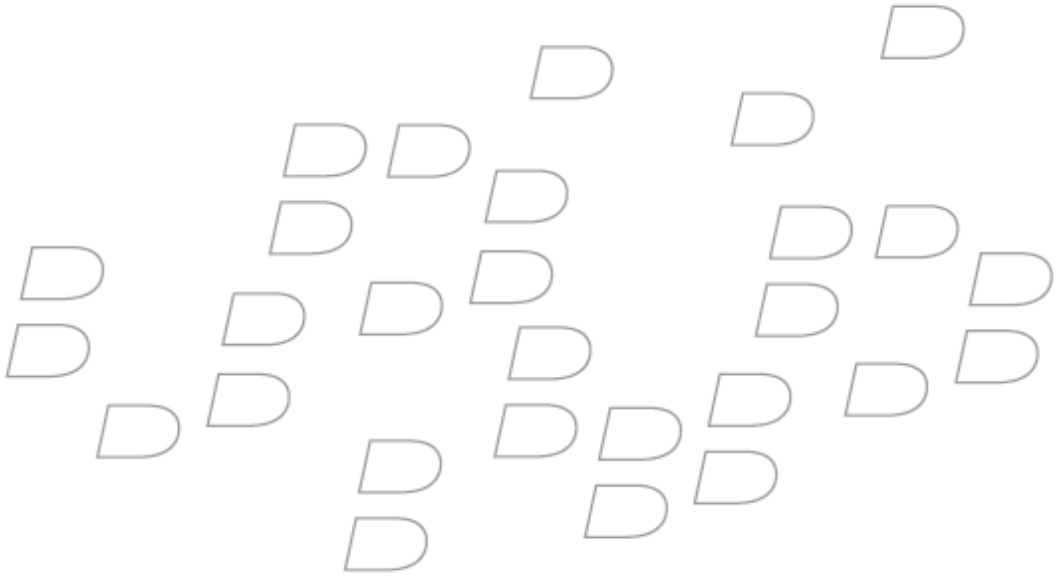


BlackBerry Java 开发环境
版本 4.1.0

BlackBerry 应用程序开发者指南
第一卷：基础

声明

本文档由 Taiguo Zhang 个人翻译完成。如有转载，复制，摘抄，必须向 Taiguo Zhang 提出。

联系方式：
Email: confach@gmail.com
MSN: confach@hotmail.com
Blogs: http://confach.cnblogs.com
http://www.36sign.com
http://www.inblackberry.com

欢迎大家和我联系。

目录

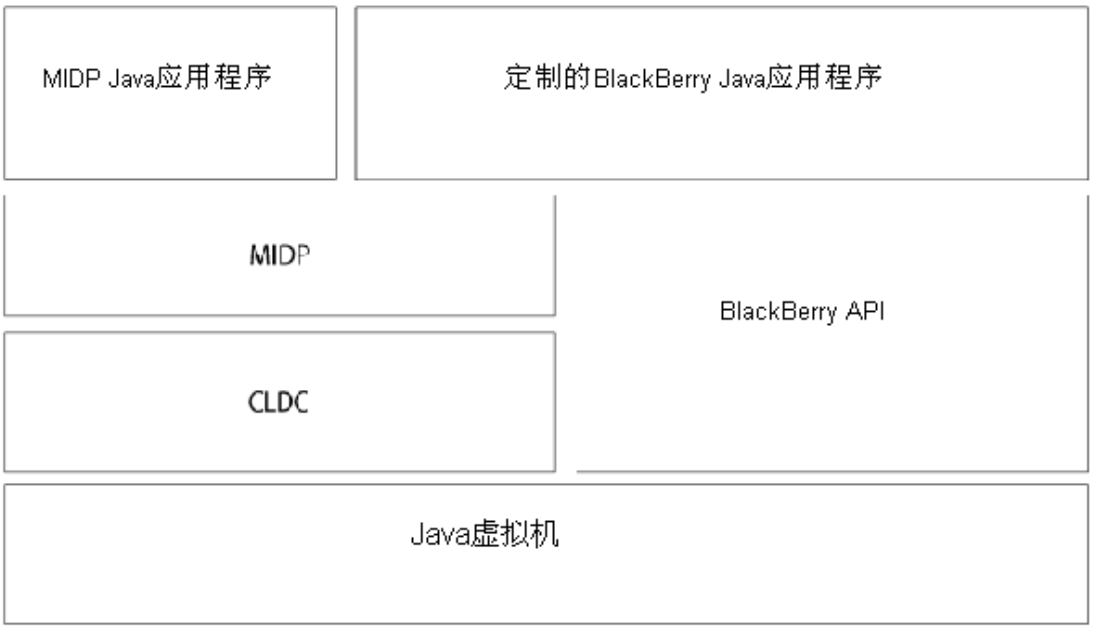
第 1 章 BlackBerry API

使用 BlackBerry API

BlackBerry Java 开发环境（简称 JDE）的设计提供了一套完整的 API 和工具,来开发在 BlackBerry 设备上运行的 Java 应用程序。

BlackBerry 设备包含了一个基于 CLDC1.1 以及 MIDP 的 Java ME（Java Platform Micro Edition）运行时环境。BlackBerry API 扩展提供了额外的功能,并且和 BlackBerry 集成得更紧密。

你可以在你的应用程序中使用 CLDC/MIDP 和 BlackBerry API。为了能让你的应用程序在任何采用 JTWI（Java Technology for Wireless Industry，无线领域的 Java 技术）的设备上运行，你仅需要使用 CLDC 和 MIDP API 来编写标准的 MIDP 应用程序。



BlackBerry 手持设备软件组件

为了查看 *API 参考*。点击任务栏的**开始>程序>Research In Motion>BlackBerry JDE 4.1.0>API Java Doc Reference**

BlackBerry API

BlackBerry API 为访问 BlackBerry 特性提供了用户界面，本地化，网络，以及其他功能。

 注：访问某些特性，如高级的加密，同步，以及消息的额外 API，是受限制的。为了使用这些 API,你必须收到专门来自 Research In Motion 的认证中心编写的许可。为了得到更多信息，参看 *BlackBerry 应用程序开发者指南第一卷：基础 第二卷：高级*。

