### 1.控件样式常用属性

**java代码：**  
<!--  
android:id —— 为控件指定相应的ID  
android:text —— 指定控件当中显示的文字，需要注意的是，这里尽量使用strings.xml文件当中的字符串  
android:grivity —— 指定控件的基本位置，比如说居中，居右等位置  
android:textSize —— 指定控件当中字体的大小  
android:background —— 指定该控件所使用的背景色,RGB命名法   
android:width —— 指定控件的宽度  
android:height —— 指定控件的高度  
android:padding\* —— 指定控件的内边距，也就是说控件当中的内容  
android:sigleLine —— 如果设置为真的话，则将控件的内容在同一行当中进行显示  
-->

       2.控件相对布局常用属性   
  
**java代码：**  
<!--  
android:layout\_above 将该控件的底部至于给定ID的控件之上  
android:layout\_below 将该控件的顶部至于给定ID的控件之下  
android:layout\_toLeftOf 将该控件的右边缘和给定ID的控件的左边缘对齐  
android:layout\_toRightOf 将该控件的左边缘和给定ID的控件的右边缘对齐

android:layout\_alignBaseline 该控件的baseline和给定ID的控件的baseline对齐  
android:layout\_alignBottom 将该控件的底部边缘与给定ID控件的底部边缘  
android:layout\_alignLeft 将该控件的左边缘与给定ID控件的左边缘对齐  
android:layout\_alignRight 将该控件的右边缘与给定ID控件的右边缘对齐  
android:layout\_alignTop 将给定控件的顶部边缘与给定ID控件的顶部对齐  
android:alignParentBottom 如果该值为true，则将该控件的底部和父控件的底部对齐  
android:layout\_alignParentLeft 如果该值为true，则将该控件的左边与父控件的左边对齐  
android:layout\_alignParentRight 如果该值为true，则将该控件的右边与父控件的右边对齐  
android:layout\_alignParentTop 如果该值为true，则将空间的顶部与父控件的顶部对齐

android:layout\_centerHorizontal 如果值为真，该控件将被至于水平方向的中央  
android:layout\_centerInParent 如果值为真，该控件将被至于父控件水平方向和垂直方向的中央  
android:layout\_centerVertical 如果值为真，该控件将被至于垂直方向的中央  
-->

### 2.Android单位

Dip:不同设备有不同的显示效果。

Px: 不同设备相同效果

Pt:point 1pt=1/72英寸

Sp:scaled pixels(放大像素)

### 3.java 内部类

内部类可以使用外部的成员变量或成员函数。

### 4．监听器

### 5．Intent对象

Intent 对象可以放置键值对。用于不同activity之间传递值；

**class** CalulateListener **implements** OnClickListener{

@Override

**public** **void** onClick(View arg0) {

//既得输入的控件值

String factorOneStr=factorOne.getText().toString();

String factorTwoStr=factorTwo.getText().toString();

//将相关的值放入intent对象

Intent intent=**new** Intent();

intent.putExtra("one", factorOneStr);

intent.putExtra("two", factorTwoStr);

intent.setClass(MainActivity03.**this**, ResultActivity.**class**);

//使用这个intent对象来启动ResultActivity

MainActivity03.**this**.startActivity(intent);

}

}

### 6.Eclipse断点失效的解决办法：

通过Run——Skip All Breakpoints来搞定，就是取消你“跳过所有断点”的设置。

为什么会这样，很可能是哪次你手太快太慌，点错了。

### 7.Android生命周期函数

Onstart: 可见

onResume:取得焦点

### 8.匿名内部类：

   使用匿名内部类课使代码更加简洁、紧凑，模块化程度更高。内部类能够访问外部内的一切成员变量和方法，包括私有的，而实现接口或继承类做不到。然而这个不是我说的重点，我说的很简单，就是匿名内部类的两种实现方式：第一种，继承一个类，重写其方法；第二种，实现一个接口（可以是多个），实现其方法。下面通过代码来说明：

Java代码 [复制代码](http://ayis.iteye.com/blog/219590) [[收藏代码http://ayis.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. public class TestAnonymousInterClass{
2. public static void main(String args[]){
3. TestAnonymousInterClass test=new TestAnonymousInterClass();
4. test.show();
5. }
6. //在这个方法中构造了一个匿名内部类
7. private void show(){
8. Out anonyInter=new Out(){// 获取匿名内部类实例
10. void show(){//重写父类的方法
11. System.out.println("this is Anonymous InterClass showing.");
12. }
13. };
14. anonyInter.show();// 调用其方法
15. }
16. }
18. // 这是一个已经存在的类，匿名内部类通过重写其方法，将会获得另外的实现
19. class Out{
20. void show(){
21. System.out.println("this is Out showing.");
22. }
23. }

public class TestAnonymousInterClass{

public static void main(String args[]){

TestAnonymousInterClass test=new TestAnonymousInterClass();

test.show();

}

//在这个方法中构造了一个匿名内部类

private void show(){

Out anonyInter=new Out(){// 获取匿名内部类实例

void show(){//重写父类的方法

System.out.println("this is Anonymous InterClass showing.");

}

};

anonyInter.show();// 调用其方法

}

}

// 这是一个已经存在的类，匿名内部类通过重写其方法，将会获得另外的实现

class Out{

void show(){

System.out.println("this is Out showing.");

}

}

程序运行的输出结果为：

this is Anonymous InterClass showing.

   所以在这里看出，匿名内部类有了自己的实现。其实很简单，使用匿名内部类是因为我这地方需要有点什么特殊的实现，所以我就在这地方把具体实现也给了出来了。然后我就在这地方获取它的实例，调用它的方法。

   接口的方式，只要把父类换成接口就行了，没必要给出代码了。

   使用匿名内部类时我们不要忘了我们的目的，我们只是在这地方想对某个类有特殊的实现。而不要想得太多，在匿名内部编写其它的方法。在匿名内部类中编写的自己的方法是不可见的。此种做法是毫无意义的，当然一般也不会这么做。在这里只是告诉初学者对于匿名内部类不要想的太多，而要这么想：匿名内部类就是重写父类或接口的方法。

 匿名内部类是没有名字的，所以我们没办法获得其类型，而只能把它当作超类或接口类型来使用。

当然，我这篇文章不是很有深度。欢迎发表意见!

### 9.java final,static

　根据程序上下文环境，[Java](http://java.chinaitlab.com/)关键字final有“这是无法改变的”或者“终态的”含义，它可以修饰非抽象类、非抽象类成员方法和变量。你可能出于两种理解而需要阻止改变：设计或效率。   
final类不能被继承，没有子类，final类中的方法默认是final的。   
final方法不能被子类的方法覆盖，但可以被继承。   
final成员变量表示常量，只能被赋值一次，赋值后值不再改变。   
final不能用于修饰构造方法。   
注意：父类的private成员方法是不能被子类方法覆盖的，因此private类型的方法默认是final类型的。

### 10．Android界面开发工具：

### 11. [Android中使用Thread线程出现的问题](http://www.cnblogs.com/ATree/archive/2010/12/10/Android-Thread.html)

很多初入Android或Java开发的新手对Thread、Looper、Handler和Message仍然比较迷惑，衍生的有 HandlerThread、java.util.concurrent、Task、AsyncTask由于目前市面上的书籍等资料都没有谈到这些问题， 今天Android123就这一问题做更系统性的总结.  
  
　　Android开发过程中为什么要线程能？  
  
　　我们创建的 Service、Activity以及Broadcast均是一个主线程处理，这里我们可以理解为UI线程.但是在操作一些耗时操作时，比如I/O读写的 大文件读写，数据库操作以及网络下载需要很长时间，为了不阻塞用户界面，出现ANR的响应提示窗口，这个时候我们可以考虑使用Thread线程来解决.  
  
　　Android中使用Thread线程会遇到哪些问题？  
  
　 　对于从事过J2ME开发的程序员来说Thread比较简单，直接匿名创建重写run方法，调用start方法执行即可.或者从Runnable接口继 承，但对于Android平台来说UI控件都没有设计成为线程安全类型，所以需要引入一些同步的机制来使其刷新，这点Google在设计Android时 倒是参考了下Win32的消息处理机制.  
  
　　1. 对于线程中的刷新一个View为基类的界面，可以使用postInvalidate（）方法在线程中来处理，其中还提供了一些重写方法比如 postInvalidate（int left，int top，int right，int bottom） 来刷新一个矩形区域，以及延时执行，比如postInvalidateDelayed（long delayMilliseconds）或postInvalidateDelayed（long delayMilliseconds，int left，int top，int right，int bottom） 方法，其中第一个参数为毫秒，如下：  
  
　　2. 当然推荐的方法是通过一个Handler来处理这些，可以在一个线程的run方法中调用handler对象的 postMessage或sendMessage方法来实现，Android程序内部维护着一个消息队列，会轮训处理这些，如果你是Win32程序员可以 很好理解这些消息处理，不过相对于Android来说没有提供 PreTranslateMessage这些干涉内部的方法.  
  
　　3. Looper又是什么呢？ ，其实Android中每一个Thread都跟着一个Looper，Looper可以帮助Thread维护一个消息队列，昨天的问题 Can't create handler inside thread 错误 一文中提到这一概念，但是Looper和Handler没有什么关系，我们从开源的代码可以看到Android还提供了一个Thread继承类 HanderThread可以帮助我们处理，在HandlerThread对象中可以通过getLooper方法获取一个Looper对象控制句柄，我们 可以将其这个Looper对象映射到一个Handler中去来实现一个线程同步机制，Looper对象的执行需要初始化Looper.prepare方法 就是昨天我们看到的问题，同时推出时还要释放资源，使用Looper.release方法.  
  
　　4.Message 在Android是什么呢？ 对于Android中Handler可以传递一些内容，通过Bundle对象可以封装String、Integer以及Blob二进制对象，我们通过在线 程中使用Handler对象的 sendEmptyMessage或sendMessage方法来传递一个Bundle对象到Handler处理器.对于Handler类提供了重写方法 handleMessage（Message msg） 来判断，通过msg.what来区分每条信息.将Bundle解包来实现Handler类更新UI线程中的内容实现控件的刷新操作.相关的Handler 对象有关消息发送sendXXXX相关方法如下，同时还有postXXXX相关方法，这些和Win32中的道理基本一致，一个为发送后直接返回，一个为处 理后才返回 ：  
  
　　5. java.util.concurrent对象分析，对于过去从事Java开发的程序员不会对Concurrent对象感到陌生吧，他是JDK 1.5以后新增的重要特性作为掌上设备，我们不提倡使用该类，考虑到Android为我们已经设计好的Task机制，我们这里Android开发网对其不 做过多的赘述，相关原因参考下面的介绍：  
  
　　6. 在Android中还提供了一种有别于线程的处理方式，就是Task以及AsyncTask，从开源代码中可以看到是针对Concurrent的封装，开 发人员可以方便的处理这些异步任务，具体的Android123在以前的文章中有详细解释，可以使用站内搜索来了解更多

### 12.Android handler作用

每一个handler都有一个与之相关联的消息队列。

线程中的run函数执行完成线程就结束。

### 13．Java实现线程的两种方法:

##### 13.1继承thread类

第二种是继承Thread类，覆盖它的run（）方法。

　　public class DoAnotherThing extends Thread {

　　public void run（）{

　　// here is where you do something

　　}

　　}

##### 13.2实现runable接口

第一是实现Runnable接口实现它的run（）方法，

　　public class DoSomething implements Runnable {

　　public void run（）{

　　// here is where you do something

　　}

}

　　这两种方法的区别是，如果你的类已经继承了其它的类，那么你只能选择实现Runnable接口了，因为Java只允许单继承的。

Java中的线程有四种状态分别是：运行、就绪、挂起、

结束（死亡（Dead）：当一个线程正常结束，它便处于死亡状态。如一个线程的run()函数执行完毕后线程就进入死亡状态。）

如果一个线程结束了也就说明他是一个死线程了。当你调用一个线程实例的 start（）的方法的时候，这个时候线程进入就绪状态，注意并不是运行状态，当虚拟机开始分配给他CPU的运行时间片的时候线程开始进入运行状态，当线 程进入等待状态，例如等待某个事件发生的时候，这时候线程处于挂起状态。

　　启动一个线程你只需要调用start（）方法，针对两种实现线程的方法也有两种启动线程的方法，分别如下：

　　DoSomething doIt = new DoSomething（）；

　　Thread myThread = new Thread（ doIt ）；

myThread.start（）；

Starts executing the active part of the class' code. This method is called when a thread is started that has been created with a class which implements Runnable.

