FORUM NOKIA

高效 MIDP 编程

版本 1.1, 2007 年 2 月 28 日

Java™

NOKIA

声明

本文信息基于其现有状况，不存在任何保证，包括销售保证、适用某一特殊用途的保证，或从任何建议、规范或范例中衍生出来的保证。此外，本文所提供的并非最终信息，在其最终发布前可能会有较大改动。本文仅用作信息通报。

诺基亚公司不承担所有因实施本文档中所表述的信息而产生的相关责任，包括侵犯任何知识产权的责任。诺基亚公司并不保证或认为使用这些信息不会构成对这些知识产权的侵犯。

诺基亚公司保留不预先通知而随时修改本规范的权力。

本文中说出现的手机用户界面用于说明目的，并不代表任何真实终端。

授权许可

本授权仅限于因个人应用而下载和打印本说明，除此之外，不存在对其它任何知识产权的授权许可。

目录

修订记录

2002 年 7 月 9 日	版本 1.0	文档首次发布
2004 年 3 月 19 日	版本 1.1	文档更新。
2007 年 1 月 16 日	版本 1.1	文档修订：文字上的微小改动，包括 术语更新。
2007 年 2 月 28 日	中文修订版 1.1	中文修订版发布

1 简介

这份指南讲述了如何提高MIDlet的效率。它涉及以下几方面的内容：

- 执行速度
- JAR 文件大小（下载成本等）
- 资源的使用（内存，网络）
- 响应性能（用户界面响应）

本指南假设读者熟悉Java™编程，同时也对基本了解 MIDP编程，如你已经读过诺基亚论坛中的文章《MIDP编程简介》 **[MIDPPROG]。**

本指南重点讨论 MIDP 的性能问题，并简要述及一些与 MIDP 关系不大的 Java 性能问题。更多信息请参阅《Java 性能的调整》**[JAVAPERF]**和《Java 平台性能》**[JAVAPP]**。

2 执行速度

资深程序员的格言是：程序往往有 90%的时间用在 10%的代码上。因此，与其努力提高所有代码的效率，不如找出代码中的“瓶颈”，使之更高效地工作，如此收益更高。

另一条众所周知的法则是：精细的设计和算法选择比逐行的代码优化更为有效。这条法则不仅对其它程序，而且对 MIDlets，同样有效。随意设计很容易降低 MIDlet 的效率。

Java 程序通常只花很少部分时间执行程序代码，大部分时间都用在执行其所调用的库代码。因此，应该了解各种库的性能（特别是各种图形库），并谨慎选择调用方式。

需要牢记的是：在不同手机上实现的 MIDP 其性能特性可能会有显著的不同。所以，在一种手机上性能极佳的实现方法在另一种手机上却不一定是最好的。例如，在某种手机上，某个特定的库方法可以用 Java 实现，但在另一种手机上，该方法却可能只是封装了一个更快的用手机本机语言编写的方法。

最后请注意，性能表现指标不仅随手机型号的不同而不同，而且在相同型号不同版本的手机之间也存在差异。制造商往往会在某个型号的产品生命周期内升级软件版本，有时甚至更换硬件元器件。

2.1 程序执行速度

2.1.1 量度

寻找程序中性能瓶颈的常用工具是 *profiler*。但是，运行于某台手机上的 MIDlet 往往不能运行 profiler。而模拟器的瓶颈与实际手机中的瓶颈有很大差别，因此运行于某台模拟器上的 MIDlet 的性能并不能向我们提示更多的有关信息。

MIDP 的通常方法是在程序中添加一些附加行来自行检测。可以测出某个调用所费的时间：

```
long startTime = System.currentTimeMillis();
doSomething();    // the thing you want to time
long timeTaken = System.currentTimeMillis() - startTime;
```

你将得到单位为毫秒的所费时间值。为了避免测试过程中因垃圾收集而造成的结果波动，在测试之前，应先调用 System.gc()。要显示测试数据，可以使用一个特殊的 MIDlet 屏幕，或者用结果数据覆盖普通屏幕上的显示。

必须确保手机系统时钟的精度。返回值可能无法精确到毫秒，所以请注意检查返回值，如果它总是十的倍数，那么保证足够长的测试时间是可以忽略这个问题的。

2.1.2 图形操作

通常，我们并不关注诸如 Form 和 List 等 MIDP 高级屏幕类的图形操作执行速度。我们只关注用于动画和游戏等的低级 Canvas 类的执行速度。Canvas 的图形操作速度随手机型号的不同有很大差别，因为执行速度不仅取决于底层硬件，而且也取决于手机自身图形库的效率。在各种支持 MIDP 的诺基亚手机上，你每秒钟可以画数以千计的短线，或数以百计的矩形或小图形（例如子图）。

