局 上下文信息 Alt+?
Alt+Shift+?
Ctrl+Shift+Space
Java 编辑器 显示工具提示描述 F2
Java 编辑器 选择封装元素 Alt+Shift+↑
Java 编辑器 选择上一个元素 Alt+Shift+←
Java 编辑器 选择下一个元素 Alt+Shift+→
文本编辑器 增量查找 Ctrl+J
文本编辑器 增量逆向查找 Ctrl+Shift+J
全局 粘贴 Ctrl+V
全局 重做 Ctrl+Y

查看
作用域 功能 快捷键
全局 放大 Ctrl+=
全局 缩小 Ctrl+-

窗口
作用域 功能 快捷键
全局 激活编辑器 F12
全局 切换编辑器 Ctrl+Shift+W
全局 上一个编辑器 Ctrl+Shift+F6
全局 上一个视图 Ctrl+Shift+F7
全局 上一个透视图 Ctrl+Shift+F8
全局 下一个编辑器 Ctrl+F6
全局 下一个视图 Ctrl+F7
全局 下一个透视图 Ctrl+F8
文本编辑器 显示标尺上下文菜单 Ctrl+W
全局 显示视图菜单 Ctrl+F10
全局 显示系统菜单 Alt+-

导航
作用域 功能 快捷键
Java 编辑器 打开结构 Ctrl+F3
全局 打开类型 Ctrl+Shift+T
全局 打开类型层次结构 F4
全局 打开声明 F3
全局 打开外部 javadoc Shift+F2
全局 打开资源 Ctrl+Shift+R
全局 后退历史记录 Alt+←
全局 前进历史记录 Alt+→
全局 上一个 Ctrl+,
全局 下一个 Ctrl+.
Java 编辑器 显示大纲 Ctrl+O
全局 在层次结构中打开类型 Ctrl+Shift+H
全局 转至匹配的括号 Ctrl+Shift+P
全局 转至上一个编辑位置 Ctrl+Q
Java 编辑器 转至上一个成员 Ctrl+Shift+↑
Java 编辑器 转至下一个成员 Ctrl+Shift+↓
文本编辑器 转至行 Ctrl+L

搜索
作用域 功能 快捷键
全局 出现在文件中 Ctrl+Shift+U
全局 打开搜索对话框 Ctrl+H
全局 工作区中的声明 Ctrl+G
全局 工作区中的引用 Ctrl+Shift+G

文本编辑
作用域 功能 快捷键
文本编辑器 改写切换 Insert
文本编辑器 上滚行 Ctrl+↑
文本编辑器 下滚行 Ctrl+↓

文件
作用域 功能 快捷键
全局 保存 Ctrl+X
Ctrl+S
全局 打印 Ctrl+P
全局 关闭 Ctrl+F4
全局 全部保存 Ctrl+Shift+S
全局 全部关闭 Ctrl+Shift+F4
全局 属性 Alt+Enter
全局 新建 Ctrl+N

项目
作用域 功能 快捷键
全局 全部构建 Ctrl+B

源代码

作用域 功能 快捷键

Java 编辑器 格式化 Ctrl+Shift+F

Java 编辑器 取消注释 Ctrl+\

Java 编辑器 注释 Ctrl+/

Java 编辑器 添加导入 Ctrl+Shift+M

Java 编辑器 组织导入 Ctrl+Shift+O

Java 编辑器 使用 try/catch 块来包围 未设置，太常用了，所以在这里列出,建议自己设置。

也可以使用 Ctrl+1 自动修正。

运行

作用域 功能 快捷键

全局 单步返回 F7

全局 单步跳过 F6

全局 单步跳入 F5

全局 单步跳入选择 Ctrl+F5

全局 调试上次启动 F11

全局 继续 F8

全局 使用过滤器单步执行 Shift+F5

全局 添加/去除断点 Ctrl+Shift+B

全局 显示 Ctrl+D

全局 运行上次启动 Ctrl+F11

全局 运行至行 Ctrl+R

全局 执行 Ctrl+U

重构

作用域 功能 快捷键

全局 撤销重构 Alt+Shift+Z

全局 抽取方法 Alt+Shift+M

全局 抽取局部变量 Alt+Shift+L

全局 内联 Alt+Shift+I

全局 移动 Alt+Shift+V

全局 重命名 Alt+Shift+R

全局 重做 Alt+Shift+Y