第 4 章 Android应用的调试

本章将学习如何处理应用bug。同时也会学习如何使用LogCat、Android Lint以及Eclipse内置的代码调试器。

为练习应用调试，我们先刻意搞点破坏。打开QuizActivity.java文件，在`onCreate(Bundle)`方法中，注释掉获取`TextView`组件并赋值给`mQuestionTextView`变量的那行代码，如代码清单4-1所示。

代码清单4-1 注释掉一行关键代码（QuizActivity.java）

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
    setContentView(R.layout.activity_quiz);

    mQuestionTextView = (TextView)findViewById(R.id.question_text_view);
    //mQuestionTextView = (TextView)findViewById(R.id.question_text_view);

    mTrueButton = (Button)findViewById(R.id.true_button);
    mTrueButton.setOnClickListener(new View.OnClickListener() {
        ...
    });

    ...
}
```

运行GeoQuiz应用，看看会发生什么，如图4-1所示。

图4-1展示了应用崩溃后的消息提示画面。不同Android版本的消息提示可能略有不同，但本质上它们都是同一个意思。当然，这里我们知道应用为何崩溃。但假如不知道应用为何出现异常，下面将要介绍的DDMS透视图或许有助于问题的排查。

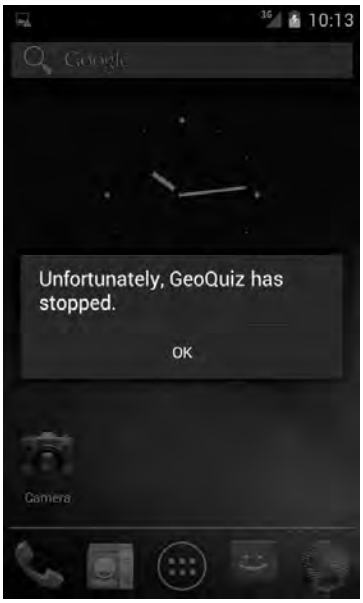


图4-1 GeoQuiz应用崩溃了

4.1 DDMS 应用调试透视图

在Eclipse中，选择Window → Open Perspective → DDMS菜单项打开DDMS透视图，如图4-2所示。

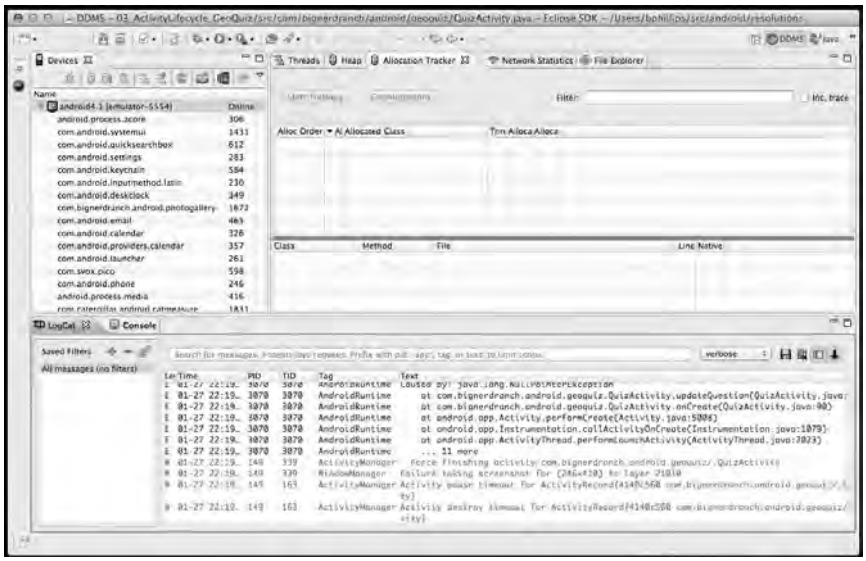


图4-2 DDMS透视图

透视图是Eclipse中预先定义的一组视图。应用调试或者代码编辑的时候，我们通常需要看到不同的视图组合。因此，Eclipse按照实际开发需要，将每组视图组合成了一幅透视图。

预定义的透视图并非不可改变。我们可以通过添加和移除视图来重新定制透视图，Eclispe会自动记住这些调整。如需重新开始调整，单击Window → Reset Perspective...菜单项返回透视图的初始状态即可。

代码编辑时使用的默认透视图叫Java透视图。当前打开的所有透视图都列在Eclipse工作区的右上角附近。点击对应透视图的按钮可实现透视图间的自由切换。

图4-2显示了DDMS透视图。DDMS（Dalvik Debug Monitor Service，调试监控服务工具）在后台处理Android应用调试所需的全部底层工作。DDMS透视图主要包含了LogCat以及Devices视图。

Devices视图用来显示连接至电脑的Android硬件和虚拟设备。设备相关的各种问题通常都可以在该视图中得到解决。