如果仅需要改变屏幕的一小部分，可以用以下方法请求重绘：

```
Canvas.repaint(int x, int y, int width, int height)
```

这样你就可以指定需要重绘的特定区域。然后 paint 方法将重绘由 Graphics 参数划定的矩形区域，有可能节约许多计算。要注意的是，如果所发出的众多重绘请求快于设备的处理速度，它将合并一

些重绘请求，并使 paint 方法的重绘矩形区域扩展到能够覆盖所有需要重绘的区域，如果重绘区域扩得过大，一些非重绘区也可能会被重绘。

另一个在大多数手机上都有效的优化方法是：如果几次重绘之间的屏幕变化比较轻微，就使用脱屏图像。这样就可以在脱屏图像上进行修改，并用 paint 方法将修改后的图像复制到屏幕上。另外，你可以只复制由 Graphics 参数划定的矩形区域。

2.1.3 垃圾收集

避免在堆上产生不必要的“垃圾”对象，经常情况下可以很方便地重用现有对象。

Immutable (不可变对象)是一个特例，也就是说，它们的状态自对象创建后就不能改变。由于不可变对象有利于代码的可靠和线程的安全，因此得到了广泛使用。

然而，如果其初始值失去意义，例如，如果它们所代表的值发生了改变，此时不可变对象极易变成垃圾对象。这样每一次改变都会创建一个具有新值的新的不可变对象，而相应的旧对象则被废弃等待作为垃圾收集。就 MIDlets 来说，通常使用不可变对象的代价要高于它所能得到的利益，因此，此种情况下最好使用可以赋新值的可重用对象。

最熟悉的例子就是 java.lang.String。大多数程序员都记不清有多少次因使用 String 而产生了垃圾对象。经典的例子是字符串的连接。请看下面的倒排字符串函数：

```
1:    static String reverse(String s)
2:    {
3:        String t = "";
4:        for (int i = s.length()-1; i >= 0; --i)
5:            t += s.charAt(i);
6:        return t;
7:    }
```

第 5 行的赋值语句并不改变字符串 t，因为 t 是不可变的。相反，它每次都创建一个新字符串，复制现存的值并添加新字符。这一方法将不必要地创建 s.length()个垃圾对象。这个典型例子说明了不可变对象所存在的问题。然而，对于字符串却有一个简单的解决办法：类 java.lang.StringBuffer 是 String 对应的的可变操作类，上面的例子可以更高效地重写为：

```
1:    static String reverse(String s)
2:    {
3:        StringBuffer t = new StringBuffer(s.length());
4:        for (int i = s.length()-1; i >= 0; --i)
5:            t.append(s.charAt(i));
6:        return t.toString();
7:    }
```

综上所述，不可变性是利大于弊的。创建少量垃圾对象不是大问题，只要不是每秒钟创建上千的垃圾对象。MIDP 虚拟机甚至就可以每秒钟轻易收集成千上万的垃圾对象。

2.1.4 多线程

多线程令你的 MIDlets 具有更好的性能，这是因为，当一个线程正等待某种条件时（如等待一个 HTTP 响应，或等待用户输入），另一个线程仍可以工作。

需要记住的是，Java 线程不一定具有抢先权，但可以相互协作。因此，代码不应在“死循环”中等待所需条件，而应该在每次循环中调用 yield 或 wait 方法，如：

```
try
{
    while (!stopped)
    {
        try
        {
```

```
                doSomething();
                synchronized(this)
                {
                    wait(500);  //milliseconds, i.e. half a second
                }
            }
        }
}
catch (InterruptedException e)
{
}
```

请注意，实际上在MIDP中从来都不会抛出InterruptedException异常（由于Thread.interrupt方法并不存在）。然而，为了与其它Java环境兼容而抛出异常，wait方法仍然将其声明，但必须使用一个空的异常处理语句。

