天坦智慧电子科技股份有限公司

(Android 通用)

作者：Newrey 余亮

日期：2015-08-06

当前版本：V 2.1

编号	版本	修改说明	修改者	修改时间
1.	1.0	第一次版本生成	余亮	2015-5-20
2.	2.0	第二次修改	Newrey	2015-8-06
3	2.1	删除不常用的说明	余亮	2015-08-12

目录

介绍

1. 为什么需要编码规范?

编码规范对于程序员而言尤为重要，有以下几个原因：

- 一个软件的生命周期中，80%的花费在于维护
- 几乎没有任何一个软件，在其整个生命周期中，均由最初的开发人员来维护
- 编码规范可以改善软件的可读性，可以让其他程序员尽快而彻底地理解新的代码
- 如果你将源码作为产品发布，就需要确任它是否被很好的打包并且清晰无误，一如你已构建的其它任何产品

命名

2. 包命名

命名规则：一个唯一包名的前缀总是全部小写的 ASCII 字母并且是一个顶级域名，通常是 com，edu，gov，mil，net，org。包名的后续部分根据不同机构各自内部的命名规范而不尽相同。这类命名规范可能以特定目录名的组成来区分部门 (department) ，项目(project)，机器(machine)，或注册名(login names)。
*例如： net.*tatans.*项目代号.项目名称*
App 包名：例如: net. tatans. coeus. launcher.

net.tatans.hyperion.social.activities
net.tatans.hyperion.im.activities
net.tatans.hyperion.im.activities 此包中包含：页面用到的 Activity 类 (activities 层级名 用户界面层)

net.tatans.hyperion.social.tools 此包中包含：公共工具方法类 （tools 模块名）
net.tatans.hyperion.social.db 此包中包含：数据库操作类
net.tatans.hyperion.social.custom 此包中包含：自定义的 View 类等
net.tatans.hyperion.social.service 此包中包含：Service 服务
net.tatans.hyperion.social.broadcast 此包中包含：Broadcast 服务

3. 类和接口 命名

命名规则：类名是个一名词，采用大小写混合的方式，每个单词的首字母大写。尽量使

你的类名简洁而富于描述。使用完整单词，避免缩写词(除非该缩写词被更广泛使用，像 URL，HTML)
接口一般使用 able、ible、er 等后缀，并且接口必须以大写 I 开头
例如： interface IButton; interface IImageSprite;

activity 类 Activity 为后缀标识，如欢迎页面类 WelcomeActivity.
Adapter 类 Adapter 为后缀标识,如商品详情适配器 ProductDetailAdapter
解析类 Hlr 为后缀标识，如首页解析类 HomePosterHlr
公共方法类 Tools 或 Manager 为后缀标识。如：线程池管理类：ThreadPoolManager
日志工具类：LogTools
数据库类 以 DBHelper 后缀标识。如城市数据库：CityDBHelper
Service 类 以 Service 为后缀标识
BroadcastReceive 以 Broadcast 为后缀标识
ContentProvider 以 Provider 为后缀标识

规约：类名必须使用驼峰规则，即首字母必须大写，如果为词组，则每个单词的首字母也必须要大写，类名必须使用名词，或名词词组。要求类名简单，不允许出现无意义的单词（*如 class XXXActivity*）。
如：class BookMarkAdd → 正确
*如：class AddBookReadPlanActivity → **错误！** 应为 class BookReadPlanAdd*

4. 方法的命名

命名规则：方法名是一个动词，采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写。
例如： public void run(); public String getBookName();

类中常用方法的命名：

1. 类的获取方法（一般具有返回值）一般要求在被访问的字段名前加上 get，如 getFirstName()，getLastName()。一般来说，get 前缀方法返回的是单个值，find 前缀的方法返回的是列表值。
2. 类的设置方法（一般返回类型为 void）：被访问字段名的前面加上前缀 set，如 setFirstName(),setLastName().
3. 类的布尔型的判断方法一般要求方法名使用单词 is 或 has 做前缀，如 isPersistent()，isString()。或者使用具有逻辑意义的单词，例如 equal 或 equals。
4. 类的普通方法一般采用完整的英文描述说明成员方法功能，第一个单词尽可能采用动词，首字母小写，如 openFile（），addCount（）。
5. 构造方法应该用递增的方式写。（参数多的写在后面）。
6. toString()方法：一般情况下，每个类都应该定义 toString(),其格式为：
7. initXXX() 初始化相关方法,使用 init 为前缀标识
8. isXXX() checkXXX() 方法返回值为 boolean 型的请使用 is 或 check 为前缀标识
9. getXXX() 返回某个值的方法，使用 get 为前缀标识
10. processXXX() 对数据进行处理的方法，尽量使用 process 为前缀标