比如说，运行应用时在Devices视图中找不到自己的设备？很容易解决，单击视图右上角向下的小三角图标，然后在弹出的菜单中选择Reset adb选项。一般来说，重启adb就可以找回所用设备。或者LogCat输出了其他设备的日志信息？没问题，在视图中点选当前工作的设备，LogCat会切换并显示该设备的日志输出。

4.2　异常与栈跟踪

现在回头来看应用崩溃的问题。为方便查看异常或错误信息，可展开LogCat窗口。上下滑动滚动条，最终应该会看到整片红色的异常或错误信息。这就是标准的Android运行时的异常信息报告，如图4-3所示。

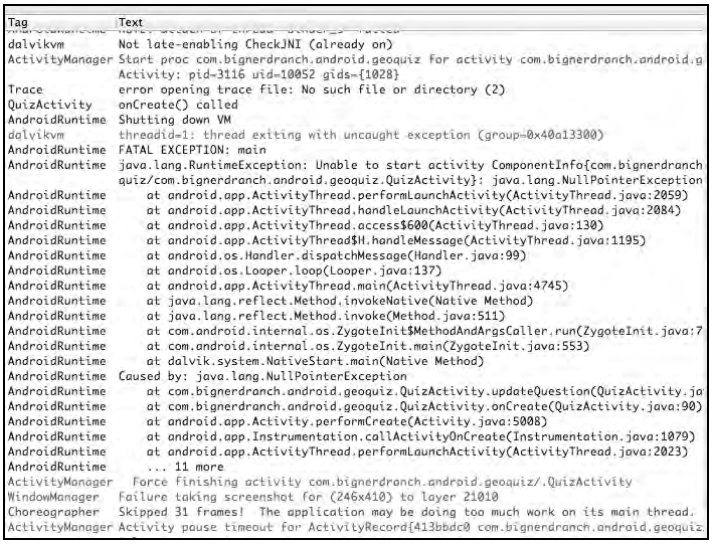


图4-3　LogCat中的异常与栈追踪

该异常报告首先告诉了我们最高层级的异常及其栈追踪，然后是导致该异常的异常及其栈追踪。如此不断追溯，直到找到一个没有原因的异常。

在大部分编写的代码中，最后一个没有原因的异常往往是我们要关注的目标。这里，没有原因的异常是`java.lang.NullPointerException`。紧接着该异常语句下面的一行就是其栈追踪信息的第一行。从该行可以看出发生异常的类和方法以及它所在的源文件及代码行号。双击该行，Eclipse自动跳转到源代码的对应行上。

Eclipse定位的这行代码是`mQuestionTextView`变量在`onCreate()`方法中的首次使用。`NullPointerException`名称的异常暗示了问题的所在，即变量没有进行初始化。

为修正该问题，取消对变量`mQuestionTextView`初始化语句的注释。

遇到运行异常时，记住在LogCat中寻找最后一个异常及其栈追踪的第一行（该行对应着源代码）。这里是问题发生的地方，也是寻找问题答案的最佳起始点。

如果发生应用崩溃的设备没有连接到电脑上，日志信息也不会全部丢失。设备会将最近的日志信息保存到log文件中。日志文件的内容长度及保留的时间取决于具体的设备，不过，获取十分钟之内产生的日志信息通常是有保证的。只要将设备连上电脑，打开Eclipse的DDMS透视图，在Devices视图里选择所用设备。LogCat将自动打开并显示日志文件保存的日志信息。

4.2.1 诊断应用异常

应用出错不一定总会导致应用崩溃。某些时候，应用只是出现了运行异常。例如，每次单击Next按钮时，应用都毫无反应。这就是一个非崩溃型的应用运行异常。

在QuizActivity.java中，修改`mNextButton`监听器代码，将`mCurrentIndex`变量递增的语句注释掉，如代码清单4-2所示。

代码清单4-2 注释掉一行关键代码(QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    ...

    mNextButton = (Button)findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length
            //mcurrentIndex = (mcurrentIndex + 1) % mQuestionBank.length
            updateQuestion();
        }
    });

    ...
}
```

