J2ME 技术

在移动设备上的应用

说明

本文是在我的毕业论文的基础上稍作修改而成，由于原论文的主旨是试图对 J2ME 平台，已及 J2ME 对移动信息设备编程（即 MIDP 编程）给出一个概述性的介绍，所以只给出了非常少量的示例，并且很多部分语焉不详。例如对于 MIDP 的 UI 编程部分，只是对各种 UI 组件给出了一个简单的描述。所以本文并非是一个 Step By Step 学习 J2ME 编程的教材，这一点希望读者注意。另外，需要说明一点，限于篇幅，原论文并未对 MIDP 的网络库及网络编程做出讨论。如果时间允许的话，我打算对各部分进行补充，并添加示例。请读者留意未来的更新版本。

本文中有两幅图参照了王森（moli）兄的《Java 手机程式设计入门》一书中的画法，特此感谢！（moli 兄，多谢了，^_^）

Java 研究组织
吴一丁
2002.6

摘要

本文介绍了 Java2 平台微型版的基本内容，以及如何使用 CLDC（连接受限设备配置）和 MIDP（移动信息设备描述）

Abstract

Introducing the Java 2 Micro Edition Platform and addressing how to write Application upon it。This article focus on the CLDC（Connected,Limited Device Configuration）and MIDP（Mobile Information Device Profile）programming with the introduction of MIDlet and its UI and RMS（Record Manage System）。

关键词（**keyword**）

Java，Java2 Platform ，J2ME， MIDP， CLDC，MIDlet，RMS

索引

第一章 J2ME 概述

1.1 J2ME 缘起

Java 的发展与 J2ME 的出现

自从 1995 年 Java 语言正式推出以来，Java 技术立刻成为被人们追逐的热门话题，然而早期的 Java 的真实能力与媒体的炒做有着相当大的差距，所以一段时间之后，Java 被冷落了。然而随着时间的推移，Java 日趋成熟，它逐渐变得更加稳定、可靠，性能也有了大的提高，到了 1999 以后，人们逐渐发现用 Java 开发 EJB 很好，用 JSP 开发 web page 也比 ASP 来得强大。然后 Java 开始向 server side 移动，作为连接客户机，数据库，应用服务器之间的中间件，展现出其跨平台，多线程，连网能力强大，类库丰富这些特有的魅力。现在 Java 又开始进入嵌入式系统，并且在手持设备，车载电脑，智能卡等领域站稳了脚跟，并逐渐成为既定的标准。有趣的是 Java 本来就是为了开发数字家电而被研发的，经过不断的变迁（由 device 到 desktop 到 server 内部），现在 Java 又回到了最初的起点，正好形成一个周而复始的圆。

J2ME，也就是 Java 2 Platform, Micro Edition，最早是在 1999 年 6 月的 JavaOne[1] 大会上被正式提出。是 Sun 专门为小型的，资源受限的消费性电子设备的应用程序开发所提供新的 Java 版本。它广泛的使用于例如蜂窝电话（cell phone），双向传唤机（two-way pager），PDA（个人数字助理），以及电视机顶盒等众多小型资源受限设备中。相对于我们平常在 desktop 上使用的 J2SE，你可以粗略的把 J2ME 理解为在微型设备上使用的 Java 平台。

移动应用与移动通讯市场

进入 2000 年以来，移动通讯市场竞争日趋激烈，各大电信服务提供商与手机、PDA 等通讯电子产品生产商为了争夺用户份额以及未来的利润，以前所未见的迅猛之势，不断推出各种新颖的服务

[1] JavaOne 是 Sun 每年举行的针对 Java 技术的大型研讨会。来自世界各地的 Java 程序员汇聚一堂，交流最新的 Java 技术。Sun 也会在那时发布 Java 的最新产品并试图指出 Java 未来的发展方向。

与新一代的通讯产品。而软件市场更是把目光投向了这被认为具有无穷潜力，标志着未来市场走向的移动应用市场。

未来科技的发展使人们相信，在今后的若干年里，大部分的这些通讯产品都将会与互联网相连，这将导致人们看待和使用通讯产品的方式的转变。用户的要求将不止局限于普通的通讯，更重要的是对日益增强地对信息的需求，用户可能需要访问 Web 站点的内容，查询企业的数据库的资料，以及管理个人的私人数据。用户需要在任何时间，任何地点，以至于使用任何设备来获取信息。未来的设备将更为人性化和用户化。而这种人性化与用户化，同时也就意味着通讯设备的应用平台必须更加多元化。

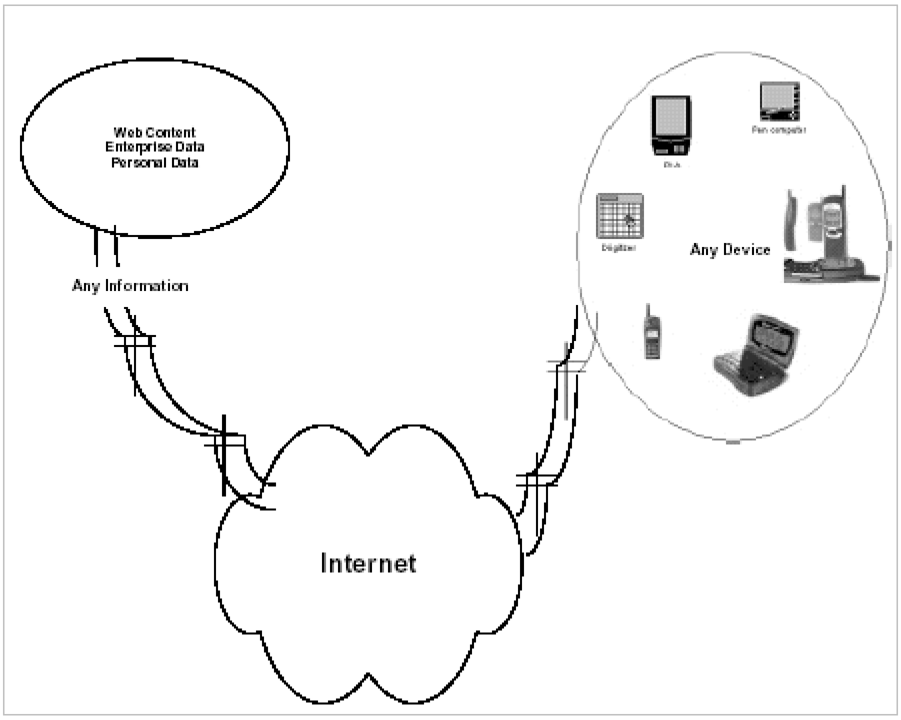

图 **1.1** 通讯产品与互联网相连

这时矛盾出现了，多元化的设备平台也带来了开发的不便。你可以设想一下，如果你开发了一个小的应用程序（假如说针对 Nokia 的某一型号的手机），这时你想在别的型号 Nokia 机上运行该程序，那么你可能需要面对一些移植工作，如果你想在 Motorola 或是西门子又或是爱利信上运行呢？，这种平台间的互相独立使得你的应用绑死在某种特定的设备上，这对于第三方软件开发商来说，简直就是噩梦。这时正是 Java 出马的时候，对于 J2ME 来说，由于其得天独厚的跨平台性与良好的可移植性，使得其在设备繁多，平台异常混乱的手机与 PDA 市场更显得如鱼得水。我们将在后面的章节中具体介绍 J2ME 是如何支持众多的设备平台的。

J2ME 的历史与前景

J2ME 这个名词虽然是最近才提出的，但是自从 Java 技术提出以来，sun 就从未停止过在小型设备应用这方面的研究，从最早的 90 年代初的 Oak 计划(Java 的前身)，到 1997 年提出的 Personal Java，1998 的 JavaCard，然后是 99 年的 KVM 计划，以及 J2ME 这个名词的提出。J2ME 作为一个独立的平台逐渐成型，而且还在不断的发展更新中。我们会在后面的章节中介绍目前 J2ME 的体系结构。

据一些无线市场分析家预计，到 2004 年，J2ME 将成为无线设备的主导开发平台。今天我们可以发现全球著名的通讯设备制造商们如 Motorola，Nokia，西门子，Panasonic 纷纷宣布支持 J2ME 技术，在美国，著名的电信网络运营商如 Nextel，SprintPCS，BT Cellnet 等也热切地加入到 J2ME 的研发中来。在日本，NTT DoCoMo 推出的 i-mode 手机获得了极大的成功，I-mode 手机部分使用了 J2ME 技术。据 Sun 公司统计，到 2001 年全球已经售出了 1500 万台支持 J2ME 技术的手机。到了今年（2002 年），Java 技术在移动设备上的应用更是被炒做的异常火热。今天的 JavaOne 大会上，J2ME 与 J2EE 一道成为了十分火爆的名词。根据 sun 的统计，目前有超过 200 家公司发售 J2ME 应用程序，今年全球将生产超过 1 台 J2ME 手机（Nokia 将生产其半数以上）。“J2ME 技术会占据移动设备领域”这句话现在说似乎还为时过早，但也并非是空穴来风。

如同前文谈到的移动设备和无线市场的特性，J2ME 的主要技术优势在于：良好的跨平台能力，与 J2EE 后端无缝的结合能力和 Java 语言的优良特色（如简单，安全）以及现有的 Java 平台上的开发工具，企业，开发人员将会为 J2ME 提供良好物力与人力基础。

未来，sun 以及 JCP 合作厂商将进一步完善与发展 J2ME 技术。这些工作一直都在进行中，如即将被发布的 MIDP NG，CLDC1.1 等等。

1.2 Java 版本的变迁

Java 的三个版本

目前 Java2 平台已经正式划分为 3 个部分，它们分别是：

- **J2SE**
  Java 2 平台标准版（Java 2 platform standard Edition），其被使用范围最广，用于我们通常使用的台式机（desktop）。

- **J2EE**
  Java 2 平台企业版（Java 2 Platform Enterprise Edition），被广泛应用于服务器上，包括 JSP，Servlet，EJB 以及目前炙手可热的 Web Service 等技术。

- **J2ME**
  Java 2 平台微型版（Java 2 Platform Micro Edition），被使用在资源受限，小型消费性电子设备上。如电视机顶盒，无线电话，掌上电脑等。

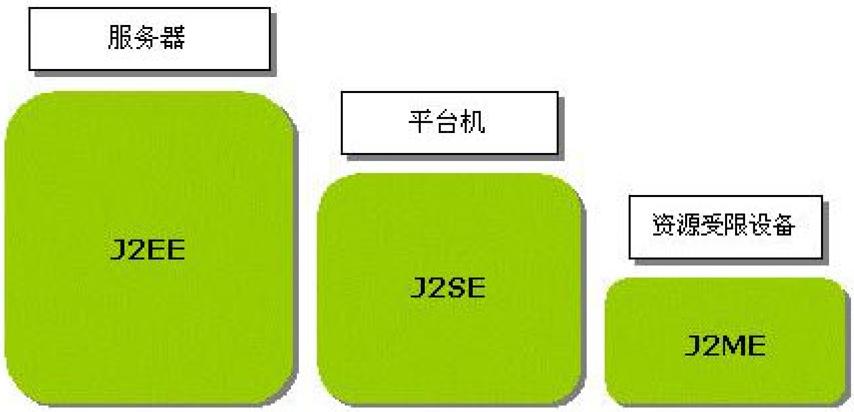

图 1.2 Java 版本的划分和针对的设备

J2ME 的优势

移动设备，PDA 市场的繁荣发展，造成了五花八门的厂商与品种异常广泛的设备。例如两种设备也许具有相同的功能，当实现的方法可能会有相当大的区别，例如对 PDA 和手持电话来说都具有

