代码规范

**一、注释/JavaDoc**

1.顶部版权声明  
2.包和引入块（每一块以空白行分隔）  
3.类或接口的声明。 在Javadoc注释，描述的类或接口的用途。

**[java]** [view plaincopy](http://blog.csdn.net/stefzeus/article/details/6425642)

1. /\*
2. \* Copyright (C) 2007 The Android Open Source Project
3. \*
4. \* Licensed under the Apache License, Version 2.0 (the "License");
5. \* you may not use this file except in compliance with the License.
6. \* You may obtain a copy of the License at
7. \*
8. \*      http://www.apache.org/licenses/LICENSE-2.0
9. \*
10. \* Unless required by applicable law or agreed to in writing, software
11. \* distributed under the License is distributed on an "AS IS" BASIS,
12. \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13. \* See the License for the specific language governing permissions and
14. \* limitations under the License.
15. \*/
16. **package** com.android.internal.foo;
17. **import** android.os.Blah;
18. **import** android.view.Yada;
19. **import** java.sql.ResultSet;
20. **import** java.sql.SQLException;
21. /\*\*
22. \* Does X and Y and provides an abstraction for Z.
23. \*/
24. **public** **class** Foo {
25. ...
26. }

**类/接口注释的内容** （1项 要求写上）  
类、接口的文档注释包含如下信息：  
1.用途。 开发人员使用某个类/接口之前，需要知道采用该类/接口的用途。  
2.如何使用。开发人员需要知道该类/接口应该如何使用，如果必要的话还需要注明不应  
该如何使用。  
3.开发维护的日志。一个有关于该类/接口的维护记录：时间、作者、摘要。  
  
**方法注释的内容** （1，5，6，7项 要求写上）  
1.类该方法是做什么的 。  
2.该方法如何工作。  
3.代码修改历史纪录。  
4.方法调用代码示范。  
5.必须传入什么样的参数给这个方法。@param  
6.异常处理。@throws  
7.这个方法返回什么。@return

**二、在Imports使用通配符的优劣**

**三、局部变量应该推迟至使用前声明并初始化**

**四、域(Field)命名**

\* 非公有，非静态字段命名以m开头。   
\* 静态域命名以s开头。  
\* 其他字段以小写字母开头。  
\* public static final 字段(常量) 全部大写，并用下划线连起来。

**[java]** [view plaincopy](http://blog.csdn.net/stefzeus/article/details/6425642)

1. **public** **class** MyClass {
2. **public** **static** **final** **int** SOME\_CONSTANT = 42;
3. **public** **int** publicField;
4. **private** **static** MyClass sSingleton;
5. **int** mPackagePrivate;
6. **private** **int** mPrivate;
7. **protected** **int** mProtected;
8. }

**五、花括号没有独自一行，它们与它前面的代码占同一行**

**六、命名规则**

1.**包** 小写。

com.chinacache.billing    
com.chinacache.billing.node

2.**类**   大小写字母混合组成，头字母大写。

class Raster;    
class ImageSprite;

3.**接口** 大小写字母混合组成，头字母大写，常以"able"、"ible"结尾 。

interface RasterDelegate;  
interface Runna ble ;  
interface Accessible ;

4.**方法** 大小写字母混合组成，第一个单词的首字母小写，其后单词的首字母大写。

run();  
r unF ast();  
g etB ackground();

5.**变量、参数**小写 ，不推荐使用下划线 ，简短明晰。

char c;  
int i;  
float m yW idth;

6.**集合、数组** 应该从命名中体现其复数的含义，例如加后缀s或前缀some。

customers ;  
postedMessages ;  
some Customers ;  
some Items ;

**七、在定义类时，应该按照访问权限的大小分别排列属性和方法。**

1. public  
2. protected  
3. 包级别(没有访问修饰符的，默认为friendly)  
4. private

**写好代码以后对照代码规范一项一项检查一下吧。**

(1)Eclipse 代码格式化  
你可以导入development/ide/eclipse下的文件，使得Eclipse按照Android代码风格规则。选择 “Window › Preferences › Java › Code Style，使用 “Formatter › Import” ,导入android-formatting.xml，”Organize Imports › Import” 导入 android.importorder.

(2)eclipse tab 设置为4个空格：  
Preferences -> General -> Editors -> Text Editors:  
Insert spaces for tabs

(3)自动格式化代码 Ctrl+Shift+

F

(4)全局 查找并替换 Ctrl+F

**八、异常**

1.不要忽视异常处理

如果像下面的代码这样，对catch后的异常作空处理，就像埋下地雷一样让人感觉到毛骨悚然：

错误的做法：

**[java]** [view plaincopy](http://blog.csdn.net/stefzeus/article/details/6425642)

1. **void** setServerPort(String value) {
2. **try** {
3. serverPort = Integer.parseInt(value);
4. } **catch** (NumberFormatException e) {
5. }
6. }

正确的做法（1）：

在方法声明时抛出异常，由客户程序员去负责消化这个异常。

**[java]** [view plaincopy](http://blog.csdn.net/stefzeus/article/details/6425642)

1. **void** setServerPort(String value) **throws** NumberFormatException {
2. serverPort = Integer.parseInt(value);
3. }

正确的做法（2）：

**[java]** [view plaincopy](http://blog.csdn.net/stefzeus/article/details/6425642)

1. /\*\* Set port. If value is not a valid number, 80 is substituted. \*/
2. **void** setServerPort(String value) {
3. **try** {
4. serverPort = Integer.parseInt(value);
5. } **catch** (NumberFormatException e) {
6. serverPort = 80;  // default port for server
7. }

正确的做法（3）：

**[java]** [view plaincopy](http://blog.csdn.net/stefzeus/article/details/6425642)

1. /\*\* Set port. If value is not a valid number, die. \*/
2. **void** setServerPort(String value) {
3. **try** {
4. serverPort = Integer.parseInt(value);
5. } **catch** (NumberFormatException e) {
6. **throw** **new** RuntimeException("port " + value " is invalid, ", e);
7. }

正确的做法（4）：

**[java]** [view plaincopy](http://blog.csdn.net/stefzeus/article/details/6425642)

1. **void** setServerPort(String value) **throws** ConfigurationException {
2. **try** {
3. serverPort = Integer.parseInt(value);
4. } **catch** (NumberFormatException e) {
5. **throw** **new** ConfigurationException("Port " + value + " is not valid.");
6. }
7. }

2、不要偷懒而捕捉一般异常

下面代码一概捕捉Exception异常，大小通吃是不对的，这样会让你在错误出现时难以定位到错误原因，一般异常无法用统一方法进行异常处理。

错误的做法：

**[java]** [view plaincopy](http://blog.csdn.net/stefzeus/article/details/6425642)

1. **try** {
2. someComplicatedIOFunction();        // may throw IOException
3. someComplicatedParsingFunction();   // may throw ParsingException
4. someComplicatedSecurityFunction();  // may throw SecurityException
5. // phew, made it all the way
6. } **catch** (Exception e) {               // I'll just catch all exceptions
7. handleError();                      // with one generic handler!
8. }