BlackBerry API 包	描述
net.rim.blackberry.api.browser	应用程序可以调用 BlackBerry 浏览器，为了得到更多信息，参看 *BlackBerry 应用程序开发者指南 第二卷：高级*。
net.rim.blackberry.api.invoke	允许应用程序调用 BlackBerry 应用程序，如任务，消息，备忘录以及电话。为了得到更多信息，*参看 BlackBerry 应用程序开发者指南 第二卷：高级*。
net.rim.blackberry.api.mail	定义了必要的功能来将内部的 RIM 消息系统对象组件转化为和 Mail API 兼容并可移植的对象。同时也提供了发送，接收，以及访问消息的功能。为了得到更多信息，*参看 BlackBerry 应用程序开发者指南 第二卷：高级*
net.rim.blackberry.api.mail.event	定义了消息事件以及监听者（Listener）接口来管理邮件事件。为了得到更多信息，*参看 BlackBerry 应用程序开发者指南 第二卷：高级*
net.rim.blackberry.api.menuitem	允许应用程序在 BlackBerry 的应用程序例如地址本，日历以及消息中增加客户定制的菜单项，为了得到更多信息，参看 *BlackBerry 应用程序开发者指南 第二卷：高级*
net.rim.blackberry.api.options	允许应用程序在 BlackBerry 设备的选项中增加选项条目。为了得到更多信息，参看 *BlackBerry 应用程序开发者指南 第二卷：高级*
net.rim.blackberry.api.pdap	允许应用程序和 BlackBerry 个人信息管理（PIM）交互，PIM 包括地址本，任务，日历。MIDP 包 javax.microedition.pim 提供了类似的功能。为了得到更多信息，参看 *BlackBerry 应用程序开发者指南 第二卷：高级*
net.rim.blackberry.api.phone	提供了访问电话应用程序的高级特性。为了得到更多信息，参看 *BlackBerry 应用程序开发者指南 第二卷：高级*
net.rim.blackberry.api.phone.phonelogs	提供了访问电话呼叫历史记录的功能。为了得到更多信息，参看 *BlackBerry 应用程序开发者指南 第二卷：高级*

net.rim.device.api.bluetooth	允许 BlackBerry 应用程序在一个蓝牙序列端口连接的基础上与打开蓝牙无线技术的设备进行通信。为了得到更多信息，参看 103 页的“*使用蓝牙序列端口连接*”.
net.rim.device.api.browser.field	允许程序在界面上显示浏览器的字段。为了得到更多信息，参看 *BlackBerry 应用程序开发者指南 第二卷：高级*
net.rim.device.api.browser.plugin	允许程序增加额外支持的MIME类型到BlackBerry浏览器上。为了得到更多信息，参看 *BlackBerry 应用程序开发者指南 第二卷：高级*
net.rim.device.api.collection net.rim.device.api.collection.util	为管理数据集合定义了接口和实用类。为了得到更多信息，参看 13 页的“*集合*”.
net.rim.device.api.compress	提供实用类来进行 GZip 和 ZLib 数据压缩。①
net.rim.device.api.i18n	提供类来支持BlackBerry设备上应用程序的本地化。为了得到更多信息，参看 13 页 的“*本地化应用程序*”.
net.rim.device.api.io	提供一个定制的 BlackBerry 类库来管理数据的输入和输出。
net.rim.device.api.mime	提供与 MIME 编码的数据流一起工作的类。
net.rim.device.api.notification	提供触发事件的通知以及响应系统以及程序的事件的方法。为了得到更多信息，参看 *BlackBerry 应用程序开发者指南 第二卷：高级*
net.rim.device.api.servicebook	允许程序增加，删除，以及访问服务约定（Service Book）②的接口。为了得到更多信息，参看 *BlackBerry 应用程序开发者指南 第二卷：高级*
net.rim.device.api.system	提供访问系统级的功能，包括键盘和滑轮的事件监听者，图像创建和支持，和应用程序控制。
net.rim.device.api.ui	提供增强的功能来控制 BlackBerry 用户界面，包括屏幕和控件布局管理，控件类型支持，焦点，滚动，以及改变监听者。为了得到更多信息，参看 39 页的“*用户界面 API*”.
net.rim.device.api.ui.component	提供了创建 UI 程序的界面组件库。为了得到更多信息，参看 39 页的“*显示用户界面组件*”。
net.rim.device.api.ui.container	提供创建 UI 程序的界面管理组件的库。为了得到更多信息，参看 49 页的“*管理用户界面组件*”。
net.rim.device.api.ui.text	提供类对文本字符串进行过滤，包含多种类型的数据，例如电话号码或 URL。
net.rim.device.api.util	提供实用的方法和接口，包含数组，哈希表，字符匹配。

① 也许有人会问，为什么 BlackBerry 需要压缩数据，又在什么地方用到呢?非常简单，就是减小数据所占用的空间,最好的一个例子是 BES 发送邮件，译者注。

② 服务订阅，Service Book,这是 BlackBerry 使用中一个非常重要的概念。功能就是你订阅的服务，例如你订阅了 MMS 的功能，它会在你 BlackBerry 手持设备上出现。译者注。

CLDC API

CLDC API 包	描述
java.io	提供数据流的系统输入和输出。
java.lang	提供 Java 编程语言基础类。
java.lang.ref	提供引用对象类，它们支持一定程度上的垃圾回收。
java.util	包含集合类，时间，以及多样的实用类。
javax.microedition.io	包含一般连接的类。

MIDP API

MIDP API 包	描述
javax.microedition.lcdui	包含 MIDP 用户界面 API，它为 MIDP 应用程序的用户界面实现提供了一组特性。
javax.microedition.lcdui.game	包含了可以为 BlackBerry 设备进行丰富游戏内容开发的类。
javax.microedition.midlet	定义了 MIDP 应用程序以及应用程序和应用程序运行的环境之间的交互。 **注**：BlackBerry IDE 可以在启动时，使参数传递到一个 BlackBerry CLDC 应用程序中。
javax.microedition.pki	定义了用来验证安全连接信息的证书。
javax.microedition.rms	为 MIDlet 提供一种机制来存储和取得持久性数据。

PDAP API

MIDP API 包	描述
javax.microedition.pim	.提供标准机制来访问 PIM 信息。

在 BlackBerry 设备上使用 Java

编译源代码，打包为.cod文件，并将.cod文件加载到BlackBerry设备上，通过虚拟机运行。

注：.cod文件名控制在128字节。

如CLDC中描述的那样，BlackBerry IDE使用一个分割的VM架构。为了降低内存的数量以及BlackBerry设备需要的处理能力，部分类加载过程，称为预验证，它在Java代码加载到BlackBerry之前发生。在将源代码打包为.cod文件之前，自动验证它。在类加载到BlackBerry设备时完成验证的提示。

限制

在 CLDC1.1 中描述的那样 BlackBerry 虚拟机有以下限制：

- 没有对象的析构（finalization）
- 没有用户类的加载
- 没有反射，因此不支持 RMI 和 Jini 网络技术。
- 没有原生方法（Native method）
- 没有 Runtime.exec()执行外部的进程

多线程

BlackBerry Java 环境提供一个真正的多线程环境来运行应用程序。这个环境允许多个应用程序同时运行，允许事件广播到多个应用程序，以及长操作和监听线程在背后运行。