显示的功能，但屏幕的实际尺寸却有很大的区别，支持的色深也不相同。重复的开发工作以及移植的不便性使得开发成本与开发难度不断提高，这就使得设备厂商要求拥有一个统一的开发平台。

面向这个问题 Sun 提出的 J2ME 的解决方法具有的优势有，从程序员的层面看—java 语言简单易学，而且功能强大，减小学习该技术的梯度。而且大量的已有的Java程序员可以平滑的过渡到J2ME编程中来，为其提供大量的人力支持。从 Java 平台来说，天生的对移植性良好的支持，以及 Java 平台的整体解决方案，即与 J2SE，J2EE 的协作可以建立端对端的 Java 解决方案。

J2ME 与 J2SE 之间的关系

如果 J2ME 可以具有 Java 平台的全部功能当然最好了，但实际上是不可能的，J2ME 针对特定的目标，必然要做出选择：什么是应该保留的，什么只好舍弃，以及需要加入些什么。例如 J2SE 中用于 UI（用户介面）设计的 AWT（Abstract Window Toolkit）库和 Swing 库，就只能舍弃，因为对于 J2ME 支持的那些资源有限的电子设备来说，一般其显示屏的物理尺寸都相对较小，并不适合 AWT 这些用于平台机的 UI 包。而且这些设备的处理器能力和内存大小也不足以支持 AWT、Swing 的事件处理模式。所以 J2ME 就定义了特有的 UI 包来适应这些设备。[1]

J2ME 与 J2SE 在类库级别上的关系图 1-3 所示

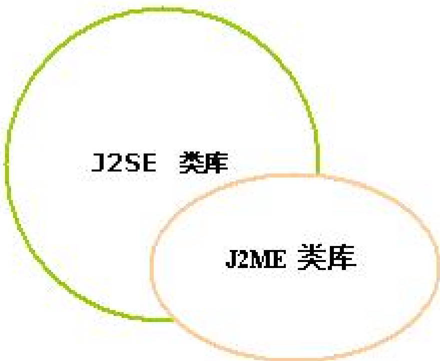

图 1-3 J2ME 与 J2SE 关系

如上图所展示的可以把类库分为 3 类

- J2SE 专有的类库
- J2SE 与 J2ME 共有的类库
- J2ME 专有的类库

[1] 具体来说对应于 MIDP，定义了 javax.microedition.lcdui 包.详细内容请参考本书第四章。

1.3 J2ME 的体系结构

J2ME 的体系结构总体概览

J2ME 其作为一个可以与 J2EE,J2SE 相提并论的平台是在 1999 年的 JavaOne 大会上提出的，随着时间的推移与技术逐渐地成熟，其平台本身的构架也逐渐演绎，嬗变。下图给出 J2ME 一个总体的构架，请注意，由于 J2ME 平台还在不停的发展之中，下图展示的结构可能与 J2ME 当前最新的结构相比存在一些细小的差别。

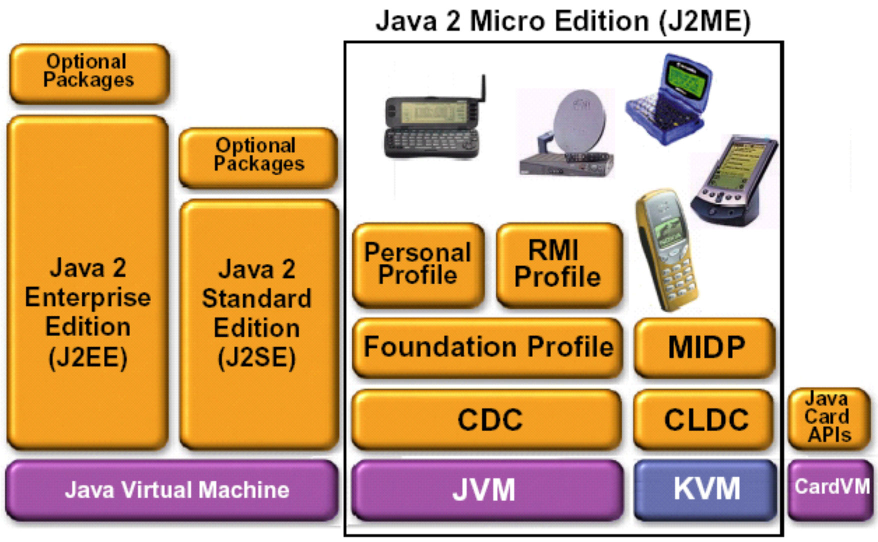

图 1-4 J2ME 平台总体结构

何谓 Configuration 与 Profile

为了使 J2ME 平台可以良好地支持范围广泛，品种诸多的消费性电子设备。以及应对不断变化渐进中的设备技术。为未来应用提供可扩展的空间。对 J2ME 平台经行模块化的设计是必不可少的。因此 J2ME 平台的设计师们提出了两个新的概念，分别为：配置（configuration）和描述（profile）。并以它们为核心把 J2ME 进行了分层化。如图 1-5 所示：

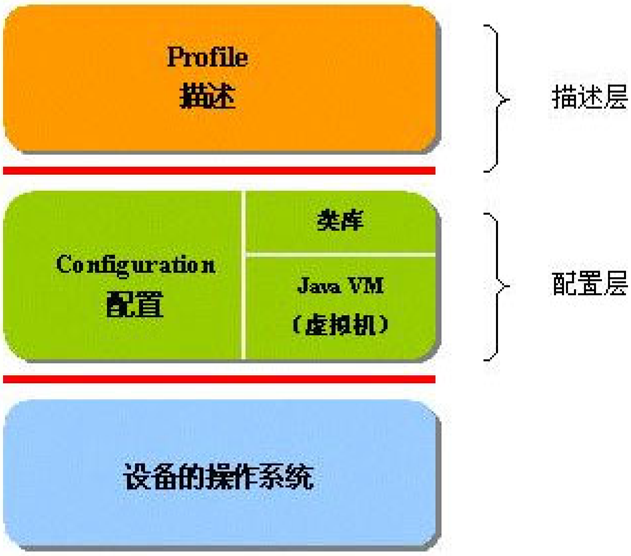

图1-5 J2ME的configuration与profile

Configuration

设备的配置（configuration）为这些设备定义了一个基本的 J2ME 运行环境，其中包括虚拟机，和核心的 java 类库。可以把配置理解为一个针对某一族设备的最小的 Java 平台。其中包括满足该族设备的 Java VM（虚拟机）功能的最小子集和针对该族设备的 Java 类库的最小集合。要注意的是，configuration 主要针对的是系统级的特性，如基本的 Java 语言的特性，虚拟机的特性，以及系统级的 java 核心类库等。

Profile

描述（profile）是构建在配置（configuration）层之上的一层类库。它比 configuration 的针对性更强。它提供 configuration 层中所缺失的功能以支持特定的设备。这些功能包括：对 UI、对持续性存储的支持等。Profile 层提供了应用程序级的接口，应用程序就建立在 Profile 层上。一个 configuration 可能会有好几个 profile（也就是说一个设备可以支持多个 profile）。请注意 profile 针对的更特定的设备，从功能划分的角度来说也是如此。例如，针对某些移动设备，可能会定义一个 SMS profile 以提供收发短消息的功能。这种功能就只能定义在 profile 中，而不能定义在 configuration 中，因为短消息这种功能是只针对手机等移动设备的。

上文抽象的介绍了一下 configuration 和 profile 这两个概念，下面具体的介绍一下 J2ME 中的 configuration 和 profile。

J2ME Configuration

J2ME 配置（Configuration）包括两种，分别是连接设备配置（CDC）和连接受限设备配置（CLDC）。正如前面提到过的，配置分别针对的是某一族系的设备。J2ME 所支持的设备主要分为两种，分别通过 CDC 和 CLDC 支持。

- 个人移动信息设备 指那些可以进行间隙性网络连接的设备如移动电话，双向寻呼机，PDA 等。这类设备从性能上来讲属于低端设备，由 CLDC 支持。

- 共享连接信息设备 指那些网络连接固定，不中断的设备如电视机顶盒，互联网电视，可试电话等。这类设备从性能上来讲属于高端设备，由 CDC 支持。

CDC

连接设备配置（Connected Device Configuration），目标设备总内存大于 2MB，其内存可以为 RAM，ROM 或是 Flash（闪存）。其虚拟机支持 Java2 虚拟机的全部功能。CDC 的虚拟机被成为 CVM（compact VM）。这里需要注意的是 CVM 虽然在功能上支持 Java2 虚拟机的全部功能。当并不等同与 J2SE 的虚拟机，为了支持特定的设备，CVM 做了从新的设计。所以从严格的意义上，图 1-3 中 CDC 下方的虚拟机应写为 CVM 更确切。但从功能的角度来讲，可以把它粗略地理解为 JVM。CDC 包含的类库与 J2SE 之间的包含关系如图 1-6 所示。

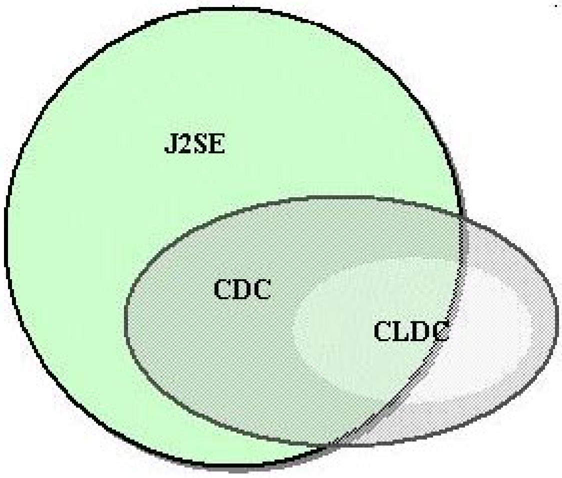

图 1-6 CDC,CLDC 与 J2SE 之间的关系

CLDC

连接受限设备配置（Connected,Limited Device Configuration ，CLDC）

CLDC 面向的目标设备是小型的，资源有限，连接受限的设备。这些设备的内存在 160~512KB 之间，处理器速度较慢，通常是靠电池给设备供电，并且网络连接常通表现为间歇性连接而且带宽有限。CLDC 为这些设备定义了一个的小型的 Java 平台（其实作的表现就是 KVM，K 虚拟机），并允许动态的向这些设备发布内容与应用程序，来方便第三方软件厂商与开发者向这些设备移植内容与应用程序。

CLDC 提供了对基本 Java 语言的支持[1]。CLDC 总的来说是与 Java 语言完全兼容的，但需注意以下几点：

- 不支持浮点数据类型[2]。即没有 float 这种数据类型。类库中也没有包含 `java.lang.Float` 这个类。之所以这样做，主要是出于对 CLDC 所面向设备的硬件功能方面的考量，如设备内存有限，大部分硬件不支持浮点等。
- 不支持类实例（class instance）的终结（finalization），即不存在 Object.finalize() 方法。
- 有限的错误处理（Error Handing），即大部分 Java.lang.Error 的子类都未被支持。这主要是由于设备的内存的限制。

CLDC 基于 Java 虚拟机规范（Java VM spec）[3]定义了专门针对于小型消费电子设备（如前文所述）的虚拟机。实质上 CLDC 在其严格的内存限制下达到了对 Java 虚拟机相当程度的兼容性。不过还是有些需要特别指出的不同点。请注意下述列出了 CLDC 中缺失的或改变了的 Java 虚拟机特性：

- 不支持浮点数据类型（没有 float 和 double）。
- 不支持 JNI (Java Native Interface)。
- 不支持用户自定义的 Java 级的类载入器（class loaders ）。
- 没有反射(reflection)[4]特性。
- 不支持线程组（thread groups）或守护线程（daemon threads）。
- 不支持类实例（class instance）的终结（finalization）。
- 没有弱参考（weak references）[5]。
- 有限的错误处理（error handling）。

---

[1] 对 java 语言的支持基于 Java Language Spec。请参考 *The Java Language Specification* by James Gosling, Bill Joy, and Guy L. Steele. Addison-Wesley, 1996, ISBN 0-201-63451-1.

