J2ME 开发环境达建

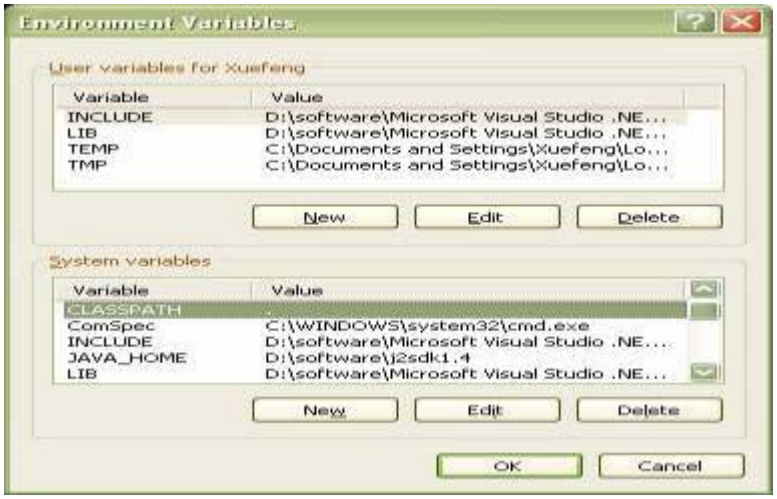
Eclipse 快速上手指南

作者：asklxf

Eclipse 是一款非常优秀的开源 IDE，非常适合 Java 开发，由于支持插件技术，受到了越来越多的开发者的欢迎。最新的 Eclipse 3.0 不但界面作了很大的增强，而且增加了代码折叠等众多优秀功能，速度也有明显的提升。配合众多令人眼花缭乱的插件，完全可以满足从企业级 Java 应用到手机终端 Java 游戏的开发。本文将带您手把手步入 Eclipse 的广阔天地，详细介绍在 Eclipse 下如何开发普通 Java 程序，Web 应用，J2EE 应用，手机 Java 程序，以及如何进行单元测试，重构，配置 CVS 等详细内容。我的开发环境是 JDK1.4.2+Eclipse3.0+Windows XP SP2，如果你在其他平台上遇到任何问题，欢迎来信交流。

1. 安装 JDK1.4

Eclipse 是一个基于 Java 平台的开发环境,它本身也要运行在 Java 虚拟机上,还要使用 JDK 的编译器，因此我们必须首先安装 JDK。JDK1.4 是目前最稳定的版本，同时也是 Eclipse 运行的必须条件。先从 SUN 的官方站点 http://java.sun.com 下载 JDK1.4 Windows 版，目前最新的是 1.4.2_06，然后运行 j2sdk-1_4_2_06-windows-i586-p.exe 安装，你可以自行设定安装目录，我把它安装到 D:\software\j2sdk1.4 目录下。接下来要配置环境变量，以便 Java 程序能找到已安装的 JDK 和其他配置信息。右键点击“ 我的电脑 ”，选择“ 属性 ”，在弹出的对话框中选择“ 高级 ”，“ 环境变量 ”，就可以看到环境变量对话框：



上面是用户变量，只对当前用户有效，下面是系统变量，对所有用户都有效。如果你希望所有用户都能使用，就在系统变量下点击“ 新建 ”，填入：

JAVA_HOME 是 JDK 的安装目录，许多依赖 JDK 的开发环境都靠它来定位 JDK，所以必须保证正确无误。

下一步，找到系统变量 Path，点击“ 编辑 ”，在最后添上 JDK 的可执行文件的所在目录，即%JAVA_HOME%\bin，我的对应目录便是 D:\software\j2sdk1.4\bin，附加到 Path 中即可，注意要以分号“ ; ”隔开：

注意：如果系统安装了多个 Java 虚拟机（比如安装了 Oracle 9i 就有自带的 JDK1.3），必须把 JDK1.4 的路径放在其他 JVM 的前面，否则 Eclipse 启动将报错。

最后一个系统变量是 CLASSPATH，Java 虚拟机会根据 CLASSPATH 的设定来搜索 class 文件所在目录，但这不是必需的，可以在运行 Java 程序时指定 CLASSPATH，比如在 Eclipse 中运行写好的 Java 程序时，它会自动设定 CLASSPATH，但是为了在控制台能方便地运行 Java 程序，我建议最好还是设置一个 CLASSPATH，把它的值设为“ . ”，注意是一个点“ . ”代表当前目录。用惯了 Windows 的用户可能会以为 Java 虚拟机在搜索时会搜索当前目录，其实不会，这是 UNIX 中的习惯，出于安全考虑。许多初学 Java 的朋友兴匆匆地照着书上写好了 Hello，world 程序，一运行却弹出 java.lang.NoClassDefFoundError，其实就是没有设置好 CLASSPATH，只要添加一个当前目录“ . ”就可以了。

2. 安装 Eclipse 3.0

配置好 JDK 后，下一步便是安装 Eclipse 3.0，可以从 Eclipse 的官方站点 http://www.eclipse.org 上下载，你会看到如下版本：

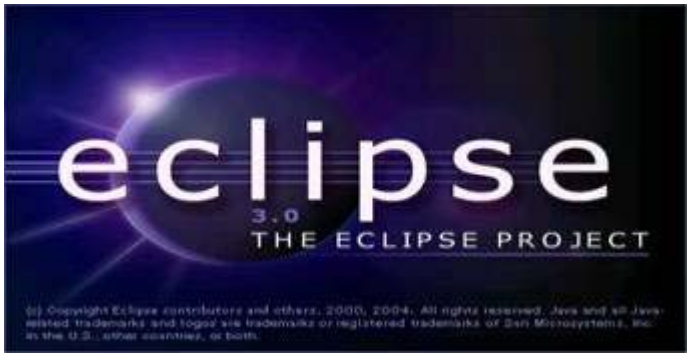
- Eclipse SDK
- RCP Runtime Binary
- RCP SDK

- Platform Runtime Binary
- Platform SDK
- JDT Runtime Binary

Eclipse SDK 包括了 Eclipse 开发环境，Java 开发环境，Plug-in 开发环境，所有源代码和文档，如果你需要所有的功能，可以下载这个版本。

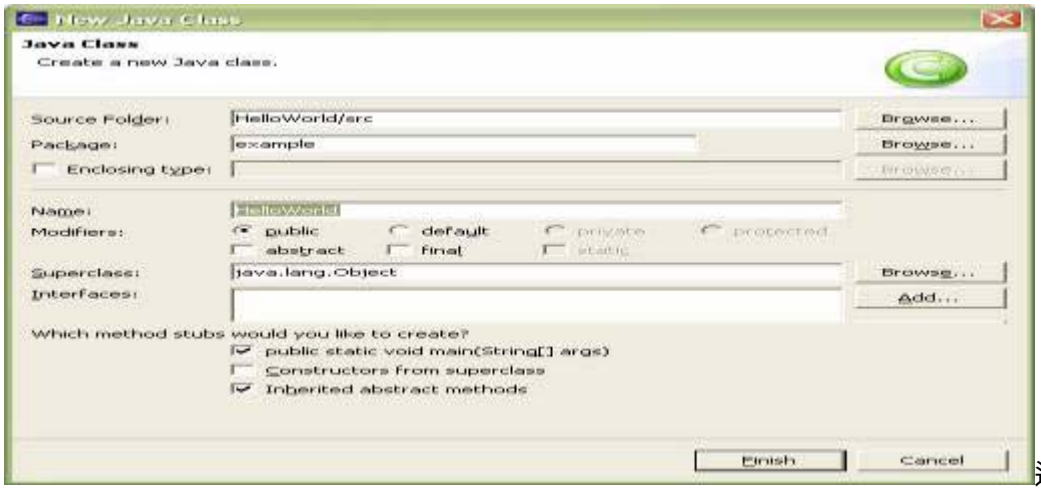
如果你和我一样,只是用Eclipse开发Java应用,而不是开发Eclipse插件或者研究Eclipse代码，那么下载一个 Platform Runtime Binary 再加上 JDT Runtime Binary 是最好的选择。

下载 eclipse-platform-3.0-win32.zip 和 eclipse-JDT-3.0.zip 后，将它们解压到同一个目录，勿需安装，直接找到目录下的 eclipse.exe 运行，出现启动画面：



稍等片刻，Eclipse 界面就出来了。如果遇到错误，启动失败，可以检查 Eclipse 目录下的 log 文件，我曾经遇到过 XmlParser 异常，仔细检查发现原来 Path 中还有一个 Oracle 的 Java1.3 版本的虚拟机，将它从 Path 中去掉后 Eclipse 启动正常。

3．第一个 Java 程序



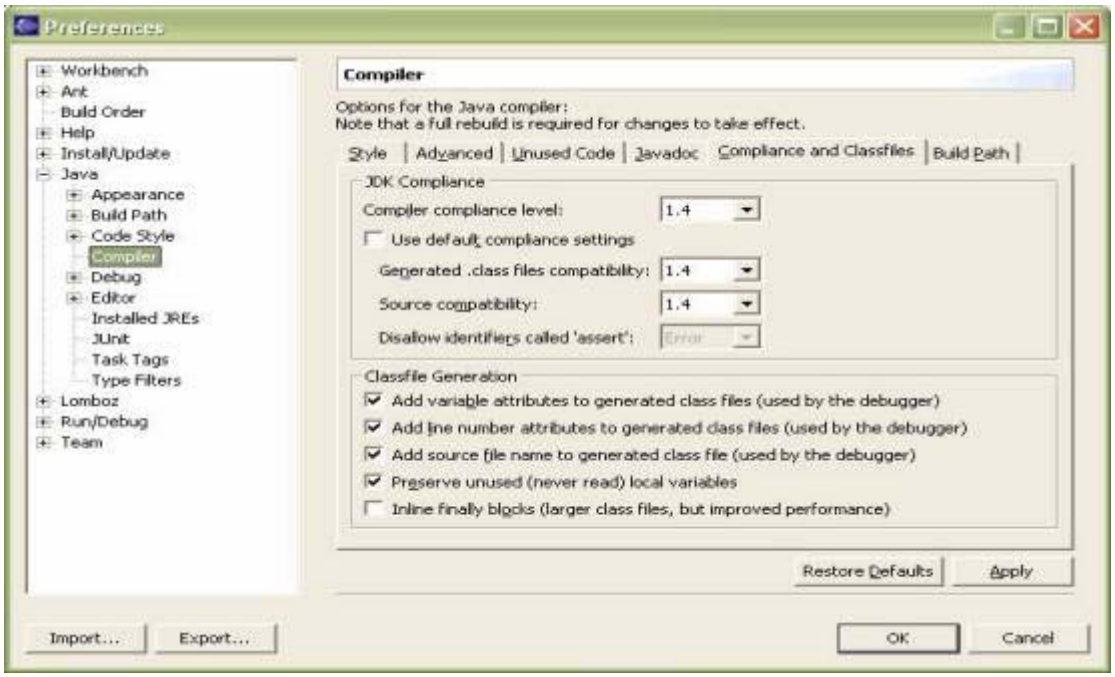
运行 Eclipse，选择菜单“File”，“New”，“Project”，新建一个 Java Project，我把它命名为 HelloWorld，然后新建一个 Java Class：

我把它命名为 HelloWorld，并且填上 Package 为 example，钩上“ public static void main(String[] args) ”，点击“ Finish ”，Eclipse 自动生成了代码框架，我们只需在 main 方法中填入：

默认设置下，Eclipse 会自动在后台编译，我们只需保存，然后选择“ Run ”，“ Run As ”，“ Java Application ”，即可在 Eclipse 的控制台看到输出。

要调试 Java 程序也非常简单，Run 菜单里包含了标准的调试命令，可以非常方便地在 IDE 环境下调试应用程序。

1.4 版本支持：



选择菜单“ Window ”，“ Preferences ”，在对话框中找到“ Java ”，“ Compiler ”，

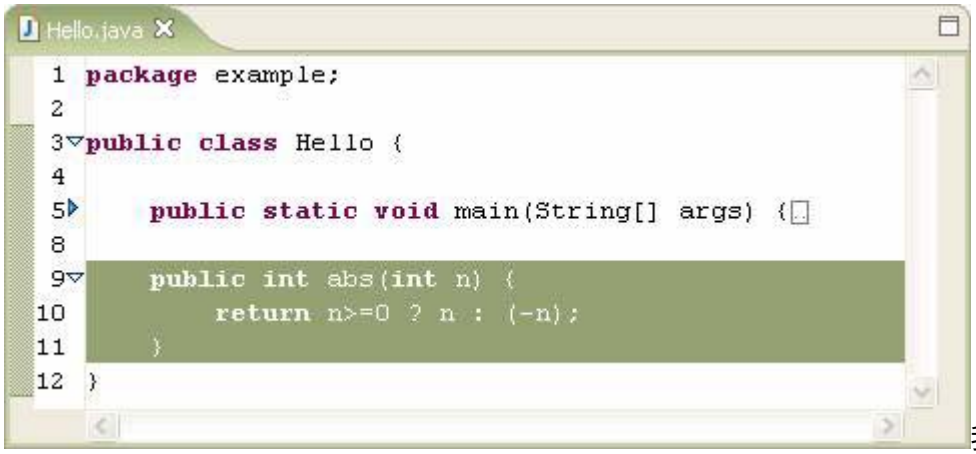
“Compliance and Classfiles”，将编译选项改成 1.4，就可以使用 JDK1.4 版的 assert（断言）语法，使得测试更加方便：

Eclipse 快速上手指南(2)

作者：asklxf

4. 在 Eclipse 中使用 JUnit

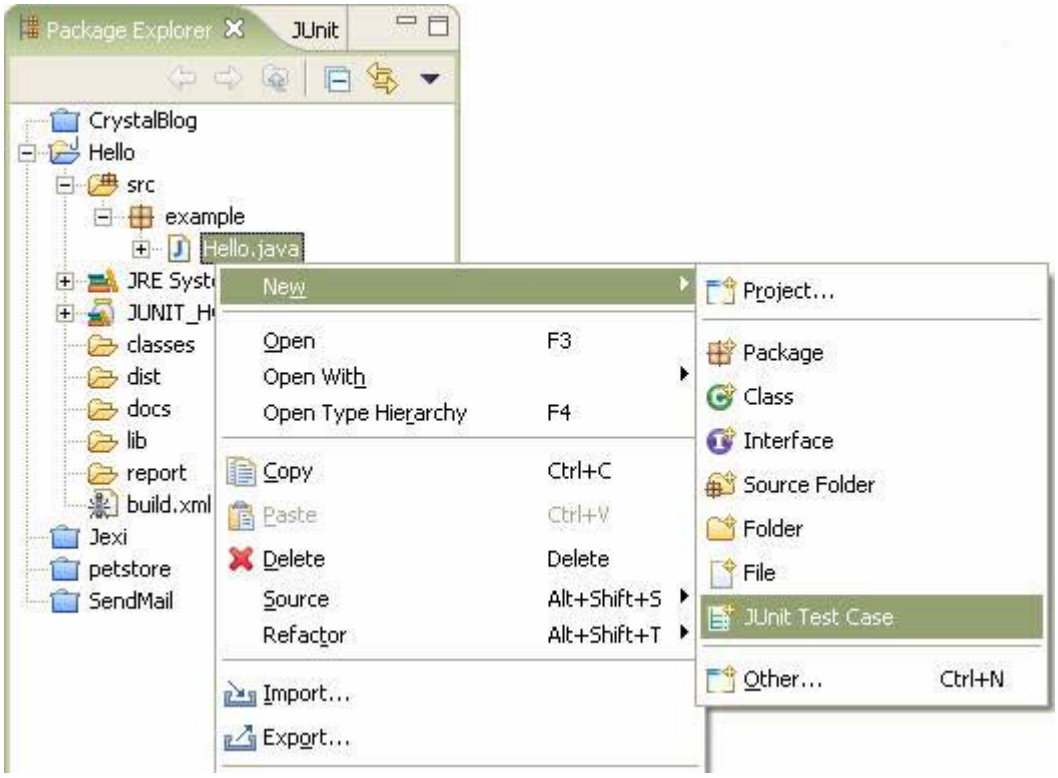
测试对于保证软件开发质量有着非常重要的作用，单元测试更是必不可少，JUnit 是一个非常强大的单元测试包，可以对一个/多个类的单个/多个方法测试，还可以将不同的 TestCase 组合成 TestSuit，使测试任务自动化。Eclipse 同样集成了 JUnit，可以非常方便地编写 TestCase。