运行GeoQuiz应用，点击Next按钮。可以看到，应用毫无响应，如图4-4所示。

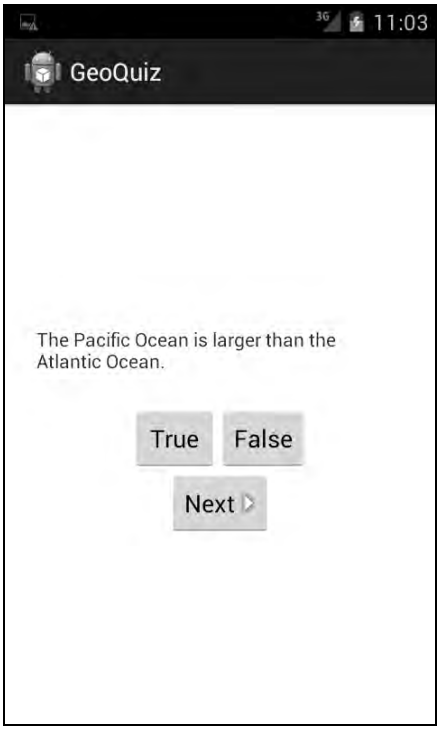


图4-4　应用不响应Next按钮的点击

这个问题要比上个更为棘手。它没有抛出异常，所以修正这个问题不像前面跟踪追溯并消除异常那么简单。有了解决上个问题的经验，这里可以推测出导致该问题的两种可能因素：

- `mCurrentIndex`变量值没有改变；
- `updateQuestion()`方法没有调用成功。

如不知道问题产生的原因，则需要设法跟踪并找出问题所在。在接下来的几小节里，我们将学习到两种跟踪问题的方法：

- 记录栈跟踪的诊断性日志；
- 利用调试器设置断点调试。

4.2.2　记录栈跟踪日志

在`QuizActivity`中，为`updateQuestion()`方法添加日志输出语句，如代码清单4-3所示。

代码清单4-3　方便实用的调试方式（QuizActivity.java）

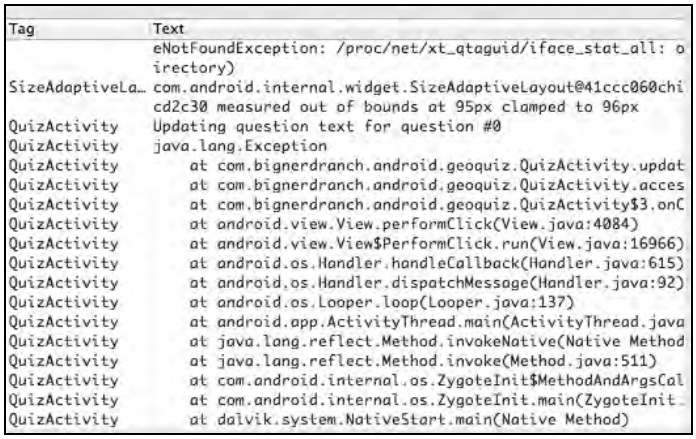
```
public class QuizActivity extends Activity {
    ...

    public void updateQuestion() {
```

```
        Log.d(TAG, "Updating question text for question #" + mCurrentIndex,
          new Exception());
        int question = mQuestionBank[mcurrentIndex].getQuestion();
        mQuestionTextView.setText(question);
    }
```