[2] 这一点在未来的 CLDC1.1 spec 中将有所改变，从其草案的公开讨论稿中可以看出，对于浮点的支持被加了回来。

[3] 对于 Java VM spec 请参考 *The Java Virtual Machine Specification* by Tim Lindholm and Frank Yellin Addison-Wesley, 1996, ISBN 0-201-63452-X.

[4] reflection，java 通过 java.lang.reflect 提供的一种功能，可以对类的能力进行分析。常用来对类的结构进行检查，在 JavaBeans 上有广泛运用。可参见 Gary Cornell 的 Core Java2 volumeI,II

[5] weak reference 提供了一种解决指向已经被 garbage collector 清除的对象的方法。在 Java2 中被介绍。具体请参考 java.lang.ref API 文档，和 sun 网站上关于 Reference Objects 和 Garbage Collection 的文章。

- 改变了的 Class 文件审核模式（class file verification）。CLDC 中新定义了一种两步审核模式，具体请参看后文。

根据 CLDC 规范，sun 在 CLDC 参考实现（CLDC reference implementation，CLDC RI）中实作了一个虚拟机，这就是我们常说的 KVM——K 虚拟机。这个“K”来自英文 Kilobyte（KB），寓意是该 VM 针对于内存以 KB 来计量单位的小型设备。

CLDC 包含了一个基本的 J2ME 运行环境，其中包括虚拟机，和核心的 java 类库。如果对 CLDC 所包含的类库进行一下考察，我们就会发现：由于 CLDC 专门针对于小型设备，它对 J2SE 类库进行了大量的简化[1]，其类库只保留了 Java 规范中定义的最核心的 3 个包，即 java.io，java.lang 和 java.util，并重新定义了一个新的包：javax.microedition。这里，你可以通过前缀来区别：java.表示核心的 java 包，javax.表示标准 java 扩展包。CLDC 与 J2SE 之间的关系如图 1-6 所示。

这里要注意的是在 CLDC 中定义的 javax.microedition 包为 javax.microedition.io，用来支持通用连接框架（GCF，Generic connection framework）[2]。CLDC 中的包和所对应的功能如表 1-1 所示。

表 1-1 CLDC 所包含的包与对应的功能

CLDC 包	对应的功能
java.io	标准 Java I/O 包，J2SE I/O 包的子集
java.lang	核心的语言包，J2SE 的子集
java.util	实用类，J2SE 标准实用类的子集
Javax.microedition.io	CLDC 的通用连接框架类和接口

Javax.microedition 中其他的包在 MIDP 中定义，来支持 CLDC 中所没有涉及的功能，如对应用程序 life-cycle 的管理,UI 界面,Event 处理,永续性存储等。详见后文关于 MID Profile 的介绍。

CLDC 中的类审核模式

CLDC 要求其底层的虚拟机能够辨别并拒绝非法的 class 文件。但由于 CLDC 本身面向小内存消耗的小型设备这一前提。其类文件检测机制与 J2SE 中定义的标准类文件审核机制有所不同。CLDC 中有一步其特有的预审核（preverification）过程。这是 CLDC 区别于通常的类文件审核过程的关键。

[1] 这些核心的 Java 包也是缩了水的，例如对于 java.util 包，由 J2SE 的 47 个类，缩小到 CLDC 的 10 个类。

[2] 关于 GCF，请参考第 7 章 MIDP 网络编程。

如图 1-7 所示，当程序的源程序被编译后，必须被预审核器预审核，然后才能生成可以被下载到目标设备上运行的类文件。之所以有这一步骤，主要是为了减轻目标设备上虚拟机中审核器的负担，加快其审核速度。所以把一部分的审核任务放在预审核器中完成。请注意，在 CLDC/MIDP 环境下开发程序，其程序经过编译后，必须经过预审核后，才能运行。

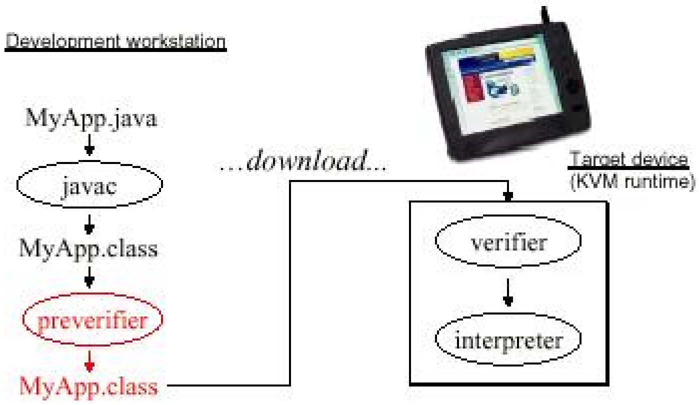

图 1—7 CLDC 的类文件审核机制

J2ME Profiles

Foundation profile

位于 CDC 上层的 profile。是 J2SE1.3 功能的子集，基本上涵盖了 J2SE 的主要功能。不过没有 GUI 的支持。（AWT 与 Swing 包没有包括在 Foundation profile 中）。对 GUI 的支持可以通过 Personal profile 来获得。

Personal 和 personal Basis profile

Personal profile 源于 sun 早先的 PersonalJava 应用环境[1]。Personal profile 可以提供 PersonalJava 的全部功能，如对 AWT 的全部支持等。用于在 Foundation Profile 上层提供对 GUI 的支持。

personal Basis profile 相对于 personal profile 更小。针对与设备要求更简单的 GUI 的场合。

[1] Personaljava3.1,sun 早先为消费性设备与嵌入式设备提供的平台。使用的是 JDK1.1API，详细情况请参考 Sun 网站。

RMI Profile

位于 CDC 的 Foundation profile 上层的 profile。以支持 RMI（Remote Method Invoke，远程方法调用）。

MID Profile

CLDC 为那些资源受限，不足以支持整个 J2SE 虚拟机环境以及 J2SE 核心类库的设备提供了运行 Java 程序的基础。然而，作为一位应用程序开发者，如果您只能通过 CLDC 提供的 API 来进行编程是几乎不可能的，因为 CLDC 中并没有提供给我们与用户、存储设备、网络直接交互的工具。CLDC 是一个基础层，其上层可以架设一系列的描述(profiles)层来提供 CLDC 所缺失的功能。每一种 profile 被设计成适应某种类型的设备的形式，而移动信息设备描述（Mobile Information Device Profile，MIDP），就是这些 profile 中的一种。

MIDP 是位于 CLDC 上层的 Profile，是目前 J2ME 平台中发展相对最成熟，最广为人知的 Profile。如同 MIDP 的名字所描述的，其针对的是移动信息设备（Mobile Information Device，MID），这类设备通常指手机或是 PDA，它们在屏幕、内存、处理器等硬件特性上有诸多限制，所以说在 MID 上开发应用程序必须要考虑一些技术上的特殊点。

下面给出 MIDP 规范中定义的移动信息设备的特性：

- 显示：96x54 （最小屏幕尺寸），
- 色深：1bit（最小色深，单色）
- 可能的输入设备：单手键盘（one-handed keyboard，指 ITU-T 手机键盘）；双手键盘（two-handed keyboard，指标准键盘，即 QWERTY 键盘，当然大小要比 PC 机用的小的多）；触摸屏
- 内存：
  128kb——用于 MIDP 组件。
  8kb——应用程序生成的永续数据（Persistent data）[1]
  32kb——Java 运行时环境（Java runtime）。
- 网络：双向的，无线的，间断的，带宽有限的网络
- 设备内核：至少要能运行 KVM
- 其他特性，如读写 non-volatile 内存（就是掉电后不会失去内容的内存，如 flash）。读写无线设备接口的 API，等等。

[1] 永续数据存储会在将来讲 RMS 时详细说明，这里有个概念就行了

如前文所述，MIDP 定义于 CLDC 的上层，用以提供对 UI、永久存储介质（persistence storage）、和网络等更高层的（相对于 CLDC）支持。那么，让我们来具体看看 MIDP 的类库。MIDP 由四个 javax.microedition 包组成，它们包括：javax.microedition.rms，javax.microedition.midlet，javax.microedition.io，javax.microedition.lcdui。如果在加上 Java 核心包（即 java.lang，java.io 和 java.util）则共有 7 个包。MIDP 中的包和所对应的功能如表 1-2 所示。

表 1-2 MIDP 所包含的包与对应的功能

MIDP 包	对应的功能
javax.microedition.io	MIDP 对网络支持
javax.microedition.lcdui	MIDP 的 UI 包
javax.microedition.midlet	定义了 MIDlet 的框架，以及 MIDlet 与环境的交互
javax.microedition.rms	MIDP 的 RMS 对永续存储介质的支持
java.io	标准 Java I/O 包，J2SE I/O 包的子集
java.lang	核心的语言包，J2SE 的子集
java.util	实用类，J2SE 标准实用类的子集

Pda profile

位于 CLDC 上层的 profile。目标是专门用来支持 PDA 设备（如 Palm 等）。目前还在定义中，还没有参考实现（RI）被实作出来。根据其草案，其目标之一是集成 CLDC 的 MIDP 和 CDC 的 Personal Profile 各自一部分的功能，如 PDAP 中会提供 AWT 的支持。

第二章 CLDC/MIDP 编程概述

2.1 J2ME 的 CLDC/MIDP 软件体系架构

目前 J2ME 平台主要分为两大体系，一是基于 CDC 的 CVM\CDC\Foundation profile 体系，该架构针对于内存需求相对较大（于 CLDC 相比），一般大于等于 2MB，处理器能力较强的设备，如电视机顶盒等。另外一种是基于 CLDC 的 KVM\CLDC\MIDP 体系，该架构针对于小内存（160～512KB）,资源受限的小型设备如手机，PDA 等。目前 J2ME 发展较为成熟的是 CLDC/MIDP 体系，这也是本文所要讨论的范围。J2ME 的 CLDC/MIDP 的软件体系架构如图 1-7 所示：