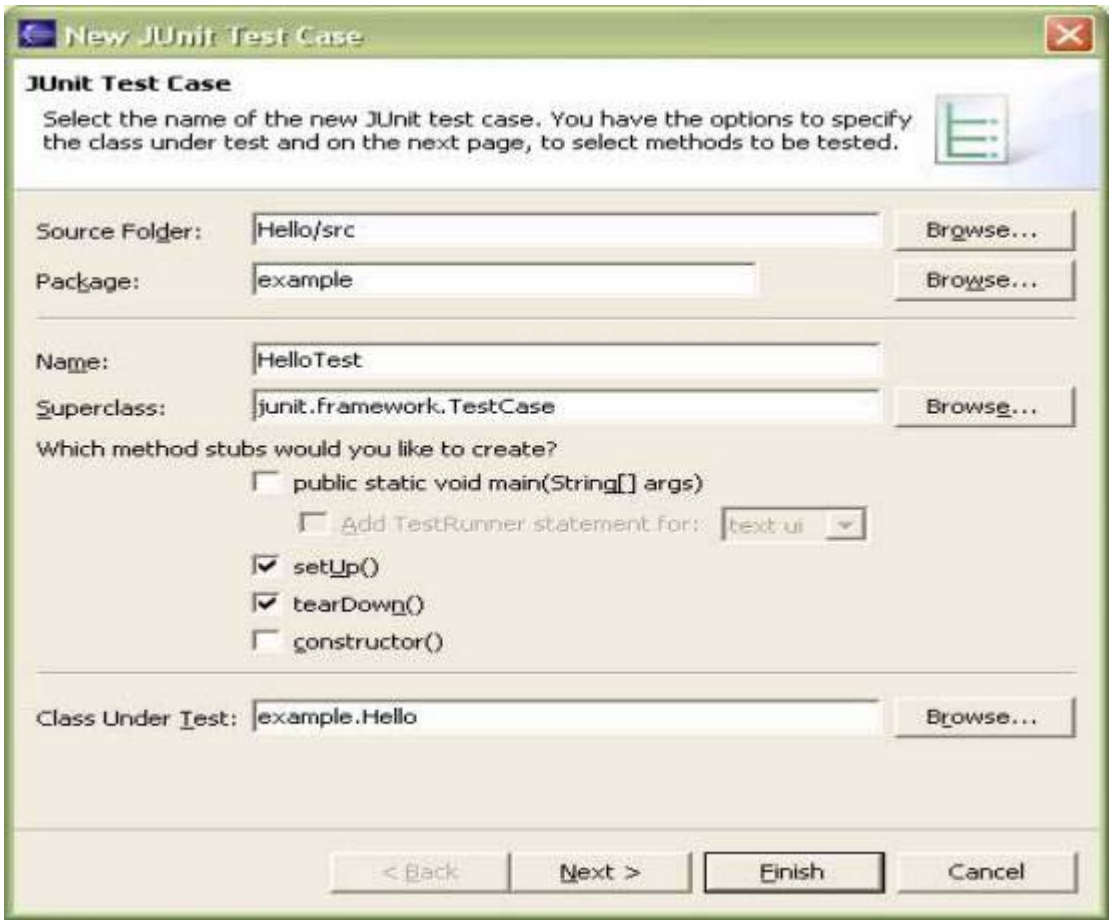


我们创建一个 Java 工程，添加一个 example.Hello 类，首先我们给 Hello 类添加一个 abs()方法，

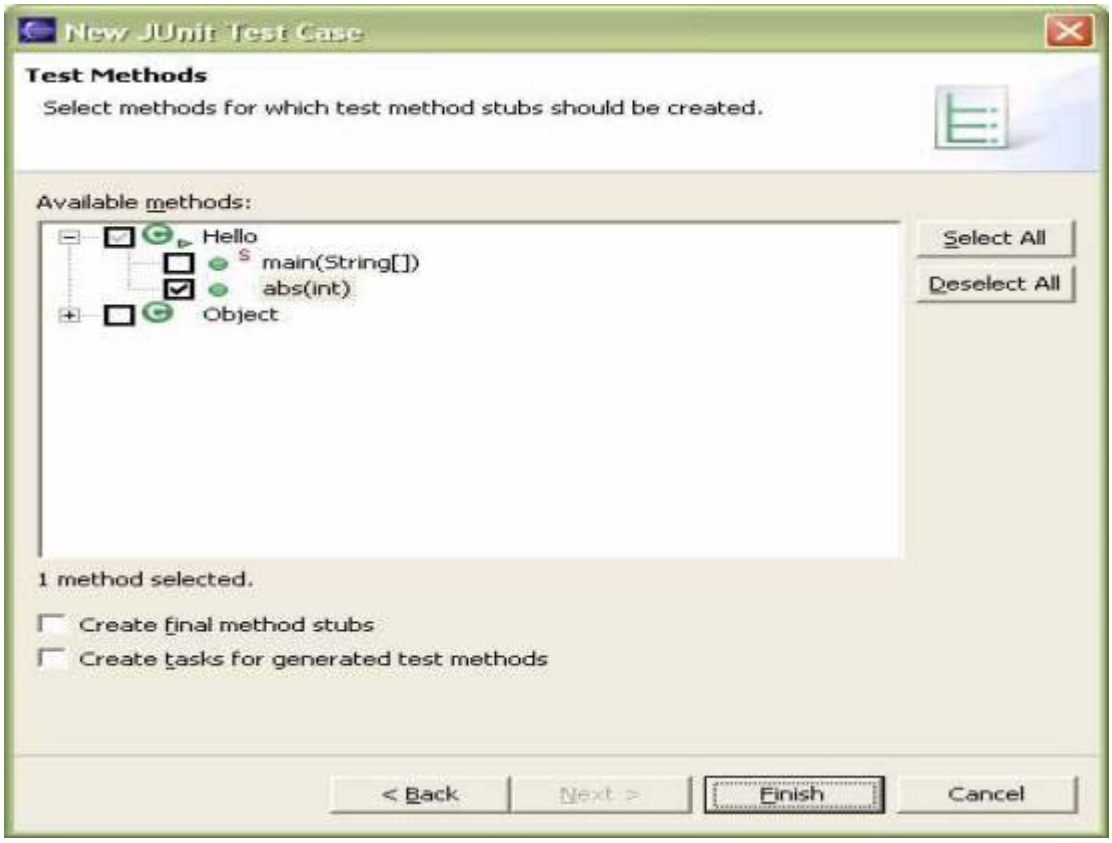
作用是返回绝对值：

下一步，我们准备对这个方法进行测试，确保功能正常。选中 Hello.java，右键点击，选择 New->JUnit Test Case：



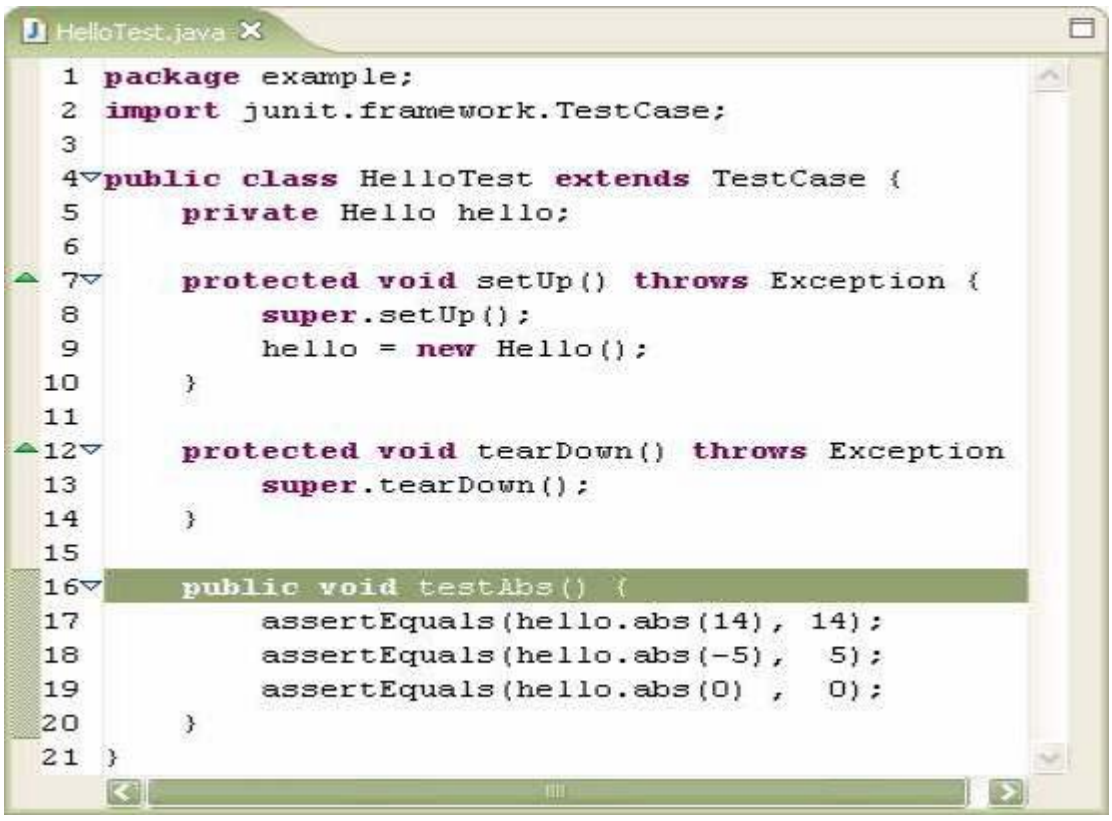


Eclipse 会询问是否添加 junit.jar 包，确定后新建一个 HelloTest 类，用来测试 Hello 类。



选中 setUp()和 tearDown()，然后点击“ Next ”：

选择要测试的方法，我们选中 abs(int)方法，完成后在 HelloTest.java 中输入：



JUnit 会以以下顺序执行测试：（大致的代码）

```
try {

HelloTest test = new HelloTest(); // 建立测试类实例

test.setUp(); // 初始化测试环境

test.testAbs(); // 测试某个方法

test.tearDown(); // 清理资源

}

catch…
```

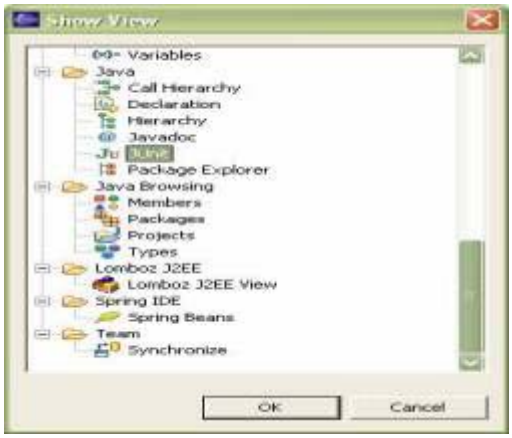
setUp()是建立测试环境，这里创建一个 Hello 类的实例；tearDown()用于清理资源，如释放打开的文件等等。以 test 开头的方法被认为是测试方法，JUnit 会依次执行 testXxx()方法。在 testAbs()方法中，我们对 abs()的测试分别选择正数，负数和 0，如果方法返回值与期待结果相同，则 assertEquals 不会产生异常。

如果有多个 testXxx 方法，JUnit 会创建多个 XxxTest 实例，每次运行一个 testXxx 方法，setUp()和 tearDown()会在 testXxx 前后被调用,因此,不要在一个 testA()中依赖 testB()。

直接运行 Run->Run As->JUnit Test，就可以看到 JUnit 测试结果：

绿色表示测试通过，只要有 1 个测试未通过，就会显示红色并列出未通过测试的方法。可以试图改变 abs()的代码，故意返回错误的结果（比如 return n+1;），然后再运行 JUnit 就会报告错误。

如果没有 JUnit 面板，选择 Window->Show View->Other，打开 JUnit 的 View：



JUnit 通过单元测试，能在开发阶段就找出许多 Bug，并且，多个 Test Case 可以组合成 Test Suite，让整个测试自动完成，尤其适合于 XP 方法。每增加一个小的新功能或者对代码进行了小的修改，就立刻运行一遍 Test Suite，确保新增和修改的代码不会破坏原有的功能，大大增强软件的可维护性，避免代码逐渐“腐烂”。

Eclipse 快速上手指南(3)

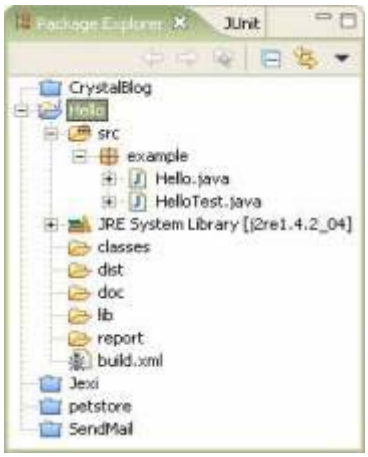
作者：asklxf

5．在 Eclipse 中使用 Ant

Ant 是 Java 平台下非常棒的批处理命令执行程序，能非常方便地自动完成编译，测试，打包，部署等等一系列任务，大大提高开发效率。如果你现在还没有开始使用 Ant，那就要赶快开始学习使用，使自己的开发水平上一个新台阶。

Eclipse 中已经集成了 Ant，我们可以直接在 Eclipse 中运行 Ant。

以前面建立的 Hello 工程为例，创建以下目录结构：



新建一个 build.xml，放在工程根目录下。build.xml 定义了 Ant 要执行的批处理命令。虽然 Ant 也可以使用其它文件名，但是遵循标准能更使开发更规范，同时易于与别人交流。

通常，src 存放 Java 源文件，classes 存放编译后的 class 文件，lib 存放编译和运行用到的所有 jar 文件，web 存放 JSP 等 web 文件，dist 存放打包后的 jar 文件，doc 存放 API 文档。

然后在根目录下创建 build.xml 文件，输入以下内容：

```
<?xml version="1.0"?>

<project name="Hello world" default="doc">

<!-- properies -->

<property name="src.dir" value="src" />

<property name="report.dir" value="report" />

<property name="classes.dir" value="classes" />

<property name="lib.dir" value="lib" />

<property name="dist.dir" value="dist" />

<property name="doc.dir" value="doc"/>

<!-- 定义 classpath -->

<path id="master-classpath">

<fileset file="${lib.dir}/*.jar" />

<pathelement path="${classes.dir}"/>

</path>

<!-- 初始化任务 -->

<target name="init">

</target>

<!-- 编译 -->

<target name="compile" depends="init" description="compile the source files">

<mkdir dir="${classes.dir}"/>

<javac srcdir="${src.dir}" destdir="${classes.dir}" target="1.4">
```

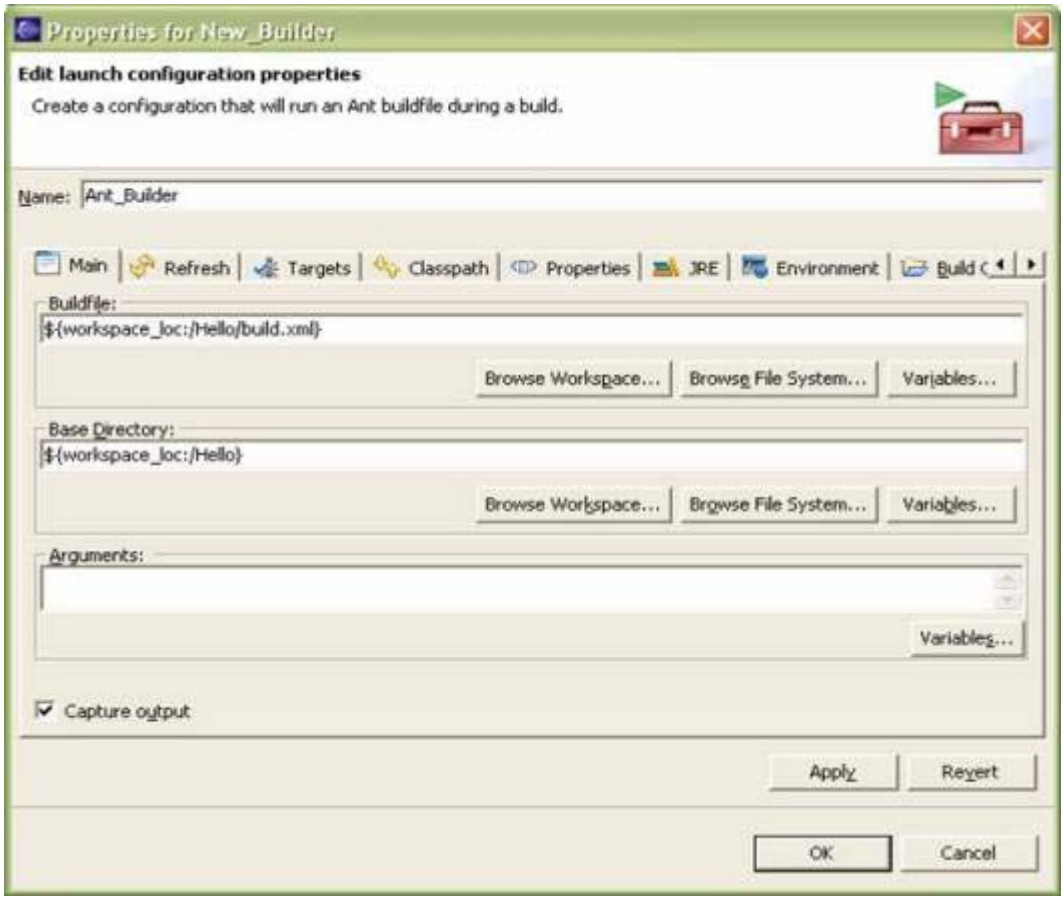
```
<classpath refid="master-classpath"/>

</javac>

</target>

<!-- 测试 -->

<target name="test" depends="compile" description="run junit test">

<mkdir dir="${report.dir}"/>

<junit printsummary="on"

haltonfailure="false"

failureproperty="tests.failed"

showoutput="true">

<classpath refid="master-classpath" />

<formatter type="plain"/>

<batchtest todir="${report.dir}">

<fileset dir="${classes.dir}">

<include name="**/*Test.*"/>

</fileset>

</batchtest>

</junit>

<fail if="tests.failed">

**********************************************************

****One or more tests failed!Check the output ...****

**********************************************************

</fail>

</target>
```

```
<!-- 打包成 jar -->
<target name="pack" depends="test" description="make .jar file">
<mkdir dir="${dist.dir}" />
<jar destfile="${dist.dir}/hello.jar" basedir="${classes.dir}">
<exclude name="**/*Test.*" />
<exclude name="**/Test*.*" />
</jar>
</target>
<!-- 输出 api 文档 -->
<target name="doc" depends="pack" description="create api doc">
<mkdir dir="${doc.dir}" />
<javadoc destdir="${doc.dir}"
author="true"
version="true"
use="true"
windowtitle="Test API">
<packageset dir="${src.dir}" defaultexcludes="yes">
<include name="example/**" />
</packageset>
<doctitle><![CDATA[<h1>Hello, test</h1>]]></doctitle>
<bottom><![CDATA[<i>All Rights Reserved.</i>]]></bottom>
<tag name="todo" scope="all" description="To do:" />
</javadoc>
</target>
```

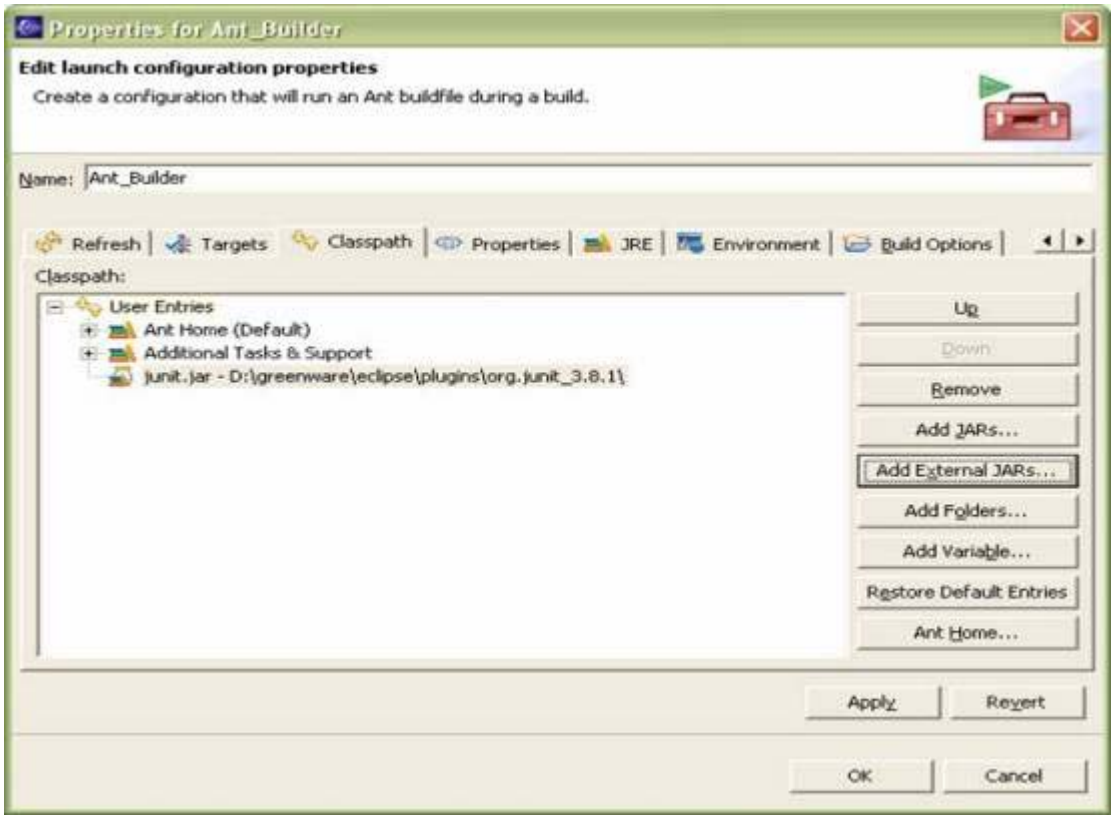
```
</project>
```

以上 xml 依次定义了 init（初始化），compile（编译），test（测试），doc（生成文档），pack（打包）任务，可以作为模板。

选中 Hello 工程，然后选择“ Project ”，“ Properties ”，“ Builders ”，“ New… ”，选择“ Ant Build ”：



填入 Name：Ant_Builder；Buildfile：build.xml；Base Directory：${workspace_loc:/Hello}（按“ Browse Workspace ”选择工程根目录），由于用到了 junit.jar 包，搜索 Eclipse 目录，找到 junit.jar，把它复制到 Hello/lib 目录下，并添加到 Ant 的 Classpath 中：