11. displayXXX() 弹出提示框和提示信息，使用 display 为前缀标识
12. saveXXX() 与保存数据相关的，使用 sav 为 e 前缀标识
13. resetXXX() 对数据重组的，使用 reset 前缀标识
14. clearXXX()removeXXX() 清除数据相关的，使用 clear 或 remove 为前缀标识
15. drawXXX() 绘制数据或效果相关的，使用 draw 前缀标识

5. 变量命名

命名规则：第一个类型标志字母小写，其后单词的首字母大写。变量名不应以下划线或美元符号开头，尽管这在语法上是允许的。变量名应简短且富于描述。变量名的选用应该易于记忆，即，能够指出其用途。尽量避免单个字符的变量名，除非是一次性的临时变量。临时变量通常被取名为 i，j，k，m 和 n，它们一般用于整型；c，d，e，它们一般用于字符型。

对于 SWING 以及类变量的命名标志，采取控件的大写的首字母进行小写缩写的格式，例如：

```
TextView txt
Button btn
ImageButton imgBtn
ImageView imgView
CheckBox chk
RadioButton rdoBtn
analogClock anaClk
DigtalClock dgtClk
DatePicker dtPk
TimePicker tmPk
toggleButton tglBtn
EditText edtTxt
ProgressBar proBar proBar
SeekBar skBar
AutoCompleteTextView autoTxt
ZoomControls zmCtl
Include ind
VideoView vdoVi
WdbView webVi
RantingBar ratBar
Tab tab
Spinner spn
Chronometer cmt
ScollView sclVi
TextSwitch txtSwt
ImageSwitch imgSwt
```

listView lVi
ExpandableList epdLt
MapView mapVi

JFrame jfMain;   //jf
JButton jbOk;   //jb
JLable jlUserName; //jl
JTree   jtTree;   //jt
DefaultMutableTreeNode dmtn MyNode;   //dmtn
ImageIcon iiMyPic;   //ii
Image imgMyPic;   //img
Icon   icoMyPic;   //ico
URL   urlLoginPic;   //url
Container conMainFrame; //con
对于 ArrayList 、 HashTable 采用首字母缩写标志_ + 变量自身类型标志+变量名
ArrayList->al_ HashTable->ht_
例如：
ArrayList<JButton> al_jbSeat=new ArrayList<JButton>();
al_ 代表 ArrayList
jb 代表 JButton
Seat 代表变量名

常规变量：
nTemp:临时通用变量
nResult:接收返回的结果
nCount:统计总数

变量的作用域及前缀

前缀	**说明**	**举例**
_	类成员变量(字段)	_optFlag
s_	静态变量	s_nAge 静态整数
arr_	数组	arr_nCount[]
al_	ArrayList	al_nFlag
ht_	HashTable	ht_nFlag

变量类型对应缩写

前缀	说明	缩写	举例
Bool[Boolean]	布尔型	b	bFlag
Char[Character]	字符型	c	cFlag
Byte	8 位带符号整数	bt	btFlag
Short	16 位带符号整数	sht	shtFlag
Int[Integer]	32 位带符号整数	n	nFlag
Long	64 位带符号整数	l	lFlag
Float	32 位单精度浮点数	f	fFlag
Double	64 位双精度浮点数	lf	lfFlag
String	字符串	s	sFlag
Date	日期时间	dt	dtFlag

例如：String sBookName;

规约：变量命名也必须使用驼峰规则，但是首字母必须小写，变量名尽可能的使用名词或名词词组。同样要求简单易懂，不允许出现无意义的单词。

如：String sBookName; → 正确

如：String sBookNameString; →错误！

6. 成员变量命名

同变量命名，但不要在私有变量前添加 m 字样！

7. 常量命名

命名规则：类常量的声明，应该全部大写，单词间用下划线隔开。

例如：static final int MIN_WIDTH = 4;

例如：static final int MAX_WIDTH = 999;

例如：static final int GET_THE_CPU = 1;

8. 异常命名

自定义异常的命名**必须以 Exception 为结尾**。已明确标示为一个异常。

```
public class MySecondException extends Throwable {

        public MySecondException() {

        super();

}

public class MyFirstException extends Exception {

        public MyFirstException() {

        super();

}
```

9. layout 命名

Activity 默认布局，以去掉后缀的 Activity 类进行命名。不加后缀

Activity 子布局，父布局名称开始，追加 item 和子布局功能说明。

例如：Activity 默认布局：homeposter.xml 子布局为 homeposter_item_poster.xml

10.id 命名

规约：layout 中所使用的 id 必须以全部单词小写，单词间以下划线分割，并且使用名词或名词词组，并且要求能够通过 id 直接理解当前组件要实现的功能。

view 的缩写详情如下

LayoutView：lv

RelativeView:rv

TextView:tv

ImageView:iv

ImageButton:im

Button:btn

11.资源命名

规约：layout 中所使用的所有资源(如 drawable,style 等)命名必须以全部单词小写，单词间以下划线分割，并且尽可能的使用名词或名词组，即使用 模块名_用途 来命名。如果为公共资源，如分割线等，则直接用用途来命名