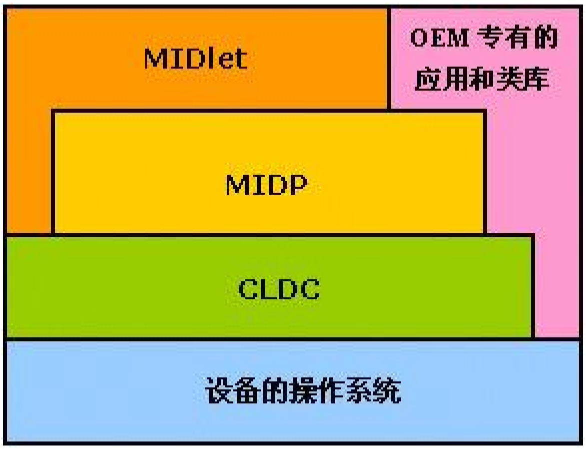

图 2-1 J2ME 的 CLDC/MIDP 软件体系架构

图 2-1 展示了 CLDC/MIDP 的软件体系架构和 MIDP 在软件体系中的逻辑位置。在 MIDP 的上层的应用程序被称为 **MIDlet**。这些 MIDlet 在底层 CLDC 的支持下，通过 MIDP 中提供的包获得更多的功能（其中包括 UI，网络，永续性存储等）。如图 1-7 所示，MIDlet 可以直接使用 MIDP 本身提供的类库以及 MIDP 从 CLDC 中继承的那部分 API。MIDlet 本身并不能直接访问其底层设备的操作系统并且也不可能做到这一点，除非你的版本是移植到特定设备平台上的。OEM 特有的应用程序（也可以称其为 OEM 特有的 MIDlet）可以通过某设备厂商提供的专用的类库来获得更多额外的功能，例如 Motorola 和 siemans 其无线开发包中都有其附加的库，以提供其设备特有的一些功能，其中有些可能访问了底层设备的操作系统。因为 KVM 不支持 JNI（java native interface），所

以要使 MIDlet 访问本地平台功能的唯一的办法是把本地代码链接到一个客户自定义的虚拟机上。这里请注意可以支持 CLDC/MIDP 的 VM 并不是只有 KVM 一种，只不过 KVM 是 sun 对 CLDC/MIDP 的一个参考实现而已。如同 MIDP 规范中指出的，设备厂商应该把参考实现移植到其设备的硬件平台上，这时厂商可能需要添加一些特有的代码来填补参考实现与特定设备之间的区别，或是增加一些特有的功能。厂商完全可以自定义自己的 VM，而且设备厂商一般都会这样做。而且也有一些软件制造商专门制作 VM，例如 IBM 的 J9 VM 等。

2.2 MIDlet 及其运行机制

在前面一节中我们介绍了 CLDC/MIDP 的软件体系结构,也给出的 MIDlet 的概念——在 MID 设备上运行的 Java 程序被称为 MIDlet（这种命名的方法如同我们熟悉的 Applet）。下面将深入讨论一些 MIDlet 所涉及的内容。

MIDlet

与通常的 Java 程序相比，MIDlet 有比较大的不同，从某种意义上来说，MIDlet 更类似与 Applet。简而言之，与 J2SE 程序相比，MIDlet 没有 main()这个程序初始入口点。同时 MIDlet 也不能调用 Runtime.exit()和 System.exit()方法来中止虚拟机的运行。（如果调用的话将会抛出 SecurityException 异常）。

MIDlet 是在 MID 设备上运行的 Java 程序，每一个 MIDlet 必须派生自抽象类 *javax.microedition.midlet.MIDlet*。MIDlet 的继承体系如图 2-2 所示:

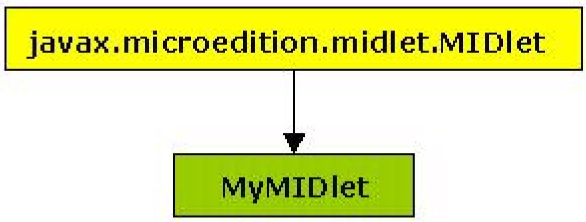

图 2-2 MIDlet 的继承体系

下面给出一个具体的 MIDlet 示例,如示例 2-1 所示:

示例 **2-1** 一个简单的 **MIDlet**

```
//HelloWorld.java,一个最简单的 MIDlet 程序。

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloWorld extends MIDlet implements CommandListener {

    private Command exitCommand;
    private TextBox tb;

    public HelloWorld(){
```

```
        exitCommand =new Command("Exit",Command.EXIT,1);
        tb =new TextBox("Hello MIDlet","Hello,World!",15,0);
        tb.addCommand(exitCommand);
        tb.setCommandListener(this);
    }
    protected void startApp(){
        Display.getDisplay(this).setCurrent(tb);
    }
    protected void pauseApp(){
    }
    protected void destroyApp(boolean u){
    }
    public void commandAction(Command c,Displayable d){
        if (c ==exitCommand){
            destroyApp(false);
            notifyDestroyed();
        }
    }
}
```

HelloWorld.java 编译后将生成一个简单的 MIDlet，该 MIDlet 可以在设备屏幕上显示“HelloWorld”字符串。程序本身非常简单，如果你熟悉 JavaApplet 编程的话，你会发现与 Applet 很类似。想要编译该程序需要 JDK1.3 和 Sun 的 MIDP 包（或是 JDK1.3 和 Sun Wireless ToolKit 1.0.3）具体编译步骤和方法请参考相关软件的说明。[1]

如果使用 J2ME Wireless Toolkit，MIDlet 在模拟器中的显示，如图 2-3 所示

[1]相关软件可以在 sun 网站下载:java.sun.com/j2me/midp

图 2-3 HelloWorld MIDlet 在模拟器中的显示

MIDlet 的 lifecycle

下面将分析 HelloWorld MIDlet 的结构，介绍其的运行的 Lifecycle。

对源程序进行观察，可以发现 MIDlet 程序的运行是由 startApp(),pauseApp()和 destroyApp()这 3 个方法控制的。它们都在 javax.microedition.midlet.MIDlet 中定义。所有的 MIDlet 都必须实现这 3 个方法。

startApp()方法用于标志一个 MIDlet 的开始执行。与 HelloWorld 程序的 constructor 不同。startApp()不光是在初始化完一个MIDlet时执行，只要该MIDLet被从Paused态激活（变为Active 态），startApp()就会被调用。

pauseApp()方法标志着 MIDlet 进入 Pause 态。而 destroyApp()方法标志着 MIDlet 进入 destroyed 态。[1]

MIDlet 的执行是通过 AMS（Application Management software）来管理的。AMS 是位于操作系统级别上来管理 MIDlet 运行的底层机制的总称，所谓 MIDlet state（MIDlet 状态）就是它一手操办，控制管理的。MIDlet state 确保了 AMS 随时可以消灭该 MIDlet，同时 MIDlet 请求进入 Pause 态，当需要时再次激活。

MIDlet State 分为 Paused，Active，destroyed 三种。当 AMS 创生一个新的 MIDlet 实体时，对应于 MIDlet，表现为其 constructor 被调用，进入 Paused 状态。当所有的准备工作都做好后，AMS 判断现在 MIDlet 可以运行了，于是调 MIDlet.startApp()方法。进入 Active 态。当 AMS 决定要把 MIDlet 转入 Paused 态，就会调用 MIDlet.pauseApp()方法，MIDlet 就会暂停执行，通常 Paused 态会用于释放所占资源。当 AMS 判断 MIDlet 不再需要，就会调用 MIDlet.destroyApp()，MIDlet 被消灭。

[1]注意：这里严格的讲应该说成：方法被调用并成功返回标志着 MIDlet 进入某个状态。

上述说明了 AMS 如何控制 MIDlet 的状态改变。程序员也可请求 MIDlet 的状态的变换，通过调用 resumeRequest, notifyPaused, notifyDestroyed 这三个方法。

我们的 HelloWorld 程序先把 destroyApp()的 unconditional 值置为 false，抛出一个 MIDletStateChangeException 异常，表示 MIDlet 这时还不想被 destroy。notityDestroyed()通知 AMS：MIDlet 进入 destroyed 态。具体的细节请参阅 MIDP API 文档。MIDlet 的状态的改变可以用图 2-4 表示：

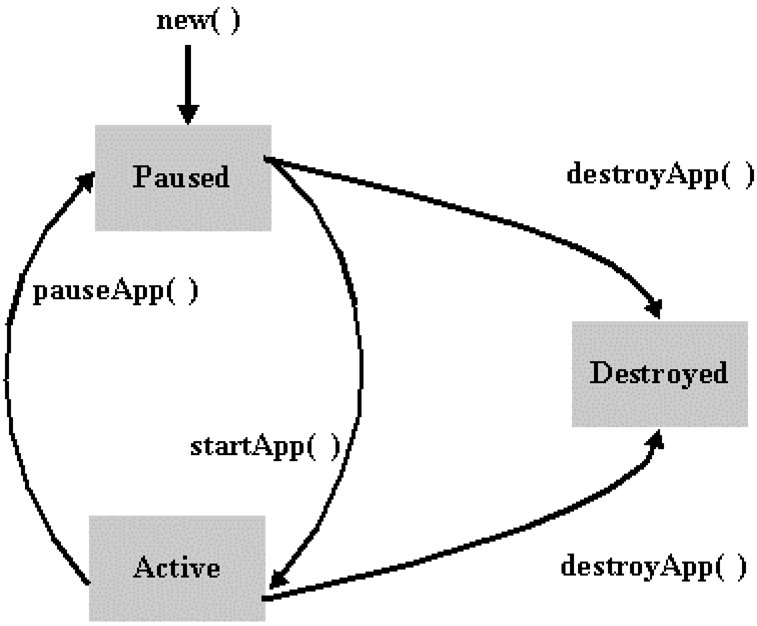

图 2-4 MIDlet 的 lifecycle

MIDlet suite

通常一组相关的 MIDlet 的集合，被称为 MIDlet suite（MIDlet 套件）。MIDlet suite 是安装发布到目标设备（或是从设备上卸载移除）的基本单位。MIDlet suite 中的所有 MIDlet 都共享同一名空间，这也是 RMS 中，同一 MIDlet suite 中的 MIDlet 之间可以访问对方的 Record Store 的原因[1]。

MIDlet suite 被封装到一个 JAR 中。这在 MIDP 规范中被称为 MIDlet suite Packaging。在 MIDlet suite 被打包的 JAR 文件中包含下面几个部分：

- 继承自 javax.microedtion.midlet.MIDlet 的类文件和其他用于实现该 MIDlet 的类文件。
- 资源文件 如图片等。

[1]见 2.3 节 可持续存储与 RMS

- 清单文件（Manifest file）用来描述 JAR 文档中的内容

除了在 JAR 文件的内部含有描述 MIDlet suite 打包信息的清单文件（manifest）外，在 JAR 文件的外部，MIDP 还定义了一个被称为 JAD（Java application descriptor）文件，即描述文件的文件，它也被用来描述 MIDlet suite 的信息。图 2-5 表示 MIDlet Suite 所包含的内容：

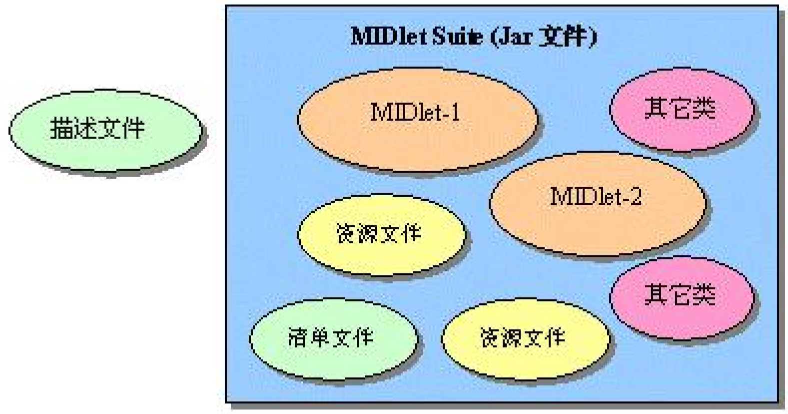

图 2-5 MIDlet Suite

清单文件和描述文件都是纯文本文件，格式也相同，文件中的每行都遵守"属性名：属性值"的格式。

MIDlet 的信息通过属性名和属性值来定义，这些属性可以通过 MIDlet.getAppProperty 方法来获得。MIDlet 属性区分大小写，以"MIDlet-"开始的属性名是 MIDP 内部保留使用的，开发者可以根据需要定义自己的属性，但不能是以"MIDlet-"前缀为开始。MIDP 规范中定义了一系列的属性，见表 2-1。