　　DoAnotherThing doIt = new DoAnotherThing（）；

　　doIt.start（）；

　　由于安全等因素Thread中的stop（）方法已经不推荐使用了，因此如果你想要停止一个线程的时候可以通过设置一个信号量，例如：

　　public class MyThread implements Runnable {

　　private boolean quit = false；

　　public void run（）{

　　while（ ！quit ）{

　　// do something

　　}

　　}

　　public void quit（）{

　　quit = true；

　　}

　　}

##### 13.3线程的同步以及生产者消费者的实现：

　如果每个线程只做它自己的事情，那么就很简单了，但是有的时候几个线程可能要同时访问一个对象并可能对它进行修改，这个时候你必须使用线程的同步在方法或者代码块使用关键字synchronized，例如：

　　public class Counter {

　　private int counter；

　　public synchronized int increment（）{

　　return ++counter；

　　}

　　public synchronized int decrement（）{

　　if（ ——counter < 0 ）{

　　counter = 0；

　　}

　　return counter；

　　}

　　}

　　每个java对象都可以最为一个监视器，当线程访问它的synchronized方法的时候，他只允许在一个时间只有一个线程对他访问，让其他 得线程排队等候。这样就可以避免多线程对共享数据造成破坏。记住synchronized是会耗费系统资源降低程序执行效率的，因此一定要在需要同步的时 候才使用，尤其在J2ME的开发中要小心。如果你要是想让线程等待某个事件的发生然后继续执行的话，那么这就涉及到线程的调度了。在java中通过 wait（），notify（），notifyAll（）来实现，这三个方法是在Object类中定义的，当你想让线程挂起的时候调用 obj.wait（）方法，在同样的obj上调用notify（）则让线程重新开始运行。

　　最后以SUN提供的Producer/Consumer的例子来结束这篇文章，内容是Producer产生一个数字而Consumer消费这个数字，这个小程序里面基本覆盖了本文所有的知识点。请详细研究一下代码

　　public class Producer extends Thread {

　　private CubbyHole cubbyhole；

　　private int number；

　　public Producer（CubbyHole c， int number） {

　　cubbyhole = c；

　　this.number = number；

　　}

　　public void run（） {

　　for （int i = 0； i < 10； i++） {

　　cubbyhole.put（i）；

　　System.out.println（"Producer #" + this.number

　　+ " put： " + i）；

　　try {

　　sleep（（int）（Math.random（） \* 100））；

　　} catch （InterruptedException e） { }

　　}

　　}

　　}

　　public class CubbyHole {

　　private int contents；

　　private boolean available = false；

　　public synchronized int get（） {

　　while （available == false） {

　　try {

　　wait（）；

　　} catch （InterruptedException e） { }

　　}

　　available = false；

　　notifyAll（）；

　　return contents；

　　}

　　public synchronized void put（int value） {

　　while （available == true） {

　　try {

　　wait（）；

　　} catch （InterruptedException e） { }

　　}

　　contents = value；

　　available = true；

　　notifyAll（）；

　　}

　　}

　　public class Consumer extends Thread {

　　private CubbyHole cubbyhole；

　　private int number；

　　public Consumer（CubbyHole c， int number） {

　　cubbyhole = c；

　　this.number = number；

　　}

　　public void run（） {

　　int value = 0；

　　for （int i = 0； i < 10； i++） {

　　value = cubbyhole.get（）；

　　System.out.println（"Consumer #" + this.number

　　+ " got： " + value）；

　　}

　　}

　　}

　　public class ProducerConsumerTest {

　　public static void main（String[] args） {

　　CubbyHole c = new CubbyHole（）；

　　Producer p1 = new Producer（c， 1）；

　　Consumer c1 = new Consumer（c， 1）；

　　p1.start（）；

　　c1.start（）；

　　}

　　}

　　SUN说输出的结果应该是如下形式，但是在我的机器上却不是这样的，做了一些改动才正确，有兴趣的朋友可以运行一下看看结果，欢迎和我讨论一下！

　　Producer #1 put： 0

　　Consumer #1 got： 0

　　Producer #1 put： 1

　　Consumer #1 got： 1

　　Producer #1 put： 2

　　Consumer #1 got： 2

　　Producer #1 put： 3

　　Consumer #1 got： 3

　　Producer #1 put： 4

　　Consumer #1 got： 4

　　Producer #1 put： 5

　　Consumer #1 got： 5

　　Producer #1 put： 6

　　Consumer #1 got： 6

　　Producer #1 put： 7

　　Consumer #1 got： 7

　　Producer #1 put： 8

　　Consumer #1 got： 8

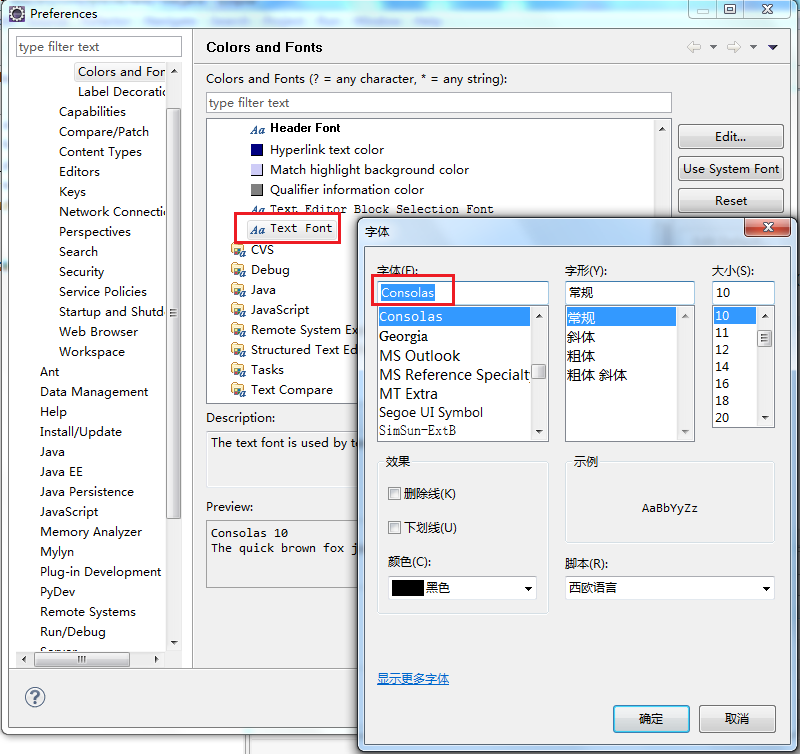
　　Producer #1 put： 9

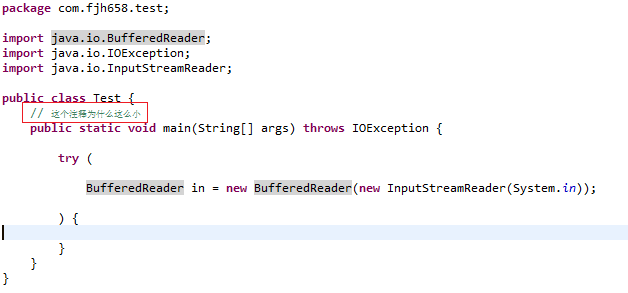
　　Consumer #1 got： 9

### 14. [关于eclipse的indigo版中文注释时字体太小的问题](http://blog.csdn.net/fjh658/article/details/7521944)

eclipse目前最新版代号indigo, 在win7上使用时中文注释时字体太小的问题. 为什么会这样?