持久数据

存储在闪存中的数据在 BlackBerry 重新设置之间持久保存。在 BlackBerry 设备上存储数据可以采用以下二种方式中的一种：

- 使用 MIDP 记录存储
- 使用 BlackBerry 持久模型

为了得到关于使用 BlackBerry API 存储持久数据的更多信息，参看 *BlackBerry 应用程序开发者指南第一卷：基础 第二卷：高级*。

网络通信

BlackBerry JDE 根据 MIDP2.0, 实现了网络通信。它提供多种连接选项，包括通过使用 HTTP 代理连接在公司防火墙背后安全连接的能力。

BlackBerry JDE 提供了以下几种连接类型：

- 流连接（StreamConnection 接口，包括：
  1. HTTP 连接（HttpConnection 接口）
  2. HTTPS 连接（HttpsConnection 接口）
  3. Socket 连接（SocketConnection 接口）
  4. 安全 socket 连接（SecureConnection 接口）
  5. 序列连接到 BlackBerry 设备的一个通信接口（CoomConnection 接口）
- 数据报连接（DtagramConnection 接口），包含
  1. UDP 数据报连接（UDPDatagramConnection 接口）

Javax.microedition.io.PushRegistry 类对 BlackBerry 设备保持了一些进入的连接。

流

BlackBerry JDE 为包含在 CLDC java.io 包里的流提供了标准的接口和类。

MIME 编码

BlackBerry IDE 提供了 MIMEInputStream 和 MIMEOutputStream 类来读写一个 MIME 编码的数据流。

类	描述
MIMEInputStream	实现一个流来读取一个 MIME 消息，然后根据 MIME 标准格式化和分解这个消息为其部分
MIMEOutputStream	实现一个输出流，这个流可以根据 MIME 标准格式化输出为其部分。本类不会完成实际的数据编码，因此你必须在写入它到本数据流治安编码它。

压缩

在 net.rim.device.api.compress 包里，BlackBerry JDE 提供类来读取使用 Zlib 或者 GZip 格式压缩的数据流。这些类的行为如 Java 标准版本里的 java.util.zip 包里对应的类一样。缺省的，压缩是允许的，BlackBerry 设备可以写有效的 GZip 和 Zlib 文件为这样压缩文件的内容。解压缩同样也是支持的。

集合

BlackBerry IDE 提供了一组接口和实用类来管理 BlackBerry 设备上的集合。
net.rim.device.api.collection 包包含了许多接口，这些接口为某些特定类型数据类型多定义了种类型的集合，例如列表，数组以及映射。这些接口定义了与 Java 标准版本集合框架的 list,set 和 map 接口类似的功能。
在你自己的类中实现这些接口，或者使用在 net.rim.device.api.collection.util 包里提供的使用类。

向量

标准的 java.util.Vector 实现了一个大小可以改变的对象数组。BlackBerry JDE 也提供了合适的类，例如 rim.device.api.util.IntVector 和 rim.device.api.util.ByteVector 来对主要类型进行工作。这些类看起来和普通的 Vector 一样,除了它们优化了在任何位置上插入的项。相反，如果你使用标准的大 Vector 作随机的改变，大量的数据会在闪存和 RAM 移动。

列表

BlackBerry JDE 在 net.rim.device.api.collection.util 包里提供了一些类来管理元素的列表

类	描述
SortedReadableList 和 UnsortedReadableList	使用这些类来维护已排序的和未排序的元素列表。SortedReadableList 类需要你使用一个比较对象来排序列表中的元素。增加到列表中的每一个元素必须被比较对象视为有效的。
IntSortedReadableList 和 LongSortedReadableList	使用这些类自动排序整形列表或与长整形关键字相关的元素。
BigSortedReadableList 和 BigUnsortedReadableList	使用这些类来存储大的数据集合（大于 10 或者 15K).这些类不会存储数据到一个数组中，因此你可以对大数据集合更有效的做出随意改变。
ReadableListCombiner	使用这个类合并 2 个或者更多的 ReadableList 对象并且将他们作为一单个 ReadableList 来存储。
ReadableListUtil	此类提供一些有用的方法如 getAt()和 getIndex()。我们可以使用此类得到只读列表中的数据。

哈希表

除了 CLDC 提供的标准 java.util.Hashtable 之外，BlackBerry JDE 包含了特定的 net.rim.device.api.collection.util.LongHashtableCollection 类，这个类提供了使用长整形作为关键字的哈希表集合。一个 LongHashtableCollection 对象，写操作作为一个映射（使用一个关

键字-元素对），读操作作为一个映射或者作为一个集合（在集合里作为一个数组来得到数据）。

事件监听者

事件监听者接口根据事件类型划分。每个应用程序注册来接收特定类型的事件。应用程序事件队列然后调度事件到一个合适的监听者。
应用程序可以实现合适的监听者接口或者在各种 Screen 对象里重写监听者方法。大多数应用程序实现了 KeyListener 和 TrackwheelListener 接口，而且注册了监听者来接收键盘和滑轮的事件。键盘和滑轮是用户和应用程序交互的主要方式。
下列的事件监听者放在 net.rim.device.api.system 包中

监听者接口	事件类型
AlertListener	实现接口来监听 alert 事件
BluetoothSerialPortListener	实现接口来监听蓝牙序列端口事件，例如打开一个蓝牙序列端口连接作为服务器或者客户端。
GlobalEventListener	实现接口来监听可以广播到所有应用程序的全局事件。
HolsterListener	实现接口来监听套装事件，例如 BlackBerry 设备从套装中插入和移开。
IOPortListener	实现接口监听 I/O 端口事件。
KeyListener	实现接口监听键盘事件，例如用户按住或释放一个键。
RealTimeClockListener	实现本接口来监听实时时钟事件，例如时钟更新。
SerialPortListener	实现此接口监听序列化端口事件，例如对于一个已经和计算机序列端口连接的 BlackBerry 设备，一个在数据正在被发送到序列化端口连接状态中的改变。
SystemListener	实现此接口来监听系统事件，例如电池状态和电源的改变。
TrackwheelListener	实现本接口监听滑轮事件，例如按住滑轮。
USBPortListener	实现本接口监听 USB 端口事件，例如对于一个已经和计算机 USB 端口连接的 BlackBerry 设备，数据正被发送到 USB 端口连接的状态。

系统功能

net.rim.device.api.system 包的类提供了访问 Java VM 和 BlackBerry 设备上系统资源的能力。

得到信号信息

RadioInfo 提供了访问信号状态信息的能力。

得到设备信息

DeviceInfo 类可以访问下列 BlackBerry 设备的信息：

- 电池电源和状态
- Blackerry 设备号
- 空闲时间
- 平台版本

系统事件通知用户

当一个事件，例如一条新消息到来的时候，Alert 类允许应用程序通知用户。

监视内存使用情况

使用一个 Memory 类提供的静态方法来得到 VM 内存使用统计信息。

Memory 类很多实用方法返回一个 MemoryStats 对象。使用 MemoryStats 类提供的实用方法得到 BlackBerry 设备上内存和可用存储空间的详细信息。

日志事件

EventLogger 允许应用程序在持久存储里存储事件日志。BlackBerry 设备维护事件队列，以至当日志满时，会删除最早的事件，并增加新的事件。用户可以按住 **Alt+lglg 键**来查看 BlackBerry 设备的系统事件日志。

使用工具

BlackBerry JDE 在 net.rim.device.api.util 包里提供了一组实用工具，这些类里的许多类提供了和 Java 标准版本里相似的功能。

- Comparator 接口定义了对象集合上的顺序的方法。
- Arrays 提供方法来操作数组，例如排序，查找，以及作为列表来查看数组。
- BitSet 类维护 bit 的集合。

net.rim.device.api.util 包包含了多个类来管理特定类型的数据集合，包括向量，哈希表，映射以及栈。

应用程序控制

应用程序控制允许系统管理员操作以下动作：

- 控制内部连接（公司防火墙背后的连接）
- 控制外部连接
- 控制本地连接（序列和 USB 连接）
- 控制访问键存储(key store)
- 控制访问特殊的 API.