表 **2-1 MIDlet** 属性

属性名	属性值及其对应的意义
MIDlet-Name	MIDlet Suite 的名称
MIDlet-Version	MIDlet Suite 的版本号码。其格式为：「主要号码.次要号码.微号码」。系统可以利用它来决定是否安装或升级。缺省值为 0.0.0
MIDlet-Vendor	提供此 MIDlet Suite 的厂商名称。
MIDlet-Icon	位于 JAR 文件中的 PNG 格式的图片文件名。系统会使用该图标来标志 MIDlet suite
MIDlet-Description	MIDlet Suite 的相关信息的简要描述。

MIDlet-Info-URL	进一步描述此 MIDlet Suite 其相关信息的 URL 地址
MIDlet-<n>	用来描述 MIDlet Suite 之中 MIDlet 的信息。<n>是一个自然数，即第一个 MIDlet 用 MIDlet-1 表示，第二个 MIDlet 用 MIDlet-2 表示，注意 n 的号码要连贯，其它依此类推。属性值的格式为："MIDlet 的名称,图标,类名" 其中，MIDlet 名称由程序开发者自己设定，图标必须要是位于 JAR 文件里头的 PNG 格式图形文件（可缺省），而类名必须是完整的"包名.类名"结构。而且该类继承自 javax. microedition.midlet.MIDlet。
MIDlet-Jar-URL	下载该 MIDlet Suite 的 JAR 文件的 URL 地址。
MIDlet-Jar-Size	代表此 MIDlet Suite 的 JAR 文件的大小(以字节为单位)
MIDlet-Data-Size	MIDlet Suite 所需要的永续储存区(persistent data)的最小字节数。该值预设为 0。
MicroEdition-Profile	执行此 MIDlet Suite 所需的 J2ME 描述（profile）的名称和版本。格式为："Profile 名称-Profile 版本"。如:MIDP-1.0。其值与系统属性 microedition.profiles 一致。
MicroEdition-Configuration	执行此 MIDlet Suite 所需的 J2ME 配置（configuration）名称及版本。格式为:"Configuration 名称-Configuration 版本"。如:CLDC-1.0。其值与系统属性 microedtion.configuration 一致

注意 MicroEdition-Profile 和 MicroEdition-Configuration 这两个属性还可以通过 System.getProperty()方法获得。对于表 2-1 给出的属性，必须出现在清单文件中的属性有：

- MIDlet-Name
- MIDlet-Version
- MIDlet-Vendor
- MIDlet-<n>
- MicroEdition-Profile
- MicroEdition-Configuration

可选的属性：

- MIDlet-Description
- MIDlet-Icon
- MIDlet-Info-URL

■ MIDlet-Data-Size

图 2-6 展示了一个清单文件的例子，该清单文件来自 J2MEWTK 中的 Demo 程序。

```
Manifest-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MIDlet-Description: Technical demonstration programs for MIDP
MIDlet-Version: 1.0.3
MIDlet-7: ManyBalls, /icons/ManyBalls.png, example.manyballs.ManyBalls
MIDlet-6: Tickets, /icons/Auction.png, example.auction.TicketAuction
Created-By: 1.3.1 (Sun Microsystems Inc.)
MIDlet-5: Stock, /icons/Stock.png, example.stock.StockMIDlet
MIDlet-Vendor: Sun Microsystems, Inc.
MIDlet-4: FontTestlet, , example.fonts.FontTestlet
MIDlet-3: Http, , example.http.HttpTest
MIDlet-2: Properties, /icons/App.png, example.PropExample
MIDlet-1: Colors, /icons/ColorChooser.png, example.chooser.Color
MicroEdition-Profile: MIDP-1.0
MIDlet-Name: SunSamples - Demos
```

图 2-6 清单文件示例（demos.jar\META-INF\MANIFEST.MF）

对于表 2-1 中的属性，必须出现在描述文件（.jad 文件）中的属性有：

■ MIDlet-Name
■ MIDlet-Version
■ MIDlet-Vendor
■ MIDlet-Jar-URL
■ MIDlet-JAR-Size

Jad 文件中的可选属性有：

■ MIDlet-Description
■ MIDlet-Icon
■ MIDlet-Info-URL
■ MIDlet-Data-Size
■ 其它自定义的属性（不以“MIDlet-”开始）

这里需要注意的，自定义的属性（不以“MIDlet-”开头的属性），应该写在描述文件（.jad 文件）中。而位于清单文件中的属性名称都应该以“MIDlet-”打头，如果清单文件与描述文件中有同名属性，那么 MIDlet 在执行时期所取得的属性值要以描述文件中所指定的为准。

图 2-7 给出一个描述文件的示例，同样来自 J2MEWTK 中附带的 demo 程序。

```
Created-By: 1.3.1 (Sun Microsystems Inc.)
MIDlet-1: Colors, /icons/ColorChooser.png, example.chooser.Color
MIDlet-2: Properties, /icons/App.png, example.PropExample
MIDlet-3: Http, , example.http.HttpTest
MIDlet-4: FontTestlet, , example.fonts.FontTestlet
MIDlet-5: Stock, /icons/Stock.png, example.stock.StockMIDlet
MIDlet-6: Tickets, /icons/Auction.png, example.auction.TicketAuction
MIDlet-7: ManyBalls, /icons/ManyBalls.png, example.manyballs.ManyBalls
MIDlet-Description: Technical demonstration programs for MIDP
MIDlet-Jar-Size: 144445
MIDlet-Jar-URL: demos.jar
MIDlet-Name: SunSamples - Demos
MIDlet-Vendor: Sun Microsystems, Inc.
MIDlet-Version: 1.0.3
Manifest-Version: 1.0
```

图 2-7 描述文件示例

清单文件和描述文件有 3 种属性必须相同，它们是 MIDlet-Name，MIDlet-Version 和 MIDlet-Vendor 属性。它们决定着 MIDlet 的版本控制，如果它们不相符，系统会认为 MIDlet 的版本不符，而不会安装该 MIDlet suite。而其它的重复属性它们的值可能会不同，如上文所述，这时描述文件中的值会覆盖清单文件中的值。

MIDP 之所以定义 JAD 文件，就是因为如果只是通过 JAR 文件中的清单文件来描述 MIDlet，那么如果要获得 MIDlet 信息，势必得下载全部的 JAR 文件。这显然是划不来的（Jar 文件可能会较大）。基于这个原因 MIDP 把清单文件中的一部分属性和一些额外的属性集成的一个单另的 JAD 文件中，这样如果想要安装某 MIDlet suite，首先会下载 jad 文件。该文件要比 Jar 文件小得多，这样就节省了大量的时间和宝贵的带宽。设备这时会根据 JAD 文件来显示 MIDlet suite 的内容，以供用户决定是否下载该 MIDlet suite。图 2-8 描述了 MIDlet 的下载与安装过程。

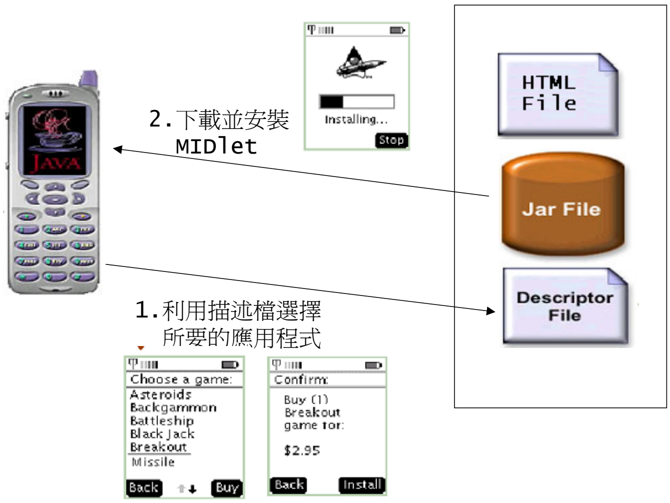

图 2-8 MIDlet 的下载与安装

MIDlet 的执行环境

MIDP 规范定义了 MIDlet 的执行环境，在同一 MIDlet suite 中的所有 MIDlet 共享相同的执行环境,MIDlet suite 中的任一 MIDlet 都可与同一 MIDlet suite 中的其他 MIDlet 交互。在 MIDlet 的执行环境中，MIDlet 可以访问的内容为：

- 实现 CLDC 和 MIDP 的类以及它们的本地代码。
- MIDlet Suite（即 JAR 文件）中的类
- MIDlet Suite 中的资源文件（JAR 文件中所有不是类文件的文件，如图片和文本文件，以及清单文件）。可以通过 java.lang.Class.getResouceAsStream()方法来获得
- 描述文件（即 JAD 文件）通过 javax.microedition.midlet.MIDlet.getAppPorperty()方法来访问。
- 通过 RMS 的 Record store 存贮在永续存储介质中的内容。[1]

图 2-9 展示了这种关系：

[1] 详见 2.3 节 可持续储存与 RMS

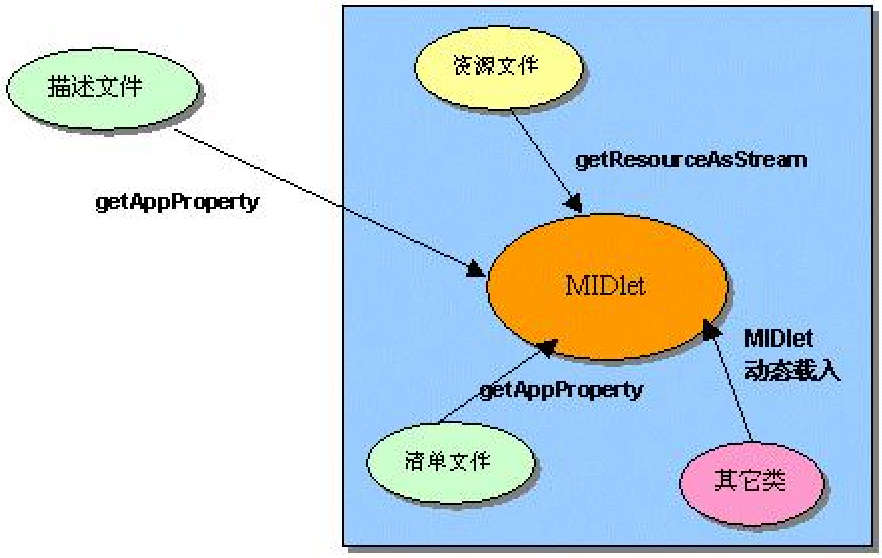

图 2-9 MIDlet 可以访问的内容

2.3 MIDP 用户界面编程概述

对于移动设备来说，其用户界面(User interface,UI)与我们日常熟悉的 PC 机 UI 有很大不同。移动设备的显示范围相比要小的多，而且输出设备没有鼠标和键盘。在移动设备上编程就不能照搬照套在台式机上进行 UI 编程的方法。

对于 CLDC 来说，其本身并没有包含任何的 UI 包。这些 UI 包被定义在 MIDP 中。MIDP 中定义的包并没有支持 AWT (Abstract Window Toolkit)，这些 UI 包是专门针对移动设备而重新设计的。MIDP 之所以不支持已有的 GUI 类如 AWT 和 Swing，原因是主要是因为 AWT 原本是为台式机设计的，而且 AWT 许多功能对于移动设备并不必要。例如，AWT 提供了大量的功能来 支持窗口管理，然而对于显示尺寸受限的手持设备来说，伸缩（resizing）窗口是不切实际的。 因此，AWT 中的窗口与布局（layout）管理器对于手持设备来说是显得很累赘。而 AWT 的事件处理模式，对 CPU 能力与内存大小都有限的移动设备来说，更是不堪重荷。基于上述原因，MIDP 放弃了的对即有 GUI 类的支持。针对移动设备而重新定义了新的 UI 包，即 javax.microedition.lcdui 包