然后在 Builder 面板中钩上 Ant_Build，去掉 Java Builder：

再次编译，即可在控制台看到 Ant 的输出：

```
Buildfile: F:\eclipse-projects\Hello\build.xml

init:

compile:
```

```
[mkdir] Created dir: F:\eclipse-projects\Hello\classes

[javac] Compiling 2 source files to F:\eclipse-projects\Hello\classes

test:

[mkdir] Created dir: F:\eclipse-projects\Hello\report

[junit] Running example.HelloTest

[junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.02 sec

pack:

[mkdir] Created dir: F:\eclipse-projects\Hello\dist

[jar] Building jar: F:\eclipse-projects\Hello\dist\hello.jar

doc:

[mkdir] Created dir: F:\eclipse-projects\Hello\doc

[javadoc] Generating Javadoc

[javadoc] Javadoc execution

[javadoc] Loading source files for package example...

[javadoc] Constructing Javadoc information...

[javadoc] Standard Doclet version 1.4.2_04

[javadoc] Building tree for all the packages and classes...

[javadoc] Building index for all the packages and classes...

[javadoc] Building index for all classes...

[javadoc] Generating F:\eclipse-projects\Hello\doc\stylesheet.css...

[javadoc] Note: Custom tags that could override future standard tags:@todo. To avoid
potential overrides, use at least one period character (.) in custom tag names.

[javadoc] Note: Custom tags that were not seen:@todo

BUILD SUCCESSFUL
```

Total time: 11 seconds

Ant 依次执行初始化，编译，测试，打包，生成 API 文档一系列任务，极大地提高了开发效率。将来开发 J2EE 项目时，还可加入部署等任务。并且，即使脱离了 Eclipse 环境，只要正确安装了 Ant，配置好环境变量 ANT_HOME=<Ant 解压目录>，Path=…;%ANT_HOME%\bin，在命令行提示符下切换到 Hello 目录，简单地键入 ant 即可。

Eclipse 平台 J2ME 开发之整合

作者：蜡笔小刀

(早就答应 mingjava 介绍一下怎样在 Eclipse 平台下整合厂商 sdk，无耐之几天碰上点小难题一时解决不好，心情郁闷。但郁闷不是办法啊，写点东西也算是排遣一下吧:)

使用 Eclipse 开发 J2ME 时，最好能将常用的一些厂商的模拟器都整合进去，比较方便。本文简要介绍一下如何将 Nokia,索爱，三星的模拟器整合到 Eclipse 环境中。注意我使用的 EclipseMe 版本为 0.4.3,新的版本可能会有点变化，希望我抛砖引玉，能有人补充更正。

厂商提供的开发包大概有三种形式：

一是提供一个厂商版本的 WTK,比如索爱,这个 WTK 里面包含了 SDK 和模拟器，开发者直接使用这个 WTK 进行开发（根使用 SUN 提供的标准 WTK 一样）；

二是提供和 WTK 配合使用的开发工具，如 Nokia。你需要自己先行安装 WTK,而 Nokia 开发工具安装的时候会让你选则 WTK 的路径。

三是提供一个独立的开发环境，但不是 WTK 的形式,如 Moto。Moto 的开发工具上有一个模拟器启动程序，有各种模拟器对应的 sdk 和编译的批处理文件。

这三种是我遇到过的，其他的没用过也不知道。对于索爱，Nokia 这样依托于 WTK 的模拟器比较容易整合到 Eclipse 中，因为 EclipseMe 也是依托于 WTK 的。如果将这些厂商的模拟器整合到一个 WTK 里面，就可以在 Eclipse 中使用了。

1. Nokia 模拟器

Nokia 的模拟器都是由安装程序安装的，其实找一个已经安装好的模拟器的整个文件夹直接拷到 wtk 的 wtklib\devices 文件夹中，就可以用了。废话两句，我常用的有 7210,3300,S60beta0.1 这些模拟器。其中 7210 支持中文，启动速度快，是 40 开发的首选。60 的模拟器都比较慢，一般只有移植的时候用一下。

2. 索爱 K700 与三星 SGH-S100,S200,C100 模拟器

索爱和三星都是以 WTK 形式提供的开发包，模拟器文件夹在他们的 WTK 的 devices 中。将他们拷到你使用的 WTK 中吧。但是要改动一个地方，否则在 Eclipse 中就不能用了。以索爱 K700 为例，打开模拟器文件夹中的配置文件"SonyEricsson_K700.properties",搜索 keyboard.handler = com.sun.kvem.midp.ConfigurableKeyboardHandler，将他用#注释调,

改成 keyboard.handler = com.sun.kvem.midp.DefaultKeyboardHandler,这样就可以在 Eclipse 中用了。但三星的这几个模拟器只能在 WTK2.2 下面用，如果不想换掉当前的 WTK，你就得再安装一个 WTK2.2,EclipseMe 可以同时支持多个 WTK，如果你建立工程的时候选择的是 WTK2.2 的配置就可以在 run 的模拟器选单中选择三星的模拟器了。但这样还是有问题，至少对我的这个版本的 EclipseMe 是这样，你需要打开 project 的属性窗口，将连接库中的内容全部删掉（原来应该是默认的 WTk2.2 的 Lib）而手工添加你需要的 lib,如 cldc1.1,midp2.0,mmapi 等。

3. Moto 模拟器

我现在还没办法将他们整合到 Eclipse 中。我发现 Moto 的模拟器不是 WTK 的标准格式，它的配置文件格式和 WTK 的模拟器不兼容。这个配置文件好像是 Moto 开发工具中的 launch.exe 读取用的。希望有哪位了解的能说明一下。

说明：本文所提主要是 Nokia 和 Midp2.0 的模拟器。Midp2.0 的模拟器基本上只有一个配置文件和几张图片，而 Nokia 的模拟器里面还带有 lib 等。对于有厂商 api 的模拟器，情况可能并不相同。这个需要具体机型具体研究，不过 Eclipse 可以指定外部 jar，也许这就是解决之道。

J2ME 入门

介绍 MIDP 应用程序的属性

作者：mingjava

MIDlet 是在 MIDP 中提出的一种应用程序模型，目前在 J2ME 中应用最为广泛。本文将主要介绍 MIDP 应用程序的属性问题。读者可以参考 MIDP Application Properties

MIDlet 可以访问两种运行时的属性值：系统和应用程序的。

系统属性的概念是在 CLDC（Connected Limited Device Configuration）中定义的，属性值被写入底层的系统，我们可以读取它们但是不能修改这些属性值。如果你想读取一个系统属性值那么你可以使用 System 类的静态方法 System.getProperty()来读取。经常有网友会询问如何读取手机号码或者 IMEI 号码，其实这些你应该参考具体机型的开发文档。各个厂商的实现都是不一样的。为了让大家查找方便这里列出在 J2ME 中定义的系统属性值，如果你的手机支持相关的 JSR，那么就可以通过上述方法取得属性值。

JSR	Property Name	Default Value1
30	microedition.platform	null
	microedition.encoding	ISO8859_1
	microedition.configuration	CLDC-1.0
	microedition.profiles	null
37	microedition.locale	null

```
   microedition.profiles                        MIDP-1.0
75 microedition.io.file.FileConnection.version 1.0
   file.separator                               (impl-dep)
   microedition.pim.version                     1.0
118microedition.locale                           null
   microedition.profiles                        MIDP-2.0
   microedition.commports                       (impl-dep)
   microedition.hostname                        (impl-dep)
120wireless.messaging.sms.smsc                  (impl-dep)
139microedition.platform                         (impl-dep)
   microedition.encoding                        ISO8859-1
   microedition.configuration                   CLDC-1.1
   microedition.profiles                        (impl-dep)
177microedition.smartcardslots                   (impl-dep)
179microedition.location.version                 1.0
180microedition.sip.version                      1.0
184microedition.m3g.version                      1.0
185microedition.jtwi.version                     1.0
195microedition.locale                           (impl-dep)
   microedition.profiles                        IMP-1.0
205wireless.messaging.sms.smsc                  (impl-dep)
205wireless.messaging.mms.mmsc                  (impl-dep)
```

应用程序属性值是在应用程序描述符文件或者 MANIFEST 文件中定义的（其中 MANIFEST 文件是打包在 jar 文件中的），当我们部署应用程序的时候会定义应用程序属性。比如下面是一个典型的 jad 文件内容：

```
MIDlet-1: HttpWrapperMidlet,,httpwrapper.HttpWrapperMIDlet
MIDlet-Jar-Size: 16315
MIDlet-Jar-URL: HttpWrapper.jar
MIDlet-Name: HttpWrapper
MIDlet-Vendor: Vendor
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
Which-Locale: en
```

其中 Which-Locale 就是应用程序属性值，我们可以通过 MIDlet 的成员方法 getAppProperty()来得到它，代码片断如下：

import javax.microedition.midlet.*;public class MyMIDlet extends MIDlet {private String suiteName;public MyMIDlet(){suiteName = getAppProperty("MIDlet-Name");... // more stuff}... // etc.}

属性值是大小写敏感的，如果属性值在系统，jad 文件和 manifest 文件中都没有定义的话，那么将返回 null。如果在 jad 文件和 manifest 文件中定义了同样的属性值，那么会出现如下两种情况：如果应用程序是 MIDP2.0 种的信任应用程序,那么 AMS 将会拒绝安装。否则在 jad 文件中的属性值会覆盖 manifest 中的值。

在 jad 文件中使用属性值有一定的好处，如果你需要更改一些数据而又不想重新编译代码、打包的话，那么你可以在 jad 中定义一些属性值。这样可以配置你的应用程序，想想是不是和你在 j2se 应用中使用属性文件类似。但是不要在 jad 文将中定义大量的数据，因为很多设备都是对 jad 文件的大小有限制的。

精通 J2ME 中的 Hello World

作者：mingjava

初学 java 的时候一般都写过如下的 HelloWorld 程序,今天我准备详细讲述一下 J2ME 中的 HelloWorld。无论你是 J2ME 开发高手还是新手都应该读读这篇文章，我想它会对你有所帮助！

在 javax.microedition.midlet 包中定义了一个非常重要的类 MIDlet，所有 J2ME 的应用程序都必须扩展这个类，只有这样才可以使得应用管理软件(Application Management Software)管理 MIDlet,包括下载、安装和删除。在被 AMS 管理的同时，MIDlet 可以和应用管理软件通信通知应用管理软件自己状态的变化，通常是通过方法 notifyDestroyed()和 notifyPaused()实现的。最后 MIDlet 还可以通过 getAppProperty(String name)读取在 jad 文件中定义的属性值。

MIDlet 有三个状态,分别是 pause,active 和 destroyed。在启动一个 MIDlet 的时候，应用管理软件会首先创建一个 MIDlet 实例并使得他处于 pause 状态，当 startApp()方法被调用的时候 MIDlet 进入 active 状态，也就是我们平时所说的运行状态。在 active 状态调用 destroyApp(boolean unconditional)或者 pauseApp()方法可以使得 MIDlet 进入 destroyed 或者 pause 状态。值得一提的是 destroyApp(boolean unconditional)方法，很多开发者对 unconditional 参数不是很理解，事实上，当 destroyApp()方法被调用的时候，AMS 通知 MIDlet 进入 destroyed 状态。在 destroyed 状态的 MIDlet 必须释放了所有的资源，并且保存了数据。如果 unconditional 为 false 的时候，MIDlet 可以在接到通知后抛出 MIDletStateChangeException 而保持在当前状态，如果设置为 true 的话，则必须立即进入 destroyed 状态。

下面是根据上述问题我编写的一个 J2ME 中的 HelloWorld 程序，他比 j2se 中的 helloworld 要复杂一点。新手可以仔细看看。

```
package com.j2medev.mingjava;

import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.lcdui.*;
```

```
public class HelloWorld extends MIDlet implements CommandListener
{
    private Display display;
    private Form mainForm;
    private StringItem stringItem;
    private Command exitCommand = new Command("Exit", Command.EXIT, 1);

    public static final String WEB_SITE = "WEB_SITE";

    protected void startApp() throws MIDletStateChangeException
    {

        initMIDlet();
        display.setCurrent(mainForm);

    }

    private void initMIDlet()
    {
        display = Display.getDisplay(this);
        mainForm = new Form("Hello World");
        stringItem = new StringItem(null, null);
        String text = getAppProperty(WEB_SITE);
        stringItem.setText(text);
        mainForm.append(stringItem);
        mainForm.addCommand(exitCommand);
        mainForm.setCommandListener(this);
    }

    protected void pauseApp()
    {

    }

    protected void destroyApp(boolean arg0) throws MIDletStateChangeException
    {
        System.out.println("exit the application");
    }

    public void commandAction(Command cmd, Displayable display)
    {
        if (cmd == exitCommand)
        {
            try
            {
```

```
                destroyApp(false);
                notifyDestroyed();
            } catch (MIDletStateChangeException e)
            {
                e.printStackTrace();
            }
        }
    }

}
```

HelloWorldMIDlet.jad 文件内容如下，如果你还不是很清楚 jad 文件请参考 MIDP 中的属性问题。尤其是当你在安装 MIDlet 到手机上的时候。

```
MIDlet-Jar-Size: 2128
MIDlet-1: HelloWorld,/icon.png,com.j2medev.mingjava.HelloWorld
MIDlet-Jar-URL: HelloWorldMIDlet.jar
MicroEdition-Configuration: CLDC-1.0
MIDlet-Version: 1.0.0
MIDlet-Name: HelloWorldMIDlet
MIDlet-Vendor: www.j2medev.com
MicroEdition-Profile: MIDP-1.0
WEB_SITE: www.j2medev.com
```

在 J2ME 中读取各种格式的文本文件

作者：mingjava

这大概是 J2ME 开发网的第一百篇文章了，虽然不具有里程碑的意义但是多少见证了 J2ME 开发网一路走来的历史。本文将对 J2ME 进行详细的总结，理清一下脉络。

学习 J2ME 最重要的部分是掌握 J2ME 的体系结构，J2ME 是 SUN 针对嵌入式消费类电子产品推出的开发平台，事实上它是一系列的技术和规范的总称。J2ME 的核心是 Configuration（配置），Configuration 的目的是定义一套纵向的 API 而服务于广泛的设备，同时 VM 也是在 Configuration 一层规定的。在 Configuration 的上面是 Profile（描述），他是运行在 Configuration 上面的，它的目的是为一类具有某些重要特性的设备提供支持。这些特性都很重要，但是它们又不是在所有的设备上都具有的。SUN 对当前市场上的设备进行硬件层的抽象根据设备的处理器、内存和网络连接能力提出了两个标准的配置，这就是 CLDC 和 CDC。目前国内提及 J2ME 就指 CLDC+MIDP 事实上是不准确的，忽略了更为强大的 CDC 是难以原谅的。配置和描述的配合异常的灵活，在一个配置上面可以同时支持多个描述，例如在 CLDC 上我们可以运行基于 MIDP 上的程序，如果设备支持的话你还可以运行 Bluetooth 这样的可选包。掌握了 J2ME 的体系结构我们就不会在 J2ME 的学习中迷路。如果你要详细了解 J2ME 一路发展的历史，可以下载 JSR 规范读读，地址是 http://jcp.org

目前流行的开发是手机开发，因此我们在本文中将主要讲述 CLDC+MIDP 的主要内容，下文的 J2ME 也特指 CLDC+MIDP,但是我们必须清楚 J2ME 并不是指 CLDC+MIDP。

J2ME 的内容并不多,如果你读读 MIDP 的 api 的话，发现总共也就是那么百十个类。比起 J2SE 的几千个类库真是小巫见大巫，之所以提供这么精简的类库给开发人员主要原因是移动信息设备的资源受限特性。在 CLDC1.0 中只定义了三个包 java.lang,java.io 和 java.util。这构成了 CLDC 的语言基础，在这层同时还包括了 KVM。在开发 J2ME 程序的时候切忌想当然，因为并不是所有的 J2SE 类库都在 J2ME 中得到了支持。

学好 J2ME 最快捷的方式就是编写 J2ME 的应用程序，多读代码、多写代码。在 J2ME 中提出了一种新的应用程序模式——MIDlet，这个类定义在 javax.microedition.midlet.MIDlet 中，我们的 MIDlet 必须扩展这个类并实现它的三个抽象方法 startApp(),pauseApp()和 destroyApp()。方法同时也反映出了 MIDlet 的生命周期，MIDlet 的生命周期是由 AMS(application management software,以前叫做 JAM)管理的。关于 MIDlet 的生命周期可以仔细参考一下 API doc。

在 CLDC 推出后两个月的时间,SUN 就推出了 MIDP1.0，让人兴奋的是在 MIDP 中提供了 GUI，这样开发人员可以很方便的编写 J2ME 应用程序了，GUI 得类库在 javax.microediton.lcdui 中定义，在 MIDP2.0 中 SUN 增加了对游戏开发的支持推出了 javax.microedition.lcdui.game 包,方便开发人员开发游戏。MIDP 中的 UI 并没有采取 AWT 或者 SWING 的设计思想，因为他们是针对 PC 的，在手机等设备上主要的交互还是通过按键完成的，因此针对鼠标键盘事件机制设计的 AWT/SWING 并不适合 J2ME 平台。在掌握 UI 的时候，我们在头脑中应该清楚的知道 J2ME 平台的界面和事件处理是区分高级和低级的。高级界面和事件处理相对简单，但是速度快、可移植性好。低级用户界面和时间处理相对复杂，但是功能强大、可移植性差。通常在开发游戏的时候我们多用低级 UI。

在 J2ME 平台中提供了一个小型的数据库，他就是 Record Management System。他的数据是存储在非挥发性存储器上的，因此不会因为程序的退出以及手机的关机而丢失，从而为 J2ME 平台提供了持久性存储。RMS 的设计异常的小巧，他主要负责存储数据和标记数据，数据存储是面向子节的，RMS 规范并没有说明什么数据能被存储，只要数据可以转换为子节，都可以被存储。RMS 是通过 id 来标志数据的，但是他并不是索引。设计小巧当然适合在手机上运行了，但是加重了开发人员的任务，我们必须负责存储数据和读取数据并表示数据，在本站有专题介绍 RMS，请参考。

在 J2ME 中非常重要的框架就是 GCF，它是在 javax.microediton.io 里面定义的，提供了联网的能力。在 MIDP2.0 中更是提供了对 TCP/IP 层联网的支持。在 GCF 中核心是 Connector，而面向接口的设计使得 GCF 的扩展性非常出色。我们在开发联网程序的时候必须要涉及到的问题就是多线程问题。因为联网操作必须在另一个线程中完成，而不能再主线程内，这样是为了避免堵塞。这时候你应该认识到其实学好 J2ME 必须要有坚实的 J2SE 的基础。

在 J2ME 中有个非常重要的概念就是可选包，可选包是针对特定设备功能提出的，比如有些设备可以支持移动多媒体，那么你就可以使用 MMAPI 进行相关的开发。

事实上设备厂商同时会开发一些针对自己设备的 API 给开发人员使用，一旦你使用了他们的 API 那么你的应用程序就丧失了可移植性，比如 Nokia 6108 的程序不能在 Motorola 388c 上运行。SUN 为了改善这些分裂 API 的问题在 JSR185 中进行了一定强度的规范，也就是我们所知道的 JTWI，JSR185 并没有提供新的 API，只是对实现 JTWI 的设备进行了规范，比如 Heap 空间至少为 256K 等。详细资料可下载 JSR 规范读读看。

希望这篇文章可帮助 J2ME 开发者理一下脉络

J2ME 概念解析

作者：asklxf

J2ME，即 Java 2 Micro Edition，是 SUN 公司推出的在移动设备上运行的微型版 Java 平台，常见的移动设备有手机，PDA，电子词典，以及各式各样的信息终端如机顶盒等等。

由于移动终端的类型成千上万，而且计算能力差异非常大，不可能像桌面系统那样仅仅两三个版本的 JVM 即可满足 Windows，Linux 和 Unix 系统，因此，J2ME 不是一个简单的微型版的 JVM。为了满足千差万别的移动设备的需求，SUN 定义了一系列的针对不同类型设备的规范，因此，J2ME 平台便是由许多的规范组成的集合。

最重要的移动终端当然是手机了，因此，我们主要讨论手机相关的 J2ME 规范。

Configuration

SUN 把不同的设备按照计算能力分为 CLDC（Connected Limited Device Configuration）和 CDC（Connected Device Configuration）两大类，这两个 Configuration 是针对设备软硬件环境严格定义的，比如 CLDC1.0 定义了内存大小为 64-512k，任何设备如果支持 CLDC1.0，就必须严格满足定义，不能有可选的或者含糊的功能。