- 阻止第三方应用程序存在 BlackBerry 设备上。

为了得到更多信息，参看 *BlackBerry Enterprise Server Handheld Management Guide* 的应用程序管理。

受限制访问的 API，类，和方法

使用了下列受限的 API，类，以及方法的应用程序可以加载到 BlackBerry 设备，但是如果他们访问了一个没有在应用程序控制下得到允许的 API 时，在运行时会抛出一个 ControlledAccessException 或者 NoClassDefFoundError 的异常。

类，方法或 API	缺省值
应用程序菜单项 API(net.rim.blackberry.api.menuitem)	允许
蓝牙 API (net.rim.device.api.bluetooth)	允许
Connector.open() (javax.microedition.io)	提示 注:内部和外部的连接由不同的应用程序控制策略来管理
DeviceKeyStore 类 (net.rim.device.api.crypto.keystore)	允许
EventInjector 类 (net.rim.device.api.system)	不允许
HTTP Filter API (net.rim.device.api.io.http)	不允许
Notification API (net.rim.device.api.notification)	允许
电 API 和 呼叫 API (用来调用电话应用程序) (net.rim.blackberry.api.phone 和 net.rim.blackberry.api.invoke)	允许(缺省，用户提示)
电话日志 API (net.rim.blackberry.api.phone.phonelogs)	允许(缺省，用户提示)
PIM API (net.rim.blackberry.api.pdap)	允许
RuntimeStore 类 (net.rim.device.api.system)	允许
SerialPort 类 (net.rim.device.api.system)	允许
Session 类 (net.rim.blackberry.api.mail)	允许
StringPatternRepository 类 (net.rim.device.api.util)	允许
USBPort 类 (net.rim.device.api.system)	允许

2

第 2 章 编写 BlackBerry Java 应用程序

应用程序管理

当 BlackBerry 设备启动时，VM 加载应用程序管理器，它管理在 BlackBerry 设备上所有运行的程序。对于其他 Java 程序，应用程序管理器的功能类似操作系统事件的中心调度员一样。

提供用户界面的应用程序扩展了 net.rim.device.api.ui.UiApplication 类。这个类为应用程序提供方法来注册事件监听者，管理线程以及 UI 组件。

没有提供用户界面的应用程序扩展了 net.rim.device.api.system.Application 类。

BlackBerry 应用程序开始于 main()函数。当一个程序开始时，它的 main()线程调用 enterEventDispatcher()来开始处理事件。这个线程运行所有绘图以及事件处理的代码，以及登等待应用程序队列里地事件。

当应用程序管理器接收到一个事件时，它将这个事件拷贝到合适的队列里，这个队列可以允许应用程序管理器指挥消息到特定的程序中。例如，前台的应用程序仅接收用户输入的消息。

编写一个例程

扩展 UiApplication 基类

每个提供用户接口的应用程序扩展了 UiApplication 基类，UiApplication 类为应用程序定义了方法来建立一个事件线程，并且显示和维护 Screen 对象。

定义 main()

在 main()中，为应用程序创建一个新的对象。调用 enterEventDispatcher()使应用程序进入事件线程并且开始处理消息。

```
public static void main(String[] args) {
  HelloWorld theApp = new HelloWorld();
  theApp.enterEventDispatcher();
}
```

定义一个构造子

为你的应用程序定义缺省的构造子。缺省的构造子调用 UiApplication.pushScreen()以显示当应用程序启动时出现的屏幕。在本例中，屏幕使一个新的 HelloWorldScreen 实例，它在下节的代码中定义：

```
public HelloWorld() {
  pushScreen(new HelloWorldScreen());
}
```

定义 main 屏幕

为了定义应用程序 UI 的主屏幕，扩展 MainScreen 类。MainScreen 类是 Screen 的子类，它实现了 TrackwheelListener 和 KeyboardListener 接口，这些接口接收和响应用户交互。如果你扩展 Screen 类或者其子类中的一个，你并不是必须实现 TrackwheelListener 和 KeyboardListener 接口。

你的类至少应该重写 2 个 MainScreen 的方法：缺省的构造子和 onClose().

在这个例子中，构造子调用了 MainScreen 的构造子。缺省地，MainScreen 提供下列特性：

- 由一个 **Close** 菜单项的缺省菜单。
- 当你点击 **Close** 或者按 **Escape** 时，缺省的是关闭动作。为了提供客户定制行为，例如显示一个对话框提示，当用户点击 **Close** 菜单项或者按 **Escape** 按钮，重写 onClose().
- 一个 RichTextField 的实例，一个可以接收焦点的只读富文本域 为了得到更多关于增加 UI 组件到屏幕中的信息，参看 40 页的“*提供屏幕导航*”
- 一个 **Select** 菜单项的上下文菜单• 为了得到更多信息，参看 60 页的“创建定制的上下文菜单“

代码实例

接下来的例子创建了一个屏幕，它包含了一个富文本域。当富文本域接收到焦点时，菜单保安一个 Close 菜单项和一个 Select 上下文菜单项。

例: HelloWorld.java

```
/**
 *  HelloWorld.java
 *  Copyright (C) 2001-2005 Research In Motion Limited. All rights
reserved.
 */
```

```
package com.rim.samples.docs.helloworld;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import com.rim.samples.docs.resource.*;

public class HelloWorld extends UiApplication {
    public static void main(String[] args) {
        HelloWorld theApp = new HelloWorld();
        theApp.enterEventDispatcher();
        }

    public HelloWorld() {
        pushScreen(new HelloWorldScreen());
        }
    }


final class HelloWorldScreen extends MainScreen {
    public HelloWorldScreen() {
        super();
        LabelField title = new LabelField("HelloWorld Sample",
LabelField.ELLIPSIS
                                    | LabelField.USE_ALL_WIDTH);
        setTitle(title);
        add(new RichTextField("Hello World!"));
        }

    public boolean onClose() {
        Dialog.alert("Goodbye!");
        System.exit(0);
        return true;
        }
    }
```