2.2 网络速度

2.2.1 带宽和延迟时间

单纯的网络速度通常按照带宽和延迟时间来衡量，其定义如下：

- *带宽*：某个开放连接中的数据传输率（通常按每秒种的比特数来计算）
- *延迟时间*：单个数据单元从源到目的地网络所需的传输时间。

这两项指标都有平均值和浮动差。即使平均值是可接受的，如果浮动差过大，用户也是常常难以接受的。

对于大数据量传输，带宽通常对网络速度的影响是主要的。而对小数据量传输，则延迟时间更为重要。

这两种性能指标的量度都取决于网络技术和网络连接的特定属性。所以，一般意义上给出精确数据是不可能的。但可以给出一些概念化数据以帮助设计。表 1 给出的一些特征值是最近从诺基亚内部研究和非正式测试中获得的：

	带宽（千比特每秒）	典型HTTP往返延迟的首次值	典型HTTP往返延迟的随后值
GSM 电路交换数据(CSD)	9.6 kbps	5 – 10 秒	2 – 3 秒
GSM 高速电路交换数据 (HSCSD)	最高 43 kbps	5 – 10 秒	2 – 3 秒
GPRS	5 – 50 kbps	5 - 8 秒（网络拥塞时可能更费时）	2 – 4 秒（网络拥塞时可能更费时）

表 1：带宽与延迟的特征值

可以看到，这些数据要比你所熟悉的宽带互联网连接慢。特别是，因为不能实时地看到并响应其它玩家的行动，较长的延迟时间使高互动性的实时多人街机游戏难以维继。

对付网络延迟的一个诀窍是使用各种异步网络操作（利用一个后台网络线程）。例如，如果你正向高分储存服务器发送一个新的高分，没有必要将 MIDlet 的其它部分挂起而等待这一操作的完成；相反，可以让后台网络线程来处理这一动作，而你则可以进入下一个游戏。

2.2.2 限制HTTP的往返传输

由于无线网络中的 HTTP 延迟时间较长，因此应该努力减少 MIDlet 使用的 HTTP 往返传输次数。尽管互联网浏览器可以进行多次 HTTP 往返传输来获取某个网页上不同的框架和图像，但 MIDlet 却应该一次性地获取它所需的全部内容。

如果 MIDlet 要从几个不同的数据源收集数据，一种可选方法是用一个代理 servlet 来收集，这样 MIDlet 仍然只需要发送一次请求。有关代理 servlet 的更多信息，请参阅后面的章节。

2.2.3 避免使用低效协议

SOAP 等基于 XML 的复杂协议可能由于数据大小、解析时间和解析代码大小等更高开销而变得非常低效。如果你正在设计用于某个 MIDP 客户端基于 XML 的协议，应该使其尽可能的简洁。

如果不需要这些协议的高级功能，则可以用一个简单的自定义协议（例如 `username=fred&password=secret`）来发送同样的信息，这样效率会高得多。诺基亚论坛文档专区中的《MIDP 1.0：联网MIDlets简介》**[MIDPNET]**和《一个网络MIDlet范例：老虎机》**[FRUITMAC]**里有对这种简单自定义协议的描述。协议的可读性对调试大有帮助（这就是HTTP等许多互联网协议可读的原因）。

一般说来，如果你使用某个简单自定义协议，那么你就要既写 MIDlet 客户端也写服务器（如 servlet）。需要注意的是版本管理，当升级客户端和服务器时如使用一个新协议，而对应的 MIDlets 仍然使用旧协议，就会产生一些问题。在客户端和服务器间的开放式通信中包含协议的版本号是很好的想法。

如果你的 MIDlet 与并非你自己写的或不受你自己控制的服务器通话，或者某台服务器还要与你的 MIDlet 之外的其它客户端通话，则你的 MIDlet 可能不得不用 XML 或 SOAP 与之通话。即使在这种情况下，你也可以使用代理 servlet 来避免这种情况：