CLDC1.0 是针对计算能力非常有限的设备定义的，只支持整数运算，不支持浮点运算，早期的 Java 手机大部分都支持 CLDC1.0，如 Nokia 3650，Siemens 6688i。

CLDC1.1 则增加了浮点运算，因此，在支持 CLDC1.1 的设备上，可以使用 float 和 double 类型的变量。现在的 Java 手机很多都能支持 CLDC1.1，如 Nokia 9500，Siemens S65。

CDC 则是针对计算能力比较强的设备定义的，如 PPC 等，CDC 平台的 JVM 基本上和桌面的 JVM 很接近了，只是可以使用的 Package 大大少于 J2SE 的包。支持 CDC 的非常高端的 Java 手机也会很快上市。

Profile

和 Configuration 相比，Profile 更多是针对软件接口的定义，Profile 有必须实现的，也有可选的功能，因此，Profile 更灵活。

最重要的 Profile 当然是 MIDP（Micro Information Device Profile），MIDP 定义了能在 Java 手机上运行的 Java 程序的规范，包括应用程序生命周期，各种 UI 界面组件，支持 Record 存储和 Http 连接等等，符合 MIDP 规范的 Java 小程序被称为 MIDlet，可以直接通过无线网络下载到手机并运行。

早期的 MIDP1.0 规范使我们能在手机上运行有 UI 界面的 Java 程序，但是 MIDP1.0 对游戏的支持不够，必须自己实现许多代码，因此，MIDP2.0 规范大大加强了对游戏开发的支持，使开发者能编写更少的代码来创建游戏。

MIDP 规范的图形界面基本上都是独立于 J2SE 的 AWT 和 Swing 组件，因为目前手机的计算能力还比较有限，但是，随着手机的 CPU 越来越快，使得 AWT 和 Swing 移植到手机上也将成为可能，因此，基于 CDC 规范的最新的 PBP 1.0（Personal Basic Profile）和 PP 1.0（Personal Profile）提供了部分 AWT 和 Swing 的支持，目前，部分高端 PDA 已经可以运行 PBP 和 PP 的 Java 程序了。可以预见，将来大部分的 AWT 和 Swing 组件都能移植到手机上。

前面已经说过，和 Configuration 相比，Profile 有许多可选包，比较实用的 Profile 还有在 JSR135 定义的 MMAPI（Mobile Media API），实现多媒体播放功能；在 JSR184 定义的 M3G API（Mobile 3D Graphics API），实现 3D 功能；在 JSR120 定义的 WMA（Wireless Messageing API），实现短消息收发。如果你的手机支持某一 Profile，如 M3G，那么便可以在 MIDlet 中使用 M3G 的 3D API 实现 3D 游戏。

如果你准备在手机上开发 J2ME 应用，选择手机时就需要注意厂商支持的 CLDC 规范，支持 MIDP1.0 还是 2.0，是否支持 MMAPI，M3G，WMA 等可选包。

Profile 虽然定义了 Java API 接口，但是底层如何实现是由各厂商自己决定的，如 M3G 定义了 3D 接口，但是底层实现既可以使用硬件加速，也可以由 C 程序模拟，或者部分由硬件实现，部分由软件实现。

比 J2ME 更精简的 Java 平台被 SUN 称为 JavaCard，运行在信用卡等芯片中，实现电子支付等功能，目前 SUN 还没有把 JavaCard 并入 J2ME 平台。

j2me 进度条与线程化模型

内容提要：
本文研究如何建立一个方便使用的线程化模型，这个线程化模型由前台的进度条 UI 和后台的背景线程组成。

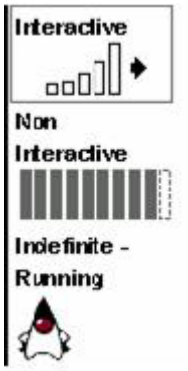
解决的问题

在 j2me 的 UI 体系中，UI 操作是在一个独立的线程中运行的。往往在 api doc 中要求程序员对接口方法立即返回。也就是说非阻塞的。你必须开启一个独立的线程来完成你自定义的复杂的工作，比如联网等可能发生阻塞的 io 操作。新的线程如果不和用户交流，告诉用户线程正在工作的话，将会显现的非常不友好。用户可能执行别的操作而扰乱程序的正常运行。一个简单的方法是提供一个进度条，这样用户就会愿意等待上一会，直到程序运行出结果。

为了将程序员从前台进度条与后台线程的通信中解脱出来，专心于后台线程的开发，有必要设计一个进度条线程模型。

应该注意到进度条有多种的形式：

A， 动画形式进度条，仅表示程序正在运行（自维护的）
B， 可交互增量形式的进度条，后台线程通过调用进度条的相应方法在程序运行中不断的改变进度条的状态
C， 进度条的表现形式应该灵活，不要固定其实现
D， 进度条对象要重复利用



进度调和后台线程的交流也有好几种情况：
A， 仅仅将进度条绘画在屏幕上，并等后台任务完成后，由后台线程跳转到成功画面。
B， 对于可取消的任务，用户可以通过点击进度条的按钮来试图 cancel 任务，后台任务应该尽快取消，并跳转到失败的画面
C， 对于不可跳转的任务，用户只有耐心等待
D， 如果背景线程运行失败，应自行跳转到失败的屏幕

进度条的设计（前台）

为了实现进度条的表现的多样性，首先抽象一个接口：
ProgressObserver.java

```
package com.favo.ui;
import javax.microedition.lcdui.Display;

/**
 * @author Favo
 *
 * 这是仿照 Smart Ticket 制作的进度条观察者，这个模型的优点是
 * 1，低耦合度。你可以通过 Form,Canvas 等来实现这个接口
 * 2，支持可中断的任务，因为背景线程是无法强制性中断的，
 * 所以就没有了在观察者中回调背景线程相应方法的必要，
 * 如果支持可中断的话，可以让背景线程来查询观察者的 isStopped()
 * 3，可以说进度条仅仅将自己绘画在屏幕上，他对后台线程毫不关心
 */
```

```
public interface ProgressObserver {
 /**
  * 将进度条复位
  */
 public void reset();

 /**
  * 将进度条设置最大
  */
 public void setMax();

 /*
  * 将自己绘制在屏幕上，如果进度条要开启自身的线程用于自动更新画面，
  * 也在这里构造并开启绘画线程（常用于动画滚动条）
  */
 public void show(Display display);

 /**
  * 滚动条退出命令，如果进度条曾经开启自身的线程用于自动更新画面，
  * （常用于动画滚动条），在这里关闭动画线程
  */
 public void exit();

 /**
  * 更新进度条
  */
 public void updateProgress(Object param1);

 public boolean isStoppable();

 public void setStoppable(boolean stoppable);

 public boolean isStopped();

 public void setStopped(boolean stopped);

 public void setTitle(String title);

 public void setPrompt(String prompt);
}
```

每个方法都很一幕了然，我解释两点：
1）“ 2，支持可中断的任务，因为背景线程是无法强制性中断的，所以就没有了在观察者中回调背景线程相应方法的必要，如果支持可中断的话，可以让背景线程来查询观察者的 isStopped() ”

如果要支持可中断线程的话，想当然的，我们希望用户按下按钮后回调后台线程的某个方法来停止线程，并且这个方法要立即返回（前面提过 UI 的用户响应不能够阻塞）。但是细细想想，线程是无法被强制停止的，而且即使能够被强制停止也很不安全。所以这个方法也只能够是通过设置某个 flag,然后立即返回。这样的话线程就和前台的 UI 紧密的耦合在一起了。与其这样，倒不如让后台线程去查询 UI 的状态。这样 UI 并不关心到底是谁在后台维护他状态。

2）如果要实现一个不交互动画 UI，那么显然这个 UI 是自维护的（也就是说 UI 单独有自己的绘画线程）。为了能够实现这种情况，可以在 show 中开启线程，在 exit 中结束线程。对于交互 UI，可以简单的忽略 exit 方法。

下面给一个利用 Form 和 Gauge 实现的交互式 UI（非自维护的），读者可以看看其中的细节，参照他可以设计自己的用 Canvas 实现的，或者自维护的等等不同的实现。

ProgressGaugeUI.java

```
package com.favo.ui;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Gauge;

/**
 * @author Favo
 * Preferences - Java - Code Style - Code Templates
 */
public class ProgressGaugeUI implements ProgressObserver, CommandListener {

 private static final int GAUGE_MAX = 8;

 private static final int GAUGE_LEVELS = 4;

 private static ProgressGaugeUI pgUI;

 private Form f;

 private Gauge gauge;

 private Command stopCMD;

 boolean stopped;

 boolean stoppable;

 int current;
```

```
protected ProgressGaugeUI() {
 f = new Form("");
 gauge = new Gauge("", false, GAUGE_MAX, 0);
 stopCMD = new Command("Cancel", Command.STOP, 10);
 f.append(gauge);
 f.setCommandListener(this);
}

public static ProgressGaugeUI getInstance() {
 if (pgUI == null) {
  return new ProgressGaugeUI();
 }
 return pgUI;
}

public void reset() {
 current=0;
 gauge.setValue(0);
 stopped=false;
 setStoppable(false);
 setTitle("");
 setPrompt("");
}

public void updateProgress(Object param1) {//这里的参数设计为提示语
 current=(current+1)%GAUGE_LEVELS;
 gauge.setValue(current * GAUGE_MAX/GAUGE_LEVELS);
 if(param1!=null && param1 instanceof String){
  setPrompt((String)param1);
 }
}

public boolean isStoppable() {
 return stoppable;
}

public void setStoppable(boolean stoppable) {
 this.stoppable = stoppable;
 if(stoppable){
  f.addCommand(stopCMD);
 }else{
  f.removeCommand(stopCMD);
 }
}
```

```
 public boolean isStopped() {
  return stopped;
 }

public void setStopped(boolean stopped) {
  this.stopped=stopped;
 }

 public void setTitle(String title) {
  f.setTitle(title);
 }

 public void setPrompt(String prompt) {
  gauge.setLabel(prompt);
 }

 public void commandAction(Command arg0, Displayable arg1) {
  if(arg0==stopCMD){
   if(isStoppable())
    stopped=true;
   else{
    setPrompt("can't stop!");
   }
  }
 }

 public void show(Display display) {
  display.setCurrent(f);
 }
public void exit() {
  // 忽略
 }
 public void setMax() {
  gauge.setValue(GAUGE_MAX);
 }
}
```

后台线程的设计

后台线程替我们作以下的内容：
1）执行我们的任务 runTask()
2）如果用户中断线程，那么 runTask()运行完后，将会跳转到我们指定的失败屏幕
3）在最后替我们调用 UI.exit()

我们需要做的：
1）提供一个前台的 UI，提供失败后跳转的画面，提供 Display 的实例
2）在 runTask()中，如果任务完成，手工跳转失败画面
3）在 runTask()中，如果任务失败，手工跳转失败画面
4）在 runTask()中改变进度栏的状态。
5）在 runTask()中查询用户是否取消，如果用户取消，应该尽快退出 runTask()

这种模型职责清晰，便于使用。但也有一个缺点：如果用户取消了任务，但是此时任务接近完成，或者已经完成。后台线程依然会显示用户取消了任务，并将会跳转到我们指定的失败屏幕。这时候会产生不一致的情况。为了解决整个问题，程序员可以在 runTask()中调用 taskComplete()来强制完成任务。这样即使用户取消了任务，依然回显示任务成功。当然你也可以不掉用 taskComplete()遵循默认的行为特点。

BackgroundTask.java

```
package com.favo.ui;

import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Alert;

/**
 * @author Favo
 * Preferences - Java - Code Style - Code Templates
 */
public abstract class BackgroundTask extends Thread {

 ProgressObserver poUI;

 protected Displayable preScreen;

 protected boolean needAlert;

 protected Alert alertScreen;

 private Display display;

 public BackgroundTask(ProgressObserver poUI, Displayable pre,
   Display display) {
  this.poUI = poUI;
  this.preScreen = pre;
  this.display = display;
  this.needAlert = false;
 }
```

```
 public void run() {
  try {
   runTask();
  } catch (Exception e) {
   Alert al = new Alert("undefine exception",
e.getMessage(), null,
                        AlertType.ALARM);
   al.setTimeout(Alert.FOREVER);
   display.setCurrent(al);
  } finally {
   if (poUI.isStoppable()) {
    if (poUI.isStopped()) {//如果用户中断了程序
     if (needAlert) {
      display.setCurrent(alertScreen, preScreen);
     } else {
      display.setCurrent(preScreen);
     }
    }
   }
   poUI.exit();
  }
 }

 /*
  * 如果任务可中断,查看 pgUI.isStopped().并尽快退出此方法;
 * 如果任务需要更新进度栏，调用 pgUI.updateProgress(“ 进度提示 ”).
 * 习惯上此方法的最后手动调用 taskComplete()以防止用户在任务接近
 * 完成时取消
  */
 public abstract void runTask();

 /**
 * 这是一个偷懒的办法,当你构造好 BackgroundTask 对象后，直接调用这个方法， *可以
帮助你初始化进度 UI,并显示出来。之后启动你的任务线程
 */
 public static void runWithProgressGauge(BackgroundTask btask, String title,
   String prompt, boolean stoppable, Display display) {
  ProgressObserver po = btask.getProgressObserver();
  po.reset();
  po.setStoppable(stoppable);
  po.setTitle(title);
  po.setPrompt(prompt);
  po.show(display);
  btask.start();
 }
```

```
 public ProgressObserver getProgressObserver() {
  return poUI;
 }

 public void taskComplete(){
  getProgressObserver().setStopped(false);
 }
}
```

如何使用

1)产生一个 ProgressObserver 对象 poUI
如果用默认的，通过调用 ProgressGaugeUI.getInstance();
2)构造 BackgroundTask 对象 bkTask，一般可以用匿名类来实现。
3）初始化 poUI-->设置后字段-->显示你的 poUI-->开启 bkTask 线程。
第三步可以用一步完成，通过调用静态方法
BackgroundTask.runWithProgressGauge(bkTask, "标题","提示", 是否可以暂停, display);

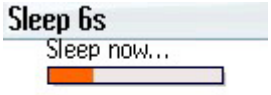
下面一个例子，看看你是否理解了，并且会使用了。
TestProgressGauge.java

```
package com.favo.ui;

import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

/**
 * @author Favo
* Preferences - Java - Code Style - Code Templates
 */
public class TestProgressGauge extends MIDlet implements CommandListener {

 Display display;

 Command workCmd;
```

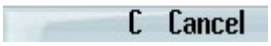
```
Command exitCmd;

Form f;

public TestProgressGauge() {
 super();
 // TODO Auto-generated constructor stub
 display = Display.getDisplay(this);
 workCmd = new Command("compute", Command.OK, 10);
 exitCmd = new Command("exit", Command.EXIT, 10);
 f = new Form("Test");
 f.setCommandListener(this);
 f.addCommand(workCmd);
 f.addCommand(exitCmd);
}

protected void startApp() throws MIDletStateChangeException {
 // TODO Auto-generated method stub
 display.setCurrent(f);
}

protected void pauseApp() {
 // TODO Auto-generated method stub

}

protected void destroyApp(boolean arg0) throws MIDletStateChangeException {
}

public void commandAction(Command arg0, Displayable arg1) {
 // TODO Auto-generated method stub
 if (arg0 == workCmd) {
  ProgressObserver poUI = ProgressGaugeUI.getInstance();
  BackgroundTask bkTask = new BackgroundTask(poUI, arg1, display) {
   public void runTask() {
    alertScreen = new Alert(
      "user cancel",
      "you press the cancel button and the screen will jump to the main Form",
      null, AlertType.ERROR);
    alertScreen.setTimeout(Alert.FOREVER);
    needAlert = true;
    //do something first
    getProgressObserver().updateProgress(null);
    try {
     Thread.sleep(3000);
```

```
    } catch (Exception e) {
     e.printStackTrace();
    }
    getProgressObserver().updateProgress("sleepd 3s...");
    if (getProgressObserver().isStopped())
     return;
    getProgressObserver().updateProgress(null);
    //do something second
    try {
     Thread.sleep(3000);
    } catch (Exception e) {
     e.printStackTrace();
    }
    getProgressObserver().setMax();
    display.setCurrent(new Form("complete"));
    taskComplete();
   }
  };
  BackgroundTask.runWithProgressGauge(bkTask, "Sleep 6s",
    "Sleep now...", true, display);
 }else if(arg0==exitCmd){
  try {
   destroyApp(false);
  } catch (MIDletStateChangeException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  }
  notifyDestroyed();
 }
}

}
```

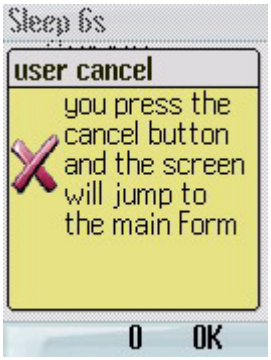
运行流程画面



按下 compute-->

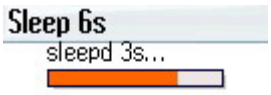


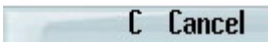
用户取消-->



回到前一屏幕-->


按下 compute-->

-->


-->

完成

希望这个模型可以加快你的开发速度。如果你有更好的解决办法，能够更清晰的解决问题或是问题的细节，欢迎讨论。

再议 j2me 进度条与线程化模型

内容提要：

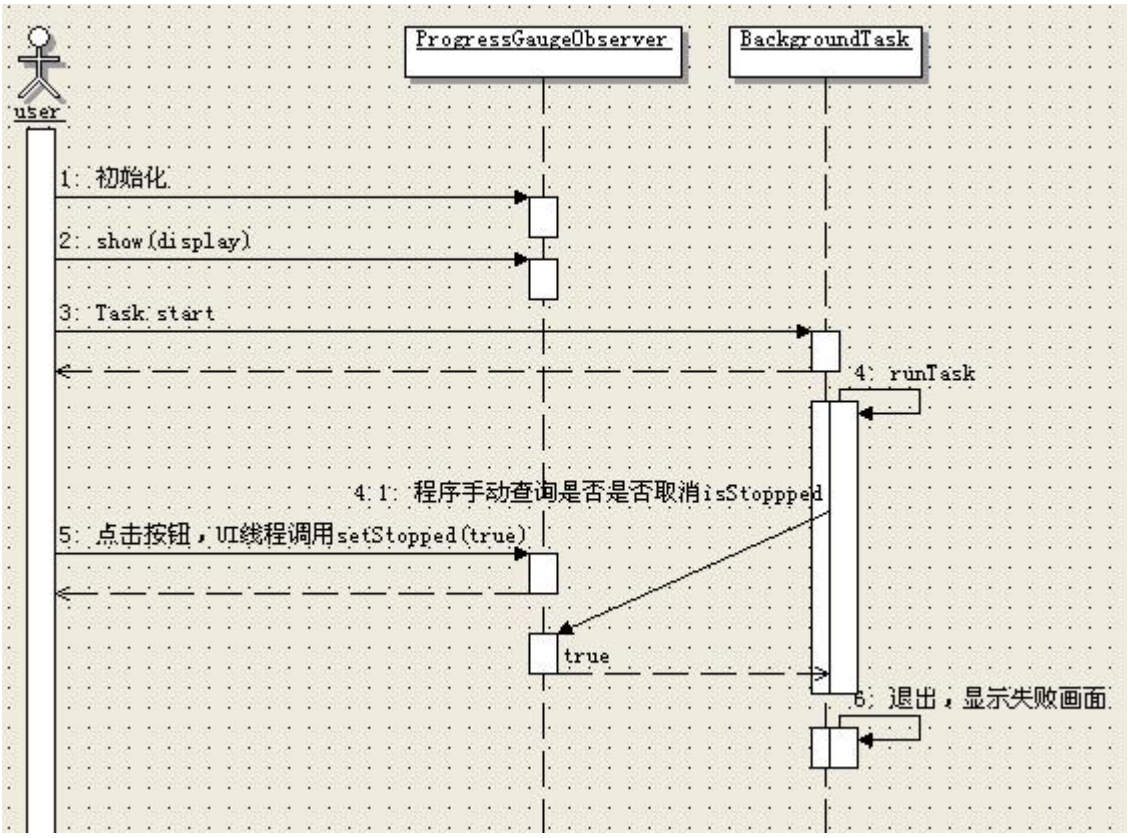
本文是《j2me 进度条与线程化模型》一文的续（以后简称原文，没看过的建议看一下）。