重用一般代码

抽象基类可以使你跨越多个类实现和重用一般功能。每个应用程序可以扩展单个基类。在BlackBerry IDE，加入基类到一个库项目中。为每个应用程序创建一个独立的项目，定义库项目的依赖。

代码实例

本指南的例程扩展了 BaseApp 类，它实现下面的功能：

- 扩展 UiApplication 类
- 实现 KeyListener 和 TrackwheelListener 接口
- 定义变量，例如一般的菜单项
- 定义一个方法创建应用程序菜单。
- 为菜单选择定义一个方法
- 定义一个抽象方法退出程序

例: BaseApp.java

```
/*
 * * BaseApp.java
 * * Copyright (C) 2001-2005 Research In Motion Limited. All rights
reserved.
 * */


package com.rim.samples.docs.baseapp;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import com.rim.samples.docs.resource.*;

public abstract class BaseApp
   extends UiApplication implements BaseAppResource, KeyListener,
TrackwheelListener {
    private MenuItem _closeItem;
    private static ResourceBundle _resources =
        ResourceBundle.getBundle(BUNDLE_ID, BUNDLE_NAME);


    /* Constructor for the abstract base class. */
    public BaseApp() {
        _closeItem = new MenuItem(_resources, MENUITEM_CLOSE, 200000,
10) {
            public void run() {
                onExit();
                System.exit(0);
                }
            };
            }
```

```
/* Override this method to add custom menu items. */
protected void makeMenu( Menu menu, int instance) {
    Field focus = UiApplication.getUiApplication().
                getActiveScreen().getLeafFieldWithFocus();

    if(focus != null) {
        ContextMenu contextMenu = focus.getContextMenu();
        if( !contextMenu.isEmpty()) {
            menu.add(contextMenu);
            menu.addSeparator();
            }
        }
    menu.add(_closeItem);
    }

/* Invoked when the trackwheel is clicked. */
public boolean trackwheelClick( int status, int time ) {
    Menu menu = new Menu();
    makeMenu( menu, 0);
    menu.show();
    return true;
    }

/* Invoked when the trackwheel is released. */
public boolean trackwheelUnclick( int status, int time ) {
    return false;
    }

/* Invoked when the trackwheel is rolled. */
public boolean trackwheelRoll(int amount, int status, int time) {
    return false;
    }

public boolean keyChar(char key, int status, int time) {
    /* Intercept the ESC key and exit the application. */
    boolean retval = false;
    switch (key) {
    case Characters.ESCAPE:
        onExit();
        System.exit(0);
        retval = true;
        break;
        }
```

```
        return retval;
        }

    /* Implementation of KeyListener.keyDown(). */
    public boolean keyDown(int keycode, int time) {
        return false;
        }

    /* Implementation of KeyListener.keyRepeat(). */
    public boolean keyRepeat(int keycode, int time) {
        return false;
        }


    /* Implementation of KeyListener.keyStatus(). */
    public boolean keyStatus(int keycode, int time) {
        return false;
        }

/* Implementation of KeyListener.keyUp(). */
    public boolean keyUp(int keycode, int time) {
        return false;
        }

    protected abstract void onExit();
    }
```

使用 BlackBerry IDE

为了编写，调试和编译应用程序，使用 BlackBerry IDE,它是 BlackBerry JDE 的一部分。


注：BlackBerry 版本 4.1 使用了 Sun JDK 5.0

创建一个工作空间

1. 在 BlackBerry IDE，选择 **File** 菜单，点击 **New Workspace**
2. 在 **Workspace name** 域，输入一个没有文件扩展名的名字。
3. 在 **Create in this directory** 域，输入一个文档。
4. 点击 **OK.**

创建一个项目

注：在包含工作空间的文件夹下的子目录中创建工程。

1. 在 BlackBerry IDE 的 **Project** 菜单，点击 **Create New Project.**
2. 在 **Project name** 域，输入一个没有文件扩张名的项目名称。
3. 在 **Create project in this directory** 域，输入在此文件夹下创建项目的文件夹名称。
4. 点击 **OK**。
5. 在工作空间文件区域里，双击项目名称来设置项目属性。

为了得到更多关于项目属性的信息，参看 BlackBerry Integrated Development Environment Online Help。

创建源文件

注：保存源文件到和项目文件相同的文件夹下。和所有 Java 程序一样，为符合你的类使用的包层次关系的源代码创建文件结构，

1. 在 BlackBerry IDE 的 **File** 菜单，点击 **New**。
2. 在 **Source file name** 域，输入一个带.java 文件扩展的文件名。
3. 在 **Create source file in this directory** 域，输入文件夹名。
4. 点击 **OK**。
5. 在编辑器区域，右击文件，然后点击 **Insert into project**.
6. 选择一个项目。
7. 点击 OK。

和 BlackBerry IDE 集成源文件管理工具

你可以通过不同的源文件控制程序来使用 BlackBerry IDE。BlackBerry IDE 允许你为源文件控制程序设置“check out”, “add new file” 和“revert”选项。在你为某一特定的源文件控制程序配置好选项后，BlackBerry IDE 可以自动 check out 文件，运行预先设置的命令，恢复改变，以及增加新创建的文件到源文件控制程序里。

1. 在 BlackBerry IDE 的 **File** 菜单，点击 **Edit－>Preferences**.
2. 点击 **Source Control** 标签。
3. 点击 **Check out** 标签。
4. 在 **Check out** 域，输入命令以能打开一个文件来编辑。例如输入：

   p4 edit %1

   注：%1 参数代表了名称和一个文件的绝对路径。例如：对于一个在 c:\mypath 的 foo.java 文件，当 BlackBerry IDE 执行命令 checkout %1,时，BlackBerry IDE 实际上运行命令 checkout c:\mypath\foo.java。
5. 点击 **Add file** 标签。
6. 在 **Add new file** 域，输入命令以增加一个新的文件到源文件控制程序中。例如输入：

p4 add %1

7. 点击 **Revert file** 标签.
8. 在 **Revert changes** 域，输入命令撤销一个源文件控制程序中的文件。例如输入：
p4 revert %1
9. 单击 **OK**。

编译项目

当你编译项目时，BlackBerry IDE 编译你的源文件到 Java 字节编码（Bytecode）,进行预验证，然后将类打包为一个.cod 文件。


注：在 Java ME 中，字节编码验证分为 2 个步骤。已经编译的代码在它加载到 BlackBerry 设备之前预先验证，因此类加载时 BlackBerry 设备仅必须进行基本的验证。BlackBerry IDE 在编译项目时自动进行预验证。