如：menu_icon_navigate.png → 正确

如：某分割线：line.png 或 separator.png →正确

btn_login_normal
按钮图片使用 btn_功能_说明
bg_head
背景图片使用 bg_功能_说明
def_search_cell
默认图片使用 def_功能_说明
icon_more_help
图标图片使用 icon_功能_说明
seg_list_line
具有分隔特征的图片使用 seg_功能_说明
sel_ok
选择图标使用 sel_功能_说明

注释

Java 程序有两类注释：实现注释(implementation comments)和文档注释(document comments)。实现注释是使用/*...*/和//界定的注释。文档注释(被称为"doc comments")由/**...*/界定。文档注释可以通过 javadoc 工具转换成 HTML 文件。

12.文件注释

所有的源文件都应该在开头有一个注释，其中列出类名、版本信息、日期和版权声明。如下：

```
/*
 * 文件名
 * 包含类名列表
 * 版本信息，版本号
 * 创建日期。
 * 版权声明
 */
```

13.类注释

每一个类都要包含如下格式的注释，以说明当前类的功能等。

```
/**
 * 类名
 * @author 作者 <br/>
 *   实现的主要功能。
 *   创建日期
 *   修改者，修改日期，修改内容。
*/
```

14.方法注释

每一个方法都要包含 如下格式的注释 包括当前方法的用途，当前方法参数的含义，当前方法返回值的内容和抛出异常的列表。

```
/**
 *
 * 方法的一句话概述
 * <p>方法详述（简单方法可不必详述）</p>
 * @param s 说明参数含义
 * @return 说明返回值含义
 * @throws IOException 说明发生此异常的条件
 * @throws NullPointerException 说明发生此异常的条件
 */
```

15.类成员变量和常量注释

成员变量和常量需要使用 java doc 形式的注释，以说明当前变量或常量的含义

```
 /**
   * XXXX含义
   */
```

16.其他注释

方法内部的注释 如果需要多行 使用/*…… */形式，如果为单行是用//……形式的注释。不要再方法内部使用 java doc 形式的注释“/**……**/”，简单的区分方法是，java doc 形式的注释在 eclipse 中为蓝色，普通注释为绿色。

17.XML 注释

规约:如果当前 layout 或资源需要被多处调用,或为公共使用的 layout(若 list_item),则需要在 xml 写明注释。要求注释清晰易懂。

代码风格规约

18.方法

- 一个方法尽量不要超过 15 行，如果方法太长，说明当前方法业务逻辑已经非常复杂，那么就需要进行方法拆分，保证每个方法只作一件事。
- 不要使用 try catch 处理业务逻辑!!!!
- 事件不要嵌套！单独写成一个事件函数，否则风格很乱

19.参数和返回值

- 一个方法的参数尽可能的不要超过 4 个！
- 如果一个方法返回的是一个错误码，请使用异常!!
- 尽可能不要使用 null， 替代为异常 或者使用空变量 如返回 List 则可以使用 Collections.emptyList()

20.神秘的数

代码中不允许出现单独的数字，字符！如果需要使用数字或字符，则将它们按照含义封装为静态常量！（for 语句中除外）

21.控制语句

判断中如有常量，则应将常量置于判断式的右侧。如：
if (true == isAdmin())...
尽量不使用三目条件的嵌套。

所有 if 语句必须用{}包括起来,即便是只有一句：

```
if (true){
//do something.....
}
if (true)
   i = 0; //不要使用这种
```

对于循环：

```
//不推荐方式_____________________________________________
while(index < products.getCount()){
```

```
    //每此都会执行一次 getCount()方法，
    //若此方法耗时则会影响执行效率
    //而且可能带来同步问题，若有同步需求，请使用同步块或同步方法
}
//推荐方式_____________________________________________
//将操作结构保存在临时变量里，减少方法调用次数
final int count = products.getCount();
while(index < count){
}
```

22.异常的捕捉处理

- 通常的思想是只对错误采用异常处理：逻辑和编程错误，设置错误，被破坏的数据，资源耗尽，等等。
- 通常的法则是系统在正常状态下以及无重载和硬件失效状态下，不应产生任何异常。
- 最小化从一个给定的抽象类中导出的异常的个数。对于经常发生的可预计事件不要采用异常。不要使用异常实现控制结构。
- 若有 finally 子句，则不要在 try 块中直接返回，亦不要在 finally 中直接返回。

23.访问控制

若没有足够理由，不要把实例或类变量声明为公有。通常，实例变量无需显式的设置(set)和获取(gotten)，通常这作为方法调用的边缘效应 (side effect)而产生。

一个具有公有实例变量的恰当例子，是类仅作为数据结构，没有行为。亦即，若你要使用一个结构(struct)而非一个类(如果 java 支持结构的话)，那么把类的实例变量声明为公有是合适的。