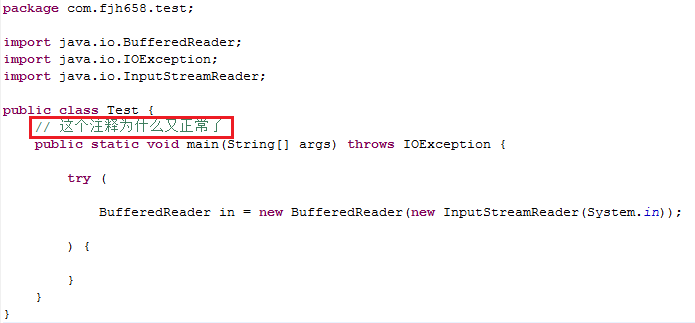
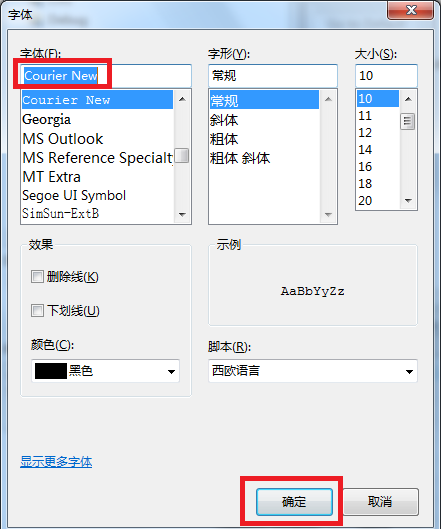
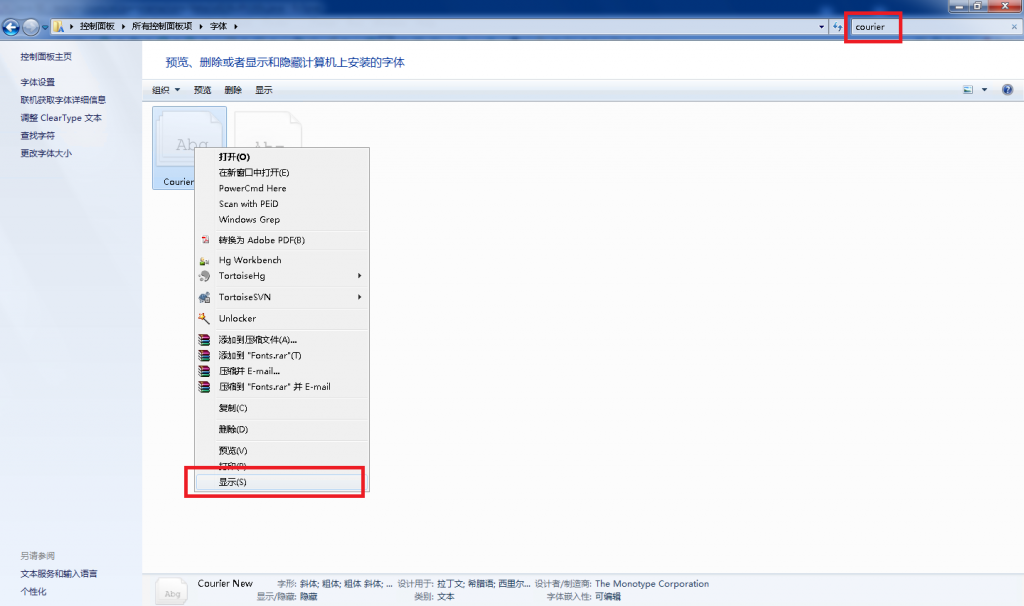
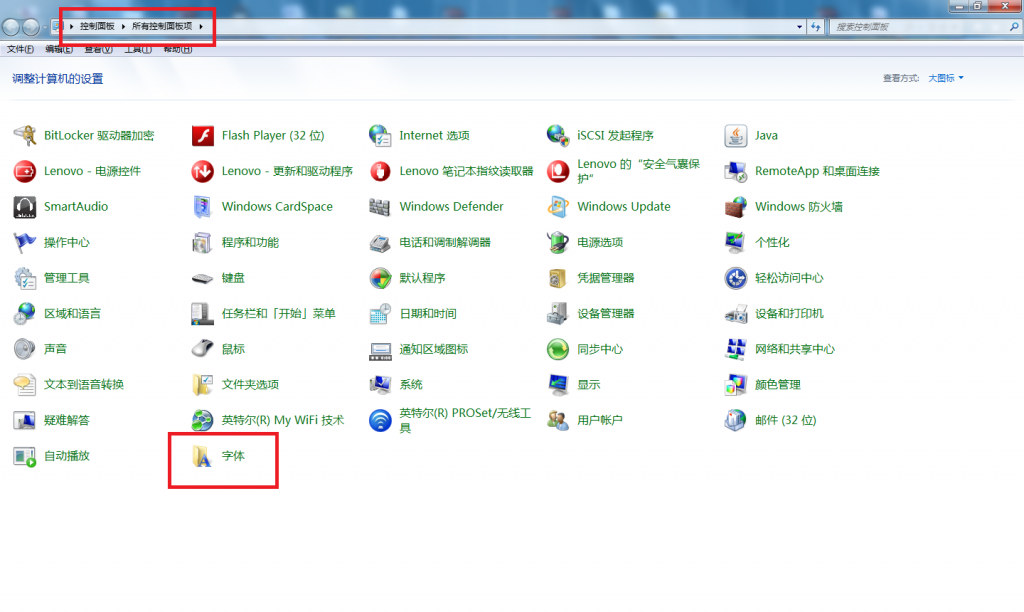
首先我们应该知道, 在win7系统中, font是有"显示"和"隐藏" 状态的. 默认情况下, eclipse使用的默认字体courier new是处于

"隐藏"下的. 这样当eclipse打开时就会使用另一种字体consolas

这时我们看到的字体就是

根据我前面说的字体的状态, 大家可能已经猜到了, 对就是将courier new变成"显示"状态

操作如下



### 15.handler与线程的关系

Handler.post将相关的进程放入队列中，而相的进程的run方法是默认是在主线程中执行的。

### 16.android looper用法:

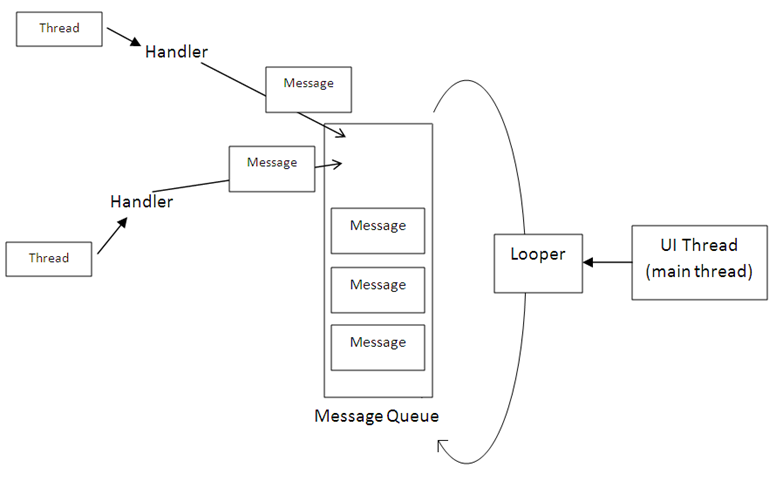
 Android系统的消息队列和消息循环都是针对具体线程的，一个线程可以存在（当然也可以不存在）一个消息队列（Message Queue）和一个消息循环（Looper）。Android中除了UI线程（主线程），创建的工作线程默认是没有消息循环和消息队列的。如果想让该线程 具有消息队列和消息循环，并具有消息处理机制，就需要在线程中首先调用Looper.prepare()来创建消息队列，然后调用 Looper.loop()进入消息循环。如以下代码所示：

Java代码  [收藏代码](javascript:void())

1. class LooperThread extends Thread {
2. public Handler mHandler;
4. public void run() {
5. Looper.prepare();
7. mHandler = new Handler() {
8. public void handleMessage(Message msg) {
9. // process incoming messages here
10. }
11. };
13. Looper.loop();
14. }
15. }

       这样该线程就具有了消息处理机制了。如果不调用Looper.prepare()来创建消息队列，会 报"java.lang.RuntimeException: Can't create handler inside thread that has not called Looper.prepare()"的错误。

       通过下图可以清晰显示出UI Thread, Worker Thread, Handler, Massage Queue, Looper之间的关系：



       解释上图中的几个基本概念：

**1.Message**

       消息对象，顾名思义就是记录消息信息的类。这个类有几个比较重要的字段：

       a.arg1和arg2：我们可以使用两个字段用来存放我们需要传递的整型值，在Service中，我们可以用来存放Service的ID。

       b.obj：该字段是Object类型，我们可以让该字段传递某个多项到消息的接受者中。

       c.what：这个字段可以说是消息的标志，在消息处理中，我们可以根据这个字段的不同的值进行不同的处理，类似于我们在处理Button事件时，通过switch(v.getId())判断是点击了哪个按钮。

       在使用Message时，我们可以通过new Message()创建一个Message实例，但是Android更推荐我们通过Message.obtain()或者 Handler.obtainMessage()获取Message对象。这并不一定是直接创建一个新的实例，而是先从消息池中看有没有可用的 Message实例，存在则直接取出并返回这个实例。反之如果消息池中没有可用的Message实例，则根据给定的参数new一个新Message对象。 通过分析源码可得知，Android系统默认情况下在消息池中实例化10个Message对象。

**2.MessageQueue**

       消息队列，用来存放Message对象的数据结构，按照“先进先出”的原则存放消息。存放并非实际意义的保存，而是将Message对象以链表的方式串联 起来的。MessageQueue对象不需要我们自己创建，而是有Looper对象对其进行管理，一个线程最多只可以拥有一个MessageQueue。 我们可以通过Looper.myQueue()获取当前线程中的MessageQueue。

**3.Looper**

       MessageQueue的管理者，在一个线程中，如果存在Looper对象，则必定存在MessageQueue对象，并且只存在一个Looper对象 和一个MessageQueue对象。倘若我们的线程中存在Looper对象，则我们可以通过Looper.myLooper()获取，此外我们还可以通 过Looper.getMainLooper()获取当前应用系统中主线程的Looper对象。在这个地方有一点需要注意，假如Looper对象位于应用 程序主线程中，则Looper.myLooper()和Looper.getMainLooper()获取的是同一个对象。

**4.Handler**

       消息的处理者。通过Handler对象我们可以封装Message对象，然后通过sendMessage(msg)把Message对象添加到 MessageQueue中；当MessageQueue循环到该Message时，就会调用该Message对象对应的handler对象的 handleMessage()方法对其进行处理。由于是在handleMessage()方法中处理消息，因此我们应该编写一个类继承自 Handler，然后在handleMessage()处理我们需要的操作。

       另外，我们知道，Android UI操作并不是线程安全的，所以无法在子线程中更新UI。但Andriod提供了几种方法，可以在子线程中通知UI线程更新界面：

* Activity.runOnUiThread( Runnable )
* View.post( Runnable )
* View.postDelayed( Runnable, long )
* Handler

       比较常用的是通过Handler，用Handler来接收子线程发送的数据，并用此数据配合主线程更新UI。那么，只要在主线程中创建Handler对 象，在子线程中调用Handler的sendMessage方法，就会把消息放入主线程的消息队列，并且将会在Handler主线程中调用该 handler的handleMessage方法来处理消息。

Java代码  [收藏代码](javascript:void())

1. package com.superonion;
3. import android.app.Activity;
4. import android.os.Bundle;
5. import android.os.Message;
6. import android.util.Log;
7. import android.os.Handler;
9. public class MyHandler extends Activity {
10. static final String TAG = "Handler";
11. Handler h = new Handler(){
12. public void handleMessage (Message msg)
13. {
14. switch(msg.what)
15. {
16. case HANDLER\_TEST:
17. Log.d(TAG, "The handler thread id = " + Thread.currentThread().getId() + "\n");
18. break;
19. }
20. }
21. };
23. static final int HANDLER\_TEST = 1;
24. /\*\* Called when the activity is first created. \*/
25. @Override
26. public void onCreate(Bundle savedInstanceState) {
27. super.onCreate(savedInstanceState);
28. Log.d(TAG, "The main thread id = " + Thread.currentThread().getId() + "\n");
30. new myThread().start();
31. setContentView(R.layout.main);
32. }
34. class myThread extends Thread
35. {
36. public void run()
37. {
38. Message msg = new Message();
39. msg.what = HANDLER\_TEST;
40. h.sendMessage(msg);
41. Log.d(TAG, "The worker thread id = " + Thread.currentThread().getId() + "\n");
42. }
43. }
44. }

       以上代码中，Handler在主线程中创建后，子线程通过sendMessage()方法就可以将消息发送到主线程中，并在handleMessage()方法中处理。

### 17.android synchronized 用法

很多网友可能看到部分[Android](http://www.ylmf.net/android/)方法中出现了synchronized字样，比如public static synchronized String getHttpResponse(String url) ，其实这个问题属于Java基础和[Android](http://www.ylmf.net/android/)无 关，使用关键字 synchronized 字面表示同步，而实际中在多线程程序里为了防止结果出现意外(会出什么意外呢? Android123举例比如一个线程正在写文件，还没有写完，但是另一个线程就开始读取了)，加入synchronized可以为该方法上锁，即只有一 个线程单独访问该方法，类似Win32中的临界区Critical Section是一个道理。不过要提示大家该方法不要滥用，在线程安全的场合下使用会增加系统开销影响性能。而有些时候还会在方法中出现 synchronized(this) 字样，不错 synchronized(this) 一般用于锁住某个循环中的内容，在iPhone Cocoa中也同样适用。