当你编译一个项目时，如果需要，BlackBerry 也编译项目依赖的任何库。

动作	过程	附加信息
编译所有项目	> 在 **Build** 菜单，点击 **Build All**	为了排除一个项目，设置项目属性。
编译所有活动的项目	> 在 **Build** 菜单，点击 **Build**。	在工作空间，活动的项目名是加粗的。为了改变哪些项目是活动的，在 **Project** 菜单，点击 **Set Active Projects**.
编译单一项目	1. 选择一个项目 2. 在 **Build** 菜单，点击 **Build Selected**。	———
创建一个工作空间 makefile	>在 **Build** 菜单，点击 **Generate Makefile and Resource.**	———

缺省的，已经编译的.cod 文件使用项目名。为了改变这个名字，双击项目文件，点击 **Build** 标签，输入 **output file name**（输出文件名）。

混淆应用程序

和传统的编译器不一样，BlackBerry 平台的编译器因为受限的无线环境而优化了。在这里无线环境的目标是最小化应用程序的大小。作为结果的.cod 文件提供了大量的类似于其他真正混淆包的混淆服务，以致能减小.cod 文件本身的大小。例如，下面的信息将从.cod 文件中移除：

- 所有调试的信息
- 本地变量名
- 源代码的行数
- 私有方法和成员名

同样，RIM 相信没有必要为应用程序提供混淆，除了已经存在的 BlackBerry 平台编译的所有应用程序缺省提供的混淆。事实上，RIM 没有对自己的产品进行任何额外的混淆。

对通过第三方工具混淆的支持没有集成到 BlackBerry JDE .同样，需要一个命令行过程来混

淆.cod 文件供 BlackBerry 设备上使用。

混淆一个.cod 文件

1. 在 BlackBerry IDE，创建应用程序


提示：在这个过程中将项目文件放到一个独立的目录中。

2. 创建临时的目录
3. 将 BlackBerry IDE 创建的 jar 文件拷贝到一个临时目录。
4. 释放.jar 文件的内容到一个临时目录。例如，在命令窗口输入下面的命令：***jar xvf SampleApplication.jar**
5. 删除释放为.jar 文件部分的.cod 文件。
6. 删除.jar 文件
7. 混淆在临时目录下包含的类文件。
8. 使用下面的命令对临时目录的内容运行预验证工具：

***preverify.exe -verbose -d . -classpath ..\lib\net_rim_api.jar;**

9. 在已混淆（和预验证）的类文件上运行 **rapc** 来创建一个.cod 文件。使用下面的命令：

***rapc.exe -verbose import=..\lib\net_rim_api.jar listing=SampleApplication.lst codename=SampleApplication SampleApplication.rapc C:\yourTempDir\SampleApplication.class**

生成 API 文档

使用一个 BlackBerry IDE 宏给代码加入注释。
一旦启用这个功能，如果你在一个函数声明前的任何一行输入/**，BlackBerry IDE 生成下面的注释:

```
/**
* <description>.
* @param menu <description>.
* @param instance <description>.
* @return <description>.
*/
```

如果你在其他行输入/**, the BlackBerry IDE 生成下面的注释：:

```
/**
* <description>.
*/
```

BlackBerry IDE 预先加载<*description*>.作为查询字符串，再查询第一个实例，然后选择这个实例，这个特性允许你输入描述，然后按 F3 转移到后面的参数。
因为 javadocs 宏依赖于分解浏览的信息，仅在成功完成一个编译后增加 javadocs。如果你的文件包含一个语法错误，并且在上面你正试着插入注释，宏不会得到函数声明。

增加一个新的编辑器宏

1. 在 **Edit** 菜单，点击 **Preferences**。
2. 点击 **Editor** 标签。
3. 点击 **Macros** 按钮。

4. 从 **When I type** 下拉列表中，选择/**.
5. 在 **Replace it with** 文本框里，输入**@javadoc**.
6. 在同一行或者每个函数声明的前一行输入/**。例如，在下面的代码段，在单词”protected”开头的插入点输入/**.

```
/** protected int makeMenu(Menu menu, int instance) { ... }
```

使用命令行

BlackBerry JDE 包含一个命令行编译器 RAPC。RAPC 编译.java 和.jar 文件到可以运行在 BlackBerry 设备模拟器或者加载到 BlackBerry 设备上的.cod 文件。
Rapc.exe 在 BlackBerry JDE 安装目录下的 Bin 下面。
RAPC 接收下面的命令行选项。

选项	描述
import	指明 RIM API 和其他依赖的库
codename	指明应用程序名（这典型是.jar 文件名）
midlet	指明应用程序是否是 MIDlet
jad	指明 JAD 文件名
\filename_1.java[附加.java 文件如果需要>]	指明.java 文件名，如果正在编译 java 文件。
\JAR_filename.jar	指明.jar 文件名，如果正在编译一个.jar 文件。

使用蓝牙开发环境

为了利用狼牙开发环境使用 BlackBerry 设备模拟器，你需要从 CSR（参看 http://www.btdesigner.com/devcasira.htm）得到普通开发系统。

利用一个 BlackBerry 设备模拟器使用蓝牙开发环境

1. 打开 BlackBerry IDE
2. 在主菜单，选择 **Edit**>**Preferences**.
3. 选择 **Simulator** 标签。
4. 选择 **Ports** 标签。
5. 在 **Communication port type** 域，选择合适的端口类型（参看 Casira Endpoint 文档）。
6. 在 Serial Port 域，输入端口信息。
7. 点击 **OK**。

使用 Eclipse 开发环境

Java 调试有线协议（Java Debug Wire Protocol, JDWP）的程序为 BlackBerry 模拟器提供一个接口。当你启动 JDWP 时，你可以使用第三方集成的开发环境。

启动 JDWP

>点击**开始>程序>Research In Motion>BlackBerry JDE 4.1.0>JDWP**.

注：在启动 JDWP 之前，你必须从 BlackBerry IDE 启动 BlackBerry 设备模拟器至少一次。你仅需要启动 JDWP 一次。为了启动一个设备模拟器，在 Eclipse 开发环境中，点击 **Run>Debug**.