如同前面AndroidRuntime的异常，Log.d(String, String, Throwable)方法记录并输出整个栈跟踪信息。借助栈跟踪日志，可以很容易看出updateQuestion()方法在哪些地方被调用了。

作为参数传入Log.d(...)方法的异常不一定是我们捕获的已抛出异常。创建一个新的Exception()方法，把它作为不抛出的异常对象传入该方法也是可以的。借此，我们得到异常发生位置的记录报告。

运行GeoQuiz应用，点击Next按钮，然后在LogCat中查看日志输出，日志输出结果如图4-5所示。



Tag	Text
	eNotFoundException: /proc/net/xt_qtaguid/iface_stat_all: o
	irectory)
SizeAdaptiveLa…	com.android.internal.widget.SizeAdaptiveLayout@41ccc060chi
	cd2c30 measured out of bounds at 95px clamped to 96px
QuizActivity	Updating question text for question #0
QuizActivity	java.lang.Exception
QuizActivity	at com.bignerdranch.android.geoquiz.QuizActivity.updat
QuizActivity	at com.bignerdranch.android.geoquiz.QuizActivity.acces
QuizActivity	at com.bignerdranch.android.geoquiz.QuizActivity$3.onC
QuizActivity	at android.view.View.performClick(View.java:4084)
QuizActivity	at android.view.View$PerformClick.run(View.java:16966)
QuizActivity	at android.os.Handler.handleCallback(Handler.java:615)
QuizActivity	at android.os.Handler.dispatchMessage(Handler.java:92)
QuizActivity	at android.os.Looper.loop(Looper.java:137)
QuizActivity	at android.app.ActivityThread.main(ActivityThread.java
QuizActivity	at java.lang.reflect.Method.invokeNative(Native Method
QuizActivity	at java.lang.reflect.Method.invoke(Method.java:511)
QuizActivity	at com.android.internal.os.ZygoteInit$MethodAndArgsCal
QuizActivity	at com.android.internal.os.ZygoteInit.main(ZygoteInit.
QuizActivity	at dalvik.system.NativeStart.main(Native Method)

图4-5 日志输出结果

栈跟踪日志的第一行即调用异常记录方法的地方。紧接着的两行表明，updateQuestion()方法是在onClick(...)实现方法里被调用的。双击该行即可跳转至注释掉的问题索引递增代码行。暂时保留该代码问题，下一节我们会使用设置断点调试的方法重新查找该问题。

记录栈跟踪日志虽然是个强大的工具，但也存在缺陷。比如，大量的日志输出很容易导致LogCat窗口信息混乱难读。此外，通过阅读详细直白的栈跟踪日志并分析代码意图，竞争对手可以轻易剽窃我们的创意。

另一方面，既然有时可以从栈跟踪日志看出代码的实际使用意图，在网站http://stackoverflow.com或者论坛http://forums.bignerdranch.com上寻求帮助时，附上一段栈跟踪日志往往有助于更快地解决问题。根据需要，我们既可以直接从LogCat中复制并粘贴日志内容，也可以选中要保存的内容，单击LogCat右上角的小软盘图标将它们保存到文本文件中。

在继续学习之前，先注释掉QuizActivity.java中的TAG常量，然后删除日志记录代码，如代码清单4-4所示。

代码清单4-4　再见，老朋友（Log.d()方法）（QuizActivity.java）

```
public class QuizActivity extends Activity {

    ...

    public void updateQuestion() {
        Log.d(TAG, "Updating question text for question #" + mCurrentIndex,
          new Exception());
        int question = mQuestionBank[mcurrentIndex].getQuestion();
        mQuestionTextView.setText(question);
    }
```