讨论了原文中使用的线程模型的不足，并针对她的缺点提出了新的改进办法并给出了改进后的实现。因原文中 UI 部分有灵活的扩展性，未作更改。

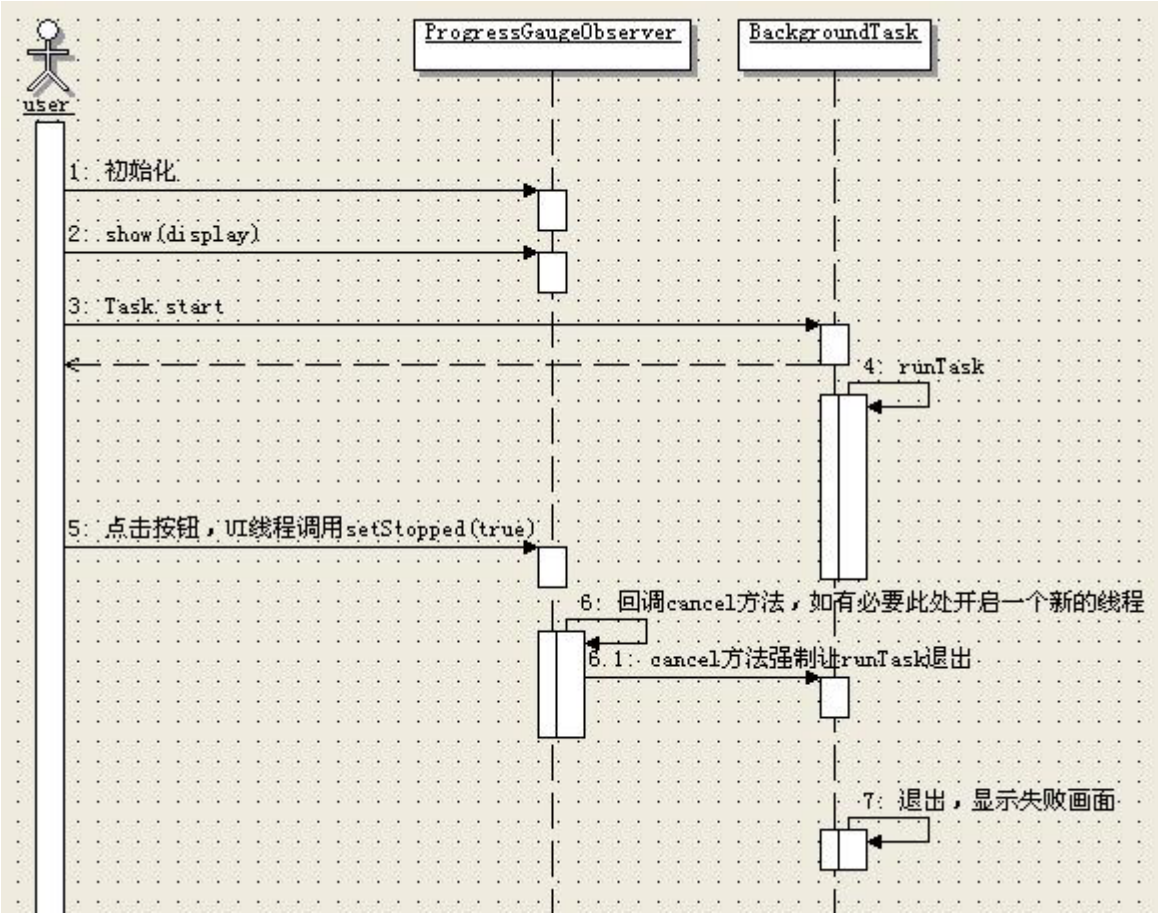
前台 UI 如何和后台线程交互

原文中模型，是一个前台的 ProgressGaugeUI 与后台线程无关的模型。这样设计的时候最大程度上的化简了通信的复杂性，实际上是一种单方向的模型（由 BackgroundTask 向 PGUI 通信）。按照这种模式的要求，程序员在 Override BackgroundTask 的 runTask()方法时，有义务定期的去查训前台的 PGUI 的运行情况，并根据这种情况做出反映。这样这种模式完全相信后台线程，将是否响应用户 cancel 命令的权利交给了后台线程，如果后台线程陷入麻烦没有响应了（比如访问一个很昂贵的网络连接），此时用户试图 cancel 也没有用，程序将会暂时的死锁，直到后台线程有时间去检查前台的状态。并且在实际情况中，到底什么时候去查询，多大的频率都是问题。在代码段中过多的此类代码，会影响对正常的流程的理解。

从下面的这个顺序图，可以看到这个具体流程：



我们需要一个方法，让我们能够强制的结束 Task。这个方法由背景线程自己提供，取名叫做 cancel()。当然没有任何一个方法可以强迫线程立即结束（曾经有，因为安全性问题而被取消）。所以 cancel()方法往往通过关闭的资源（一个连接,一个流等）来迫使 runTask 发生异常被中断，runTask 有义务根据自己的约定捕捉此类异常并立即退出。一图胜千言，让我们看看这种方法的流程。

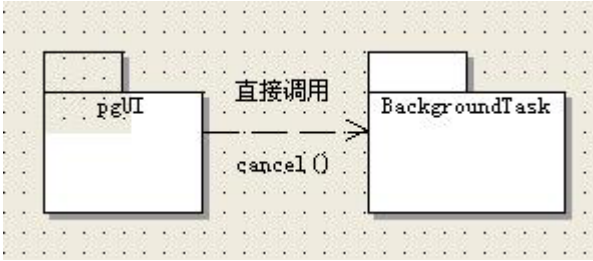


很显然的，关键在于前台的线程对后台的线程进行了回调，这样就可以解决问题了。但是新的问题来了，这样做迫使我们将前台与后台线程紧密的耦合在了一起（因为要回调嘛）。能不能既实现回调又避免前台 UI 与后台线程的紧密耦合呢？

通过 Cancelable 接口降低耦合度

幸好，我门可以利用接口来实现这一点。

先前的模型是这样的：



为了降低耦合，我们建立一个接口

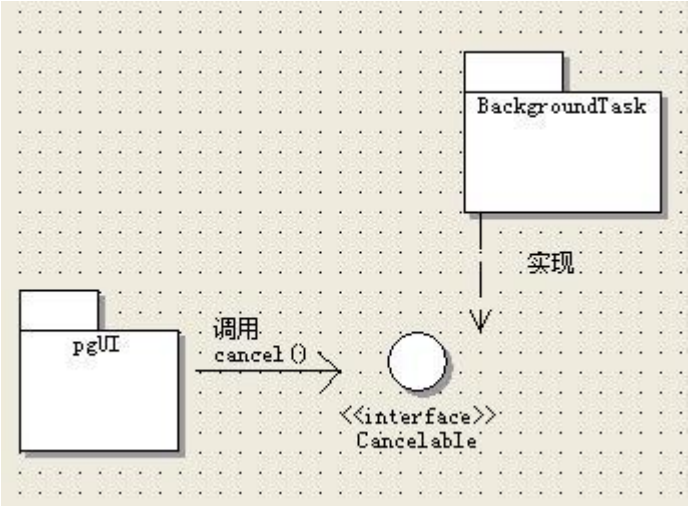
```
public interface Cancelable {
```

```
/**
* 本方法非阻塞，应该立即返回（如有必要开启新的线程）
* 此外应避免对此方法的重复调用
*/
public void cancel();
}
```

接下来在 ProgressObserver 加入对这个方法的支持

```
public interface ProgressObserver {
......
......
/**
* 设置取消 Task 时回调的函数对象
* @param co
*/
public void setCancelalbeObject(Cancelable co);
}
```

这样，就可以在用户按下取消按钮的时候，就可以进行对 Cancelable.cancel()的回调。这样灵活性大大增强了。



用 J2ME 实现简单电子邮件发送功能

在 GCF 中并没有提供给我们能够发送电子邮件的 API，J2ME 的可选包也没有提供相关的功能。那么我们能用 J2ME 实现发送电子邮件功能嘛？答案是肯定的。本文将主要讲述如何在 J2ME 中实现发送电子邮件的功能。

这里一个非常重要的思想就是代理。我们知道 GCF 提供给我们进行联网的能力了，比如通过 Http 联网。在 MIDP2.0 中甚至提供了 socket 联网的 API。那么我们可以通过他们连接服务器端的程序比如 servlet，然后 servlet 可以通过 JavaMail 提供的接口发送邮件。那么我们需要做的只是通过 Http 协议或者其他协议把邮件的标题、内容、收件人等发送给 servlet。就是这个简单的思想却是非常灵活非常有用。

首先我们构造一个 Message 类来代表发送的消息。它包括主题、收件人和内容三个字段。

```
package com.j2medev.mail;
```


```
public class Message
{

    private String to;
```

```
private String subject;

private String content;

public Message()
{

}

public Message(String to, String subject, String content)
{
    this.to = to;
    this.subject = subject;
    this.content = content;
}


public String getContent()
{
    return content;
}


public void setContent(String content)
{
    this.content = content;
}
public String getSubject()
{
    return subject;
}


public void setSubject(String subject)
{
    this.subject = subject;
}


public String getTo()
{
    return to;
}
public void setTo(String to)
```

```
{
        this.to = to;
    }

    public String toString()
    {
        return to+subject+content;
    }
}
```

在用户界面的设计上，我们需要两个界面。一个让用户输入收件人和主题，另一个用于收集用户输入的内容。由于 TextBox 要独占一个屏幕的，因此我们不能把他们放在一起。

```
/*
 * Created on 2004-12-8
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.j2medev.mail;

import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;

/**
 * @author P2800
 *
 * TODO To change the template for this generated type comment go to Window -
 * Preferences - Java - Code Style - Code Templates
 */
public class MainForm extends Form implements CommandListener
{

    private MailClient midlet;

    private TextField toField;

    private TextField subField;

    private boolean first = true;
```

```
    public static final Command nextCommand = new Command("NEXT", Command.OK, 1);

    public MainForm(MailClient midlet, String arg0)
    {
        super(arg0);
        this.midlet = midlet;
        if(first)
        {
            first = false;
            init();
        }

    }

    public void init()
    {
        toField = new TextField("To:", null, 25, TextField.ANY);
        subField = new TextField("Subject:", null, 30, TextField.ANY);
        this.append(toField);
        this.append(subField);
        this.addCommand(nextCommand);
        this.setCommandListener(this);

    }

    public void commandAction(Command cmd,Displayable disp)
    {
        if(cmd == nextCommand)
        {
            String to = toField.getString();
            String subject = subField.getString();
            if(to == "" && subject == "")
            {
                midlet.displayAlert("Null to or sub",AlertType.WARNING,this);
            }
            else
            {
                midlet.getMessage().setTo(to);
                midlet.getMessage().setSubject(subject);
                midlet.getDisplay().setCurrent(midlet.getContentForm());
            }
        }
    }

}
```

```
package com.j2medev.mail;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.TextBox;
import javax.microedition.midlet.MIDlet;


public class ContentForm extends TextBox implements CommandListener
{
    private MailClient midlet;

    private boolean first = true;

    public static final Command sendCommand = new Command("SEND", Command.ITEM,
            1);

    public ContentForm(String arg0, String arg1, int arg2, int arg3,
            MailClient midlet)
    {
        super(arg0, arg1, arg2, arg3);
        this.midlet = midlet;
        if (first)
        {
            first = false;
            init();
        }

    }

    public void init()
    {
        this.addCommand(sendCommand);
        this.setCommandListener(this);
    }

    public void commandAction(Command cmd, Displayable disp)
    {
        if (cmd == sendCommand)
        {
            String content = this.getString();
            midlet.getMessage().setContent(content);
            System.out.println(midlet.getMessage());
```

```
            try
            {
                synchronized (midlet)
                {
                    midlet.notify();
                }

            } catch (Exception e)
            {
            }

        }
    }
}
```

最后我们完成 MIDlet,在其中包括联网的程序代码,由于本站已经提供了很多关于 J2ME 联网的介绍，因此这里不再进行更多的解释。

```
package com.j2medev.mail;

import java.io.DataOutputStream;
import java.io.IOException;

import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;


public class MailClient extends MIDlet
{
    private MainForm mainForm;

    private ContentForm contentForm;

    private Display display;

    private Message message;

    public Message getMessage()
    {
        return message;
    }
```

```
public void setMessage(Message message)
{
    this.message = message;
}

public void displayAlert(String text, AlertType type, Displayable disp)
{
    Alert alert = new Alert("Application Error");
    alert.setString(text);
    alert.setType(type);
    alert.setTimeout(2000);
    display.setCurrent(alert, disp);

}


public ContentForm getContentForm()
{
    return contentForm;
}

public Display getDisplay()
{
    return display;
}


public MainForm getMainForm()
{
    return mainForm;
}

public void initMIDlet()
{
    MailThread t = new MailThread(this);
    t.start();
    message = new Message();
    display = Display.getDisplay(this);
    mainForm = new MainForm(this, "Simple Mail Client");
    contentForm = new ContentForm("Content", null, 150, TextField.ANY, this);
    display.setCurrent(mainForm);
}


protected void startApp() throws MIDletStateChangeException
```

```
    {

        initMIDlet();

    }

    protected void pauseApp()
    {


    }

    protected void destroyApp(boolean arg0) throws MIDletStateChangeException
    {


    }

}

class MailThread extends Thread
{
    private MailClient midlet;

    public MailThread(MailClient midlet)
    {
        this.midlet = midlet;
    }

public void run()
    {
        synchronized(midlet)
        {
            try
            {
                midlet.wait();
            }
            catch(Exception e)
            {
                e.printStackTrace();
            }

        }
        System.out.println("connecting to server.....");
        HttpConnection httpConn = null;
```

```
        DataOutputStream dos = null;

        try
        {
            httpConn =
(HttpConnection)Connector.open("http://localhost:8088/mail/maildo");
            httpConn.setRequestMethod("POST");
            dos = new DataOutputStream(httpConn.openOutputStream());
            dos.writeUTF(midlet.getMessage().getTo());
            dos.writeUTF(midlet.getMessage().getSubject());
            dos.writeUTF(midlet.getMessage().getContent());
            dos.close();
            httpConn.close();
            System.out.println("end of sending mail");
        }
        catch(IOException e)
        {}
    }}
```

在服务器端，我们要完成自己的 servlet。他的任务比较简单就是接受客户端的数据然后通过 JavaMail 发送到指定的收件人那里。如果您对 JavaMail 还不熟悉请参考如下文章。这里直接给出 servlet 代码。

使用 JavaMail 实现收发电子邮件功能

使用 Servlet 发送电子邮件

```
package com.j2medev.servletmail;

import java.io.DataInputStream;
import java.io.IOException;
import java.util.Properties;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.mail.*;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import java.util.*;
import java.net.*;
```

```
public class MailServlet extends HttpServlet
{
    private static String host;

    private static String from;

    public void init(ServletConfig config) throws ServletException
    {
        super.init(config);
        host = config.getInitParameter("host");
        from = config.getInitParameter("from");
        System.out.println(host + from);
    }

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException
    {

        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException
    {

        DataInputStream dis = new DataInputStream(request.getInputStream());
        String send = dis.readUTF();
        String subject = dis.readUTF();
        String content = dis.readUTF();
        try
        {
            Properties props = System.getProperties();
            // Setup mail server
            props.put("mail.smtp.host", host);
            // Get session
            Session session = Session.getDefaultInstance(props, null);
            // Define message
            MimeMessage message = new MimeMessage(session);
            // Set the from address
            message.setFrom(new InternetAddress(from));
            // Set the to address
            message.addRecipient(Message.RecipientType.TO, new InternetAddress(
                    send));
            // Set the subject
            message.setSubject(subject);
```

```
            // Set the content
            message.setText(content);
            // Send message
            Transport.send(message);
        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

web.xml

```
<?xml version="1.0" ?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
 <servlet>
  <servlet-name>ServletMail</servlet-name>
  <servlet-class>com.j2medev.servletmail.MailServlet</servlet-class>
  <init-param>
   <param-name>host</param-name>
   <param-value>smtp.263.net</param-value>
     </init-param>
     <init-param>
      <param-name>from</param-name>
      <param-value>eric.zhan@263.net</param-value>
     </init-param>
 </servlet>

 <servlet-mapping>
  <servlet-name>ServletMail</servlet-name>
  <url-pattern>/maildo</url-pattern>
 </servlet-mapping>
 <welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
 </welcome-file-list>

 <error-page>
  <error-code>404</error-code>
  <location>/error.jsp</location>
 </error-page>

</web-app>
```

总结：本文更多要介绍的其实还是这个代理的思想，利用所学的知识灵活应用。不要局限于 J2ME 提供给你的 API。文章中实现的客户端服务器都比较简单，也不够友好，如果感兴趣可以稍微修饰一下，对提高能力还是有帮助的。在 MIDP 中只是提供了 RMS 作为持久存储用，因此实现接受存储邮件不是很方便。如果手机支持 FileConnection 的话可以编写一个接受邮件的客户端。

游戏栏目

通过游戏代码学 J2ME

作者: npc7776

通过一款游戏的代码分析，从中学习掌握 MIDP 的开发知识。
以一个简单的射击类游戏为说明，代码中较多的涉及控制操作，显示操作的内容。
这个游戏由 7 个类组成，其中可以分为
主类 : planemain.java
游戏动作类: cortrol.java
游戏属性类: plane.java bullet.java
游戏辅助类:about.java cover.java backdrop.java
首先来介绍主类的代码
planemain.java

```
package npc7776; // 包名

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;//引入相关的 J2ME 包

/*******************************************
* 类功能介绍：主要屏幕用于控制现实启动封面及启动游戏
*******************************************/

public class planemain
    extends MIDlet
    implements CommandListener { // 注释一
  private Display display = null;
  private List mainList = null;
  private Command cmdQuit, cmdOk;//注释二

  public planemain() {
    String option[] = {
        "继续", "新开一局", "最高分", "帮助", "关于"};
    mainList = new List("选项", List.IMPLICIT, option, null);
    display = Display.getDisplay(this);
```

```
  cmdQuit = new Command("退出", Command.EXIT, 1);
  cmdOk = new Command("选择", Command.OK, 2);
  mainList.addCommand(CmdQuit);
  mainList.addCommand(CmdOk);
  mainList.setCommandListener(this);
}                              //注释三

public void startApp() {
  display.setCurrent(new cover(this)); //注释四
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional) {
  mainList = null;
  display = null;
}

/***********************************************
 * 功能介绍: 返回到主界面
 * 输入参数:
 * 返回参数:
 ***********************************************/

public void goBack() {
  display.setCurrent(mainList); // 显示主屏幕
}                                    //注释五

public void commandAction(Command c, Displayable d) {//注释六
  if (c == cmdOk) {
    tch (mainList.getSelectedIndex()) {
      case 0:
        break;
      case 1:
        display.setCurrent(new cortrol(this));
        break;
      case 2:
        break;
      case 3:
        break;
      case 4:
        display.setCurrent(new about(this));
        break;
```

```
      }
    }
    if (c == cmdQuit) {
      destroyApp(true);
      notifyDestroyed();
    }
  }                                  //注释七