连接 Eclipse 开发环境

注：在完成本节的任务之前，安装和配置 Eclipse 开发环境。

完成下面的步骤：

1. 扩展 Sun JRE。
2. 加入 API 文档。
3. 设置 Builder。
4. 设置项目变量。

扩展 Sun JRE

1. 建立你的工作空间和项目。
2. 启动 Eclipse 平台。
3. 在 Eclipse 任务栏，点击 **Window>Preferences**.
4. 扩展 **Java** 项。
5. 选择 **Installed JREs**。
6. 点击 **Add**。
7. 在 Add JRE 的窗口的 **JRE type** 域，选择 **Standard VM**。
8. 在 **JRE name** 域，为 JRE 输入一个名字。
9. 在 **JRE home directory** 域，输入 Sun JRE 的位置。例如：

   ```
   C:\Java\jdk1.5.0_02\jre.
   ```

10. 务必使 **Use default system libraries** 域没有选。
11. 点击 **Add External JARs**。
12. 浏览你的 BlackBerry IDE 安装目录下的 **lib** 目录，例如：

    ```
    C:\Program Files\Research In Motion\BlackBerry JDE 4.1.0\lib
    ```

13. 选择 **net_rim_api.jar**.
14. 点击 **Open**。

加入 API 文档

1. 加入 RIM.jar 到你的项目。
2. 在 Add JRE 窗口，扩展 **net_rim_api.jar** 文件。
3. 选择 **Javadoc location**.
4. 点击 **Edit**。
5. 点击 **Browser**。
6. 找到找到你的 BlackBerry IDE 目录下的 **docs\api** 目录。例如：

C:\Program Files\Research In Motion\BlackBerry JDE 4.1.0\docs\api

7. 点击 **OK**。
8. 点击 **OK**。
9. 在 Installed JREs 窗口，选择新创建的 JRE，缺省的是 **RIM JVM**。
10. 在 Add JRE 窗口，点击 **OK**。

设置 Builder

1. 在 Eclipse 任务栏，点击 **Project>Properties**.
2. 选择 **Builders**。
3. 点击 **New**。
4. 在 Choose configuration type 窗口，选择 **Program**.
5. 点击 OK。
6. 在 New_Builder 窗口属性的 **Name** 域，为你的 Builder 输入一个名字。
7. 在 **Location** 域，点击 **Browser File System**。
8. 找到你的 BlackBerry IDE 目录下的 **Bin** 目录，例如：

C:\Program Files\Research In Motion\BlackBerry JDE 4.1.0\bin

9. 选择 **rapc.exe** 文件。
10. 点击 **Open**。

设置项目变量

1. 在 **Working Directory** 域，点击 **Variables**。
2. 在 Select Variable 窗口，选择 **build project**.
3. 点击 **OK**。
4. 在 **Arguments** 域，输入：

-quiet [desired space separated java, class, jar, or jad files] import="C\Program Files\Research In Motion\BlackBerry JDE 4.1.0\lib\net_rim_api.jar" codename=C:\Development\ProjectName

例如：

-quiet C:\Development\TestProject*.java import="C:\Program Files\Research In Motion\BlackBerry JDE 4.1.0\lib\net_rim_api.jar" codename=C:\Development\TestProject.

5. 点击 **OK**。
6. 在 New_Builder 窗口属性里，点击 **Build Options** 标签。
7. 在 Run the builder 部分，验证下面的选项是否选择了。
   - **After a "Clean"**
   - **During manual builds**
   - **During auto builds**
8. 点击 **OK**。
9. 在属性窗口，点击 **OK**。

注：RAPC 不支持通配符，如果输入的路径发生错误，使用空格分隔文件列。例如将 **C:\Development\TestProject*.java** 代替为 **C:\Development\A.java C:\Development\B.java**.

设置连接时间

当在 Eclipse 开发环境里调试时，为了阻止连接超时，为正在调试的程序设置超时连接值。

1. 在 Eclipse 的任务栏，点击 **Windows**>**Preferences**.
2. 扩展 **Java** 项。
3. 选择 **Debug**。
4. 在 **Communication** 部分的 **Debugger timeout** 域，输入值。
5. 在 **Launch timeout** 域输入值。

注：你在 **Debugger timeout** 和 **Launch timeout** 设置的值依赖你的计算机处理速度。如果设置这些域之后连接问题继续出现，增加超时的值。

使用 Eclipse 开发环境进行调试

1. 在 Eclipse 点击 **Run**>**Debug**.
2. 选择 **Remote Java Application**.
3. 点击 **New**。
4. 点击 **Source** 标签。
5. 确认你的程序是否列出。
6. 点击 **Close**。
7. 打开 **JDWP** 程序，为了得到更多信息，参看 27 页的“启动 JDWP “。
8. 在 Eclipse 任务栏，点击 **Run**>**Debug**.
9. 在 **Remote Java Application** 项下面，选择一个应用程序。
10. 点击 **Debug**。

注：如果出现下面的错误信息：“Failed to connect to remote VM. Connection timed out”,增加调试超时时间。为得到更多信息参看 29 页的”设置连接时间”.

编程指南

编写高效的代码

使用本地变量

不管什么时候，尽量使用本地变量。访问本地变量比访问类的成员高效。

使用速记判断 Boolean 条件

为了代替第一个例子中没有必要的判断 Boolean 条件，使用第二个例子中的速记。最后编译的代码会更短：

```
if( boolean_expression == true ) {
    return true;
    }
```

```
else {
    return false;
 }
// Do this
return( boolean_expression );
```

使类为 final

当你创建一个代码库时，如果你知道他们永远不会被扩展，那么将他们标记为 final。final 关键字的出现允许编译器生成更高效的代码。


注：缺省，BlackBerry JDE 编译器标记你应用程序.cod 文件中不会扩展的类为 final。

使用 int 代替 long

在 Java 中，一个 long 代表的是 64 位的整数。因为 BlackBerry 设备使用的是一个 32 位的处理器，如果你是用 int 代替 long，操作将会快 2－4 倍。

避免垃圾回收

避免调用 System.gc()进行垃圾回收。这个操作会占用许多时间，特别是在内存受限的 BlackBerry 设备上。让虚拟机进行垃圾回收。

对字符串使用静态变量

当定义 String 类型的静态字段(也成类字段)，可以用静态变量(非 final)代替常量(final)加快程序速度。反之，对于原始数据类型，例如 int，也成立。
例如，你可能创建一个如下的 String 对象：

```
private static final String x = "example";
```

对于这个静态常量（由 final 关键字标识），你使用常量的每个时候都会创建一个临时的 String 对象。在字节代码中，编译器去掉”x”，代替它的是字符串“example”，以致每次引用”x”时 VM 都会进行一次哈希表查询。
相比之下，度于静态变量(非 final 关键字)，字符串只创建一次。仅当初始化“x”时，VM 才进行哈希表查询。


注：你可以使用公共常量（也就是 final 字段），但是标记变量为私有。

避免 String(String)的构造子

避免使用 java.lang.String(String)构造子，因为它创建了一个没有必要的 String 对象，这个对象是作为参数提供的一个字符串的拷贝。因为 String 对象创建后不可以修改，所以拷贝典型没有必要。


注：当使用字符串构造子时，编译器会由警告。

```
String str = new String("abc"); // 避免.
String str = new String("found " + n + " items"); // 避免.
```

在 Java 程序里，每个引用的字符串都作为一个 java.lang.String 类的对象。换言之，你可以编写如下面的代码来创建一个 String。

```
String str = "abc"; // 建议.
String str = "found " + n + " items"; // 建议.
```

编写有效的循环

在一个循环外考虑循环不变的代码。

```
//避免
for( int i = 0; i < vector.size(); i++ ) {
...
}
```

在这个实现中，在每次的迭代中 vector.size()被调用，这是低效的。如果你的容器可能不止一个元素，将容器的大小赋值给本地变量。下面的代码移除了循环不变的代码:

```
// 建议
int size = vector.size();
for( int i = 0; i < size; ++i ) {
...
}
```