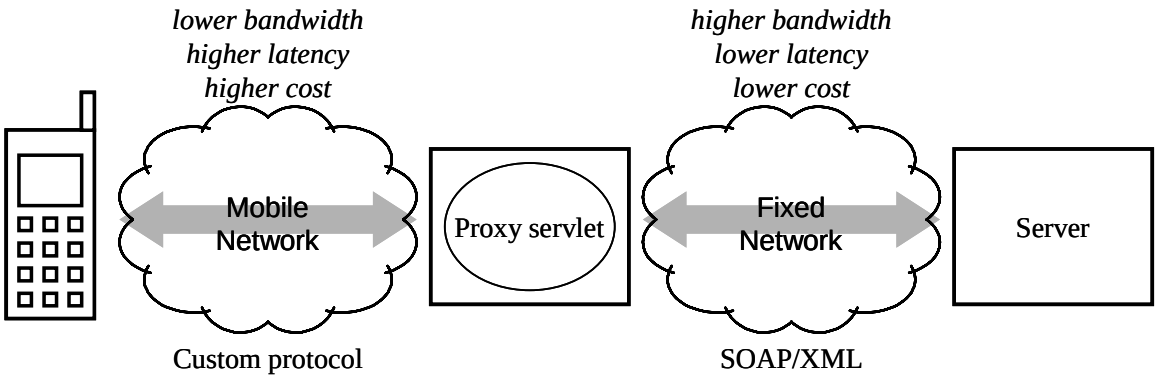

图 1：代理 Servlet

代理 servlet 可以在 SOAP 或 XML 协议与 MIDlet 自定义协议之间实现转换。它根据每个 MIDlet 请求对应地发送一系列 SOAP 或 XML 请求——由于代理 servlet 和服务器之间通过一个快速固网连接，因而这样做的效率很高。

2.2.4 Socket 连接

MIDP 2.0 提供了一些接口，用于简化基于 UDP，TCP 和 TLS 的通信。可以用一个 UDPDatagramConnection 来发送和接收 UDP 数据报消息。也可以以用一个 ServerSocketConnection 接受来自外部的 TCP 连接请求。对每个所接受的来自外部的 TCP socket 连接，都会创建一个 SocketConnection。SocketConnection 也用于向外的 TCP 连接。不是用 HTTP 连接，相反地，我们向

设计师提供了能有所帮助的意见，使其所创建的代码具有更高的性能表现和数据吞吐量。这些都基于对某些现有 MIDlets 性能表现方面的现实性观察。

从底层平台、TCP 协议，及承载层的角度看，以非常小的数据块（每次一个字节或类似）收发数据是非常低效的。这将导致平台负担不必要的额外开销，而且无法充分利用 TCP 和蜂窝无线网络的的全部能力。只要有可能，MIDlets 都应该以较大的单位收发数据。

关于通过TCP sockets接收数据，MIDlets不应以polling方式使用输入流接口的`available()`方法，即在一个（繁忙）循环中询问可以读取多少数据。这种方法非常低效，可能导致严重的性能表现异常。为了实现高效率，MIDlet应该更智能化地使用一些流读取方法，这些方法能规定所读取的数据量。当然，确切使用哪种接口取决于具体MIDlet的要求。

3 JAR 文件的大小

请尽量缩小 JAR 文件的尺寸，其理由如下：

- MIDlet 套件的下载将更快：例如，GSM 电路交换数据下载每千字节的 JAR 文件耗时 1 秒。
- 许多 WAP 网关往往被配置成小 WAP 页，而且不支持大文件（特别的，超过 30KB 的 JAR 文件就会遭遇这个问题）
- 面向广阔市场的 MIDP 手机可能只具备不足以应付 MIDlet JAR 文件的有限存储空间（存储空间随手机型号不同而有很大不同）
- 用户更偏爱小的 MIDlet，因为这样就可以在手机中安装更多的 MIDlet

3.1 小尺寸设计

有两种尺寸比较重要：MIDlet 套件的 JAR 文件的尺寸以及 MIDlet 套件安装在手机上所占用的空间尺寸（如果 JAR 文件不是按其打包状况安装的）。后一个尺寸取决于手机的具体实现。但是， JAR 文件的大小却是所需安装空间的预测标识。

由于 JAR 文件格式为每个类文件设置独立的头，因此通常情况下尽可能使用较少的类。由于这个原因，MIDlet 不象普通的 Java 程序那样“面向对象”。特别是：

- **每个‘事物’只有一个类**－例如，如果用一个类就可以完成所有事情，就不要把它分成“模型”、“视图”和“控制器”等类
- **限制使用各种接口**－接口是一种特殊的类，按照定义，它不提供功能；仅当在同一个发布的 MIDlet 中需要处理多重实现时才使用接口
- **使用“无名包”**－将所有的 MIDlet 类放在同名的包中只会增加 MIDletJAR 文件的尺寸－只为导入库添加 `package` 语句
- **考虑使用源代码预处理器，而不用‘static final’常量**－每个这类常量都会占用 JAR 文件空间
- **限制使用静态初始化**－Java 类文件格式并不直接支持静态初始化，而是在执行时才作分配；例如，静态初始化某个字节数组(‘`static byte[] data = {(byte)0x02, (byte)0x4A, …};`’)在生成的类文件中将导致每个值占用 4 个字节，而不是 1 个字节