}
```

注释:

一 . 通过代码:public class planemain extends MIDlet implements CommandListener 定义一个名为 planemain 的类,这个类通过 MIDlet

派生,MIDlet 是一个虚类,从这个类派生时需要重载 startApp,pauseApp,destroyApp 这 3 个方法,他们是在 MIDlet 状态发生改变时会被调用。

1.当程序第一次运行或是恢复运行时，startApp 方法会被调用。

2.当程序暂停执行时，pasuseApp 方法会被调用。

3.当程序退出时，destroyApp 方法会被调用。

同时 planemain 类实现了 CommandListener 接口,实现这个接口是为了让 planemain 对象能够处理来自窗口的菜单命令。

二 .private Display display = null; private List mainList = null; private Command cmdQuit, cmdOk;这里我们定义了显示类，List

类和俩个事件按钮。

三 .public planemain(){} 在构造方法中，定义了 List 加载的内容,设定当前窗体,并定义监听事件。mainList.addCommand(CmdQuit) 的作用

是把命令添加到 mainList 中去，然后调用 mainList.setCommandListener(this)将命令监听器设置在 planemain 对象上。

四 . 显示封面俩秒。startApp 这里我们在窗体开始后，设定窗体显示。注意我们将当前设定赋予 cover 类。在 cover 类中我们只现实封面特定

的时间(cover 类将在后面写到).

五 .goBack 类由 cover 类调用,在封面结束后,显示本页内容. 这里只做理解,等以后写到 cover 类就会明白.

六 .commandAction 方法是 CommandListener 接口所要求实现的方法,用于菜单命令处理.

七 .整个 commandAction 方法主要是监听控制按钮动作.在选取项目后点击确认分别作不

同的处理.当选取继续时可以向其他页调用转回,选取新

开游戏时将当前页面交给 cortrol 类.当选择关于时,将当前页面控制权交给 about 类.最后当我们点下的是退出按钮时,则退出当前游戏,此时程序

将会调用 destroyApp()方法.mainList.getSelectedIndex()意思是得到被选中的选项的位置索引.

辅助类 cover 类

```
package npc7776;

import javax.microedition.lcdui.*;
import java.util.*;

/**************************************
* 类功能介绍：启动显示封面,由 main 类调用,封面显示俩秒
**************************************/

public class cover
    extends Canvas {
  private planemain mainScreen = null;
  private Timer timer = null;
  Image img = null;
  public cover(planemain m) {
    mainScreen = m;
    timer = new Timer();
    TimerTask tsk = new TimerTask() {
      public void run() {
        mainScreen.goBack();//返回主窗口
      }
    };
    timer.schedule(tsk,2000);//注释一
  }
  public void paint(Graphics g){
    int width = getWidth();
    int height = getHeight();
    g.setColor(255,255,255);
    g.fillRect(0,0,width,height);//清屏
    try{
        img = Image.createImage("/res/cover.png");
    }catch (Exception e){}
    g.drawImage(img,width/2,height/2,Graphics.HCENTER|Graphics.VCENTER);

  }                          //注释二
```

}

注释：

一　.在构造类中有一个内嵌类 TimerTask,他完成的 run()方法，当执行时会调用主体类的 goBack()方法,而 timer.schedule(tsk,2000)方法执行结果正是在 2000 毫秒后执行 TimerTask
类的对象 tsk 的 run()方法。关于 J2ME 中的定时器这里再详细的介绍一下,在 J2ME 中，定时器用于
在指定时间执行任务或者重复执行任务,J2ME 中自 MIDPv1.0 开始就提供了对定时器的支持。要
使用定时器就要需要涉及到 TIMER(定时器)类和 TimerTask(定时器任务)类。TimerTask 类是一个
虚类,用于实现一个具体的任务。Timer 类用于通过定期执行由 TimerTask 对象表示的任务。要
使用定时器功能，就必须从 TimerTask 类派生新类,并且重载 void run()方法,然后通过 Timer 对
象来定时执行任务。

二　.我们可以看到 cover 继承了 Canvas,那么当它构造完成后会主动执行本方法。他的主要目
的是实现显示一张图到显示区。这里介绍一下图像.创建 Image 对象不是通过对象的构造方法，
而是要通过调用 Image 类的 createImage 方法来实现的,如果图像是通过资源内的图像文件创建,
那么图像文件格式必须为 PNG 格式，因为在 MIDP 中只支持 PNG 格式的图像资源（MIDPv2.0 中支持
透明的 PNG 图像）。一般把图像资源保存在工程下/res/下面。利用 Graphics 输出图像时需要调
用的方法是 void drawImage(Image img,int x,int y,int anchor),此外,在 MIDPv2.0 中 Graphics
类也添加了新的方法用于旋转图像,即 void drawRegion(Image img,int x_src,int y_src,int width,
int height,int transform,int x_dest,int y_dest,int anchor),通过 drawRegion 方法可以在
不创建新的图像对象的情况下输出旋转的图像。参数 transform 表示旋转的方式,允许的取值为

Sprite.TRANS_NONE 不旋转

Sprite.TRANS_ROT90,Sprite.TRANS_ROT180,Sprite.TRANS_ROT270 顺时针旋转 90,180,270

Sprite.TRANS_MIRROR 沿水平线翻转

Sprite.TRANS_MIRROR_ROT90,Sprite.TRANS_MIRROR_ROT180,Sprite.TRANS_MIRROR

_ROT270

沿水平线翻转后再顺时针旋转 90 度,180 度，270 度

辅助类 backdrop 类

```
package npc7776;

import javax.microedition.lcdui.*;

import javax.microedition.lcdui.game.*;

/**************************************

* 类功能介绍:设置背景及清除屏幕

**************************************/

public class backdrop {

private static final int[][] map1 = {

{

0, 0, 0, 0, 0, 0, 0, 0}

, {

3, 0, 0, 0, 0, 0, 0, 0}

, {

6, 0, 0, 0, 0, 0, 0, 0}

, {

6, 0, 0, 0, 0, 0, 1, 2}

, {

6, 0, 0, 0, 0, 0, 4, 5}

, {

6, 0, 0, 0, 0, 0, 7, 8}

, {
```

```
6, 0, 0, 0, 0, 0, 0, 0}

, {

9, 0, 0, 0, 1, 2, 3, 0}

, {

0, 0, 0, 0, 4, 5, 6, 0}

, {

0, 0, 0, 0, 7, 8, 9, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

3, 0, 0, 0, 0, 0, 0, 0}

, {

6, 0, 0, 0, 0, 0, 0, 0}

, {

6, 0, 0, 0, 0, 0, 1, 2}

, {

6, 0, 0, 0, 0, 0, 4, 5}

, {

6, 0, 0, 0, 0, 0, 7, 8}

, {
```

```
6, 0, 0, 0, 0, 0, 0, 0, }

, {

9, 0, 0, 0, 1, 2, 3, 0}

, {

0, 0, 0, 0, 4, 5, 6, 0}

, {

0, 0, 0, 0, 7, 8, 9, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

3, 0, 0, 0, 0, 0, 0, 0}

, {

6, 0, 0, 0, 0, 0, 0, 0}

, {

6, 0, 0, 0, 0, 0, 1, 2}

, {

6, 0, 0, 0, 0, 0, 4, 5}

, {

6, 0, 0, 0, 0, 0, 7, 8}

, {
```

```
6, 0, 0, 0, 0, 0, 0, 0}

, {

9, 0, 0, 0, 0, 0, 0, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

3, 0, 0, 0, 0, 0, 0, 1}

};

private static final int[][] map2 = {

{

0, 0, 0, 0, 0, 0, 0, 0}

, {

3, 0, 0, 0, 0, 0, 0, 0}

, {

6, 0, 0, 0, 0, 0, 0, 0}

, {

6, 0, 0, 0, 0, 0, 1, 2}

, {

6, 0, 0, 0, 0, 0, 4, 5}

, {
```

```
6, 0, 0, 0, 0, 0, 7, 8}

, {

6, 0, 0, 0, 0, 0, 0, 0}

, {

9, 0, 0, 0, 1, 2, 3, 0}

, {

0, 0, 0, 0, 4, 5, 6, 0}

, {

0, 0, 0, 0, 7, 8, 9, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

3, 0, 0, 0, 0, 0, 0, 0}

, {

6, 0, 0, 0, 0, 0, 0, 0}

, {

6, 0, 0, 0, 0, 0, 1, 2}

, {

6, 0, 0, 0, 0, 0, 4, 5}

, {
```

```
6, 0, 0, 0, 0, 0, 7, 8}

, {

6, 0, 0, 0, 0, 0, 0, 0}

, {

9, 0, 0, 0, 1, 2, 3, 0}

, {

0, 0, 0, 0, 4, 5, 6, 0}

, {

0, 0, 0, 0, 7, 8, 9, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

3, 0, 0, 0, 0, 0, 0, 0}

, {

6, 0, 0, 0, 0, 0, 0, 0}

, {

6, 0, 0, 0, 0, 0, 1, 2}

, {

6, 0, 0, 0, 0, 0, 4, 5}

, {
```

```
6, 0, 0, 0, 0, 0, 7, 8}

, {

6, 0, 0, 0, 0, 0, 0, 0}

, {

9, 0, 0, 0, 0, 0, 0, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

0, 0, 0, 0, 0, 0, 0, 0}

, {

3, 0, 0, 0, 0, 0, 0, 1}

};

private static final int TILE_WIDTH = 32; //图片宽

private static final int TILE_HIEGHT = 32; //图片高

private static final int TILE_NUM_COL = 8; //列

private static final int TILE_NUM_ROW = 36; //排

private int[][] currentMap; //地图

private TiledLayer dixing; // 地形

private int groundColor; // 背景颜色

private int screenHeight; // 计算 Y 轴

private int dixingScroll; //屏幕 Y 位置

public backdrop(int screenHeight) {
```

```
this.screenHeight = screenHeight;
try {
setMap(2);
}
catch (Exception e) {}
}
public void setMap(int LeveL) throws Exception {
Image tiledImages = null;
switch (LeveL) {
case 1:
tiledImages = Image.createImage("/res/terrain1.png");
currentMap = map1;
groundColor = 0xF8DDBE;
break;
case 2:
tiledImages = Image.createImage("/res/terrain2.png");
currentMap = map2;
groundColor = 0xF8DDBE;
break;
default:
tiledImages = Image.createImage("/res/terrain1.png");
currentMap = map1;
groundColor = 0xF8DDBE;
break;
```

```
}
dixing = new TiledLayer(TILE_NUM_COL, TILE_NUM_ROW, tiledImages, TILE_WIDTH,
TILE_HIEGHT);
for (int row = 0; row < TILE_NUM_ROW; row++) {
for (int col = 0; col < TILE_NUM_COL; col++) {
dixing.setCell(col, row, currentMap[row][col]);
}
}
// 注释一
/* dixingScroll = 1 - dixing.getCellHeight() * dixing.getRows() +
this.screenHeight;
dixing.setPosition(0, dixingScroll); */
dixing.setPosition(0, 0);
}
public void scrollTerrain(Graphics g) {
/*if (dixingScroll < 0) {
dixingScroll += 2;
dixing.setPosition(0, dixingScroll);
dixing.paint(g);
}
else if (dixingScroll == 0) {
dixing.setPosition(0, 0);
dixing.paint(g);
}*/
```

```
if (dixingScroll > (this.screenHeight-dixing.getCellHeight()*dixing.getRows()-1))
{

dixingScroll -= 2;

dixing.setPosition(0, dixingScroll);

dixing.paint(g);

}

else if (dixingScroll ==
(this.screenHeight-dixing.getCellHeight()*dixing.getRows()-1)) {

dixing.setPosition(0, dixingScroll);

dixing.paint(g);

}

}

public TiledLayer getTerrain() {

return dixing;

}

public int getGroundColor() {

return groundColor;

}

}
```

这里来介绍一下 Layer 类,一般来讲，游戏都会有背景，有的甚至用几层背景来表示画面的层次感。在游戏开发中可以通过作图完成对背景的处理，不过在 MIDP v2.0 中可以通过 Layer 对象轻松处理游戏背景、物体的移动和实现动画.图层是可以在游戏画布中单独绘制并能够移动的一些对象，一般来说，游戏都会有超过一个的图层。例如，背景图层用来绘制游戏的背景，场景图层用来显示当前的场景，物体图层用来显示当前场景中移动的物体。

在 MIDP 中 TiledLayer 类用来显示背景和场景，Sprite 类用来显示移动的物体，这两个类都是直接从 Layer 类中派生的。由于 Layer 是无法直接使用的，因此在处理背景时需要用到 TiledLayer （分块图层）类。TiledLayer 类能够按照单元的方式组织整个大的背景图层。在 TiledLayer 类中有两个概念：单元和图像分块。

1.单元是指将整个图层分为相同大小的区域，假设游戏屏幕尺寸为：130×60，每个单元的面积为 10×10，那么可以把屏幕分为 13×6 个单元。

2.图像分块是指将一幅大的图像按照单元大小分为多个块，对于屏幕的每个单元都可以用一个图像进行填充。

如果要创建显示一个图层需要以下步骤。

- 装入图像。
- 创建一个 TiledLayer 对象。
- 把图像装入 TiledLayer 对象，并设定图层的大小和每个分块的尺寸。TiledLayer 对象会根据图像的尺寸计算分块的数量，并且通过每个分块的尺寸和单元格的数 量计算图层的面积，实际这个过程是在创建 TiledLayer 对象时完成。
- 把图像分块按需要填入单元格，以后如果需要修改某个单元格的图像，则可以重新设置。
- 调用 TiledLayer 对象的 paint 方法进行绘图。
- 调用 Canvas 对象的 flushGraphics 方法刷新屏幕。

注释一：这里控制后面的地图移动方向,从上到下或者从下到上,一种算法。

辅助类 about 类

```
package npc7776;
import javax.microedition.lcdui.*;

public class about extends Form implements CommandListener{
  private Command cmdBack = null;
  private planemain mainScreen = null;
  public about(planemain m) {
    super("关于");
    mainScreen = m ;
```

```
    cmdBack = new Command("返回",Command.BACK,1);
    this.addCommand(cmdBack);
    this.setCommandListener(this);
    this.append("开发者");
    this.append("写上名字");
    }
    public void commandAction(Command c,Displayable d){
      if (c == cmdBack){
        mainScreen.goBack();
      }
    }
}
```

一个程序在完成发布时会说明此程序的作者信息，版本信息，版权信息等作者想自主的说明展示。这个类就是做这个事情的。它独立为一个窗口，并实现监听接口，能转回到发起页。要注意的是他继承了 Form 类。这里的 this.append("开发者")和 this.append("写上名字")添加一个文字标题到 Form 对象的最后面，返回被添加对象的位置。相当于调用 append (new StringItem(null,str)),只是简单的应用了 Form 类.

还有三个比较重要的类没有写,近期就可以完成。如果有什么写得不对的地方请大家提出，有什么不明白的地方也可以写出,我会尽力解答的,谢谢 mingjava 得意见，我再尽量把他们整理在一起。或者等我写完的时候，把他们放在 2--3 篇幅内。这里顺便把这个游戏需要的图片上传。再 JB 中打包的方式 Wizards --> Archive Builder---> 把图片加入--->工程窗口会出现一个 MIDlet suite,这样就可以运行了。有的时候 MIDlet Suite 是 0 bytes，点到 MIDlet Suite 右键 Make 即可。

```
package npc7776;

import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;

public class Plane {
  Image img;              //图片
  int x , y;              // 飞机所在位置的坐标
  int frameX, frameY;     //单个飞机图片的像素大小
  int scnX , scnY;        // 屏幕高宽
  Sprite plane;           //精灵类 plane
  int Move = 3;           //移动的距离
  boolean b = true;       // 一个 boolean 值
  public Plane(Image imgs,int fX,int fY,int sX,int sY) {
    scnX = sX;
```

```
    scnY = sY;
    frameX = fX;
    frameY = fY;
    img = imgs;
    x = y = sX /2 ;  // 飞机开始的位置
    plane = new Sprite(img,frameX,frameY);// 构造 plane
  }
  // 画飞机
  public void drawSelf(Graphics g){

    if (b == true){
      plane.setPosition(x,y) ;
      plane.paint(g) ;
      b = false;
    }else
      plane.paint(g) ;

  }
//恢复飞机桢数
  public void hf(){
  plane.setFrame(0);}
//移动左方法
  public void moveLeft(){
    getXY();
    plane.setFrame(1) ;
    if (x - Move > 0){
      plane.move(-Move,0) ;

    }
  }
  public void moveRight(){
    getXY();
     plane.setFrame(2) ;
    if (x + Move + frameX < scnX){
      plane.move(Move,0) ;

    }
  }
  public void moveUp(){
    getXY();
     plane.setFrame(0) ;
    if (y - Move >0){
      plane.move(0,-Move) ;
```

```
    }
  }
  public void moveDown(){
    getXY();
     plane.setFrame(0) ;
    if (y+frameY+Move < scnY){
      plane.move(0,Move) ;

    }
  }
  //获得当前飞机 X,Y 点
  public void getXY(){
    x = plane.getX() ;
    y = plane.getY() ;
  }
}
```

这是个 Sprite 类。说到这个类首先先讲讲 Layer 类,在 MIDPv2.0 中可以通过 Layer (图层)对象轻松处理游戏背景，物体的移动和实现动画。图层是可以在游戏画布中单独绘制并能够移动的一些对象，一般来说，游戏都会有超过一个的图层。例如，背景图层用来绘制游戏的背景，场景图层用来显示当前的场景，物体图层用来显示当前场景中移动的物体。例如，用游戏的一个场景来举例，游戏的背景图是星空，占据整个屏幕；然后在上面覆盖建筑物图层，这一图层没有覆盖整个屏幕，而且是透明的，没有覆盖和透明的区域上就会显示出背景；然后最上面才是物体图层显示移动的主角和障碍物。在 MIDP 中 TiledLayer 类用来显示背景和场景，Sprite 类用来显示移动的物体，这两个类都是直接从 Layer 类中派生的。Sprite 对象主要用来解决物体在游戏中的动画、移动和检测物体间的碰撞。在使用 Sprite 类时，需要先了解下面几个概念：画面帧、动画序列、对准点。

1 画面帧是一段动画中的一幅图像，多幅连贯的帧连续播放就可以产生动画效果。

2 动画序列是指播放动画时播放各个帧的顺序。

3 对准点是对于一个 Sprite 对象所包含的图像内的一个坐标点，在旋转时可以这个点为中心，或者在移动对象时以这个点作为基准点。默认情况下，这个点位于图像的左上角，也就是图像中的坐标 (0 , 0)位置。

为了能够使用精灵，首先需要生成画面帧。我们这里用到的飞机 PNG 图片（三里有下载）,一共三个飞机在那个图片中，第一个飞机像素大小为 24*24,整个图片 24*72 的.把第一个飞机作为序号 0(第一帧序号从 0 开始，而不是从一开始),依此类推序号 1，2。其次画面帧是可以旋转的,类似于前面介绍的 Image 类一样,Sprite 类旋转可以支持 7 种旋转方式，这里就不过多介绍了，旋转的时候是以图片的中心进行旋转。还有个碰撞判断这里就先不做介绍，会在以后的游戏中介绍。

大家对 Sprite 类大概有个理解，现在在来说程序。
public Plane(Image imgs,int fX,int fY,int sX,int sY),这里传递了一个形参,再以后别的类中应用。
public void drawSelf(Graphics g){....}在这里的判断，是为了确定飞机刚刚显示在屏幕上的初始位置。
下面 hf,moveLift,moveRight,moveUp.moveDown 这些方法控制飞机地移动，这里要说一下 plane.setFrame(1)就是得到图片序号 1 就是当飞机往左移动的时候，显示图片中序号 1 在屏幕上。方法里的判断语句是为了让飞机不要飞出屏幕，算法还是很容易的。

```
package npc7776;
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;
import java.util.*;
/*************************************
* 类功能介绍：控制子弹，将所有子弹的状态放入
* 纪录集当需要刷新或更新时再取出
*************************************/
public class bullet {
  Sprite bullet;
  Image img;
  Vector bullets = new Vector();
  public bullet() {
    try{
      img = Image.createImage("/res/bullet.png") ;
    }catch (Exception e){}
    bullet = new Sprite(img,8,8);
  }
  //加入子弹
  public void addBullet(int x, int y,int width){
    Sprite b = new Sprite(img,8,8);
    b.setPosition(x- bullet.getWidth() /2 + width/2,y) ;
    bullets.addElement(b) ;
  }
```