约定俗成

24.变量赋值

避免在一个语句中给多个变量赋相同的值。它很难读懂。例如：

```
fooBar.fChar = barFoo.lchar = 'c';
```

不要将赋值运算符用在容易与相等关系运算符混淆的地方。例如：

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

应该写成

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

```
}
```

不要使用内嵌(embedded)赋值运算符试图提高运行时的效率，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;
```

25.圆括号

一般而言，在含有多种运算符的表达式中使用圆括号来避免运算符优先级问题，是个好方法。

即使运算符的优先级对你而言可能很清楚，但对其他人未必如此。你不能假设别的程序员和你一样清楚运算符的优先级。

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

26.返回值

设法让你的程序结构符合目的。例如：

```
if (booleanExpression) {
        return true;
} else {
        return false;
}
```

应该代之以如下方法：

```
return booleanExpression
```

类似地：

```
if (condition) {
        return x;
}
return y;
```

应该写做：

```
return (condition ? x : y);
```

27.条件运算符"?"前的表达式

如果一个包含二元运算符的表达式出现在三元运算符" ? : "的"?"之前，那么应该给表达式添上一对圆括号。例如：

```
(x >= 0) ? x : -x
```

21 种代码的坏味道

应该在编程中尽量避免这 21 种“坏味道”。

28.Duplicated Code

代码重复几乎是最常见的异味了。他也是 Refactoring 的主要目标之一。代码重复往往来自于 copy-and-paste 的编程风格。

29.Long method

它是传统结构化的“遗毒“。一个方法应当具有自我独立的意图，不要把几个意图放在一起。

30.Large Class

大类就是你把太多的责任交给了一个类。这里的规则是 One Class One Responsibility，一个类一个职责。

31.Divergent Change

一个类里面的内容变化率不同。某些状态一个小时变一次，某些则几个月一年才变一次；某些状态因为这方面的原因发生变化，而另一些则因为其他方面的原因变一次。面向对象的抽象就是把相对不变的和相对变化相隔离。把问题变化的一方面和另一方面相隔离。这使得这些相对不变的可以重用。问题变化的每个方面都可以单独重用。这种相异变化的共存使得重用非常困难。

32.Shotgun Surgery

这正好和上面相反。对系统一个地方的改变涉及到其他许多地方的相关改变。这些变化率和变化内容相似的状态和行为通常应当放在同一个类中。

33.Feature Envy

对象的目的就是封装状态以及与这些状态紧密相关的行为。如果一个类的方法频繁用 get 方法存取其他类的状态进行计算，那么你要考虑把行为移到涉及状态数目最多的那个类。

34.Data Clumps

某些数据通常像孩子一样成群玩耍：一起出现在很多类的成员变量中，一起出现在许多方法的参数中…..，这些数据或许应该自己独立形成对象。