**1. synchronized 方法：**通过在方法声明中加入 synchronized关键字来声明 synchronized 方法。如：   
　　public synchronized void accessVal(int newVal);   
　 　synchronized 方法控制对类成员变量的访问：每个类实例对应一把锁，每个 synchronized 方法都必须获得调用该方法的类实例的锁方能执行，否则所属线程阻塞，方法一旦执行，就独占该锁，直到从该方法返回时才将锁释放，此后被阻塞的线程方能获得 该锁，重新进入可执行状态。这种机制确保了同一时刻对于每一个类实例，其所有声明为 synchronized 的成员函数中至多只有一个处于可执行状态（因为至多只有一个能够获得该类实例对应的锁），从而有效避免了类成员变量的访问冲突（只要所有可能访问类成员变 量的方法均被声明为 synchronized）。   
　　在 Java 中，不光是类实例，每一个类也对应一把锁，这样我们也可将类的静态成员函数声明为 synchronized ，以控制其对类的静态成员变量的访问。   
　 　synchronized 方法的缺陷：若将一个大的方法声明为synchronized 将会大大影响效率，典型地，若将线程类的方法 run() 声明为 synchronized ，由于在线程的整个生命期内它一直在运行，因此将导致它对本类任何 synchronized 方法的调用都永远不会成功。当然我们可以通过将访问类成员变量的代码放到专门的方法中，将其声明为 synchronized ，并在主方法中调用来解决这一问题，但是 Java 为我们提供了更好的解决办法，那就是 synchronized 块。   
  
**2. synchronized 块：**通过 synchronized关键字来声明synchronized 块。语法如下：   
　　synchronized(syncObject) {   
　　//允许访问控制的代码   
　　}   
　　synchronized 块是这样一个代码块，其中的代码必须获得对象 syncObject （如前所述，可以是类实例或类）的锁方能执行，具体机制同前所述。由于可以针对任意代码块，且可任意指定上锁的对象，故灵活性较高。   
　　对synchronized(this)的一些理解  
　　一、当两个并发线程访问同一个对象object中的这个synchronized(this)同步代码块时，一个时间内只能有一个线程得到执行。另一个线程必须等待当前线程执行完这个代码块以后才能执行该代码块。   
　　二、然而，当一个线程访问object的一个synchronized(this)同步代码块时，另一个线程仍然可以访问该object中的非synchronized(this)同步代码块。   
　　三、尤其关键的是，当一个线程访问object的一个synchronized(this)同步代码块时，其他线程对object中所有其它synchronized(this)同步代码块的访问将被阻塞。   
　　四、第三个例子同样适用其它同步代码块。也就是说，当一个线程访问object的一个synchronized(this)同步代码块时，它就获得了这个object的对象锁。结果，其它线程对该object对象所有同步代码部分的访问都被暂时阻塞。   
　　五、以上规则对其它对象锁同样适用

### 18.android程序安装：

1、最简单的就是从“电子市场”上下载，上面有很多的免费软件，可以很方便的下载和安装程序。同类型的替代软件还有飞流下载和地瓜游戏。

2、可以从本站（<http://www.myapks.com/>）下载软件安装包，存放在手机的SD卡上，然后使用文件管理器进行安装，这样就可以在Android手机里直接安装。如果手机上没有文件管理器，建议使用方法1在电子市场上先搜索es关键字，安装ES文件浏览器免费版本。如果连电子市场都没有，那么使用方法3先安装一个文件管理器。

3、可以用USB数据线连接电脑，然后在电脑上对Android手机进行安装，但是首先需要在电脑上安装APK安装器或者豌豆荚或者是魔乐手机助手等软件，可以批量安装程序，非常方便。

以上第2和第3种方法都需要对手机进行设置，“菜单键”-settings（设置）-Applications（应用程序）-Unknown sources（选择“未知源”）和进入Development（开发）-USB debugging（选择“USB 调试”）

### 19.android中的broadcastReceiver

本文来自<http://blog.csdn.net/hellogv/> ，引用必须注明出处！

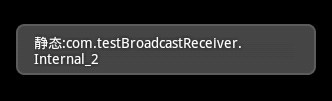
       前面分别讨论了[Activity](http://blog.csdn.net/hellogv/archive/2010/11/06/5992198.aspx)和[Service](http://blog.csdn.net/hellogv/archive/2010/11/08/5994952.aspx)， 这次就轮到BroastcastReceiver，Broastcast是应用程序间通信的手段。BroastcastReceiver也是跟 Intent紧密相连的，动态/静态注册了BroastcastReceiver之后，使用sendBroadcast把Intent发送之后，系统会自 动把符合条件的BroastcastReceiver启动，跟嵌入式系统的中断类似。

        本文主要演示了如何静态/动态注册BroastcastReceiver，向系统索取电量信息，以及枚举信息的字段。本文运行截图如下：





上图是发送Intent至内部动态注册的BroadcastReceiver，接收到之后显示消息名称。动态注册BroadcastReceiver用到registerReceiver()。

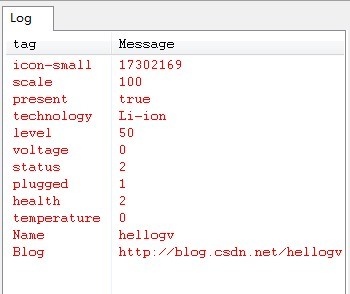


上图是发送Intent至内部静态注册的BroadcastReceiver，接收到之后显示消息名称。静态注册比动态注册麻烦点，先新建一个类继承BroadcastReceiver，然后到AndroidManifest.xml 添加

**[xhtml]** [view plaincopyprint?](http://blog.csdn.net/hellogv/article/details/5999170)

1. <receiver android:name="clsReceiver2">
2. <intent-filter>
3. <action
4. android:name="com.testBroadcastReceiver.Internal\_2"/>
5. </intent-filter>
6. </receiver>

第一个name是类名，第二个是action的名称。



上图是枚举Intent消息的字段，这个功能比较适合懒人，把收到的Intent消息的字段全部分解了，再看看哪个需要的，懒得记住。实现这部分的代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/hellogv/article/details/5999170)

1. //当未知Intent包含的内容，则需要通过以下方法来列举
2. Bundle b=intent.getExtras();
3. Object[] lstName=b.keySet().toArray();
5. for(int i=0;i<lstName.length;i++)
6. {
7. String keyName=lstName[i].toString();
8. Log.e(keyName,String.valueOf(b.get(keyName)));
9. }

main.xml的代码如下：

**[xhtml]** [view plaincopyprint?](http://blog.csdn.net/hellogv/article/details/5999170)

1. <?xml version="1.0" encoding="utf-8"?>
2. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3. android:orientation="vertical" android:layout\_width="fill\_parent"
4. android:layout\_height="fill\_parent">
6. <Button android:id="@+id/Button01" android:layout\_width="wrap\_content"
7. android:layout\_height="wrap\_content" android:text="发送至内部动态注册的BroadcastReceiver"></Button>
8. <Button android:id="@+id/Button02" android:layout\_width="wrap\_content"
9. android:layout\_height="wrap\_content" android:text="发送至内部静态注册BroadcastReceiver"></Button>
10. <Button android:id="@+id/Button03" android:layout\_width="wrap\_content"
11. android:layout\_height="wrap\_content" android:text="发送至系统BroadcastReceiver"></Button>
12. </LinearLayout>

testBroadcastReceiver.java的代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/hellogv/article/details/5999170)

1. package com.testBroadcastReceiver;
3. import android.app.Activity;
4. import android.content.BroadcastReceiver;
5. import android.content.Context;
6. import android.content.Intent;
7. import android.content.IntentFilter;
8. import android.os.Bundle;
9. import android.util.Log;
10. import android.view.View;
11. import android.widget.Button;
12. import android.widget.Toast;
14. public class testBroadcastReceiver extends Activity {
15. Button btnInternal1,btnInternal2,btnSystem;
16. static final String INTENAL\_ACTION\_1 = "com.testBroadcastReceiver.Internal\_1";
17. static final String INTENAL\_ACTION\_2 = "com.testBroadcastReceiver.Internal\_2";
18. static final String INTENAL\_ACTION\_3 = "com.testBroadcastReceiver.Internal\_3";
19. @Override
20. public void onCreate(Bundle savedInstanceState) {
21. super.onCreate(savedInstanceState);
22. setContentView(R.layout.main);
23. btnInternal1=(Button)this.findViewById(R.id.Button01);
24. btnInternal1.setOnClickListener(new ClickEvent());
25. btnInternal2=(Button)this.findViewById(R.id.Button02);
26. btnInternal2.setOnClickListener(new ClickEvent());
27. btnSystem=(Button)this.findViewById(R.id.Button03);
28. btnSystem.setOnClickListener(new ClickEvent());
29. //动态注册广播消息
30. registerReceiver(bcrIntenal1, new IntentFilter(INTENAL\_ACTION\_1));
31. }
32. class ClickEvent implements View.OnClickListener{
34. @Override
35. public void onClick(View v) {
36. if(v==btnInternal1)//给动态注册的BroadcastReceiver发送数据
37. {
38. Intent intent = new Intent(INTENAL\_ACTION\_1);
39. sendBroadcast(intent);
40. }
41. else if(v==btnInternal2)//给静态注册的BroadcastReceiver发送数据
42. {
43. Intent intent = new Intent(INTENAL\_ACTION\_2);
44. sendBroadcast(intent);
45. }
46. else if(v==btnSystem)//动态注册 接收2组信息的BroadcastReceiver
47. {
48. IntentFilter filter = new IntentFilter();//
49. filter.addAction(Intent.ACTION\_BATTERY\_CHANGED);//系统电量检测信息
50. filter.addAction(INTENAL\_ACTION\_3);//第三组自定义消息
51. registerReceiver(batInfoReceiver, filter);
53. Intent intent = new Intent(INTENAL\_ACTION\_3);
54. intent.putExtra("Name", "hellogv");
55. intent.putExtra("Blog", "http://blog.csdn.net/hellogv");
56. sendBroadcast(intent);//传递过去
57. }
58. }
60. }
62. /\*
63. \* 接收动态注册广播的BroadcastReceiver
64. \*/
65. private BroadcastReceiver bcrIntenal1 = new BroadcastReceiver() {
67. public void onReceive(Context context, Intent intent) {
68. String action = intent.getAction();
69. Toast.makeText(context, "动态:"+action, 1000).show();
70. }
71. };

74. private BroadcastReceiver batInfoReceiver = new BroadcastReceiver() {
76. public void onReceive(Context context, Intent intent) {
77. String action = intent.getAction();
78. //如果捕捉到的action是ACTION\_BATTERY\_CHANGED
79. if (Intent.ACTION\_BATTERY\_CHANGED.equals(action)) {
80. //当未知Intent包含的内容，则需要通过以下方法来列举
81. Bundle b=intent.getExtras();
82. Object[] lstName=b.keySet().toArray();
84. for(int i=0;i<lstName.length;i++)
85. {
86. String keyName=lstName[i].toString();
87. Log.e(keyName,String.valueOf(b.get(keyName)));
88. }
89. }
90. //如果捕捉到的action是INTENAL\_ACTION\_3
91. if (INTENAL\_ACTION\_3.equals(action)) {
92. //当未知Intent包含的内容，则需要通过以下方法来列举
93. Bundle b=intent.getExtras();
94. Object[] lstName=b.keySet().toArray();
96. for(int i=0;i<lstName.length;i++)
97. {
98. String keyName=lstName[i].toString();
99. Log.e(keyName,b.getString(keyName));
100. }
101. }
102. }
103. };