```
  //移动和删除子弹
  public void move(){
    for (int i = 0; i< bullets.size() ; i++){
      Sprite bb = (Sprite)bullets.elementAt(i) ;
      bb.move(0,-3) ;
      if(bb.getY() < 0 ){
        bullets.removeElementAt(i) ;
      }
    }
  }
  //画子弹
  public void drawSelf(Graphics g){
    for(int i = 0; i< bullets.size() ;i++){
      Sprite bs = (Sprite)bullets.elementAt(i) ;
      bs.paint(g) ;
    }
  }
}
```

这里比前面的飞机类只不过多了一个 Vector 类。Vector 是使用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，都允许直接按序号索引元素,这个类我接触的不多,请朋友门多多指教

public void addBullet(int x, int y,int width){..}这里 int x, int y,是取的飞机的 x,y。

b.setPosition(x- bullet.getWidth() /2 + width/2,y)，设置子弹出现位置是从飞机中间的坐标。

move 的方法中,if(bb.getY() < 0)如果子弹超过屏幕自动消失。

游戏的主要控制类: cortrol.java

```
package npc7776;
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;
/*************************************
*类功能介绍:游戏的主要控制程序
*************************************/

public class cortrol

extends GameCanvas

implements CommandListener, Runnable {
```

```
Image imgs;

Graphics g;

Plane plane;

backdrop back;

bullet bullet;

Thread thread = null;

planemain mainScreen;

public cortrol(planemain mainScreen) {

super(true);

this.mainScreen= mainScreen;

g = this.getGraphics();

try {

jbInit();

}

catch (Exception e) {

e.printStackTrace();

}

plane = new Plane(imgs, 24, 24, this.getWidth(),this.getHeight());

bullet = new bullet();

back = new backdrop(this.getHeight());

thread = new Thread(this);

thread.start();

}

private void jbInit() throws Exception {
```

```
imgs = Image.createImage("/res/planes.png");
setCommandListener(this);
addCommand(new Command("Exit", Command.EXIT, 1));
}
public void commandAction(Command command, Displayable displayable) {
if (command.getCommandType() == Command.EXIT) {
mainScreen.goBack();
}
}
public void keyInput() {
plane.hf();
if ( (this.getKeyStates() & this.LEFT_PRESSED) != 0) {
plane.moveLeft();
}
if ( (this.getKeyStates() & this.RIGHT_PRESSED) != 0) {
plane.moveRight();
}
if ( (this.getKeyStates() & this.UP_PRESSED) != 0) {
plane.moveUp();
}
if ( (this.getKeyStates() & this.DOWN_PRESSED) != 0) {
plane.moveDown();
}
if ( (this.getKeyStates() & this.FIRE_PRESSED) != 0) {
```

```
bullet.addBullet(plane.x, plane.y, 24);
}
}
public void stop(){
thread = null;
}
public void clear() {
g.setColor(back.getGroundColor());
g.fillRect(0, 0, this.getWidth(), this.getHeight());
}
public void run() {
while (true) {
clear();
keyInput();
plane.drawSelf(g);
bullet.drawSelf(g);
bullet.move();
back.scrollTerrain(g);
this.flushGraphics();
try {
thread.sleep(100);
}
catch (Exception e) {}
}
```

```
}

}
```

结合以前写的这里大部分都可以看懂，主要来讲下线程和按键。

在 JAVA 中,线程主用于实现一些并发任务,如果要定义一个作为线程运行的类,就必须实现 Runnable 接口,重载 void run()方法,并在 run 方法内实现线程的运算,然后创建一个线程 Thread 对象,并启动线程.J2ME 的线程 Thread 类是 JDK 线程类的一个精简版本，定义了线程控制方法,用于实现对线程的控制.上面例子中

thread = new Thread(this)	是给 cortrol 对象创建一个线程 Thread 对象,
thread.start()	就是启动线程
thread.sleep(100)	这是在 run()里的线程休眠,括号内的 100 为参数,表示的休眠时间，以毫秒计算,当线程强行中断的时候会抛出异常.

设备上的按键在逻辑上被分为俩类:标准按键 keyCode 和游戏按键 gameAction,这些键值都作为 Canvas 的静态常量进行定义.这里介绍下游戏按键.

游戏按键由以下值组成:

UP DOWN LEFT RIGHT : 方向按键.

FIRE GAME_A GAME_B GAME_C GAME_D : 开火键和 4 个游戏按键.

标准按键是物理上的定义,因为标准按键在不同的设备上都是固定的.而游戏按键是逻辑上的定义,由设备自行定义.比如:某些手机的数字 5,0,7,9 会被映射到 4 个方向,对于某些有方向键盘的设备,其中方向键盘就是会被映射到 4 个方向.上面这些是 Canvas,下面介绍在 MIDP v2.0 里的 GameCanvas 中定义的处理键盘事件,在 GameCanvas 中定义了静态常量表示每个键是否被按下,具有下面这些值:

UP_PRESSED DOWN_PRESSED LEFT_PRESSED RIGHT_PRESSED:上,下,左,右键

GAME_A_PRESSED GAME_B_PRESSED GAME_C_PRESSED GAME_D_PRESSED:A,B,C,D 游戏键

如果要检查特定的按键是否被按下,就需要检查将 getKeyStates()返回的值与这些键值进行按位与(&)并根据计算结果来判断.注意:为了程序能在不同的设备上运行,就一定要考虑不同设备间键盘布局的特点,在开发程序的时不要假设按键的布局方式.。

这个游戏只是用于学习目的，有很多地方可以进一步改进，大家可以对以上代码进一步

的优化和补充，使其更好玩。

如果以上内容有什么不对的地方,请 email :npc_gp@hotmail.com,大家一起探讨研究.

图片下载

J2ME Game 开发笔记（二）

作者：happyfire　　文章录入：蜡笔小刀

整理近期的一些 blog，包括 NokiaS60,Motorola,Eclipse 的一些内容

Nokia S60 的几个问题

1．不能每帧调用 System.gc(),否则严重降低 fps
2．Nokia S60 机器的不同机型对于 translate 和 setClip 的处理不一样。在 Nokia N-Gage QD 等机型中，setClip 是相对于 translate 以后的坐标计算的，而在 Nokia 6600,6670 等机型中，setClip 不受 translate 的影响，永远只相对于屏幕左上角(0,0)点计算。所以如果在 Nokia6670 中，使用先 translate 再 setClip 的方法画子图，则会出现错误。为了统一代码，在 Nokia S60 中不要使用 translate,即使用，两次 translate 之间不要进行 setClip.
修改后的画子图函数为：

```
public static void drawSubImg(Graphics g,Image img,int x,int y,int sx,int sy,int swidth,int sheight)
 {
  g.setClip(x,y,swidth,sheight);
  g.drawImage(img,x-sx,y-sy,GLT) ;
  g.setClip(0,0,width,height) ;
 }
```

3．部分 Nokia 机型（6600，6670 等）退出后报错 null pointer exception 的解决方法
不要在在主 while 循环中调用 destroyApp,而改成检测一个标志，退出主循环后再调用 destroyApp

```
boolean exit ;
...
while(!exit){
...
  if(...){
    exit = true ;
  }
...
}

destroyApp(true);
```

注：可在 destroyApp 内部调用 notifyDestroyed

Nokia"不能运行应用程序"错误新解

Nokia 手机运行 J2ME 程序的时候出现“ 不能运行应用程序 ”的错误，一般都是内存不足引起的，但今天遇到这样的错误，却发现是另一个原因。即当使用 nokia 的 UI API，DirectGraphics 的 drawImage 时，如果旋转参数设置不当，也会出现“ 不能运行应用程序 ”的错误。

Eclipse 集成 Motorola 模拟器

在 Eclipse 的菜单/工具条中选择 Run->External Tools,打开面板后，选择 program,然后 new 一个新的配置
1 在 Location 中填入 Moto 模拟器的路径，如：C:\Program Files\Motorola\SDK v4.2 for J2ME\EmulatorA.1\bin\emujava.exe，Moto 的不同模拟器支持 n 种不同机型，需要看 moto sdk 的文档才知道。
2 在 Arguments 里填入执行的参数，包括 jad 路径，模拟器使用的机型。如：
"${project_loc}\deployed\${project_name}.jad" -deviceFile Resources\V600.props
我是让模拟器执行 deployed 里面的 jad/jar,${project_loc}是工程路径,${project_name}是工程名。这里选择的机型是 V600.

说明：这种方法的局限在于只能执行 jar,所以每次运行前必须打包。实际使用前需要为没种机型配置一个 run,由于使用了通配参数，所以所有的工程都可以使用一个配置

Motorola 手机 J2ME 应用问题

1 应用程序图标
必须在 jad 文件 Midlet-Icon 属性中指定图标文件，Midlet-1 中指定的图标无效
Moto V 系列图标大小应为 15*15,其他尺寸无法显示。

2 左右软键问题
Motorola 手机操作系统设定是：右软键确认，左软键取消。所以，我们的程序应该和这个习惯保持一致。

3 Key Code
Moto V 的 key code 不同于其他 Midp2.0 机器
左软键：21
右软键: 22
中键: 20
up: 1
down: 6
left: 2
right: 5

MIDP2.0 Canvas 全屏问题

MIDP2.0 Canvas 可以调用 setFullScreenMode(true)将 Canvas 设置成全屏，但设置成全屏后新的 Canvas width & height 的获得对于不同手机却并不一样。

1 MotoV 系列

调用 setFullScreenMode(true)后，将触发 sizeChanged 事件，此事件从系统接受两个参数，即为 Canvas 全屏后的 width & height,通过这个事件可以获得新的宽高。

```
protected void sizeChanged(int w, int h)
{
    width = w ;
    height = h ;
}
```

但要注意，此事件并不是同步的，就是说如果你调用了 setFullScreenMode(true)之后，立即使用新的 width,height,有可能获得错误的结果。

2 SE K700

调用 setFullScreenMode(true)后，不会触发 sizeChanged,而是通过 getWidth 和 getHeight 获得新的宽高。SE 的 setFullScreenMode 调用后是立即返回的，所以可以获得正确的 width & height

－－－－－－－－－－

对于其他机型暂时还不了解

百宝箱应用编译打包事宜

1 编译时，设置 javac 的 target vm 为 1.1 即可通过移动检测。wtk 中无法实现。在 Eclipse 中可以在 java-compiler－Compliance and Classfiles 中做以下设置：

Compiler compliance level: 1.4
Generated .class files compatibility: 1.1
Source compatibility: 1.3

2 用 eclispe 打混淆包。但 eclipse 编写 jad 中文会出现乱码，所以用 wtk 编写正确的 jad,然后用 wtk 打包（注意不能覆盖 eclispe 打的包），这是为了用 wtk 获得正确的 jad 和 manifest 文件。将 elcipse 打包出的 jar 解压，用 wtk 生成的 mainifest 代替原 jar 中的 mainifest 文件，然后用 winrar 打包（zip 格式，可选最大压缩，注意要选择所有的文件后打包，不要将外面的整个目录打包）.最后将 jad 中的 jar size 改为这个最新的 jar 的字节数。

另：1. Nokia S60,SE k700 机器中显示的游戏名字为 MIDlet-1 中的名字，而 Nokia40 为 MIDlet-Name 中的名字

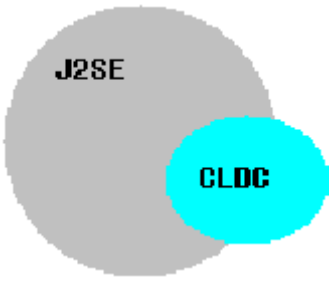
2. 根据 sp 提供的资料 Nokia 7650 游戏不能用中文名（其实 NGageQD 可以）

移动新技术栏目

企业管理软件在 J2ME 无线平台中的应用

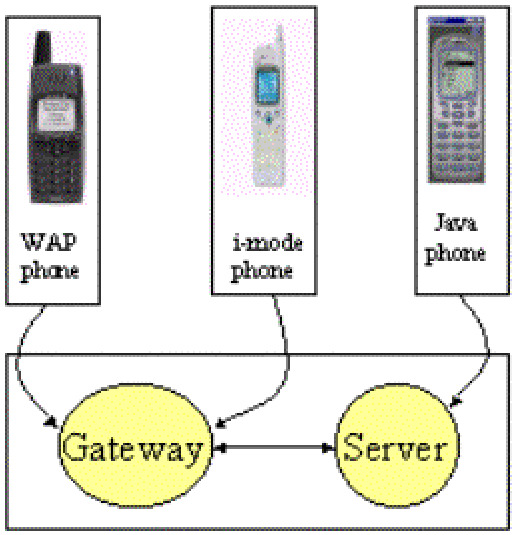
作者：刘金柱 来源：IBM developerWorks
本文主要叙述传统计算机管理软件与支持JAVA虚拟机的无线设备进行数据库交互的体系结构。随着无线开发平台软、硬件环境的成熟和完善，PC 软件系统与无线设备的数据交换将成为普遍流行的整体解决方案，给传统计算机管理软件带来一次新技术的革命。
延伸的无线设备
用户在当今，计算机管理系统广泛服务于全球各地的企业、组织中，为企业创造着巨大价值。系统终端用户已经被覆盖到企业 Intranet 或广域的 Internet 中，使用户更加容易简单的获得系统服务。这样就满足用户对获得系统服务的需求了吗？企业的采购员可能想在路途中知道当前仓库中的动态存货量，销售人员在和客户谈判时可能想立刻获知当前在库产品价格和数量以争取到一笔大的销售定单，你无法预知你的系统用户在什么时刻需要获得计算机服务，传统的管理软件约束了用户更多正常行为。无线技术应用延伸了客户端用户对空间位置的需求，使这些企业需求变为现实。
在无线设备编程中的 J2ME 体系
无线接入设备的可定制化编程能力被第三代 (3G) 和 GRPS 宽带无线网络所推进,从而可以在无线网络中传输视频信号和高质量的音频信号，而在这些基础网络硬件的推进下，无线设备已不象过去那样提供制造商所固有的编码程序，各种无线设备开发平台逐步形成，并成为业界生产和制造标准，这些设备将支持动态程序下载、自动安装、用户触发执行等功能。Java 语言最初是用户消费类型产品的编程，非常适合无线设备编程的特点，例如支持广泛的网络协议、安全性、平台交叉的兼容性、纯面向对象的开发过程等。

Java 技术经过多年的发展，已经是一种非常成熟的编程语言，但是使用传统的 Java 虚拟机和类库并不能够运行在这些资源相当受限的设备上（例如普通手机内存只支持几十到几百字节）。1999 年由 JCP(Java Community Process,JCP)公布了互联受限设备配置（Connected,Limited Device Configuration,CLDC），和移动设备描述（Mobile Information Device Profile,MIDP）。CLDC 为大量不同制造商、不同类型的互联资源受限设备的 Java 平台提供了"最低公共标准"的核心类库和 Java 语言、虚拟机等特征，并将定义的范围缩到最小，已能够适应广泛的无线设备，而 MIDP 则基于 CLDC 定义的平台之上，针对移动设备而扩展的一些 API。这些规范和通用框架被称为 Java 2 平台微缩版（Java 2 Platform,Micro Edition,J2ME）



但是 J2ME 并不能定义所有设备制造商、其他原始设备制造商所拥有的特性，列如 Motorola Java 手机(下面的无线设备就以 KJava 手机为例)可能与 PalmOS 提供不同的手机特性，为此各厂商提供自己的 OEM 类库来访问特定的功能。

J2ME 提供了开发无线应用的标准类库，作为这篇文章最关心的部分，J2ME 提供了通用的支持输入\输出的互联框架，使用这个框架允许无线设备具有网络通讯输入\输出的可编程能力，如下图所示：



调用格式：

```
Connector.open(":"
```

":"");

所有的连接都通过调用 javax.microedition.io.Connector 类中的方法 open 来创建，通过这个连接类，我们可以方便的实现无线设备与 PC 服务器进行 TCP/IP 的 socket 的通讯：

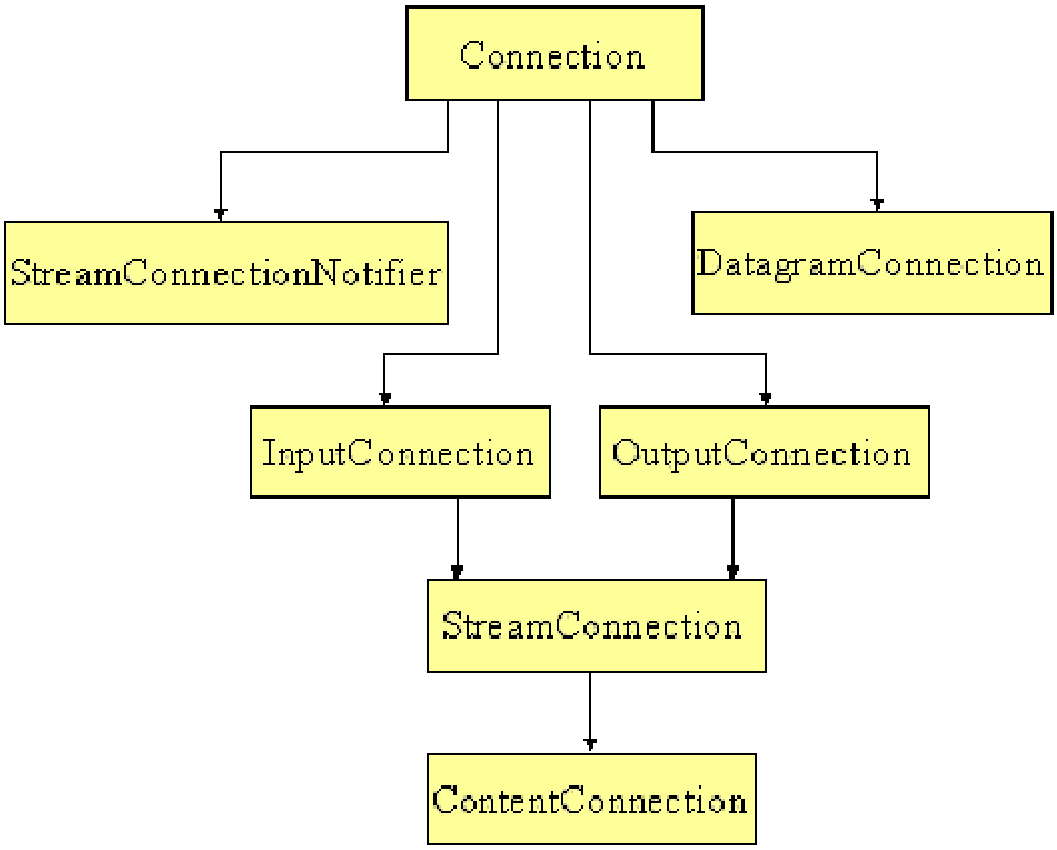
```
StreamConnection conn=(StreamConnection)

Connector.open("socket://www.cn-java.com:8080");

InputStream in = conn.openInputStream();

OutputStream out= conn.openOutputStream();
```

在企业实际案例中，应用更广泛的是透过防火墙来访问企业数据库的 Http 方式，我们也可以简单的使用下面的方式来实现无线设备与 PC 服务器进行 TCP/IP 的 http 通讯：

```
HttpConnection conn=(HttpConnection)
```

```
Connector.open("http://www.cn-java.com/index.php");

InputStream in=conn.openInputStream();