注释掉TAG常量会移除未使用的变量警告。当然，也可以删除TAG常量，但最好不要这样做。因为，保不准什么时候还会用它来记录日志消息。

4.2.3　设置断点

要使用Eclispe自带的代码调试器跟踪调试上一节中我们遇到的问题，首先需要在updateQuestion()方法中设置断点，以确认该方法是否被调用。断点会在断点设置行的前一行代码处停止运行，然后我们可以逐行检查代码，看看接下来到底发生了什么。

在QuizActivity.java文件的updateQuestion()方法中，双击第一行代码左边的灰色栏区域。可以看到，灰色栏上出现了一个蓝色圈圈。这就是我们设置的一处断点，如图4-6所示。

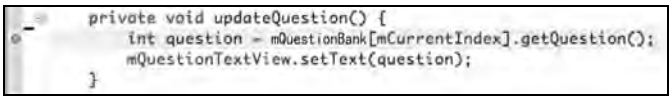


图4-6　已设置的一处断点

启用代码调试器并触发已设置的断点，我们需要调试运行而不是直接运行应用。要调试运行应用，右键单击GeoQuiz项目，选择Debug As → Android Application菜单项。设备会报告说正在等待调试器加载，然后继续运行。

应用启动并加载调试器运行后，应用将会暂停。应用首先调用QuizActivity.onCreate(Bundle)方法，接着调用updateQuestion()方法，然后触发断点。

若是首次使用调试器，会看到询问是否打开调试透视图的提示窗口弹出，如图4-7所示。单击Yes按钮确认。

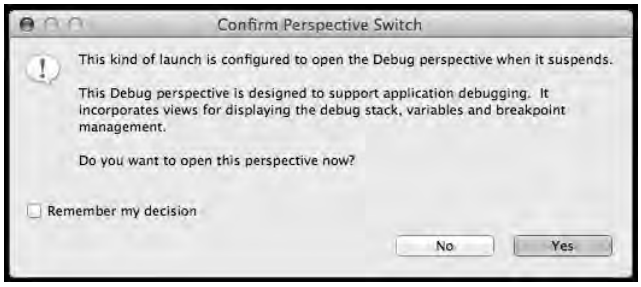


图4-7　切换至调试透视图

Eclipse随后打开了代码调试透视图。调试透视图中间部分是代码编辑视图。可以看到QuizActivity.java代码已经在其中打开了，断点设置所在行的代码也被加亮显示了。应用在断点处停止了运行。

代码编辑视图上方是代码调试视图，如图4-8所示。该视图显示了当前的栈。

图4-8 代码调试视图

可使用视图顶部的黄色箭头按钮单步执行应用代码。从栈列表可以看出`updateQuestion()`方法已经在`onCreate(Bundle)`方法中被调用了。不过，我们需要关心的是检查Next按钮被点击后的行为。因此单击Resume按钮让程序继续运行。然后，再次点击Next按钮观察断点是否被激活（应该被激活）。

既然可以重复断点暂停然后再Resume的过程，也就可以趁机了解下调试透视图中的其他视图。右上方是变量视图。程序中各对象的值都可以在该视图中观察到。该视图首次出现时，只能看到`this`的值（`QuizActivity`本身）。单击this旁边的三角展开按钮或点击右箭头键可看到全部变量值，如图4-9所示。

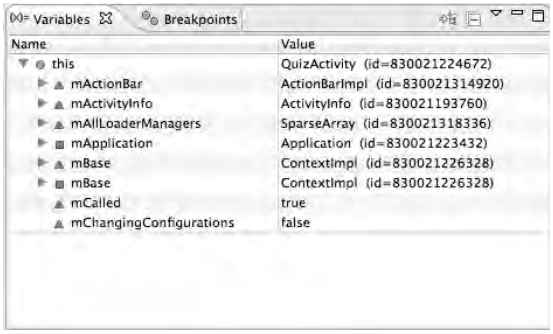


图4-9 变量查看视图

变量名旁边的彩色图形表明了该变量的可见性：

- 绿色圆圈　公共变量；
- 蓝色三角　默认变量（包内可见）；
- 黄色菱形　保护变量；
- 红色正方形　私有变量。

展开this变量后出现的变量数量之多有点让人吃惊。除QuizActivity类中实例变量外，它的Activity超类、Activity超类的超类（一直追溯到继承树的顶端）的全部变量也都列在了this下面。