106. }

clsReceiver2.java的代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/hellogv/article/details/5999170)

1. package com.testBroadcastReceiver;
3. import android.content.BroadcastReceiver;
4. import android.content.Context;
5. import android.content.Intent;
6. import android.widget.Toast;
8. /\*
9. \* 接收静态注册广播的BroadcastReceiver，
10. \* step1:要到AndroidManifest.xml这里注册消息
11. \*      <receiver android:name="clsReceiver2">
12. <intent-filter>
13. <action
14. android:name="com.testBroadcastReceiver.Internal\_2"/>
15. </intent-filter>
16. </receiver>
17. step2:定义消息的字符串
18. step3:通过Intent传递消息来驱使BroadcastReceiver触发
19. \*/
20. public class clsReceiver2 extends BroadcastReceiver{
21. @Override
22. public void onReceive(Context context, Intent intent) {
23. String action = intent.getAction();
24. Toast.makeText(context, "静态:"+action, 1000).show();
26. }
27. }

### 20.抽象类与接口

一、抽象类的定义  
抽象类：包含一个抽象方法的类，称为抽象类。  
抽象方法：只声明而未实现的方法称为抽象方法，抽象方法必须在抽象类中定义。  
抽象类和抽象方法必须使用abstract进行声明。   
例如：以下定义了一个抽象类  
abstract class A{  
private String name ;  
public abstract void print() ;//抽象方法  
public void setName(String name){//非抽象方法  
   this.name = name ;  
}  
};   
抽象类中能否有构造方法？——能  
abstract class A{  
private String name ;  
public A(){  
     System.out.println(“抽象类中的构造方法”);  
}  
public abstract void print() ;  
public void setName(String name){  
   this.name = name ;  
}  
};  
抽象类能否实例化？——不能  
abstract class A{  
private String name ;  
public abstract void print() ;//抽象方法  
public void setName(String name){//非抽象方法  
   this.name = name ;  
}  
};   
   public class X{  
     public static void main(String[] argc)  
     {  
           A a=new A();  
     }  
}  
分析：正常情况下，一个类产生了一个对象之后，此对象就可以调用类中的公共方法了。但是在抽象类中的方法定义的是抽象的，没有实现的。  
所以，抽象类无法直接实例化 。

抽象类的定义上能否用final修饰？——不能  
final：被final定义的类不能有子类  
抽象类：必须有子类，而且子类（普通类）必须实现抽象类中的全部抽象方法

二、接口的定义  
接口的作用：   
    把方法的名称或数据提供给用户  
接口的定义：   
    只包含抽象方法或全局常量的类称为接口。  
注意：  
    所有的方法都是抽象的方法

方法接口：  
    public interface A{  
         public abstract void add( Object a, Object b);  
         public abstract void sub( Object a, Object b);  
         public abstract void multi( Object a, Object b);  
         public abstract void divi( Object a, Object b);  
};

常量接口：  
    public interface Constants{  
        public static final String DATA\_SOURCE\_NAME=“MyDB”;  
        public static final String USERNAME=“root”;  
        public static final String PASSWORD=“123”;  
};

Log.v

Log.d

### 21.调试相关IP说明

182.150.230.240

106.187.47.186

数据库：

+--------------------------------+------+----------------------------------+-------+-----------------+-----------+---------------+-----------------+

| host\_id | user | passwd | email | host\_ip | host\_port | host\_local\_ip | host\_local\_port |

+--------------------------------+------+----------------------------------+-------+-----------------+-----------+---------------+-----------------+

| FFFFFFFFFFFFFFFF | test | E10ADC3949BA59ABBE56E057F20F883E | no1 | 182.150.230.240 | 54247 | 192.168.1.100 | 1000 |

| FFFFFFFFFFFFFFFF106.187.47.186 | | | | | 0 | | 0 |

+--------------------------------+------+----------------------------------+-------+-----------------+-----------+---------------+-----------------+

### 22.设置android默认启动的AVD：

那可以在run as  --> Run configurations  
你那个项目的target中选择

### 23.time ->schedule

receivetimer.schedule(

**new** TimerTask() { **public** **void** run()

{

Bundle tmp=**new** Bundle();

tmp.putString("Type", "Timeout");

Message tmpmsg=Recv\_Handler.obtainMessage();

tmpmsg.setData(tmp);

Recv\_Handler.sendMessage(tmpmsg);

SingleInstance inst;

inst = SingleInstance.*getInstance*();

**if**(inst.TransferMode==**false**)

{

inst.ConnectState=**false**;

inst.HostState=**false**;

}

Stored\_num=0;

inst.ID="0";

inst.TOKEN="0";

} }, 8000);////设定为8s

### 24.socket.recieve(！)

### 25.java file stream（！）

使用Java操作文本文件的方法详解   
摘要: 最初java是不支持对文本文件的处理的，为了弥补这个缺憾而引入了Reader和Writer两个类   
最初java是不支持对文本文件的处理的，为了弥补这个缺憾而引入了Reader和Writer两个类，这两个类都是抽象类，Writer中 write(char[] ch,int off,int

length)，flush()和close()方法为抽象方法，Reader中read(char[] ch,int off,int length)和close()方法是抽象方法。子类应该分别实现他们。

　　当我们读写文本文件的时候，采用Reader是非常方便的，比如FileReader，InputStreamReader和BufferedReader。其中最重要的类是InputStreamReader，

它是字节转换为字符的桥梁。你可以在构造器重指定编码的方式，如果不指定的话将采用底层操作系统的默认编码方式，例如GBK等。当使用FileReader读取文件

的时候。

**[java]** [view plaincopy](http://blog.csdn.net/smartcat86/article/details/4085739/)

1. FileReader fr = new FileReader("ming.txt");
2. int ch = 0;
3. while((ch = fr.read())!=-1 )
4. {
5. System.out.print((char)ch);
6. }

其中read()方法返回的是读取得下个字符。当然你也可以使用read(char[] ch,int off,int length)这和处理二进制文件的时候类似，不多说了。如果使用

InputStreamReader来读取文件的时候

while((ch = isr.read())!=-1)  
{  
System.out.print((char)ch);   
}  
这和FileReader并没有什么区别，事实上在FileReader中的方法都是从InputStreamReader中继承过来的。read()方法是比较好费时间的，如果为了提高效率

我们可以使用BufferedReader对Reader进行包装，这样可以提高读取得速度，我们可以一行一行的读取文本，使用readLine()方法。

BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream("ming.txt")));  
String data = null;  
while((data = br.readLine())!=null)  
{  
System.out.println(data);   
}  
当你明白了如何用Reader来读取文本文件的时候那么用Writer写文件同样非常简单。有一点需要注意，当你写文件的时候，为了提高效率，写入的数据会先

放入缓冲区，然后写入文件。因此有时候你需要主动调用flush()方法。与上面对应的写文件的方法为：

**[java:showcolumns]** [view plaincopy](http://blog.csdn.net/smartcat86/article/details/4085739/)

·········10········20········30········40········50········60········70········80········90········100·······110·······120·······130·······140·······150

1. FileWriter fw = new FileWriter("hello.txt");
2. String s = "hello world";
3. fw.write(s,0,s.length());
4. fw.flush();
5. OutputStreamWriter osw = new OutputStreamWriter(new FileOutputStream("hello2.txt"));
6. osw.write(s,0,s.length());
7. osw.flush();
8. PrintWriter pw = new PrintWriter(new OutputStreamWriter(new FileOutputStream("hello3.txt")),true);
9. pw.println(s);

不要忘记用完后关闭流！下面是个小例子，帮助新手理解。其实有的时候java的IO系统是需要我们多记记的，不然哪天就生疏了。

**[java:showcolumns]** [view plaincopy](http://blog.csdn.net/smartcat86/article/details/4085739/)

·········10········20········30········40········50········60········70········80········90········100·······110·······120·······130·······140·······150

1. import java.io.\*;
2. public class TestFile2
3. {
4. public static void main(String[] args) throws IOException
5. {
6. FileReader fr = new FileReader("ming.txt");
7. char[] buffer = new char[1024];
8. int ch = 0;
9. while((ch = fr.read())!=-1 )
10. {
11. System.out.print((char)ch);
12. }
13. InputStreamReader isr = new InputStreamReader(new FileInputStream("ming.txt"));
14. while((ch = isr.read())!=-1)
15. {
16. System.out.print((char)ch);
17. }
18. BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream("ming.txt")));
19. String data = null;
20. while((data = br.readLine())!=null)
21. {
22. System.out.println(data);
23. }
24. FileWriter fw = new FileWriter("hello.txt");
25. String s = "hello world";
26. fw.write(s,0,s.length());
27. fw.flush();
28. OutputStreamWriter osw = new OutputStreamWriter(new FileOutputStream("hello2.txt"));
29. osw.write(s,0,s.length());
30. osw.flush();
31. PrintWriter pw = new PrintWriter(new OutputStreamWriter(new FileOutputStream("hello3.txt")),true);
32. pw.println(s);
33. fr.close();
34. isr.close();
35. br.close();
36. fw.close();
37. osw.close();
38. pw.close();
39. }
40. }

java中多种方式读文件  
一、多种方式读文件内容。  
1、按字节读取文件内容  
2、按字符读取文件内容  
3、按行读取文件内容  
4、随机读取文件内容

**[java:showcolumns]** [view plaincopy](http://blog.csdn.net/smartcat86/article/details/4085739/)

·········10········20········30········40········50········60········70········80········90········100·······110·······120·······130·······140·······150