OutputStream out= conn.openOutputStream();
```

在 CLDC 定义的框架中，所有 Connection 都继承了实现了 Connection 接口，如下图所示：



在 MIDP 提供了一套标准的用户界控件面库和屏幕模型，屏幕模型是 MIDP 界面的核心概念，屏幕模型可以帮助开发者根据不同需求组装成不同的屏幕，有两种屏幕类型：Canvas 和 Screen。Canvas 可以允许开发者在屏幕上画置图形、响应用户输入的低级对象，功能灵活，但编程难度较大，而 Screen 则是一个封装了完整用户界面的高级对象（Alert、List、TextBox、Form）。一个完整的应用可以由两种类型的屏幕交互组成。

如果屏幕类被扩展了 Form 类型，则可以在 Form 中包含下面的 Item 类：
StringItem -显示字符串
ImageItem -显示图象
TextField -受限的文本输入
DateField -显示日期时间
Gauge -显示图形化的值
ChoiceGroup - 单选和多选框

一般情况一个简单的界面应用可以由 Screen 类型的屏幕完成，如果应用界面比较复杂，则需要使用 Canvas 方式将画面一一画出，并自己写程序负责事件的响应机制。

在原有旧系统上实现无线设备的数据交换

应用无线设备让企业管理系统更加高效和灵活,但是更多的计算机管理软件系统在当初设计时可能并未考虑过实现无线设备应用,在企业的软件投资过程中,投资应当受到合理的保护,给遗留系统添加无线设备访问能力成为一个值得研究的课题。

1．可行性分析

传统企业管理系统一般为 C/S、B/S 或混合而成,支持无线设备的首要条件是网络环境。KJava 手机或其他 Java 无线设备可以通过 Http 或 Socket 协议访问 Internet 资源，因此，企业数据库必须能够通过互联网进行访问，如果企业管理系统由于硬件环境不能接入 Internet，则与普通无线设备进行交互的可能性不大。在物理环境下，企业数据库必须能够与外界 internet 进行数据交换。

无线设备作为一种资源受限设备，相较 PC 程序而言只能完成比较简单的界面交互，内存和 CPU 处理能力的限制使不进行大规模数据计算和图形处理，屏幕和键盘输入设备也不是很方便，开发成本也较高。因此，需要分析哪些业务数据是有必要与无线设备进行数据通信。由于各种 Kjava 无线设备厂商生产的设备在除了支持 J2ME 标准外，一般都会增加一些 OEM 特性，以能够访问其设备特性，因此需要考虑到支持市面上哪类家族的无线设备，是 Nokia 系列还是 Moto 系列，也可以为不同家族的手机开发出不同 J2ME 程序（在 UI 控制方面有自己的开发包）。

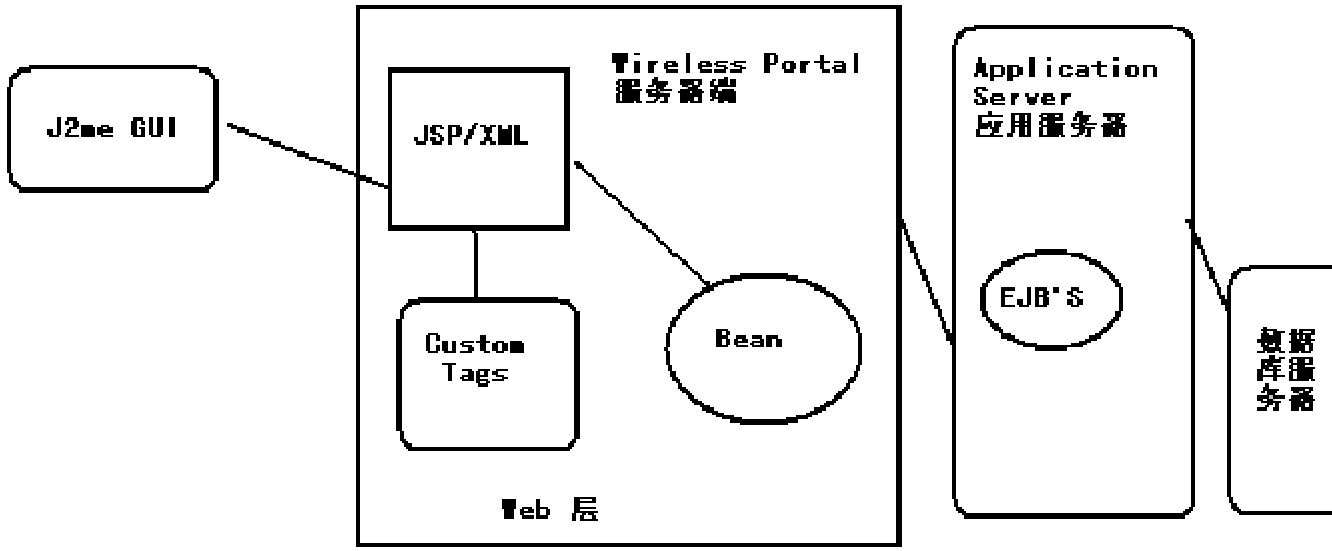
2．制定客户端与服务器端通讯接口标准

无线设备中的 Java 虚拟机只提供了设备的可编程能力，数据的 IO 传输等，因此，在开发服务器端系统和客户端（无线设备）系统前需要制订出一套完善的通讯接口。要制订这一个接口首先应该清楚的是当两者通讯时，我们一般会使用流的方式，既把一个请求数据或响应数据编码成一个长的流，流内包含的数据使用约定的分割符号进行界定，例如发送登录命令的请求流的编码顺序可能是这样的：
login jackliu 123456 解释：login 表示登录请求，一个空格后表示跟随一个用户名，另一个空格跟随口令，但也可能是这样的 01 _name[jackliu]name_ ps[123456]ps_ 解释：01 表示登录请求，_name[和]name_之间表示用户姓名，_ps[和]ps_之间表示登录口令，可以看出流的编码方式是多样的，只要能够让服务器端和客户端识别，遵循简单、通用的原则制定即可(也可以使用 xml 语言来描述,但是我觉得目前手机内存和 CPU 计算能力较弱,适合 J2ME 的 xml 解析器也不多)。

3．开发服务器端响应系统

无线设备不支持直接访问数据库的能力，而企业管理系统多数都由大型数据库支持，因此我们需要编写一套服务器端程序来接受无线设备的请求指令，通过解析请求、执行逻辑、响应结果的方式把数据再传回给无线设备。旧的原有系统很可能采用非 Java 语言开发，如果服务器端系统使用 Java 语言开发可能对原软件供应商存在技术障碍，同时原有系统逻辑不容

易与 Java 集成。在这种情况下，服务器端程序就不一定使用 Java 编写，由于是使用 Http 或 Socket 通讯,使服务器端程序可以由任何支持 Http 协议或 socket 协议的程序语言编写。



4．开发无线设备客户端

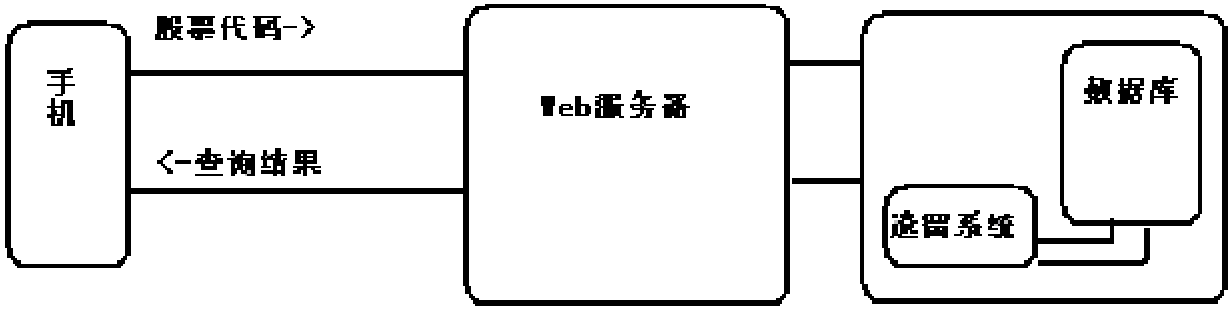
使用 J2ME 提供的 API 和 MIDP 类库开发无线设备的客户端系统，由于 MIDP 提供的标准界面类库往往不能完全满足各类资源受限设备的特性，可能会是使用到具体无线设备家族的 OEM 类，类如专门为 Motorola 开发时的 LWT 类库，一般来说使用了这些类库会影响 KJAVA 程序的可移植性，但由于 OEM 类是基于 CLDC 规范之上的类库，所以可以很容易的更换成其他厂商的 OEM 包以能够运行在其他的 Java 无线设备中。

对一个遗留系统进行改造的例子

通过一个例子来说明我们应该如何改造一个遗留的旧系统，以让他支持无线设备。我们来描述一下这个旧系统的基本功能和用户的新需求：

A 证券公司，几年前，开发了一套基于 C/S 结构的证券系统，数据库被共享在 Internet 上，公司可以通过证券系统查询各种股价信息和交易数量。现在，公司希望能够功过手机等类似无线设备时时能够查询到这些股票信息，并且要求保留原有的系统平台功能。

我们分析这套系统基于 C/S 结构，无法直接和手机使用 http 通信，需要单独为此开发一套基于 http 协议的后台服务程序，让手机通过 http 来调用这个服务程序，框架如下图：



这里仅实现一个按股票代码查询股价信息的功能来说明这个开发结构。

1．制订通讯协议标准

手机发出请求某支股票价格询问命令协议格式：

```
_command[007]command_ _uid[]uid_ _stockNo[]stockNo_
```

服务端响应协议格式：

```
_command[008]command_ _stockTitle []stockTitle_ _stocktPrice[]stockPrice _
```

2．开发服务端程序

假设服务器端我们使用 Servlet 开发，为了能够接收到请求，我们打算设计一个 MyWirlessServlet 来实现，基本代码如下：

```
import javax.servlet.*;

import javax.servlet.http.*;

import java.io.*;

import java.util.*;

import java.sql.*;

public class MyWirlessServlet extends HttpServlet{

public void init() throws ServletException{}

public void doGet(HttpServletRequest request,

HttpServletResponse response) throws ServletException, IOException{
```

```
//不在 doGet 种操作，手机与其通讯时要采用 doPost 方式，所以我们要在 doPost 写处理
}
public void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException{
DataInputStream in = new DataInputStream
(new BufferedInputStream(request.getInputStream()));
DataOutputStream out = new DataOutputStream
(new BufferedOutputStream(response.getOutputStream()));
String reqData=in.readLine(); //获取请求命令行
….//此处根据 reqData 来访问数据库，把结果编码成响应格式
String repData="_command[008]command_ _stockTitle
[A 股票]stockTitle_ _stockPrice[12.85]stockPrice _";
out.writeUTF(repData);//用 UTF8 格式编码发送给手机
out.flush();
out.close();
}
```

3．开发手机端程序

这段程序使用了标准的 J2ME 开发包

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.io.*;
public class HttpDemo extends MIDlet implements CommandListener{
```

```
private Form form;
private Command exitCommand=new Command("Exit",Command.EXIT,1);
private Command okCommand =new Command("OK",Command.OK,2);
private Display display;
public HttpDemo() {
display=Display.getDisplay(this);
}
protected void pauseApp() {
form=null;
display.setCurrent(null);
}
protected void startApp() {
form=new Form("股价查询：");
form.addCommand(exitCommand);
form.addCommand(okCommand);
form.setCommandListener(this);
display.setCurrent(form);
StringBuffer sb=new StringBuffer();
String url="http://www.cn-java.com:8080/ MyWirlessServlet ";
HttpConnection conn =null;
DataInputStream in=null;
OutputStream out=null;
try{
```

```
conn =(HttpConnection)Connector.open(url);

conn.setRequestMethod( HttpConnection.POST ); //设置连接为 Post 方式

conn.setRequestProperty( "User-Agent", "Profile/MIDP-1.0
Configuration/CLDC-1.0" );

conn.setRequestProperty( "Content-Language", "en-US" );

out= conn.openDataOutputStream();

//发送请求给出股票代号是 999 的股票价格信息

out.write("_command[007]command_ _uid[jackliu]uid_ _stockNo[999]stockNo_";

out.flush();

out.close();

in= conn.openDataInputStream();

//读返回得结果，这里我们没有做解析处理

sb.append(is.readUTF());

}catch (Exception e){

sb.append(e.toString());

}finally{

try{

if(in!=null)in.close();

if(out!=null)out.close();

if(conn!=null)conn.close();

}catch(java.io.IOException ioe){}

}

//将股票价格信息显示到手机屏幕上

form.append(sb.toString());
```

```
}

protected void destroyApp(boolean parm1) {}

public void commandAction(Command c,Displayable s){

if(c==exitCommand){

destroyApp(false);

notifyDestroyed();

}

}

}
```

模拟屏幕测试如下：

参考信息：

- [Programming Wireless Devices with the java 2 Platform,Micro Edition]--Roger Riggs
- sun 无线开发网址 http://wireless.java.sun.com/
- MIDP API DOC http://servlet.java.sun.com/logRedirect/wirelessindex-leftnav/http://java.sun.com/j2me/docs
- Wirless ToolKit http://servlet.java.sun.com/logRedirect/wirelessindex-leftnav/http://java.sun.com/products/j2mewtoolkit/
- MIDP Network Programming using HTTP http://wireless.java.sun.com/midp/articles/network/

关于作者

刘金柱,金蝶软件金融事业部高级软件工程师,您可以通过 E-mail:suntoday@eyou.com 与他取得联系。

移动通信设备中 J2ME 开发的现状和前景展望

作者：董向辉文章来源：IBM developerWorks
在消费电子和嵌入式设备的广阔领域中，目前最受关注的是移动通信设备，因此有必要介绍一下移动通信设备的主要操作系统平台，这对 J2ME 的开发是相当重要的。移动通信设备主要包括 PDA 和智能手机，现在和将来都还会有一部分设备处于 PDA 和智能手机交界的位置。其中手机的市场远比 PDA 要大得多，所以许多国外谈到 J2ME 的文章都是以 Wireless 应用为主。

PDA 也即掌上电脑，一般是指类似于 Palm 公司出品的 Palm 这样的设备，主要的操作系统有 Palm OS 和 Pocket C 两大阵营。Palm OS 来自 Palm 公司，是一个开放的系统，在 PDA 市场上占主导地位，已经有非常多的第三方厂商开发的应用和一大批非常忠实和狂热的用户。目前采用 Palm OS 的主要有 Palm 公司的 Palm 系列和 Sony 公司的 Clie 系列（Handspring 公司的 Visor 也属于这类产品，但是已经宣布退出传统 PDA 市场，主要发展 PDA 和无线通讯结合的产品 Treo）。

Pocket PC 是微软及其合作伙伴 Casio、Compaq、Hewlett Packard 和 Symbol 推出的，基于 Win CE 3.0，也是一个开放的标准系统，功能可以扩展（以往的 Win CE 是一个封闭的不可扩展的系统），在这个平台上厂商可以自己开发软件。具体产品如 Compaq 的 iPAQ。

另外，Sharp 公司 Zaurus 系列 PDA 采用 Linux 的一个针对嵌入式系统的版本。作为拥有高达 64M 内存的高端 PDA，可以满足 CDC 的标准，预装了 Personal Java 的虚拟机。

在智能手机领域，主要的平台是 Symbian 的 EPOC。EPOC 最早由 Psion 开发，主要面向智能手机，也有 PDA 的特征，Psion 就是第一批主要的 PDA 厂商之一。Psion 宣布 EPOC 的第一个版本是开放的 OS，并向其它厂商授权，随后 Psion 与 Ericsson，Nokia，及稍后的 Motorola 建立了名为"Symbian"的联盟,目前主要股东为 Motorola, Nokia, Panasonic, Psion 和 SonyEricsson，而且 Siemens4 月也宣布加入。Symbian 在欧洲有很好的基础，为大多数主要手机厂商所采用，典型的产品如 Norkia 的 9210。

在这些操作系统平台中，Palm OS 上已经有了 MIDP 的参考实现，但是并不是最适合 PDA 的 Profile。Symbian 实现了 CLDC 和 MIDP，作为智能手机的操作系统，是理想的 MIDP 应

用平台。在 Compaq 的 iPaq 上已经实现了 CDC（基于 Pocket PC），另外前面提到过，IBM 的 WebSphere Micro Environment 在 Pocket PC 上实现了 CLDC，CDC 和 MIDP。Sharp 的 Zaurus 实现了 Personal Java（基于 Linux）。由于这些主要操作系统平台都是开放的，基本可以相信，主要的移动通信设备操作系统都将一直提供 J2ME 的支持，但是具体支持哪个标准则并不一定死板地按照 CLDC 和 CDC 的最初定义，和设备的硬件条件和厂商的选择有关，或者可能同时实现多个标准。

关于移动信息设备上 J2ME 应用的具体开发，现有的技术基本分为以下几类：

采用早期的 KVM 和 com.sun.kjava 包，或者加上第三方的 kawt 类库。有很多早期的文章和代码，但是这一技术将被 PDA Profile 取代，而且不保证兼容性，不建议开发者采用这一方法，而是等待 PDA Profile 的最后完成及其实现。
采用 CLDC 和 MIDP 开发。由于 MIDP 标准发布较早，智能手机的市场也远比 PDA 大得多，这是目前大量的无线应用文章的主要内容，不过它最适合的目标是智能手机，虽然智能手机可能有一定的 PDA 的功能，但是这一技术并不是最适合 PDA 的。
采用 CDC 和 Personal Profile，这样的技术不太多见，目前的移动信息设备硬件能力还有所不足。但是下一代的 PDA 和智能手机肯定可以支持。

就现有的 PDA 硬件条件而言，J2ME 应用要和已有的应用竞争还有比较多的困难：
空间：现在一般的 Palm 类设备内存从 2M 到 16M 不等（目前较高档的机型都已支持扩展卡，可以扩展到 128M，但是卡上运行程序的速度要慢得多，一般主要用来放数据，运行程序不能依赖于卡），Palm 上一般现有程序的标准大小是几十 K 到 300K 以下。而仅 MIDP 的 Palm 参考实现库在 Palm 上就需要将近 600K，再加上作为基础的 CLDC 库，加上程序，就快要接近 1M 了。和已有的应用相比，目前是没有太大竞争力的。
速度：由于 Palm 一直信奉的是"简单就是美"的原则，目前的 Palm 系列 PDA 主要用的是 33MHz 的 Motorola 68000 系列 DragonBall，这虽然带来了成本的降低和节电性，但 J2ME 应用的性能很难让人满意了。
功能：目前最适合 Palm 的 PDA Profile 尚未完成，如果用 MIDP 开发则不太合适，用早期的 KVM 和 com.sun.kjava 则因为不是 J2ME 标准，程序兼容性无法保证。

虽然如此，但是在移动信息设备上开发 J2ME 应用的前景还是非常美好的：
硬件的发展：目前的 Palm 配 8M 到 16M 内存已经是主流，下一代超过 32M 是必然的趋势。Palm OS 5 终于开始支持 32 位的 ARM RISC 处理器，其速度应该能够提高 10 倍以上。事实上，Sharp 公司今年第一季度发售的 Zaurus SL-5500 已经采用了 Intel 206MHz StrongARM 处理器和 64M 内存，在这样的硬件上速度和内存都不会成为问题。
标准的完善：PDA Profile 今年内将最后完成，可以预计很快会在主要操作系统上得到实现。目前在 JCP 也有许多其他标准非常有意义，例如关于蓝牙技术（Bluetooth），游戏，定位等。
平台的广泛性：移动信息设备的主要操作系统平台都是开放的，J2ME 的标准也是开放的，因此基本上所有的主要操作系统平台都支持或将支持 J2ME，这带来的广阔市场（智能手机的市场尤其巨大，据称 Norkia 到今年就可以出货 5000 万只 Java 手机，而到 2003 年可达 1 亿只。）和真正的"一次编写，到处运行"是任何其他技术无法比拟的。对于开发者而言，可以不受操作系统的限制，不用担心在这个多变的市场上因平台的变化而影响自己的应用。

开发的方便性：J2ME 虽然和 J2SE 有着许多不同，但仍然属于 Java 技术，具有 Java 方便

开发的优点，也使得 Java 程序员学习移动信息设备开发没有太大困难。