3.2 使用扰码器

扰码器是一个程序，它修改已编译的 Java 程序，删除所有不必要的信息（如较长的方法和变量名），从而使‘反编译’的结果难于理解。作为一个保护方法，它对 MIDlet 没有什么价值。因为 MIDlet 太小了，通过一些处理，反编译一个经过扰码处理的 MIDlet 也能将其猜个八九不离十。

但是，删除所有不必要的信息可以缩小 JAR 文件的大小，这是非常有用的。缩小的比例因扰码器的不同而不同，也因 MIDlet 的不同而不同。但是，我们自己的 MIDlet 测试显示：一般可以压缩 10%。这个数字通常比扰码器声称的缩小比例要小。可能由于 MIDlet 很小，并且有一些固定开销，也可能是因为 PNG 位图文件等资源占用了 MIDlet 中 JAR 文件的较大部分。

3.3 库

在一般的软件开发中，开发并使用一些经常用到的库是明智的做法。但是，如果整个库都被包含在 MIDlet 套件中，你可能就要为许多实际上并不需要的功能增加系统开销。

最好回到使用“剪切加粘贴”的重用方式。对开发者的开发进度来说，这样做是低效率的，但对 JAR 文件的空间使用却更为高效。

使用库时，需要考虑是否真正需要库中所有的类。也许可以删除一些类文件。甚至删除一些不用的方法然后重新编译某些类。

需要记住的是，编写 MIDP 库时应该尽量减少各个类之间的依赖关系，这样，不需要的类就可以被安全地真正删除。遗憾的是，减少类之间相互依赖的典型方法是使用 Java 接口，而这又会增大库的尺寸(见 3.1 节)。

3.4 保持小资源

MIDlet 套件通常包含相应的 PNG 位图等资源。应该尽可能使其保持小尺寸、小数量。

用不同的位图编辑工具储存 PNG 位图时，位图文件的大小会有很大的不同，它们并不都对尺寸进行优化。可以尝试其中的一些工具，并用储存尺寸最小的工具来保存位图（即使你喜欢用其它工具进行编辑）。

3.5 合并图像文件

尽量减少类文件有助于降低每个 JAR 文件的大小，同样，尽量减少图像文件对此也很有帮助。一个经常使用的窍门是将许多图像合并到一个文件中去。

图 2：合并后的图像文件

从文件中载入大图像后，可以用以下方法绘出每个帧：

```
g.setClip(x, y, FRAME_WIDTH, FRAME_HEIGHT);
g.drawImage(fiveMenImage, x - FRAME_WIDTH * frameNumber, y,
            Graphics.TOP | Graphics.LEFT);
```

这里，`frameNumber`的值从 0 到 4；将其按{0, 1, 2, 3, 4, 3, 2, 1}的顺序进行循环，你可以制造出一幅某人正在行走的动画。如果要用`paint`方法进一步绘画，需要记住的是：有必要再次改变图像的剪辑窗口。

4 资源利用

4.1 堆内存

一般说来，移动电话的堆内存都不大。例如，某些早期诺基亚 MIDP 手机仅提供 150KB 的 MIDP 堆内存。后来一些机型，如诺基亚 6600 移动电话，重启手机后如果没有运行其他附加应用，可以具有多达 9MB 的堆内存。一般而言，S60 终端的情况比 Series 40 终端要复杂很多，因为前者可以同时运行几个应用。诸如 `Runtime.getRuntime().freeMemory()` 和 `Runtime.getRuntime().totalMemory()`等方法在 S60 终端中并不会给出正确值，因为堆内存的数量根据当前的需求而动态变化。因而就无法给出堆内存的确切数量。

必须确保释放不再用到的屏幕（如开机视屏，splash screen），以便作为垃圾收集。不经常使用的屏幕（如帮助，选项等）应该在用到时才创建，并在每次使用后通过垃圾收集释放其所占空间。这样做是通过牺牲执行速度而换取了额外的堆内存。

要特别注意“内存泄漏”。当不再拥有某个对象，但却在某些地方仍然存在对它的引用时（这样它就不会被作为垃圾收集），就会发生内存泄漏。当不能很快摒弃对某个不再用到的对象引用时，应将引用设为空（`null`）。