MIDP 的 UI 模式与类库结构

基于上述已经讨论过的原因，MIDP 建立了一个与 AWT 不同，专用于移动设备，并且功能紧缩的 GUI 包。 MIDP 的 lcdui 包包含了 high-level 与 low-level 这两种类型的 API，而且针对这两者分别定义了分别各自的事件处理模式。

high-level API 面向于那些非常注重移动信息设备间可移植性（portability）的应用程序。 为了达到这种可移植性，High-level API 采用了更高级的抽象。因此，对具体 GUI 可做的视感（look and feel）控制很有限。开发者不能自定义这些 high-level 组件的可视化外观（如形状，颜色，字体等）。在一般的 GUI 设计不同，与 UI 组件的交互（如导航条，窗口的滚动，用户界面之间的联系等）全部都由具体的实现来执行，开发者不需要关心这些交互的过程。底层的具体实现会根据设备的硬件特性与 UI 风格而做出必要的配接工作，这样就使得应用具有良好的可移植性。注意，所有实现了 High-level API 的类都继承了 javax.microedition.lcdui.Screen 类。

low-level API 面向的是那些需要精确定位和控制图像元素，以及需要获得 low-level 输入事件的应用程序。 该 API 使开发者可以全面掌控 display 上所画之物。low-level API 由 javax.microedition.lcdui.Canvas 与 javax.microedition.lcdui.Graphics 这两个类实现。注意，并不是所有访问 low-level API 的 MIDlet 都能确保可移植的，这是因为该 API 中也提供了某

些机制来访问一些底层细节，而这些细节可能与某种具体设备密切相关，这些就丧失了一部分的可移植性。

MIDP GUI 模型

MIDP GUI 模型的核心是屏幕（screen），屏幕在 MIDP 规范中抽象为一个封装了设备特有的图形显示和用户输入的对象。而每个 MIDlet 会有唯一的一个 display 对象（javax.microedition.lcdui.Display），它管理着屏幕的显示。当调用 Display 对象的 setCurrent( ) 方法时，屏幕就被显示。一个应用程序可以有多个屏幕，而在一个 Display 对象中，同一时间只能显示一个屏幕。屏幕只是 MIDP 规范中定义的一个抽象，从开发者的角度来说，具体指的就是 displaybale 对象。共有两种类型的 Dispalyable 对象。

- Screen：封装高级的 UI 对象，如 Alert，List，TextBox，Form 等 UI 组件。
- Canvas：允许开发者自己定义处理图像和用户输入的低级对象。

MIDP 的 UI 类的之间的层次结构如图 2-10 所示：

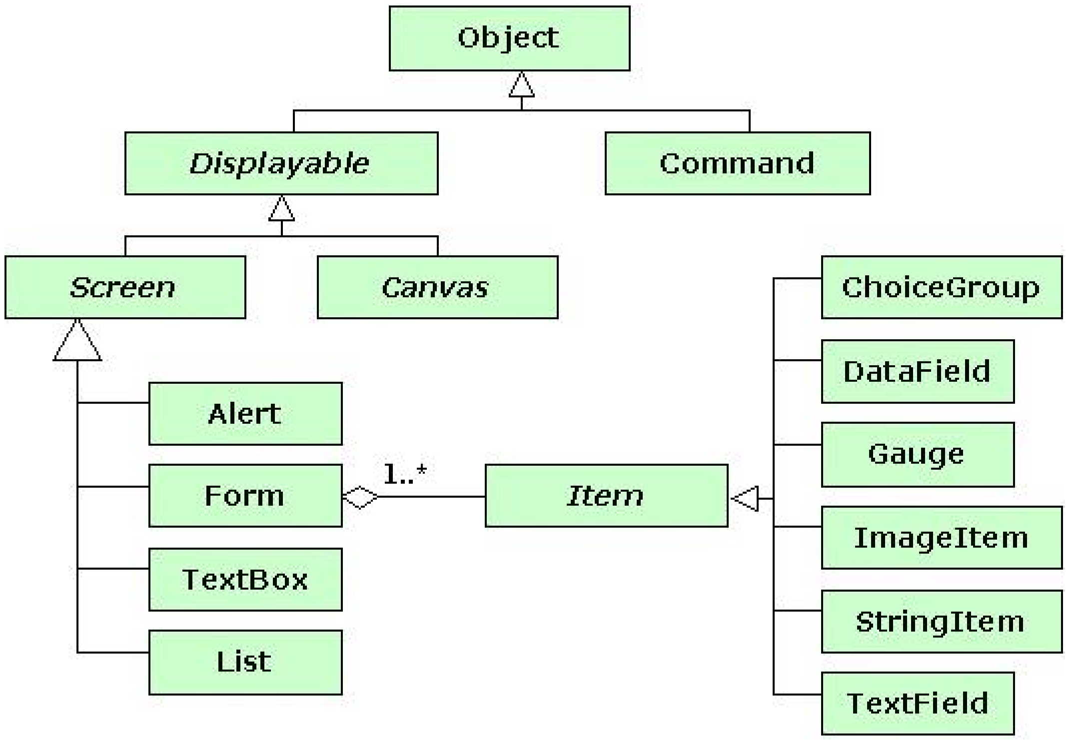

图 2-10 MIDP UI 库的类层次图

MIDP 的 **UI** 包

MIDP 的 GUI 类都包含在 javax.microedition.lcdui 包中，其中类的层次结构如图 2-10 所示，表 2-2 给出了这些类的简单介绍。

表 **2-2 MIDP** 之 **lcdui** 包中的接口与类

接口	简要介绍
CommandListener	应用程序通过该接口接收 high-level 事件。
ItemStateListener	应用程序侦听该接口判断 item 的内部状态是否改变
Choice	定义了 List 和 ChoiceGroup 中所含元素的公用 API
类	简要介绍
Alert	Screen 的子类，可以在处理下一屏幕之前，给用户显示数据并等待一定的时间的 GUI 组件
AlertType	标志 alert 性质的一个实用类
Canvas	Low-level GUI 的基类，用来处理 low-level 事件并可以通过调用 graphic 来重画 display 对象
ChoiceGroup	可置于 Form 中的一组可选择元素
Command	一种封装了 action 语义信息的结构
DateField	用来表示日期或时间的组件，该组件应被置于 Form 中
Display	系统显示与输入设备的管理器
Displayable	有能力被置于 display 之上的一种对象
Font	用来表示字体和字体属性的类
Form	Screen 的子类，可包含 item（如 images, text, text fields，choice groups 等）的 GUI 组件
Gauge	一个条状图形来显现一个值，用于 form 中，可理解为 MIDP 中的进程条。
Graphics	图形类，用来提供简单的二维几何变换能力。
Image	图片类，用来放置图形图片的数据
ImageItem	当 Image 对象被加载到 form 或者 alert 上时，用来提供布局控制

Item	所有可以被加载到 Form 上的组件的超类（也有一些 item 可以被加到 Alert 上）
List	Screen 的子类，显示一个选项或几个选项的列表。
Screen	所有 high-level UI 类的超类。
StringItem	一种可包含 String 的 item
Canvas	Low-level GUI 的基类，用来处理 low-level 事件并可以通过调用 graphic 来重画 display 对象
TextBox	Screen 的子类，允许用户输入并编辑文本。
TextField	可编辑的文本组件，被放置于 Form 中。
Ticker	显示一行不停滚动滑过屏幕的文本。可被加载于所有的 Screen 对象上，但不能加载到 Canvas 上。
Display	系统显示与输入设备的管理器

MIDP GUI 编程的一般模式

想要在 MIDP 设备上显示东西，首先你需要获得设备的 Display 对象，然后使用 display 对象的 setCurrent()方法把 Displayable 对象显示出来。这就是 MIDP GUI 编程的一般思路，其中 Display 对象起到了设备的显示管理器的作用。一个 MIDlet 终其一生只能获得一个 Display 对象的实例。

通过给 Display 对象的静态方法 getDisplay()提供一个 MIDlet 的 reference[1]，该 MIDlet 就获得了 Display 对象的实例。通常会在 MIDlet 的 startApp()方法中进行这一过程。

```
...
public void startApp( ) {
     display = Display.getDisplay(this);
   }
...
```

然后通过把可以把 Displayable 对象传递给 Display 的 setCurrent()方法，来显示该 Dispalyable 对象。

```
public void setCurrent(Displayable d);
```

[1] getDisplay()方法的函数原型是 public static Display getDisplay(MIDlet);

```
public void setCurrent(Alert alert,Displayable nextDisplayable);
```

同时 Display 对象还提供了 getCurrent()，可以用来返回当前被显示的 Displayable 对象的 reference。

```
public Displayable getCurrent();
```

抽象命令（**Command** 对象）与 **Command** 的侦听

由于 MIDP 其 UI 需要适应数量广泛，特性可能互不相同的目标设备，所以 MIDP 需要高度抽象其输入机制。也就是说，理想情况下，开发者不需要考虑设备的特殊性。例如键的数量与位置，键的绑定等。MIDP 定义了抽象的 Command 机制，而具体的实现根据与该设备想适合的机制来提供支持。对于开发者来说，Command 对象就类似于普通 GUI 编程的 button，可以设定其标题，如（ok，cancel，goback 等）；当用户调用该 command 是应用程序会做出适当的相应。而具体 command 如何在屏幕上显示，以及如何于具体设备的按键等建立联系是具体实现的责任，开发者不需要考虑。

Command 对象有 3 个参数，分别为：

- Label ：command 的标题，也就是 Command 所显示的内容，如 OK 等
- CommandType：这个值用来表示 command 的类型，大多数的设备都会有一个标准的键盘及其功能的映射方法，会根据这个值来映射 command 对应的按键以及功能。例如如果 CommnadType 被设定为 BACK，具体实现会将该 command 映射到设备的"goback"键（一般设备都会有）和这个键对应的标准的"回退"操作上。
- Priority：command 的优先级。

如一个显示"OK"的 command：

```
Command c =new Command("OK ",Command.OK,0);
```

Command 的事件处理模式采取的是标准的 Listener 模式，需要使用 Displayable 对象的 setCommandListener()方法来注册 Command 的 Listener：

```
public void setCommandListener(CommandListener l)
```

这样，Command 事件就传送到 MIDlet 的 *CommandListener()*接口，应用通过实现 commandAction()来处理相应的 Command 事件：

```
public void commandAction(Command c, Displayable d)
```