35.Primitive Obsession

面向对象的新手通常习惯使用几个原始类型的数据来表示一个概念。譬如对于范围，他们会使用两个数字。对于 Money，他们会用一个浮点数来表示。因为你没有使用对象来表达问题中存在的概念，这使得代码变的难以理解，解决问题的难度大大增加。好的习惯是扩充语言所能提供原始类型，用小对象来表示范围、金额、转化率、邮政编码等等。

36.Switch Statement

基于常量的开关语句是 OO 的大敌，你应当把他变为子类、state 或 strategy。

37.Parallel Inheritance Hierarchies

并行的继承层次是 shotgun surgery 的特殊情况。因为当你改变一个层次中的某一个类时，你必须同时改变另外一个层次的并行子类。

38.Lazy Class

一个干活不多的类。类的维护需要额外的开销，如果一个类承担了太少的责任，应当消除它。

39.Speculative Generality

一个类实现了从未用到的功能和通用性。通常这样的类或方法唯一的用户是 testcase。不要犹豫，删除它。

40.Temporary Field

一个对象的属性可能只在某些情况下才有意义。这样的代码将难以理解。专门建立一个对象来持有这样的孤儿属性，把只和他相关的行为移到该类。最常见的是一个特定的算法需要某些只有该算法才有用的变量。

41.Message Chain

消息链发生于当一个客户向一个对象要求另一个对象，然后客户又向这另一对象要求另一个对象，再向这另一个对象要求另一个对象，如此如此。这时，你需要隐藏分派。

42.Middle Man

对象的基本特性之一就是封装，而你经常会通过分派去实现封装。但是这一步不能走得太远，如果你发现一个类接口的一大半方法都在做分派，你可能需要移去这个中间人。

43.Inappropriate Intimacy

某些类相互之间太亲密，它们花费了太多的时间去砖研别人的私有部分。对人类而言，我们也许不应该太假正经，但我们应当让自己的类严格遵守禁欲主义。

44.Alternative Classes with Different Interfaces

做相同事情的方法有不同的函数 signature，一致把它们往类层次上移，直至协议一致。

45.Incomplete Library Class

要建立一个好的类库非常困难。我们大量的程序工作都基于类库实现。然而，如此广泛而又相异的目标对库构建者提出了苛刻的要求。库构建者也不是万能的。有时候我们会发现库类无法实现我们需要的功能。而直接对库类的修改有非常困难。这时候就需要用各种手段进行 Refactoring。

46.Data Class

对象包括状态和行为。如果一个类只有状态没有行为，那么肯定有什么地方出问题了。

47.Refused Bequest

超类传下来很多行为和状态，而子类只是用了其中的很小一部分。这通常意味着你的类层次有问题。

48.Comments

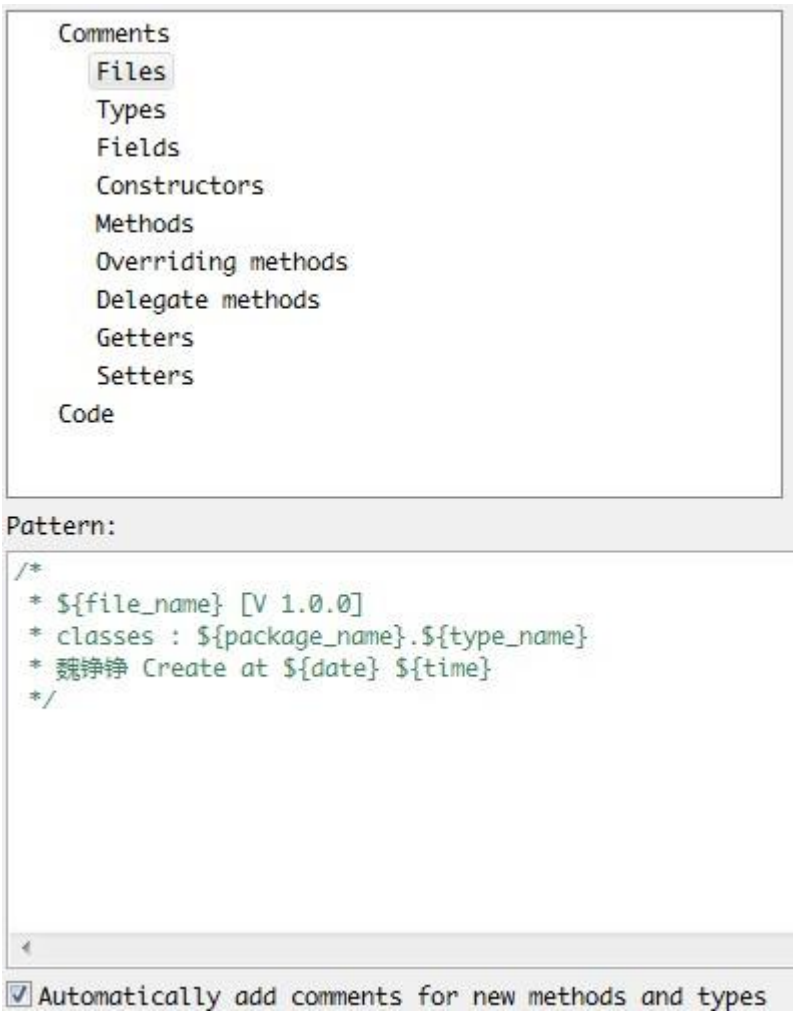
经常觉得要写很多注释表示你的代码难以理解。如果这种感觉太多，表示你需要 Refactoring。

Eclipse 配置方法

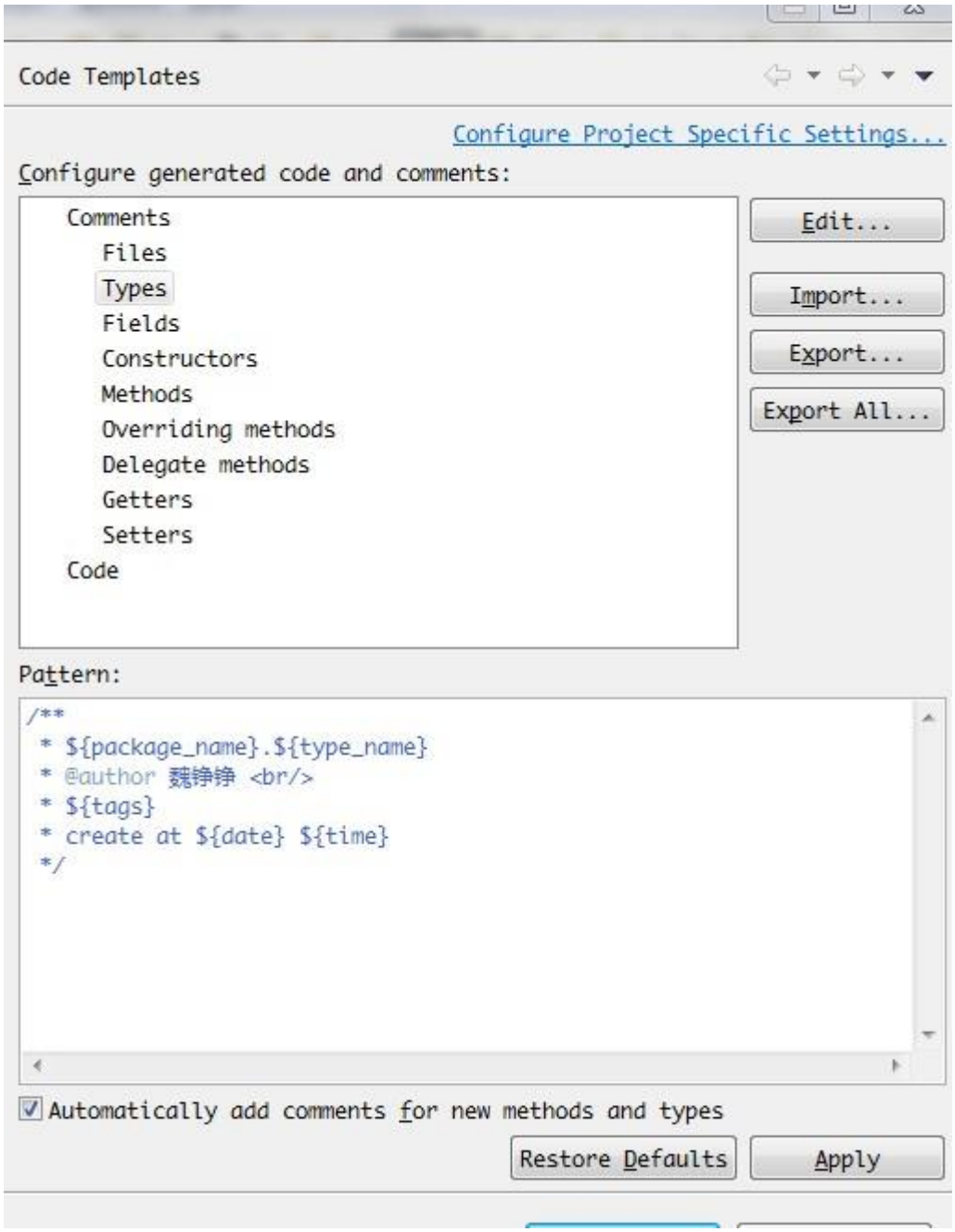
49.注释模板

在 eclipse 的 preferences 中，选择 java → code style →code Template

1. 添加文件创建日志模板



2.设置类注释模板