理解 MIDlet 应用如何使用堆内存是很有用的。如果需要在某些 MIDP 终端上优化 MIDlet 的堆内存使用，这方面的理解将帮助你发现在 MIDlet 代码中那些地方作些改进就能大大节省堆内存。例如下列

方法可能适用于某些情况：为大型数据对象或集合使用更为紧凑的数据表现；寻找大型但却放置很少数据的数组，并使用具有更高效数据表现的其他方法；也可以使用更为紧凑的编码图像。

4.2 网络

随着分组交换数据（如 GPRS）的广泛使用，MIDP 设备正日益成为主流。所以，它有可能成为 HTTP 连接所使用的网络技术。你的用户可能正在为你的 MIDlet 所形成的网络服务按信息包或按兆字节付费，因此你有责任谨慎使用用户手中的钱。

最小化你所发送的信息包的数量和大小。例如，仅当发生变化时才发送信息包，而不要发送“什么都没发生”这样的消息。可能的话，避免将你的 MIDlet 设计成向服务器发送常量流消息，例如，让 MIDlet 和服务器预测各种状态的变化，仅当状态与预测内容不同时才发送消息。

5 响应性能

实际上，用户并不在意你的 MIDlet 有多快，而更在意你的 MIDlet 感觉起来有多快。有很多窍门能让 MIDlet 感觉起来更快，尽管其中的一些方法使实际速度有所下降。

5.1 Indicating liveness 进程提示

当 MIDlet 的显示停止变化时用户最能体会到。此时他们会怀疑 MIDlet 或手机发生了故障。如果进行大型计算或 HTTP 网络访问这类需时较长的工作，最好显示一个动画指示器，如类似标尺的进度指示器。最流行的网络浏览器在下载网页时会出现进度指示器，这就是一个很好的例子。

5.2 响应

如果某个 MIDlet 不能迅速反应用户的按键操作，就会被感觉为响应迟钝。遗憾的是，在阐述性能的规范中对“快速”的定义不甚明确。就 MIDlet 而言，正常速度的响应是用户对本地应用的体验，根据手机型号的不同而有所不同。当然，任何超过一秒的操作都会被认为非常漫长。

要提高响应速度，必须确保事件回调（如 `Canvas.keyPressed` 或 `CommandListener.commandAction`）的快速返回，因为负责重绘屏幕的线程可能也会调用它们。如果要做一个耗时较长的动作，应该启动一个独立的线程，并设法让用户可以选择退出。

需要记住的是：仅有速度是不够的，用户必须看到工作有所进展。对每一次按键，都应该给用户一个看得见的（听得到的）反应。

5.3 延迟掩盖

延迟掩盖的好方法是使用“炫目视屏(splash screen)”，当MIDlet的其它部分还在对自身实施初始化时，可以预先显示这种屏幕。MIDP `Alerts`非常适用于炫目视屏，如果你想在炫目视屏超时或被取消时仍能方便地回调，可以使用`Form`(表单)。诺基亚论坛的“老虎机”范例[FRUITMAC]中就有这样一个炫目视屏。

在老虎机范例中还有一个掩盖网络延时的窍门：它一开始就显示滚轮，然后才向服务器发出 HTTP 请求来获得结果。滚轮一直快速旋转，直到收到 HTTP 响应。然后，它逐渐放慢速度并停在服务器给出的结果值上。这样，MIDlet 看上去好像很积极地在工作，虽然它实际上只是在等待服务器的响应。

6 可用性考虑

可用性是应用设计的重要课题，在移动终端中则更为重要，因为移动终端显示屏小，输入机制有限。考虑可用性问题时应该想到：运行 MIDlet 的移动终端可以发生很多情况。这些情况如突然断电，没有网络覆盖，以及接到来电等。

6.1 终止 MIDlets

可以因为很多理由而去终止一个 MIDlet。不管这种理由是有目的的还是意外的，该 MIDlet 的行为应不致于给用户造成麻烦。需要保存数据以使该 MIDlet 在下次启动时具有同样的场景。表 2 给出一些能导致 MIDlet 终止的事件。它也展示了 MIDlet 的行为表现和退出点。

如果是诺基亚终端中的一个正常退出，Java 应用管理器 (Java Application Manager, JAM) 会调用该 MIDlet的`destroyApp()`方法。这为实现一个自动保存机制提供了完美平台，以便将应用数据保存到RMS。然后用户就可以在下次使用时具有同样的状态。如果应用不能在五分钟内关闭，JAM会立即杀掉KVM。这会限制向服务器保存数据或从服务器提取数据。