前文所给出的示例 2-1，HelloWorld 程序说明了这一过程。

Displayable 类与 Command 类及 CommandListener 接口的 UML 图，如图 2-11 所示

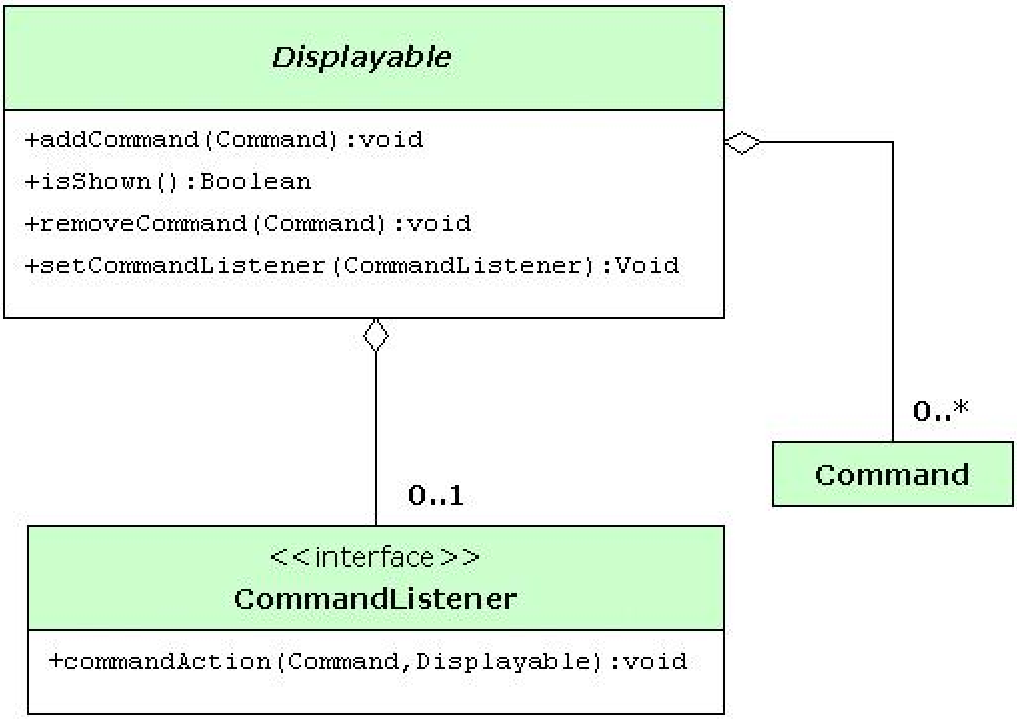

图 2-11 Displayable,Command 和 CommandListener

High level 界面编程

使用 Screen

High-level UI 组件主要指 Screen 的 4 个子类（list，TextBox，Alert，Form）。这些 Screen 的子类都封装了功能完全的 UI 元素。Screen 类是一个抽象类，每个 Screen 都具备如下两种属性：标题（title）和 ticker。screen 的 title 是显示在文档内容上方的一个简单的文本串。ticker 是一个可以在标题的上方显示一行滚动信息的图形组件。这两种组件是否被显示都是可选的，默认情况下，标题会被显示，而 ticker 不会被显示。 如果不显示 ticker，ticker 所占用的显示区域会被屏幕替代。

Ticker 与 title 如图 2-11 所示

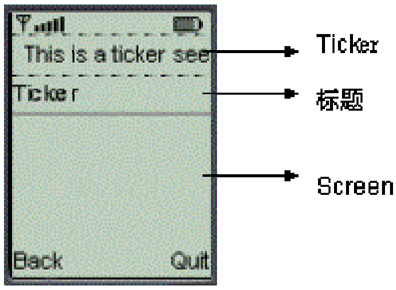

图 2-11 Screen 对象的 Ticker 与 Title

Screen 类中定义了下面的方法被用来分别设置和获取 screen 的标题和 ticker：

```
public void setTitle(String title);
public String getTitle();
public void setTicker(Ticker ticker);
public Ticker getTicker();
```

下面分别介绍 Screen 的 4 个子类 list，TextBox，Alert 和 Form。

List

List 是包含一个选择列表的 Screen。List 中的每个选项由一个文本串和一个可选的图标组成。有 3 种类型的 List，它们分别是 implicit 型（隐含型），exclusive（互斥型）和 multiple choice 型（多选型）。

- 隐含型 快速选择单一的项目，类似下拉菜单，焦点指向的选项会被默认选择
- 互斥型 一次只能选择一个选项，与单选框（radio button box）类似
- 多选型 可以选择多个选项，类似于复选框（checkbox）。

图 2-11 展示了上述 3 种 list 类型。

隐含型 List　　互斥型 List　　多选型 List

图 2-11 3 种 List 类型

可以在 List 的构造器，设定 listType 变量来指定其类型。即通过把 ListType 分别设为 IMPLICIT（隐含），EXCLUSIVE（互斥），和 MULTIPLE（多选）。如生成一个标题为"Choose one"的互斥型 list：

```
List list = new List("Choose one", Choice.EXCLUSIVE);
```

TextBox

TextBox 是允许用户输入并编辑文本的 Screen。为了创建 TextBox 对象，需要了解其构建器的参数。TextBox 的构建式为：

```
public TextBox(String title, String text, int maxSize, int constraints);
```

TextBox 构建器的 4 个参数中，title 用来表示 screen 的标题。text 和 maxSize 用来决定 textbox 的初始（预设）文本和 最大尺寸，而通过 constraints 变量（限定符变量），可以给 TextBox 设置输入限制。限定符使得设备可以使用输入限制来使用户更容易进行输入，并获得符合一定格式的输出。表 2-3 给出了限定符可能的值和相应的含义

表 **2-3 TextBox** 的 **constraints** 属性

Constraints 的值	含义
TextField.ANY	用户可以输入任何字符
TextField.EMAILADDR	输入的必须是一个 E-mail 地址
TextField.NUMBER	输入的必须是一个整数
TextField.PASSWD	输入的文本将被掩码(即被星号代替),这样使得输入字符不可见
TextField.PHONENUMBER	输入的必须是一个电话号码
TextField.URL	输入的必须是一个 URL

TextField.PASSWD 限定符可以通过使用位与运算符（"|"）与任何其他限定符混合使用。(如 TextField.PHONENUMBER|TextField.PASSWD 表示 TextBox 的输入被限定为电话号码，并且其输入内容被星号所隐藏)

图 2-13 给出限定符为 TextField.PASSWD,TextField.EMAILADDR 和 TextField.URL 的 TextBox 的示例。

TextField.PASSWD　TextField.EMAILADDR　TextField.URL

图 2-13 TextBox

Alert

Alert 是在处理下一屏幕之前，在一定的时间内给用户显示数据的 Screen。Alert 用来提示用户出现了错误或是给用户显示一些信息。Alert 的构建式为：

```
public Alert(String title);
public Alert(String title, String alertText, Image alertImage,
   AlertType alertType);
```

构建器中的 AlertType 是一个类，其中封装了 5 种 alert 类型分别是：INFO, WARNING, ERROR, ALARM, 和 CONFIRMATION。可以通过 AlertType.playSound()方法来发出不同的声音。Alert 的持续时间通过 Alert.setTimeout()方法来设定。

图 2-14 给出一个 Alert 的示例：

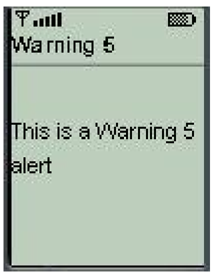

图 2-14 Alert 示例

Form

Form 是一种可以集成其他 GUI 元素的 Screen。这些 GUI 元素包括：图片（image），字符串(string)，和 Item 类的子类(即 ChoiceGroup, DateField, Gauge, ImageItem, StringItem, TextField 共 6 种）可以通过 Form 的 append，insert，delete，get，set 这些方法来对 Form 中包含的 GUI 元素进行操作。

Form 除了可以添加图片和字符串之外，更重要的是功能在于，Form 还可以把 Item 类的 6 种子类集成到自身中来，组成更功能更复杂的 GUI 模型。下面分别介绍这 6 种 Item。

- **ChoiceGroup**

与 List 类似，但 ChoiceGroup 在只能在 Form 中使用，而且只支持互斥（exclusive）和多选（multiple choice）模式。图 2-15 给出一个互斥的 ChoiceGruop 的示例

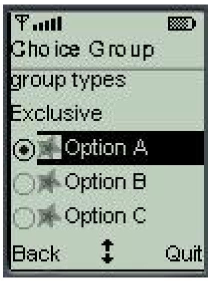

图 2-14 ChoiceGroup 示例

- **DateField**

DateField 用于显示日期和时间，DateField 显示的模式可以被配置为 DateField.DATE, DateField.TIME,或是 DateField.DATE_TIME 这三种，前 2 种只显示日期或是时间，第 3 种即显示日期又显示时间。

图 2-16 给出一个 DateField.DATE_TIME 模式的 DateField。

图 2-15 DateField 示例

■ Gauge

Gauge 可以用图形方式来显现一个值，用于 form 中。Gauge 分为交互（interactive）模式和非交互（non-interactive）模式。交互模式的 Gauge 用户可以改变 Gauge 当前的值。而非交互模式则不可改变，非交互模式一般用于表示一个进程条。

图 2-17 给出 Gauge 的一个示例：

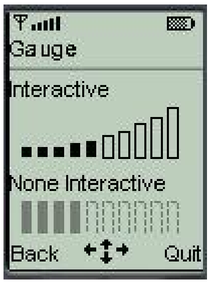

图 2-17 Gauge 示例

■ ImageItem

ImageItem 用来在 Form 中显示图片，ImageItem 与 Image 的不同之处在于：ImageItem 是把 Image 经过封装的类，可以对其摆放位置进行控制。通过给 ImageItem 的构造器的 Layout 变量传递参数，就可以控制 Image 的摆放位置。这些参数在 ImageItem 类中事先定义，主要有 LAYOUT_CENTER LAYOUT_DEFAULT 等等。

图 2-18 给出一个 ImageItem 的示例：

图 2-18 ImageItem 示例

StringItem

StringItem 是可以封装多个 String 的对象，当 String 被添加到 Form 时会自动转化为 StringItem。

TextField

TextField 类似与 TextBox，不过它放置于 Form 中。类似 TextBox，它也可以应用自身类中定义的输入限定符（constraints）。TextField 类中定义的限定符主要有：TextField.ANY，TextField.EMAILADDR，TextField.NUMBER，TextField.PASSWD，TextField.PHONENUMBER，和 TextField.URL 图 2-19 给出了一个 TextField 的示例：

图 2-19 TextField 示例

low-level 界面编程

当开发者需要直接掌控图形元素在屏幕上的画法，以及处理低级的事件如键的按下与释放时，就必须使用 Low-Level 用户界面 API。也就是说需要使用 Canvas 类和 Graphics 类。Canvas 类提供了绘制 Graphics 类的方法和处理键（key）事件和 game action 事件的回调机制。而 Graphics 类，可以精确的在 Canvas 上绘制直线、矩形、圆弧、文本、和图像等图形元素。

Low-level 界面编程的一般模式

Canvas 是一个抽象类，进行 Low-level 编程需要实现一个 Canvas 的子类，并且重载 Canvas 的抽象方法 paint()。

```
protected abstract void paint(Graphics g)
```

在 paint()中，可以对传递来的 Graphics 对象进行操作，如使用 Graphsics 对象的 getWidth()和 getHeight()获得有效的绘制区，setGrayScale()和 setColor()方法对灰度和颜色经行控制。drawLine()方法进行画线，drawRect()和 fillRect()方法绘制并填充矩形，drawString()绘制文本等等。当实作好了 Canvas 的子类之后，由于 Canvas 也是 Displayable 类的子类，所以使用 Display 的 setCurrent()方法，就可以在屏幕上显示该 Canvas。