```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

TatansDb 操作 Sqlite 数据库

首先创建一个测试实体类 User.java

```
package com.devchina.ormdemo;

import java.util.Date;
 @Table(name="表名")
public class User {

   private int id;
   private String name;
   private String email;
   private Date registerDate;
   private Double money;

   //////////////getter and setter 不能省略哦///////////////
   public int getId() {
      return id;
   }
   public void setId(int id) {
      this.id = id;
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public String getEmail() {
      return email;
   }
   public void setEmail(String email) {
      this.email = email;
   }
   public Date getRegisterDate() {
      return registerDate;
   }
   public void setRegisterDate(Date registerDate) {
      this.registerDate = registerDate;
```

```
    }
    public Double getMoney() {
        return money;
    }
    public void setMoney(Double money) {
        this.money = money;
    }

}
```

```
package com.devchina.ormdemo;

import java.util.Date;
import java.util.List;

import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

public class TatansOrmDemoActivity extends TatansActivity {


    @ViewInject(id=R.id.textView) TextView textView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.fragment_main);

        TatansDb db = TatansDb.create(this, "数据库名");

        User user = new User();
        user.setEmail("afinal@tsz.net");
        user.setName("探索者");
        user.setRegisterDate(new Date());

        db.save(user);

        List<User> userList = db.findAll(User.class);//查询所有的用户

        Log.e("TatansOrmDemoActivity", "用户数量："+ (userList!=null?userList.size():0));

        textView.setText(userList.get(0).getName()+":"+user.getRegisterDate());
    }
}
```

1.save

2.findAll

3.findAll(clazz, orderBy);//orderBy--用表里的一个字段名称即 User 对象里面的属性就行了，不用再加什么 order by id desc 什么的

4.findAllByWhere(clazz, strWhere)

5.findAllByWhere(clazz, strWhere, orderBy)

6.findDbModelBySQL(strSQL)

7.findById(id, clazz)

TatansActivity 注入式布局

```
public class TatansDemoActivity  extends TatansActivity  {

        //无需调用 findViewById 和 setOnclickListener 等
        @ViewInject(id=R.id.button,click="btnClick")  Button  button;
        @ViewInject(id=R.id.textView)  TextView  textView;

        public void onCreate(Bundle  savedInstanceState)  {
              super.onCreate(savedInstanceState);
              setContentView(R.layout.main);
        }

        public void btnClick(View  v){
              textView.setText("text  set  form  button");
        }
}
```