事件	说明	显示等待提示	退出类型	退出点
关电源		否	非正常	关电源
无法解决的错误	如内存用完，稍后讲述	是	正常	应用列表
超出 Java 运行内存	如其他错误，见上述内容	Error Note	非正常	应用列表
自然退出	MIDlet 自己的退出方法	是	正常	应用列表
“后退”命令	当将“后退”用于退出时	是	正常	应用列表
短按“结束”键		是	正常	应用列表
长按“结束”键		是	正常	应用列表

表 2：一些能终止 MIDlet 的事件

6.2 应用状态

MIDlet 运行中会发生很多事件，必须记住的是：应用赖以开发的平台是一台移动电话。表 2 提供了当 MIDlet 运行时出现的一些最常见事件和行为。

事件	说明	行为
收到 SMS	收到 SMS 时的提示音和震动。 没有其他的干扰影响	无
来电	出现来电提示	MIDlet 运行于后台，同时显示一条提示信息。拒绝或接听后，手机恢复显示 MIDlet 并立即继续执行 MIDlet。
电池不足提示	提示加警示音	MIDlet 运行于后台，同时显示一条提示信息。过了一段时间后手机恢复显示 MIDlet 并立即继续执行 MIDlet。

表 3： 会中断 MIDlet 的一些事件

必须记住的是：来电提示超时或接听电话后一段时间，手机恢复显示MIDlet并立即继续执行MIDlet，因而需要实现一个自动暂停。通过暂停该MIDlet，用户在就绪后就能返回到这个MIDlet。由于诺基亚终端并没有在MIDlet类中实现PauseApp()方法，必须使用其他方法。

通常当屏幕上出现一些图像时，它们是用Canvas类创建的，这个类中含有hideNotify()方法。当cavas从显示屏消失后调用该方法，例如，当某条提示或某个来电消息遮盖了原有显示时。可以使用这个hideNotify()方法来实现一种机制，如停止运行线程。应该向用户指出正处于暂停模式，例如，显示一个暂停信息，或在菜单中加入一个“继续”项。

Canvas的showNotify()方法调用先于显示canvas到屏幕。使用这个方法就可以实现一些方法用于重启线程或恢复被保存的状态，但重要的是：用户可作出自己的选择。也就是说，用户决定后才能继续运行MIDlet, 而不是自动继续。

通过调用其isShown()方法可以查询Displayable对象真实可见或者不可见。这并非一个事件驱动的动作，所以需要被监听。下面的范例说明了实现一个自动暂停方法的简单方式。

Canvas class:

```
public void hideNotify() {
  pauseOnCanvas(); }

public void showNotify() {}

public void pauseOnForm() {
  canvasPaused = false;
  stop();
  myForm.pauseForm();
  parent.setDisplayable(myForm); }

public void pauseOnCanvas() {
  if (myForm.pauseOnForm() == true)
    pauseOnForm();
  else {
    paused = true;
    stop();
    repaint(); }
}

public void resumeCanvas() {
  canvasPaused = false;
  start();
}
```

Form class:

```
public void pauseForm() {
  addCommand(cmResume);
  formPaused = true;
}

public void resumeForm() {
  removeCommand(cmResume);
  formPaused = false;
  parent.setDisplayable(myCanvas);
  myCanvas.resumeCanvas();
}
```

7 参考文献

[FRUITMAC] *MIDP 1.0：老虎机范例*, 诺基亚论坛，2004, http://www.forum.nokia.com

[JAVAPERF] *Java*性能调试， Jack Shirazi, O'Reilly, 2000, ISBN: 0-596-00015-4

[JAVAPP] *Java*平台的性能表现：战略与战术，Steve Wilson and Jeff Kesselman, Addison-Wesley, 2000, ISBN: 0-201-70969-4

[MIDPNET] *MIDP 1.0：网络MIDlet介绍*，诺基亚论坛，2004, http://www.forum.nokia.com

[MIDPPROG] *MIDP 1.0：MIDlet编程介绍*，诺基亚论坛，2004, http://www.forum.nokia.com

8 请对本文进行评价

请花一点时间对对本文评分，以帮助我们改进文档质量，也让我们能了解您所发现的最有价值的资源。