1. import java.io.BufferedReader;
2. import java.io.File;
3. import java.io.FileInputStream;
4. import java.io.FileReader;
5. import java.io.IOException;
6. import java.io.InputStream;
7. import java.io.InputStreamReader;
8. import java.io.RandomAccessFile;
9. import java.io.Reader;
10. public class ReadFromFile {
11. /\*\*
12. \* 以字节为单位读取文件，常用于读二进制文件，如图片、声音、影像等文件。
13. \* @param fileName 文件的名
14. \*/
15. public static void readFileByBytes(String fileName){
16. File file = new File(fileName);
17. InputStream in = null;
18. try {
19. System.out.println("以字节为单位读取文件内容，一次读一个字节：");
20. // 一次读一个字节
21. in = new FileInputStream(file);
22. int tempbyte;
23. while((tempbyte=in.read()) != -1){
24. System.out.write(tempbyte);
25. }
26. in.close();
27. } catch (IOException e) {
28. e.printStackTrace();
29. return;
30. }
31. try {
32. System.out.println("以字节为单位读取文件内容，一次读多个字节：");
33. //一次读多个字节
34. byte[] tempbytes = new byte[100];
35. int byteread = 0;
36. in = new FileInputStream(fileName);
37. ReadFromFile.showAvailableBytes(in);
38. //读入多个字节到字节数组中，byteread为一次读入的字节数
39. while ((byteread = in.read(tempbytes)) != -1){
40. System.out.write(tempbytes, 0, byteread);
41. }
42. } catch (Exception e1) {
43. e1.printStackTrace();
44. } finally {
45. if (in != null){
46. try {
47. in.close();
48. } catch (IOException e1) {
49. }
50. }
51. }
52. }
53. /\*\*
54. \* 以字符为单位读取文件，常用于读文本，数字等类型的文件
55. \* @param fileName 文件名
56. \*/
57. public static void readFileByChars(String fileName){
58. File file = new File(fileName);
59. Reader reader = null;
60. try {
61. System.out.println("以字符为单位读取文件内容，一次读一个字节：");
62. // 一次读一个字符
63. reader = new InputStreamReader(new FileInputStream(file));
64. int tempchar;
65. while ((tempchar = reader.read()) != -1){
66. //对于windows下，rn这两个字符在一起时，表示一个换行。
67. //但如果这两个字符分开显示时，会换两次行。
68. //因此，屏蔽掉r，或者屏蔽n。否则，将会多出很多空行。
69. if (((char)tempchar) != 'r'){
70. System.out.print((char)tempchar);
71. }
72. }
73. reader.close();
74. } catch (Exception e) {
75. e.printStackTrace();
76. }
77. try {
78. System.out.println("以字符为单位读取文件内容，一次读多个字节：");
79. //一次读多个字符
80. char[] tempchars = new char[30];
81. int charread = 0;
82. reader = new InputStreamReader(new FileInputStream(fileName));
83. //读入多个字符到字符数组中，charread为一次读取字符数
84. while ((charread = reader.read(tempchars))!=-1){
85. //同样屏蔽掉r不显示
86. if ((charread == tempchars.length)&&(tempchars[tempchars.length-1] != 'r')){
87. System.out.print(tempchars);
88. }else{
89. for (int i=0; i<charread; i++){
90. if(tempchars[i] == 'r'){
91. continue;
92. }else{
93. System.out.print(tempchars[i]);
94. }
95. }
96. }
97. }
98. } catch (Exception e1) {
99. e1.printStackTrace();
100. }finally {
101. if (reader != null){
102. try {
103. reader.close();
104. } catch (IOException e1) {
105. }
106. }
107. }
108. }
109. /\*\*
110. \* 以行为单位读取文件，常用于读面向行的格式化文件
111. \* @param fileName 文件名
112. \*/
113. public static void readFileByLines(String fileName){
114. File file = new File(fileName);
115. BufferedReader reader = null;
116. try {
117. System.out.println("以行为单位读取文件内容，一次读一整行：");
118. reader = new BufferedReader(new FileReader(file));
119. String tempString = null;
120. int line = 1;
121. //一次读入一行，直到读入null为文件结束
122. while ((tempString = reader.readLine()) != null){
123. //显示行号
124. System.out.println("line " + line + ": " + tempString);
125. line++;
126. }
127. reader.close();
128. } catch (IOException e) {
129. e.printStackTrace();
130. } finally {
131. if (reader != null){
132. try {
133. reader.close();
134. } catch (IOException e1) {
135. }
136. }
137. }
138. }
139. /\*\*
140. \* 随机读取文件内容
141. \* @param fileName 文件名
142. \*/
143. public static void readFileByRandomAccess(String fileName){
144. RandomAccessFile randomFile = null;
145. try {
146. System.out.println("随机读取一段文件内容：");
147. // 打开一个随机访问文件流，按只读方式
148. randomFile = new RandomAccessFile(fileName, "r");
149. // 文件长度，字节数
150. long fileLength = randomFile.length();
151. // 读文件的起始位置
152. int beginIndex = (fileLength > 4) ? 4 : 0;
153. //将读文件的开始位置移到beginIndex位置。
154. randomFile.seek(beginIndex);
155. byte[] bytes = new byte[10];
156. int byteread = 0;
157. //一次读10个字节，如果文件内容不足10个字节，则读剩下的字节。
158. //将一次读取的字节数赋给byteread
159. while ((byteread = randomFile.read(bytes)) != -1){
160. System.out.write(bytes, 0, byteread);
161. }
162. } catch (IOException e){
163. e.printStackTrace();
164. } finally {
165. if (randomFile != null){
166. try {
167. randomFile.close();
168. } catch (IOException e1) {
169. }
170. }
171. }
172. }
173. /\*\*
174. \* 显示输入流中还剩的字节数
175. \* @param in
176. \*/
177. private static void showAvailableBytes(InputStream in){
178. try {
179. System.out.println("当前字节输入流中的字节数为:" + in.available());
180. } catch (IOException e) {
181. e.printStackTrace();
182. }
183. }
184. public static void main(String[] args) {
185. String fileName = "C:/temp/newTemp.txt";
186. ReadFromFile.readFileByBytes(fileName);
187. ReadFromFile.readFileByChars(fileName);
188. ReadFromFile.readFileByLines(fileName);
189. ReadFromFile.readFileByRandomAccess(fileName);
190. }
191. }
192. 二、将内容追加到文件尾部
193. import java.io.FileWriter;
194. import java.io.IOException;
195. import java.io.RandomAccessFile;
196. /\*\*
197. \* 将内容追加到文件尾部
198. \*/
199. public class AppendToFile {
200. /\*\*
201. \* A方法追加文件：使用RandomAccessFile
202. \* @param fileName 文件名
203. \* @param content 追加的内容
204. \*/
205. public static void appendMethodA(String fileName,
207. String content){
208. try {
209. // 打开一个随机访问文件流，按读写方式
210. RandomAccessFile randomFile = new RandomAccessFile(fileName, "rw");
211. // 文件长度，字节数
212. long fileLength = randomFile.length();
213. //将写文件指针移到文件尾。
214. randomFile.seek(fileLength);
215. randomFile.writeBytes(content);
216. randomFile.close();
217. } catch (IOException e){
218. e.printStackTrace();
219. }
220. }
221. /\*\*
222. \* B方法追加文件：使用FileWriter
223. \* @param fileName
224. \* @param content
225. \*/
226. public static void appendMethodB(String fileName, String content){
227. try {
228. //打开一个写文件器，构造函数中的第二个参数true表示以追加形式写文件
229. FileWriter writer = new FileWriter(fileName, true);
230. writer.write(content);
231. writer.close();
232. } catch (IOException e) {
233. e.printStackTrace();
234. }
235. }
236. public static void main(String[] args) {
237. String fileName = "C:/temp/newTemp.txt";
238. String content = "new append!";
239. //按方法A追加文件
240. AppendToFile.appendMethodA(fileName, content);
241. AppendToFile.appendMethodA(fileName, "append end. n");
242. //显示文件内容
243. ReadFromFile.readFileByLines(fileName);
244. //按方法B追加文件
245. AppendToFile.appendMethodB(fileName, content);
246. AppendToFile.appendMethodB(fileName, "append end. n");
247. //显示文件内容
248. ReadFromFile.readFileByLines(fileName);
249. }
250. }

### 26.broadcast（！）

1.在应用程序的代码中进行注册

RegisterReceiver(receiver,filter)

unRegisterReceiver()

2.androidManifext.xml注册

应用程序关闭也能receive消息

### 27.window 目录浏览

1.直接进入d:

2.列出所有文件：dir

### 28.Android AVD安装软件

1、运行模拟器并成功开机进入待机界面（不知道如何开机?[请点此](http://www.dboy365.com/archives/110)）

2、点击PC开始菜单里面的“运行”输入cmd，如果是win7则直接在开始菜单里的搜索框中输入cmd并回车，开启cmd命令窗口

3、将您下载好的APK包复制到您模拟器的安装地址中的Tools目录下，例如我的安装地址是：“D:\android\tools”

如果您还没有APK文件，想先学习一下安装过程，那就去新浪微博下载一个做测试吧，可[点此直接下载](http://3g.sina.com.cn/tv/soft/weibo/weibo_10105010.apk)

下载后改个短一些的名字，然后复制到上面说的目录下，这里就用ceshi.apk来介绍吧

4、点击模拟器的安装地址中的Tools目录下的emulator.exe,注意，点击后只会一闪而过，但一定要记得运行

5、在cmd命令窗口中输入cd d: 回车    注：d:是您的模拟器安装所在目录，自己更换一下

6、再输入d:回车

7、继续输入cd D:\android\tools回车   注：D:\android\tools是您模拟器的安装目录及tools子目录，自己更换一下

8、最后输入adb install ceshi.apk回车    注：ceshi.apk是您下载并放在tools目录下的文件名

C:\Users\D-BOY>cd d:

D:\

C:\Users\D-BOY>d:

D:\>cd D:\android\tools

D:\android\tools>adb install ceshi.apk

至此，安装步骤就结束了，您进入刚才开机的手机模拟，从目录中找找，就能找到该软件了；

最后，还要特别向大家介绍“eoemarket”,有了它，安装软件、游戏就方便多了，直接从这个软件里安装就可以，嘿嘿，[这里提供直接下载](http://cid-493b21cad5775b5d.office.live.com/self.aspx/.Public/eoemarket.apk)，大家用上面的方法安装上，一劳永逸的操作；

### 28.Android broadcast

Broadcastreceiver 如接收来电，拍照完成等消息。

### 29.package is not signed.

Change the gpgcheck=0

### 30.adapter30.adapter设计模式

ContentHandler有很方法要实现，不是直接实现的，所以我们会去继承DefaultHandler.

### 31.Rails installation

Rails installation path cannot containing space in windows.

# 第二章 错误解决

### 1. Please ensure that adb is correctly located at 'D:\android-sdk-windows\platform-tools\adb.exe' and can be executed. "

今天遇见了这个问题 因为我更新了最新的adt 出现ADB server didn't ACK, failed to start daemon 然后你想启动一个程序 这个时候又会出现 Please ensure that adb is correctly located at 'D:\android-sdk-windows\platform-tools\adb.exe' and can be executed. "

首先呢 查看是否查看任务管理器，是否有关闭所有adb.exe 有的关掉

将将android-sdk-windows\platform-tools和android-sdk-windows\tools都加到环境变量中去

测试一下能启动不能 如果不能

运行一下命令行

adb kill-server   
adb start-server

这个时候可能出现

D:\android-sdk-windows\platform-tools>adb kill-server

D:\android-sdk-windows\platform-tools>adb start-server  
\* daemon not running. starting it now on port 5037 \*  
ADB server didn't ACK  
\* failed to start daemon \*

 如果还是有毛病 这个时候唯一的选择，我这里呢 没有听从网上的 说啥重启机器 重启eclipse，我的办法是

在eclipse中的android管理器中也就是 avd Manager中  选择 第一个选项 虚拟设备virtural device从里面找到一个模拟机启动 当启动起来之后   关掉你的eclipse 在重新启动 就好了

这里说命的一点是 一定要注意 虚拟设备一定要先开 然后在重启eclipse 至少我的方法是这样子的

注意一个顺序问题

### 2. [java.net.SocketException: Permission denied](http://yyyyy5101.iteye.com/blog/691707)

在AndroidMainfest.xml中加入   
<uses-permission android:name="android.permission.INTERNET" />

# 第三章 Linux Part

### 1.Linux startup script

### 2. vim

ggyG Copy all

edit in columns

* Ctrl + V to go into column mode
* Select the columns and rows where you want to enter your text
* Shift + i to go into insert mode in column mode
* Type in the text you want to enter. Dont be discouraged by the fact that only the first row is changed.
* Esc to apply your change (or alternately Ctrl+c)

### 3.vi Find And Replace Text Command

by [nixCraft](http://www.cyberciti.biz/tips/about-us) on November 13, 2010 · [6 comments](http://www.cyberciti.biz/faq/linux-unix-vim-find-replace-text-string-words/#comments)· last updated at November 17, 2010

How do I find and replace (substitute) test using vi or vim text editor under UNIX / Linux / BSD or Apple OS X operating systems?  
  