我们现在只需关心变量mCurrentIndex的值。在变量视图里滚动查看直到找到mCurrentIndex。显然，它现在的值为0。

代码看上去没问题。为继续追查，需跳出当前方法。单击Step Over按钮右边的Step Return按钮（为跳过助手方法access$1(QuizActivity)，因此我们要单击Step Return按钮两次）。

查看代码编辑视图，我们现在跳到了mNextButton的 OnClickListener方法，正好是在updateQuestion()方法被调用之后。真是相当方便的调试，问题解决了。

接下来我们来修复代码问题。不过，在修改代码前，必须先停止调试应用。停止调试有两种方式：

- 停止程序，选中DDMS设备视图中的程序运行进程，单击红色的停止按钮杀掉进程。
- 断开调试器，在视调试图直接单击Disconnect按钮即可，如图4-8所示。

断开调试器要比停止程序更简单些。

然后切换到Java透视图，在OnClickListener方法中取消对mCurrentIndex语句的注释，如代码清单4-5所示。

代码清单4-5　取消代码注释（QuizActivity.java）

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...

    mNextButton = (Button)findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            //mcurrentIndex = (mcurrentIndex + 1) % mQuestionBank.length
            mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length
            updateQuestion();
        }
    });

    ...
}
```

代码修复后，记得清除断点设置。选中变量视图旁的断点视图。（如看不到断点视图，可选择Window → Show View → Breakpoints菜单项打开）在断点视图中选择断点，单击视图上方的深灰色X按钮完成清除。

至此，我们已经尝试了两种不同的代码跟踪调试方法：

- 记录栈跟踪诊断性日志；
- 利用调试器设置断点调试。

没有哪种方法更好些，它们各有所长。通过实际应用中的比较，也许我们会有自己的偏爱。

栈跟踪记录的优点是，在同一日志记录中可以看到多处的栈跟踪信息；缺点是，必须学习如何添加日志记录方法，重新编译、运行并跟踪排查应用问题。相对而言，代码调试的方法更为方便。以调试模式运行应用后（选择Debug As → Android Application菜单项），可在应用运行的同时，在不同的地方设置断点，寻找解决问题的线索。

4.2.4 使用异常断点

如一时无法设置合适的断点，我们仍然可以使用调试器来捕捉异常。在QuizActivity.java中，再次注释掉`mQuestionTextView`变量的赋值语句。选择Run → Add Java Exception Breakpoint...菜单项调出异常断点设置窗口，如图4-10所示。

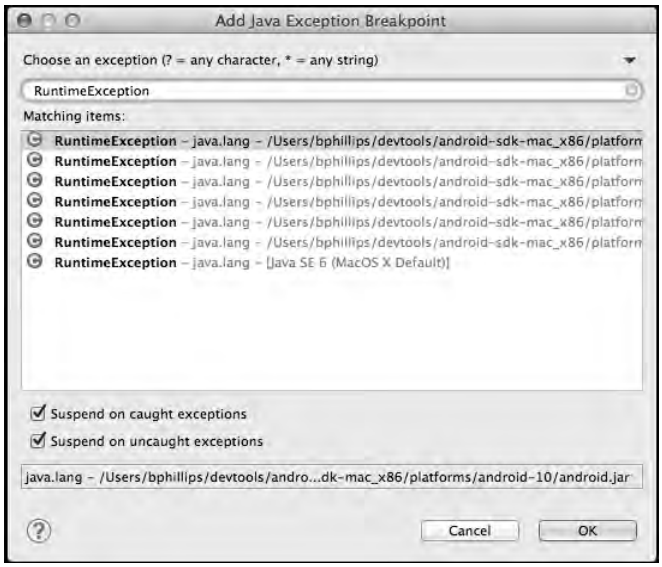


图4-10 设置异常断点

通过该对话窗口设置所需的异常断点。这样，无论任何时候，只要应用抛出异常就可以触发该断点。根据需要，可限制断点仅针对未捕获的异常生效。当然，也可以设置为两种类型的异常都生效。

在Android的世界里，通常由框架来捕捉住大多数异常，然后切换到调试窗口并停止应用的进程。这意味着，设置异常断点时，通常需选择Suspend on caught exceptions选项。