TatansCache 缓存框架

看桌面代码

TatansApp 获取 App 程序名和版本信息

```
  /**
   * 获取应用程序名称
   */
  public static String getAppName(Context context)

 /** [获取应用程序版本名称信息]
   *
   * @param context
   * @return 当前应用的版本名称
   */
  public static String getVersionName(Context context)
```

TatansDensity 常用单位换算

```
/**
 * dp 转 px
 *
 * @param context
 * @param val
 * @return
 */
public static int dp2px(Context context, float dpVal)

  /**
   * sp 转 px
   *
   * @param context
   * @param val
   * @return
   */
  public static int sp2px(Context context, float spVal)

  /**
   * px 转 dp
   *
   * @param context
   * @param pxVal
   * @return
   */
 public static float px2dp(Context context, float pxVal)

   /**
    * px 转 sp
    *
    * @param fontScale
    * @param pxVal
    * @return
    */
  public static float px2sp(Context context, float pxVal)
```

TatansKeyBoard 软键盘操作

```
/**
```

```
    * 打卡软键盘
    *
    * @param mEditText
    *                  输入框
    * @param mContext
    *                  上下文
    */
   @SuppressLint("NewApi")
public static void openKeybord(EditText mEditText, Context mContext)

/**
     * 关闭软键盘
     *
     * @param mEditText
     *                  输入框
     * @param mContext
     *                  上下文
     */
    @SuppressLint("NewApi")
public static void closeKeybord(EditText mEditText, Context mContext)
```

TatansLog 日志 Log 统一管理

详见 Api：https://192.168.1.249/svn/coeus/tools
默认过滤器是（tatans）

TatansLog.d(“str”)

TatansLog.e(“a”)

TatansPreferences 关于 xml 的统一操作

详见 Api：https://192.168.1.249/svn/coeus/tools

```
TatansPreferences.put(this,"key",3);
TatansPreferences.get(this,"key",”取不到之后默认的值”);
```

TatansSdCard 关于 sd 卡的相关辅助

详见 Api：https://192.168.1.249/svn/coeus/tools

TatansToast 关于 toast 的相关统一操作

详见 Api：https://192.168.1.249/svn/coeus/tools

```
TatansToast.show(this,"消息",10000);
```

Gson 解析 JSON 数据

Maven 依赖添加：

```
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.3.1</version>
</dependency>
```

```
String jsonData =
"[{\"username\":\"arthinking\",\"userId\":001},{\"username\":\"Jason\",\"use
rId\":002}]";
try{
        JsonReader reader = new JsonReader(new StringReader(jsonData));
        reader.beginArray();
        while(reader.hasNext()){
                reader.beginObject();
                while(reader.hasNext()){
                        String tagName = reader.nextName();
                        if(tagName.equals("username")){
                                System.out.println(reader.nextString());
                        }
                        else if(tagName.equals("userId")){
                                System.out.println(reader.nextString());
                        }
                }
                reader.endObject();
        }
        reader.endArray();
}
catch(Exception e){
        e.printStackTrace();
}
```

通过把 JSON 数据映射成一个对象，使用 Gson 对象的 fromJson()方法获取一个对象数组进行操作：

创建 JSON 数据对应的一个 POJO 对象 User.java：

```
public class User {
        private String username ;
        private int userId ;
        public String getUsername() {
                return username;
        }
        public void setUsername(String username) {
                this.username = username;
        }
        public int getUserId() {
                return userId;
        }
        public void setUserId(int userId) {
                this.userId = userId;
        }
}
```

使用 Gson 对象获取 User 对象数据进行相应的操作：

```
Type listType = new TypeToken<LinkedList<User>>(){}.getType();
Gson gson = new Gson();
LinkedList<User> users = gson.fromJson(jsonData, listType);
for (Iterator iterator = users.iterator(); iterator.hasNext();) {
        User user = (User) iterator.next();
        System.out.println(user.getUsername());
        System.out.println(user.getUserId());
}
```

如果要处理的 JSON 字符串只包含一个 JSON 对象，则可以直接使用 fromJson 获取一个 User 对象：

```
String jsonData = "{\"username\":\"arthinking\",\"userId\":001}";
Gson gson = new Gson();
User user = gson.fromJson(jsonData, User.class);
System.out.println(user.getUsername());
System.out.println(user.getUserId());
```