示例 2-2 给出一个简单的使用了 Canvas 的 MIDlet，该程序在 Canvas 上显示 2 行文本。

示例 **2-2** 一个简单的 **Canvas** 程序

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

class MyCanvas extends Canvas {
  public void paint(Graphics g) {
    g.setColor(255, 0, 0);
    g.fillRect(0, 0, getWidth(), getHeight( ));
    g.setColor(255, 255, 255);
   //g.setFont(Font.getFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN,
        Font.SIZE_LARGE));
    g.drawString("It's a Canvas!", 5, 20, g.TOP | g.LEFT);
    //g.setFont(Font.getFont(Font.FACE_SYSTEM,
Font.STYLE_BOLD|Font.STYLE_ITALIC ,
        Font.SIZE_MEDIUM));
    g.drawString("Wu_Yiding", 35, 50, g.TOP | g.LEFT);
  }
}

public class MyMidlet extends MIDlet {
  public MyMidlet( ) {
  }

  public void startApp( ) {
    Canvas canvas = new MyCanvas( );
    Display display = Display.getDisplay(this);
    display.setCurrent(canvas);
  }

  public void pauseApp( ) {
  }

  public void destroyApp(boolean unconditional) {
  }
}
```

该程序的运行结果如图 2-20（左图）所示，如果把程序中关于字体设置的两行的注释去掉，则 setFont()方法会对文本的字体做进一步的调整。结果如图 2-20（右图）所示。

图 2-20 简单的 Canvas 示例

Canvas 的低级事件处理

Canvas 所处理的低级事件包括：键事件（key event）和动作事件（action event，也称 game action event），和点击事件（pointer event）[1]。如为了响应键事件。应用需要重载 Canvas 的 keyPressed()，keyReleased()和 keyRepeated()方法。具体的细节请参考 MIDP API 文档。

[1] 并非所有的 MIDP 设备都支持点击事件，所以在某些设备上，应用可能永远不会响应该事件

2.4 可持续存储与 RMS

在某种场合下，也许会要求程序中生成的对象在程序结束后并不立即被消灭，而是把它存储到某个地方，以供我们以后取用。可持续存储（persistent Storage，或称永续存储）指的是就是这种机制。J2SE 中提供的可持续存储机制（通过 JBDC 和 Object Serialization API）并不适合用于移动设备，所以 MIDP 中定义了 RMS 来提供这种机制。

RMS(Record Management System)简介

RMS(Record Management System)指的是 MIDP 中用来管理 MIDlet 在可持续存储介质（persisent Storage ）上所经行的数据存储的记录管理系统。你可以把 RMS 理解为 MIDP 上的简单的数据库管理系统。

Record Store 与 Record

在 RMS 中，基本的存储单位被称为 record store（RS），record store 中的每一项被称为一个 record。由此可见 RMS 中数据库的结构是非常简单的。record store 是一个由 record 组成的二进制文件，具体它如何存储在 persisent 介质上并保证其完整性是由具体的手机平台来实现的。例如系统重启或掉电后的数据恢复等.

record 以字节数组的形式储存。每个 record 都被给予一个唯一的 recordID。recordID 以自然数表示，按 1，2，3... 顺序排列。请注意 recordID 只是用来标志一个 record，并不是说相邻的两个 record 就需要连续的 recordID，具体 record 如何排列是由底层实现来确定的。RecordID 的分配只是简单的单步增长，并不存在 recordID 的循环利用。（也就是说，如果我删除了一个 record，再分配 record 时，recordID 为曾经分配过的最大 ID 加 1，与是否删除过 record 无关）recordID 的最大值为 2,147,483,647（即 int 型的最大值），对于储存受限的移动设备来说这已经很够用了。

我们使用 recordID 来检索 record store 中的记录。（通过使用 getRecord 方法）。

MIDP spec 规定在同一 MIDlet suite 中的 MIDlet 创建的 record Store 可以共享，不同 MIDlet suite 则不能。这一点可以从同一 MIDlet suite 的 MIDlet 创建的 record Store 被置于同一目录，而不同 MIDlet suite 的 record Store 被置入不同目录看出来。一个 MIDlet suite 中可以创建多个 record Store。Record store 的名字是大小写敏感的，最长 32 个 unicode 字符（即 32×2byte）。同一 MIDlet suite 中 Record store 不能重名，如果是不同的 MIDlet suite 则可以。

图 2-21 展示了 RMS 的 Record store 和 Record 之间的关系。

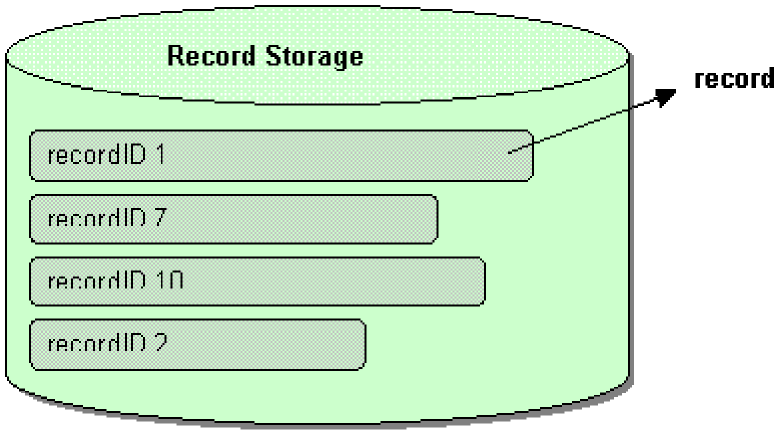

图 2-21 Record Store 与 Record

RMS 包

MIDP 在 javax.microedition.rms 包中定义了 RMS 所用到的接口，类，和异常，

接口	对应的功能
RecordFilter	定义查找 record 的过滤器
RecordComparator	定义 record 的比较操作
RecordListener	侦听 record 的添加，修改，删除。
RecordEnumeration	用来构造一种机制来枚举某 record Store 中的 record
类	对应的功能
RecordStore	表示一个 Record Store
异常	对应的功能
InvalidRecordIDException	当 RecordID 不合法时抛出。
RecordStoreException	通用的 Record Store 异常
RecordStoreFullException	表示 Record Store 已满
RecordStoreNotFoundException	无法找到 Record Store 时抛出
RecordStoreNotOpenException	当对一个还未打开的 Record Store 操作时抛出。

RMS 编程

对 **Record Store** 进行操作

如在RMS包中所示，MIDP通过RecordStore类来具体实作出record store，当使用Record Store之前，必须首先打开它，你可以通过 RecordStore 的 openRecordStore()方法来创建一个新的 Record Store 并打开或是打开一个已经存在的 record store。

openRecordStore()方法的声明如下：

```
public static RecordStore openRecordStore(String recordStoreName,boolean
createIfNecessary)
```

请注意，openRecordStore()方法的第二个参数，如果为 true，表示如果 record Store 名字（第一个参数）不存在则用这个给出的 record Store 名字新建一个 Record Store，并打开它。

例如有如下代码片断：

```
String rs_name="myRS";
RecordStore rs;
    try{
        // 创建一个 recordStoreName 为 myRS 的 Record Store
        rs = RecordStore.openRecordStore( rs_name, true);
    }catch(Exception e){
        System.out.println("Error
              opening"+rs_name+":"+e.toString());
    }
……
```

对于 Record Store 操作的方法主要还有：

- getName 返回打开或创建的 Record Store 的名字
- getNumRecords 返回当前 Record Store 的 Record 数目
- getSize 返回当前 Record Store 的总字节数
- getSizeAvailable 返回 Record Store 可以返回的潜在剩余字节数
- getNextRecordID 返回下一个 Record 的 recordID
- getVersion 返回 Record Store 的版本
- getLastModified 返回 Record Store 的最后修改时间。

- closeRecordStore 关闭当前 Record Store
- deleteRecordStore 删除一个 Record Store

使用 Record

当建立或打开了一个 Record Store 后，就可对其中的 Record 进行操作，例如可以给 record store 中添加一个 record，例如如下代码段：

```
String meg1= "fay,I love you,please don't leave me alone";
//数据被存储到 record 之前，必须先转换为一个 byte 数组。
byte[] data= meg1.getBytes();
   try {
      rs.addRecord(data, 0, data.length);
      System.out.println("Writing record: " + meg1)
   } catch(Exception e){
      System.out.println("Error adding record:"+e.toString());
   }
```

注意 addRecord 方法以一个 byte 数组，偏移值，和长度为参数。创建 record 时，从 byte 数组中，以给定的偏移值开始，长度为给定长度[1]来创建一个 Record。

如果要从已有的 record store 中读取一个 record：

```
String meg1;
byte[] data;
   try {
      data = store.getRecord(1);//1 是这里假设的 recordID
      meg1 = new String(data);
   }catch(Exception e){
      System.out.println("Error opening record:"+e.toString());
   }
```

归纳一下，用来操作 Record 的一些常用方法如下

- addRecord 给 record store 添加一个记录。
- getRecord() 以 recordID 为参数得到一个 record。

[1] 严格意义上来讲上，record 的长度是给定长度的子集。

- deleteRecord(), 删除 recordID 标志的 record
- getRecordSize(),以 recordID 为参数，得到 record 的大小的字节数
- setRecord(),通过给出的一个 byte 数组来覆盖一个由 recordID 指定的 record

record 的过滤，比较，侦听和枚举

除了上述基本的record Store和record操作之外，RMS还提供了4个接口，来方便程序员对record进行过滤，比较，侦听和枚举。

RecordFilter 接口用来定义查找 record 的过滤器，程序员通过实现 matches()方法，来定义自己的过滤器。Matches()方法以一个 record 为参数（即 byte 数组），返回 boolean 来标志该 record 是否满足该过滤器。

RecordComparator 接口用来定义 record 的比较操作，程序员通过实现 compare()方法，来定义自己的比较操作，compare()方法以两个 record 为参数，并返回 EQUIVALENT，FOLLOWS 或 PRECEDES 这 3 个静态 int 常量中的一个（这 3 个常量是 RecordComparator 接口中事先定义好的）。EQUIVALENT 表示两个 record 被认为相同；FOLLOWS：第一个 record 在第二个之后；PRECEDES：第一个在第二个之前。

RecordListener 接口用来侦听 record Store 中 record 的添加，修改和删除。需要实做 3 个方法：recordAdded()，recordChanged()和 recordDelted()。每一种方法有两个参数，分别是 RecordStore 和 recordID。

RecordEnumeration 接口用来定义 record 的枚举。通常可以简单的使用 RecordStore 的 enumerateRecords()方法，该方法会返回一个实现了 *RecordEnumeration* 接口的类的实例。限于篇幅，这里不再赘述，详细内容可参考 MIDP API 文档。

结束语

本文通过对 J2ME 平台以及 CLDC/MIDP 编程的概述性的介绍，展示了 Java 技术如何被应用到移动设备上。以及在 J2ME 平台上进行 MIDlet 编程基本概念和一般方法。

参考文献

【1】 Roger Riggs *Programming Wireless Devices with the Java 2 Platform, Micro Edition* Addison Wesley 2001
【2】 Qusay Mahmoud *Learning Wireless Java* O’Reilly 2001
【3】 Yu Feng,Dr. Jun Zhu *Wireless Java Programming with Java[tm] 2 Micro Edition* Sams 2001
【4】 王森 《Java 手机程式设计入门》知域（台湾），2001
【5】 JCP Specification *Mobile Information DeviceProfile (JSR-37)*
【6】 JCP Specification *Connected, Limited Device Configuration Specification Version 1.0a*
【7】 KVM White Paper *Java 2 Platform MicroEdition (J2ME) Technologyfor Creating Mobile Devices* Sun 2000
【8】 James Gosling, Bill Joy, and Guy L. Steele.*The Java Language Specification* Addison-Wesley, 1996
【9】 Tim Lindholm, Frank Yellin *The Java Virtual Machine Specification* Addison-Wesley, 1996