接下来我们来选择要捕捉的异常类型。输入`RuntimeException`，选择随后出现的任何选项。`RuntimeException`是`NullPointerException`、`ClassCastException`及其他常见异常的超类，

因此该设置基本适用于所有异常。

不过，为能捕捉住各种子类异常，我们还需做一件事。切换到调试透视图，在断点视图中，应该能看到刚设置的RuntimeException断点。单击该断点并勾选Subclasses of this exception选框，这样在NullPointerException异常抛出时，断点随即被触发。

调试GeoQuiz应用。这次，调试器很快就定位到异常抛出的代码行。真是太棒了。

异常断点影响极大。在调试的时候，如仍然保留了设置的异常断点，那么在一些系统框架代码或者我们无需关注的地方有异常发生时，断点都会被触发。因此，如不需要的话，建议清除它们。

4.3　文件浏览器

为检查应用运行过程或结果，DDMS透视图还提供了其他一些强大的工具。文件浏览器就是其中一个非常方便的工具。单击透视图右边标签组上的文件浏览器视图按钮打开它，如图4-11所示。

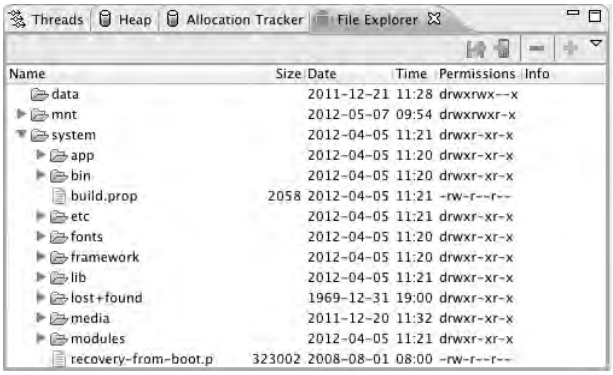


图4-11　文件浏览器

文件浏览器用来浏览设备的文件系统以及上传和下载文件。我们无法在物理设备上查看应用的/data目录，如图4-12所示。但在模拟器上可以，这意味着我们可以查看应用的个人数据存储区。Android最新版本的存储区位于/data/data/[your package name] 目录下。

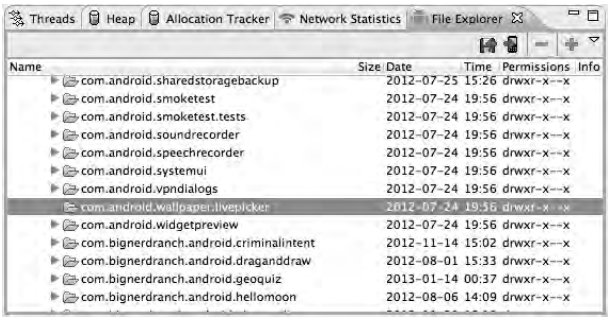


图4-12　GeoQuiz应用的data 目录（模拟器中）

该目录下目前什么也没有。但在第17章，我们会在这里看到应用写入到该私人存储区的数据文件。

4.4 Android 特有的调试工具

大多数Android应用的调试和Java应用的调试都差不多。然而，Android也有其特有的应用调试场景，比如说应用资源问题。显然，Java编译器并不擅长处理此类问题。

4.4.1 使用Android Lint

该是Android Lint发挥作用的时候了。Android Lint是Android应用代码的静态分析器（static analyzer）。实际上，它是无需代码运行，就能够进行代码错误检查的特殊程序。基于对Android框架知识的掌握，Android Lint深入检查代码，找出编译器无法发现的问题。Android Lint检查出的问题通常值得关注。

我们会在第6章看到Android Lint对于设备兼容问题的警告。此外，Android Lint能够对定义在XML文件中的对象类型做检查。在QuizActivity.java中，如代码清单4-6所示，人为制造一处对象转换错误。

代码清单4-6 不匹配的对象类型（QuizActivity.java）

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
    setContentView(R.layout.activity_quiz);

    mQuestionTextView = (TextView)findViewById(R.id.question_text_view);

    mTrueButton = (Button)findViewById(R.id.true_button);
    mTrueButton = (Button)findViewById(R.id.question_text_view);

    ...
}
```