Both vi and vim text editor comes with substitute command for finding and replacing text.

### 4. rm mytest >> appendfirst.txt 2>> append\_second.txt

Add the first cmd error to the append\_second.txt

### 5.create yum repos.

[user@machine] mkdir -p MyRepo/i386 MyRepo/noarch MyRepo/i686 MyRepo/SRPMS

[user@machine] mv myrpm.noarch.rpm MyRepo/noarch

[user@machine] mv myrpm.src.rpm MyRepo/SRPMS

[user@machine] mv mybinrpm.i386.rpm MyRepo/i386

[user@machine] createrepo MyRepo

/etc/yum.repos.d

Now, if we want to install this group on a machine, we create a .repo file in /etc/yum.repos.d like this:

[MyRepo]

name=My Repository

baseurl= file:///path/to/MyRepo

gpgcheck=1

gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-MyRepo

If you didn't sign your rpms and do not wish to verify signatures, set gpgcheck=0 in your .repo file. If you did sign your rpms, then you'll need to export your gpg key into a file so yum can import the file into the rpm database.

## Syntax

The syntax is as follows:  
:%s/WORD-To-Find-HERE/Replace-Word-Here/g  
OR  
**:%s/FindMe/ReplaceME/g**

## Examples

The substitute command can be used as per your requirements.

### Task: VI / Vim Basic Find and Replace

To find each occurrence of 'UNIX', and replace it with 'Linux', enter (press ESC, type : and following command):  
:%s/UNIX/Linux/g

### Task: Find and Replace with Confirmation

Find a word called 'UNIX' and replace with 'Linux', but ask for confirmation first, enter:  
:%s/UNIX/Linux/gc

### Task: Find and Replace Whole Word Only

Find whole words exactly matching 'UNIX' to 'Linux'; and ask for confirmation too:  
:%s/\<UNIX\>/Linux/gc

### Task: Case Insensitive Find and Replace

Find 'UNIX' (match UNIX, unix, UnIx, Unix and so on) and replace with 'Linux':  
:%s/unix/Linux/gi  
Same command with confirmation:  
:%s/unix/Linux/gic

### Task: Case sensitive Find and Replace

Find each 'UNIX' and replace with 'bar':  
:%s/UNIX/bar/gI  
Same command with confirmation:  
:%s/UNIX/bar/gIc

## How Do I Replace In the Current Line Only?

Find 'UNIX' and replace with 'Linux' in the current line only (note % is removed from substitute command):  
:s/UNIX/Linux/g  
NOTE: You need to prefix % the substitute command to make changes on all lines:  
:%s/UNIX/Linux/g

## How Do I Replace All Lines Between line 100 and line 250?

:{START-n},{END-n}s/word1/word2/g  
Find 'UNIX' and replace with 'Linux' all lines between line 100 and line 250, enter:  
:100,200s/UNIX/Linux/g  
OR  
:100,200s/UNIX/Linux/gc

### 3.samba服务配置

#### 3.1 start smb:

1./etc/init.d/smb

2.smbpasswd –a username

#### 3.2 selinux part:

# If you want to use the useradd/groupadd family of binaries please run:

# setsebool -P samba\_domain\_controller on

#

# If you want to share home directories via samba please run:

# setsebool -P samba\_enable\_home\_dirs on

#

# If you create a new directory you want to share you should mark it as

# "samba\_share\_t" so that selinux will let you write into it.

# Make sure not to do that on system directories as they may already have

# been marked with othe SELinux labels.

#

# Use ls -ldZ /path to see which context a directory has

#

# Set labels only on directories you created!

# To set a label use the following: chcon -t samba\_share\_t /path

#

# If you need to share a system created directory you can use one of the

# following (read-only/read-write):

# setsebool -P samba\_export\_all\_ro on

# or

# setsebool -P samba\_export\_all\_rw on

开启Samba用户HOME目录权限  
/usr/sbin/setsebool -P samba\_enable\_home\_dirs=1  
开启单个共享目录（path）权限  
chcon -t samba\_share\_t path  
开启所有共享目录  
只读权限  
/usr/sbin/setsebool -P samba\_export\_all\_ro on  
读写权限  
/usr/sbin/setsebool -P samba\_export\_all\_rw on  
查看某个目录（path）是否开启了权限  
ls -ldZ path

#### 3.dom and sax’s difference

DOM为 XML 文档的已解析版本定义了一组接口。解析器读入整个文档，然后构建一个驻留内存的树结构，然后您的代码就可以使用 DOM 接口来操作这个树结构。

SAX API是一个基于事件的 API，适用于处理数据流，即随着数据的流动而依次处理数据。  
SAX API 在其解析您的文档时发生一定事件的时候会通知您。在您对其响应时，您不作保存的数据将会被抛弃。

### 4.shell scripts

#### 1.cmdline variable:

<http://www.freeos.com/guides/lsst/index.html>

#### 2. omit path from PATH

path\_append () { path\_remove $1; export PATH="$PATH:$1"; }

path\_prepend () { path\_remove $1; export PATH="$1:$PAT H"; }

path\_remove () { export PATH=`echo -n $PATH | awk -v RS=: -v ORS=: '$0 != "'$1'"' | sed 's/:$//'`; }

# 第四章 Ruby

## 1.rails

##### 1.rake routes

Resource pointed out in previous answers is awesome and that is where I got started. I still refer that in case I am stuck somewhere. One thing I find missing in the recourse is that it doesn't include the explanation of reading the routes table i.e. output of command rake routes and it takes time to fit the pieces together. Although if you read through the whole guide patiently, you can fit the pieces together.

On my system it gives following output(excerpt relevant to resources :messages)

messages GET /messages(.:format) {:action=>"index", :controller=>"messages"}

POST /messages(.:format) {:action=>"create", :controller=>"messages"}

new\_message GET /messages/new(.:format) {:action=>"new", :controller=>"messages"}

edit\_message GET /messages/:id/edit(.:format) {:action=>"edit", :controller=>"messages"}

message GET /messages/:id(.:format) {:action=>"show", :controller=>"messages"}

PUT /messages/:id(.:format) {:action=>"update", :controller=>"messages"}

DELETE /messages/:id(.:format) {:action=>"destroy", :controller=>"messages"}

All the columns in this table give very important information:

Route Name(1st Column): This gives the name of the route, to which you can append "\_url" or "\_path" to derive the helper name for the route. For example, first one is the "messages", so you can use meesages\_path and messages\_url in your views and controllers as a helper method. Looking at the table you can tell messages\_path will generate a path of form "/maessages(.:format)". Similarly, other route names generated are "new\_message", "edit\_message" and "message". You can also control the naming of routes.

HTTP Verb(2nd Column): This gives the information about the http verb which this route will respond to. If it is not present, then it means this route will respond to all http verbs. Generally browsers only support, "GET" and "POST" verbs. Rails simulate "PUT" and "DELETE" by passing a parameter "\_method" with verb name as value to simulate "PUT" and "DELETE". Links by default result in a "GET" verb and form submissions in "POST". In conjunction with the first column, if you use messages\_path with http "GET" it would match first route and if you use it with "POST" it will match second route. This is very important to note, same url with different http verbs can map to different routes.

URL Pattern(3rd Column): Its like a limited featured regular expression with syntax of its own. ":id" behaves like (.+) and captures the match in parameter "id", so that you can do something like params[:id] and get the captured string. Braces () represent that this parameter is optional. You can also pass these parameters in helpers to generate the corresponding route. For example if you used message\_path(:id => 123) is will generate the output "/messages/123".

Where this routes(4th Column): This column generally tells the controller and the corresponding action which will handle requests matching this route. There can be additional information here like constraints if you defined any.

So if "localhost:3000/magazines" is the page you want, you should check the routes table with url pattern as "/magazines(.:format)" and disect it yourself to find out what you need. I would recommend you read the whole guide from top to bottom if you are just starting rails.

(This might be just an overkill to write all this here, but I faced many problems due to this info not being available in a consolidated manner. Always wanted to write it out and finally did. I wish it was available on http://edgeguides.rubyonrails.org/routing.html in a separate section.)

##### 2.execute ruby unit test case

Ruby test\_case.rb

ruby home\_controller\_test.rb -n test\_should\_test(test specific test case)

test "should test" do

v = false

assert !v

end

##### 3. rake task:

You can see all these rake tasks and their descriptions by running rake --tasks –describe

##### 4.Ajax push(comet),

**Juggernaut**

**Ajax push技术核心揭秘**

**引言**

在web应用普及的今天，用户开始将更多的关键应用向Web迁移, 广大Web应用开发者与推广者在享受到了成功地喜悦。与同时很多用户已经开始抱怨我们的Web应用总是那么被动与迟钝，何时才能让他们的应用更加主动实时的,让发生在服务端的事件第一时间内通知给他们。然而开发人员也不得不面对这样的事实，在Web天生的无状态与非连接制约下无法他们无法对应用的实时性更进一步的提升。

在近几年的发展特别是Aajx的出现让我们的web应用找到了新的兴奋点，然而这仍然没有解决前面提到的问题，难道我们真的无路可走了？

经过笔者的探索发现我们是可以实现基于web的实时主动通知的， 即采用Ajax push技术,她可以让我们web应用拥有前所未有激动人心的功能和使用体验。笔者将通过本文带领逐步大家去实现这个激动人心的体验。笔者目的也非常简单,目的在于为大家提供更多的参考资料 ,无论对错希望本文能起到抛砖引玉的作用.如果能借此引发大家对Web实时推式通知技术的更为广泛的讨论我将感到万分荣幸!