TBIS 系统全局音频管理

这是一个 Android 系统的音频管理类，当一个声音开启时候，可以控制另一个声音。

```
AudioManagerUtil amu = new AudioManagerUtil(context, new AudioManagerCallBack() {
                @Override
```

```
            public void onFocusSuccess() {
                // TODO Auto-generated method stub
                super.onFocusSuccess();
            }
            @Override
            public void onFocusFail(int errCode) {
                // TODO Auto-generated method stub
                super.onFocusFail(errCode);
                Log.d("TEST", "onFocusFail");
            }

            @Override
            public void onFocusGain() {
                // TODO Auto-generated method stub
                super.onFocusGain();
                Log.d("TEST", "onFocusGain");
            }

            @Override
            public void onFocusLoss() {
                // TODO Auto-generated method stub
                super.onFocusLoss();
                Log.d("TEST", "onFocusLoss");
            }

            @Override
            public void onFocusLossTransient() {
                // TODO Auto-generated method stub
                super.onFocusLossTransient();
                Log.d("TEST", "onFocusLossTransient");
            }

            @Override
            public void onFocusLossTransientDuck() {
                // TODO Auto-generated method stub
                super.onFocusLossTransientDuck();
                Log.d("TEST", "onFocusLossTransientDuck");
            }
        });
```

具体请参考 API：https://192.168.1.249/svn/coeus/audiodoc

TBIS 网络框架

具体请参考 API 文档： https://192.168.1.249/svn/coeus/tools

Maven 依赖添加：

```
<dependency>
  <groupId>net.tatans.coeus</groupId>
  <artifactId>coeus-network</artifactId>
  <version>0.1.7-release</version>
</dependency>
```

Gradle 依赖添加

```
compile 'net.tatans.coeus:coeus-network:1.0.7-release'
```

get 方法：

方法一(回调方式):

```
TatansHttp fh = new TatansHttp();
fh.get("http://www.baidu.com", new HttpRequestCallBack()
{
@Override
public void onLoading(long count, long current) { //每1秒钟自动被回调一次
textView.setText(current+"/"+count);
}
@Override
public void onSuccess(String t) {
textView.setText(t==null?"null":t);
}
@Override
public void onStart() {
//开始http请求的时候回调
}
@Override
public void onFailure(Throwable t, String strMsg) {
```

```
//加载失败的时候回调
}
});
```

方法二(静态变量):

```
/**
*
* @param url
*                  发送请求的URL
* @param context
*                  当前Context上下文
* @return  服务器响应字符串
* @throws Exception
*/
String sRequest=HttpUtil.getRequest(String url, Contextcontext);
```

使用 TatansHttp 上传文件 或者 提交数据 到服务器（post 方法）

方法一(回调方式):

```
HttpRequestParams params = new HttpRequestParams();
params.put("username", "michael yang");
params.put("password", "123456");
params.put("email", "750749212@qq.com");
params.put("profile_picture", new File("/mnt/sdcard/pic.jpg")); //  上传文件
params.put("profile_picture2", inputStream); //  上传数据流
params.put("profile_picture3", new ByteArrayInputStream(bytes)); //  提交字节流
TatansHttp fh = new TatansHttp();
fh.post("http://www.yangfuhai.com", params, new HttpRequestCallBack(){
@Override
public void onLoading(long count, long current) {
textView.setText(current+"/"+count);
}
@Override
public void onSuccess(String t) {
textView.setText(t==null?"null":t);
```

```
}
});
```

方法二(静态变量):

```
/**
*
* @param url
*                  发送请求的URL
* @param context
*                  当前Context上下文
* @return  服务器响应字符串
* @throws Exception
*/
//没有参数的post请求
```

String sRequest=HttpUtil.postRequest(String url, Contextcontext);
/**
*
* **@param** url
* 发送请求的URL
* **@param** rawPapams
* Map<String,String>
* **@param** context
* 当前Context上下文
* **@return** 服务器响应字符串
* **@throws** Exception
*/
//有参数的post请求

```
Map<String, String> map2 = new HashMap<String, String>();
map2.put("id", sId);
map2.put("pageNo", "2");
```

String sRequest=HttpUtil.postRequest(“www.baidu.com”,map2,Context);

使用 TatansHttp 下载文件：

方法一(回调方式):

支持断点续传，随时停止下载任务 或者 开始任务

```
TatansHttp fh = new TatansHttp();
//调用download方法开始下载
HttpHandler handler = fh.download("http://other.tatans.net/apksource/net.tatans.setting/1.1.1.apk",
new HttpRequestParams(),DirPath.getMyDir(“路径”,”apknam.apk”),//默认路径在tatans
true,//true:断点续传 false:不断点续传（全新下载）
new HttpRequestCallBack() {
@Override
public void onLoading(long count, long current) {
textView.setText("下载进度："+current+"/"+count);
}
@Override
public void onSuccess(File t) {
textView.setText(t==null?"null":t.getAbsoluteFile().toString());
}
});
//调用stop()方法停止下载
handler.stop();
```

方法二(静态变量)

```
/**
*
* @param file
*                下载文件存放路径
* @param rawPapams
*                Map<String,String>
* @param context
*                当前Context上下文
```

* **@return**服务器响应字符串

* **@throws**Exception
*/
//有参数的post请求
HttpUtil.downLoadFile(File file,String url,Map<String ,String> rawParams, Context context)