因为使用了错误的资源ID，代码运行时，会导致`TextView`与`Button`对象间的类型转换出现错误。显然，Java编译器无法检查到该错误，但Android Lint却可以在应用运行前就捕获到该错误。

在包浏览器中，右键单击GeoQuiz项目，选择Android Tools → Run Lint：Check for Common Errors菜单项打开 Lint Warnings视图，如图4-13所示。

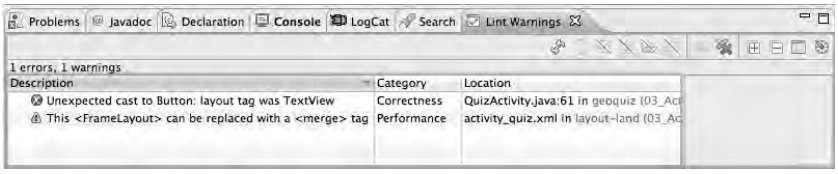


图4-13 Lint发出了警告

可以看到，Android Lint报告了一处错误及一个警告信息。我们已经知道了类型转换错误发生的原因。现在，对照代码清单4-7修正代码错误。

代码清单4-7　修正类型不匹配的代码错误（QuizActivity.java）

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
    setContentView(R.layout.activity_quiz);

    mQuestionTextView = (TextView)findViewById(R.id.question_text_view);

    mTrueButton = (Button)findViewById(R.id.question_text_view);
    mTrueButton = (Button)findViewById(R.id.true_button);

    ...
}
```

注意，Android Lint提示的警告信息和类型转换问题关联性不大。该警告建议在layout-land/activity_quiz.xml中使用`merge`标签。可惜Android Lint搞错了。这里`FrameLayout`是用来以某种特定的方式放置其他组件的，不应该以`merge`标签来替换它。

4.4.2　R类的问题

对于引用还未添加的资源，或者删除仍被引用的资源而导致的编译错误，我们已经很熟悉了。通常，在添加资源或删除引用后再重新保存文件，Eclipse会准确无误的重新进行项目编译。

不过，有时这些编译错误会一直出现或是出现得莫名其妙。如遇这种情况，请尝试如下操作。

- 运行Android Lint

  选择Window → Run Android Lint菜单项。Lint会检查并梳理项目资源文件。

- 清理项目

  选择Project → Clean菜单项。Eclipse会重新编译整个项目，消除错误。

- 重新检查资源文件中XML文件的有效性

  如果最近一次编译时未生成R.java文件，则会引起项目资源引用错误。通常，这是由布局XML文件中的拼写错误引起的。因无法校验布局XML文件的有效性，Eclipse往往无法进行输入错误警示。修正错误并保存XML文件，Eclipse会重新生成新的R.java文件。

- 删除gen目录

  如果Eclipse无法生成新的R.java文件，我们可以删除整个gen目录。Eclipse会重新编译项目并创建一个新的gen目录，内含功能完备的R类。

如仍存在资源相关问题或其他问题，建议仔细阅读错误提示并检查布局文件。慌乱时往往找不出问题所在。休息冷静一下，再重新查看Android Lint报告的错误和警告。我们或许能够从中找出代码错误或拼写输入错误。

若存在无论如何都无法解决的问题或其他Eclipse相关问题，还可以访问http://stackoverflow.com网站或本书论坛 http://forums.bignerdranch.com寻求帮助。