**Ajax push的广阔前景**

通知技术笔者把它们从概念上分为两种：**被动的拉式通知技术**和**主动的推式通知技术**。这两种说法目前网上已经有大量的文章,我在这里只做简单的介绍:

1、 **被动的拉式通知技术**又称Pull方式如下图

|  |
| --- |
| 浏览器 |

|  |
| --- |
| 服务器 |

|  |
| --- |
| internet |

|  |
| --- |
| 定时请求 |

|  |
| --- |
| 根据请求响应 |

图：pull方式

拉方式需要客户机不定时的检查服务器已获知是否发生新的事件或数据是否有变化，这种方式并不实时，但在web上比较容易实现。

2、 **主动的推式通知技术**又称push方式如下图

|  |
| --- |
| 浏览器 |

|  |
| --- |
| 服务器 |

|  |
| --- |
| internet |

|  |
| --- |
| 发生事件主动发送 |

|  |
| --- |
| 首次建立通信连接 |

|  |
| --- |
| 事件 |

图：push方式

推式客户机与服务只需建立好连接之后，每当服务器有特殊事件发生时才通知客户机，该方式实时性非常强，但目前在Web上实现较为复杂

这两种方式后者有非常显著的优点，而**Ajax push**就是需要在web上实现的后者的通信技术，如果**Ajax push**能被的完美实现，那么基于web的IM软件、基于Web的关键业务报警系统、基于web的实时监控系统、更智能人性的Web信息系统、甚至是基于web的远程控制系统、等等都将可以实现，同时彻底摆脱客户端部署与安装，避免服务端的高负载。而webMsn、GMAIL这些系统中的很多被我们认为是不可思议的特性也能被任何一个web程序员轻松的开发出来。这是多么美好的时刻，更加值得我们去期待。

**突破观念的束缚**

Web应用的优点在于易于部署，但web是无状态非连接的，从这个角度来看服务器无法对客户端进行实时的推式通知，可能会有人对我说的不屑于顾，不是已经有很多系统都实现了web上的通知吗？其实不然，目前实现web上动态通知的技术概括下来基本上有以下两种方法：

1、 **定期刷新法**。定期刷新法又可以分为整体页面间隔刷新和异步间隔刷新两种方法。

a) **整体页面间隔刷新法，**该方法在早期聊天室中使用，该方法实现非常简单，只需要在Html头中加入如下代码：

<META HTTP-EQUIV="refresh" CONTENT="10" URL="你的页面">

该方法目前已经很少使用，主要因为如果现在网页过于复杂加载时间长，如果频繁刷新会对用户造成非常差的体验，同时会传输大量重复的网络数据无疑加大了服务器及网络的负担。

b) **异步间隔刷新法，**由于采用异步刷新方法，即使刷新平凡都不会造成页面闪烁，同时减少了不必要的展现数据，是目前采用的较多的方法，实现上又分为Ajax刷新即XMLHttp刷新和隐藏贞刷新。我给出简单现示例如下:

**Ajax刷新**

var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP.5.0");

function refreshUi()

{

xmlhttp.open("POST","你接收请求的页面",true);

xmlhttp.setRequestHeader("Content-Type","application/x-www-form-urlencoded;charset=utf-8");

xmlhttp.send(你好提交的数据);

var s=xmlhttp.responseText;//获取服务器收到的数据

UpdateUi(s);//更新你的界面上的数据

}

setInterval(refreshUi,20);

服务段可以是一个可以是jsp,asp,serverlet,aspx,等服务端处理程序，然后通过Response对象向客户端输出需要的数据即可

**隐藏贞刷新**

即在页面上放入隐藏的FrameSet或则IFrame，然后通过对该贞的提交操作获取数据然后刷新页面

function refreshUi()

{

var hiddenForm= document.frames[1].document.forms[0];//获取隐藏贞中的表单信息

hiddenForm.submit();//对表单进行提交

updateUi(hiddenForm);//对界面进行更新

}

setInterval(refreshUi,20);

但采用定期刷新法后不管是Ajax提交还是隐藏贞提交都有共同的缺点是跟新不实时，刷新速度不能过快否则将严重影响客户端或服务器的性能，同时很多客户的数据在相当长的时间内一般不会更新，这样的会造成无大量无意义的刷新，同时增大服务器的负载。但由于该方法实现简单，所以大量的web应用程序采用了该方法实现通知技术，属于非实时拉动（被动）式通知技术。

2、 **客户端插件法**，通过为客户段编写第三方插件的形式嵌入到客户端网页中，然后同服务器建立通信连接实现即时通知技术，该技术有点可以实现服务端到客户端的主动推式通知技术，目前主要是以ActiveX或是java Applet的方式提供插件。缺点是实现技术复杂，需要让客户安装插件，容易出现不兼容的情况，同时在目前网络病毒泛滥的情况下，很多客户端浏览器上的安全策略是禁止安装第三方的插件，这对部署和维护都带来了相当大的难度。

从上面的技术实现上看目前这两种方法目前缺点都非常明显要么非实时性，同时服务器负担巨大；要么安装插件，没有发挥web的优点。所以我们必须跳出常规方法寻找另外的方法，幸运的是经过笔者研究和尝试基本上找到实现服务器实时主动通知客户端的实现思路。

**揭开神秘的面纱**

事先说明，笔者在当前文章中只会提供实现的原理和参考代码段，并不提供把该功能设计成组件的相关体系结构和类设计。而关于**Ajax push**组件的设计笔者正在编写另一篇文章。

回到正题，我们先从客户端谈起，由于Web无状态非连接的特性，如果我们要在web上实现push方式，那么我们首要的是必须与服务器建立通信连接。然而我们要求是不安装任何插件，那我们首先应当想到XmlHttpRequest对象，这个非常对，那么我们如何它与服务器建立连接呢？很简单，由于该通信连接不能阻塞浏览器的主体运行所以我们不能采用同步请求方式，所以我们必须采用异步通信方式，而恰好得的XmlHttpRequest是提供的异步请求方式对于请求应答时常没有什么特别的限制，这正好符合我们与服务器建立长期连接的需求，也许微软也打算在将来的某个时间提供对浏览器push模式的支持。以下是建立长期通信连接的实例片断：

|  |
| --- |
| var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP.5.0");  function ConnectServer()  {  xmlhttp.open("POST","http://127.0.0.1:5565",true);//建立异步通信 xmlhttp.setRequestHeader("Content-Type","application/x-www-form-urlencoded;charset=utf-8");  var m=xmlhttp.send("当前浏览起的标识");  }  xmlhttp.onreadystatechange = function()  {  if(xmlhttp.readyState == 4)  {  //判断返回值是否正常  if(xmlhttp.status == 200)  {  //执行你的方法  var s=xmlhttp.responseText;//获取服务器发过来的信息  //执行你处理该事件的相关代码  }  }  setTimeout(ConnectServer(),100);//重新建立下一次连接  }  ConnectServer(); |

注意，127.0.0.1是实例的服务器，在书写程序时可以更换该地址，每次onreadystatechange被回调后应该建立另一次连接，应为每次服务器应答客户机后客户机就会断开连接，所以每次服务器应答之后客户机就应该建立新的连接以等待服务器的下一次通知。

现在客户机已经能够和服务器建立长久的连接，那么服务器该如何通知客户机呢？好的，现在我们就来解决该问题，当客户机一个xmlhttp请求发送到服务器后，服务器接收到该请求不立即应答，而将该请求挂起，直到服务器产生需要通知客户机的事件时才应答客户机。而如何挂起该请求了？有两种方案：

1、 是在web服务器上实现一个Serverlet（java）或httpHandle(.net)来接收该请求，当请求的调用到来时我们可以阻塞该线程上的调用，直到服务器产生通知客户机的事件。

2、 自行实现一个简单的httpserver来接收应答并处理。即用一个线程来监听指定端口，当请求到达时将用于应答的套接字（socket）存储在内存中的列表中，当服务器有事件通知时从列表中检索出对应的套接字并应答。

由于第一种方案会阻塞线程，由于web服务器应答各种请求的线程数是有限的所以第一种方案会带来性能损失和不稳定因素。所以第二种方案才是可行的。基于java的参考代码如下：

接收xmlhttp请求的代码片断

|  |
| --- |
| Public static HashMap<String, Socket> clientSockets=new HashMap<String, Socket>();  private boolean serverStarted=true;  .....  .....  .....  java.net.ServerSocket ss;  ss = new java.net.ServerSocket(5565);//设置监听端口  while(serverStarted) //循环应答  {  try {  java.lang.Thread.sleep(100);  } catch (InterruptedException e) {  break;  }  java.net.Socket s =ss.accept();  byte[] b =new byte[s.getInputStream().available()] ;  s.getInputStream().read(b);  String requestStr=new String(b,"utf-8");  //从请求字符串中分析出客户端发来的标示符  String clientId=parseRequestStr(requestStr);  clientSockets.put(clientId,s);  }  …….. |

当服务器发生事件需要通知客户段时的代码片断

|  |
| --- |
| java.net.Socket s=clientSockets.get(clientId);//从列表中检索出需要的客户段套接字  if(s==null)  return;  java.io.PrintWriter p=new java.io.PrintWriter(s.getOutputStream());  p.println("HTTP/1.1 200OK");  p.println("Content-Type:text/html; charset:utf-8");  p.println("Content-Length:"+msg.length());//msg为服务器要发到客户端的信息  p.println();  p.println(msg);  p.flush();  s.getOutputStream().flush();  s.getOutputStream().close();  s.close();  clientSockets.remove(clientId);//移出发送的Socket |

好了倒此为止服务器主动通知客户机**Ajax push**技术的基本实现原理和参考代码段已经给出，相信大家根据笔者所给出的技术要点举一反三实现出更好的推式技术。

**结束语**

最后我还是要补充的说几句，笔者在文章中只是给出的关键实现，但在实际中应用该技术还需要考虑很多，我在这里举几个要考虑的比较重要点：

1、 关于客户段异常，如果连接失败如何最省资源的自动再次连接，如何检测连接已经断开

2、 服务器需要对列表中的套接字进行定期验证以保证列表中的连接可用

3、 很多消息要连续通知客户机是对消息考虑队列的处理

好了我就说这里，如果大家有更多的问题和建议请E-mail联系我,希望大家看过本文后也能向别人问起“你的应用今天**Ajax push**了吗？”

##### 5. Could not determine content-length of response body. Set content-length of the response or set Response#chunked = true

gem 'thin'

rails server thin

##### 6.WEB SERVER

Path translation

Web servers are able to map the path component of a Uniform Resource Locator (URL) into:

A local file system resource (for static requests)

An internal or external program name (for dynamic requests)

For a static request the URL path specified by the client is relative to the web server's root directory.

Consider the following URL as it would be requested by a client:

http://www.example.com/path/file.html

The client's user agent will translate it into a connection to www.example.com with the following HTTP 1.1 request:

GET /path/file.html HTTP/1.1

Host: www.example.com

The web server on www.example.com will append the given path to the path of its root directory. On an Apache server, this is commonly /home/www (On Unix machines, usually /var/www). The result is the local file system resource:

/home/www/path/file.html

# 第六章 Linux C++

### 1.codeblock installation:

*Code::Blocks ->* ***xwWidget (> libwxGTK2.8)*** *->* ***GTK+ (> libgtk-x11-2.0)*** *-> X*

#### 1.1 install libgtk2.0-dev

*sudo yum install libgtk2.0-dev*

#### 1.2 install xwWidget

#### 1.3 install CB

./bootstrap

./configure

make

make install

###### error1

...(configure aborts with some unspecific error messagelike that), then you might consider also running dos2unix bootstrap acinclude.m4 codeblocks.pc.in configure.in Makefile.am before running bootstrap. Once you've run the bootstrap script, jump to next step.

aclocal:configure.in:61: warning: macro `AM\_OPTIONS\_WXCONFIG' not found in library

...then aclocal is having trouble finding the wxWidgets .m4 files. You can do one of two things: To just get bootstrap to find the path this time do:

export ACLOCAL\_FLAGS="-I `wx-config --prefix`/share/aclocal"

###### error2:/bin/sh: zip: command not found

sudo yum install zip

###### error3:./codeblocks: error while loading shared libraries: libwx\_gtk2u\_aui-2.9.so.4: cannot open shared object file: No such file or director

**Create a file called /etc/ld.so.conf.d/myapp.conf:**  
# vi /etc/ld.so.conf.d/myapp.conf

**Activate Your Library Path**

You must run the following command to activate path:  
# ldconfig

# ldconfig -v | less

# 第六章 git

## 1.ignore files

Add .gitignore to ignore files.

git ls-files --others -i --exclude-from=.git/info/exclude

git ls-files --others -i --exclude-standard

git clean -ndX

Explanation:

git help clean

git config --global alias.abc "commit -am"