另外，如果你迭代的项的顺序并不重要，你可以向后迭代来避免栈上多余的本地变量，并且可以使比较更加快速。

```
for( int i = vector.size() - 1; i >= 0; --i ) {
...
}
```

优化子表达式

假如你使用相同的表达式 2 次，不要依赖编译器为你优化。使用本地变量，如下代码：

```
one( i+1 ); two( i+1 ); // Avoid.
int tmp = i+1; one( tmp ); two( tmp ); // Prefer.
```

优化除法操作

除法操作在 BlackBerry 设备上可能慢，因为处理器没有硬件触发指令。

在你的代码中，当一个正数除以 2 时，使用向右移一位(>>1)代替.仅当你知道你正在处理的正数时使用“向右移位”（>>）.

```
midpoint = width / 2; // Avoid.
int = width >> 1; // Prefer.
```

避免 java.util.Enumeration

避免使用 java.util.Enumeration 对象，除非你想隐藏数据（换句话说，为了返回一个数据的枚举代替数据本身。

```
// Avoid.
for (Enumeration e = v.elements(); e.hasMoreElements();) {
o = e.nextElement();
...
}
```

为一个 Enumeration 对象请求一个向量或者哈希表速度慢，并且创建了不必要的垃圾。代替它的是，迭代元素本身，如下面的例子：

```
// Prefer.
for( int i = v.size() - 1; i >=0; --i ) {
o = v.elementAt( i );
...
}
```

如果向量可能被其他线程修改，同步迭代，如下例子所示：

```
synchronized( v ) {
    for( int i = v.size() - 1; i >=0; --i ) {
        o = v.elementAt( i );
        ...
        }
}
```

注：Java SE 使用一个 Iterator 对象实现类似的功能，但是 iterator 在 Java ME 不可用。

使用 instanceof 进行转型

使用 instanceof 代替捕捉一个 ClassCastException 异常来判断转型是否成功。

```
// Avoid.
try
{
    (String)x.whatever();
    }
catch( ClassCastException e ) {
        ...
}

// Prefer.
if( x instanceof String ) {
    (String)x.whatever();
    }
else {
...
}
```

使用 instanceof 比用 try/catch 要快。当转型失败发生异常时才使用 try/catch。在紧跟由一个 instanceof 检查的条件语句的第一个代码块里，BlackBerry IDE 编译器和虚拟机被优化为仅对一个类检查。在由一个 instanceof 检查的条件语句后面的转型利用了这个优化。

例如，编译器可以优化第一个例子，但是第二个不能:

```
// Prefer.
if ( a instanceof <type> ) {
    <type> instance = (<type>)a;
```

```
    x.method(instance);
    instance.method(x, y, z);
}

// Avoid.
if( a instanceof <type> ) {
    x.method( (<type>)a );
}
```

使用 instanceof 判断条件

为了编写较小而快的代码，如果使用 instanceof 判断条件，不要显式判断一个变量是否为 null。当"e"为 null 时，表达式 e instanceof <type>判断为 false。

```
// Avoid.
if( e != null && e instanceof ExampleClass ) {
if( e == null || ! ( e instanceof ExampleClass)

// Prefer.
if( e instanceof ExampleClass ) { ... }
if( ! ( e instanceof ExampleClass ) ) { ... }
```

避免使用 StringBuffer.append(StringBuffer)

CLDC 不包含 StringBuilder.append(StringBuilder)方法。采用将一个 string buffer 加到另一个的方法会创建一个 String 的中间对象。代替它的是，应用程序可以使用 net.rim.device.api.util.StringUtilities.append(StringBuffer dst, StringBuffer src[, int offset, int length]).

```
// Avoid.
public synchronized StringBuffer append(Object obj) {
    return append(String.valueOf(obj));
}
// Prefer.
public synchronized StringBuffer append(Object obj) {
    if (obj instanceof StringBuffer) {
        StringBuffer sb = (StringBuffer)obj;
        net.rim.device.api.util.StringUtilities.append( this, sb, 0,
sb )
        return this;
}
return append(String.valueOf(obj));
}
```

减小代码大小

当编写应用程序时可以采用下面的指南来减小编译后代码的大小。

设置适合的访问方式

当你创建代码库时，为字段和方法使用合适的访问权限可以显著减小编译后代码的大小。特殊的是，完成以下操作:

- 不管什么时候，只要可能就将字段声明为 private。除了好的编码实践外，这可以使编译器优化.cod 文件。
- 当可能时，使用缺省(包)的访问方式来代替 public 访问(也就是，忽略 public 和 protected 关键字)

避免创建接口

当创建 API 库时，避免创建接口，除非你预知 API 的多个实现。接口会产生更大更慢的代码。

使用内部的静态类

当创建一个内部的类隐藏一个在其他类里的类时，但是内部类没有引用外部类对象，声明这个内部类为 static。这个操作压缩了对外部类引用的创建。

例如，下面的代码需要一个对外部类对象的引用。

```
// Avoid.
class outer {
    int i;
    class inner {
        inner() {}
        int example() { return i; }
        }
    }
```

比较而言，下面的代码仅仅定义了内部类名的范围：

```
// Prefer.
class outer {
    static class inner {
        ...
    }
}
```

前一个例子是下面的缩写版本：

```
class outer {
...
}
class outer$inner {
...
}
```

当在内部类的方法里需要访问外部类数据时，仅仅使用一个非静态的内部类。如果为命名范围使用一个类，那么使这个类为 static。

避免没有必要的初始化

在类里避免没有必要的字段初始化，这些类里，字段有缺省值。如果在一个类里没有初始化一个字段，它会自动使用下面的缺省值初始化字段。

- 对象引用初始化为 null
- int,byte 或 long 初始化为 0
- boolean 初始化为 false

例如，下面的代码段没有不同：

```
// Avoid.
class BadExample {
    private int fieldsCount = 0; // Avoid.
    private Field _fieldWithFocus = null; // Avoid.
    private boolean _validLayout = false; // Avoid.
    }

// Prefer.
class BetterExample {
    private int fieldsCount; // Prefer.
    private Field _fieldWithFocus; // Prefer.
    private boolean _validLayout; // Prefer.
    }
```


注:在一个方法里，必须显式初始化本地变量。

导入单独的类

一个应用程序仅使用了来自一个包的少量的类，这个程序应该导入单独的类，而不是整个库。

```
// Avoid.
import net.rim.blackberry.api.browser.*;

// Prefer.
import net.rim.blackberry.api.browser.Browser;
```

在 BlackBerry 设备上使用时间

在对时间敏感的应用程序里，不要为任何事物依赖时间区域，除了显示本地时间给用户。

BlackBerry 设备钟

BlackBerry 设备操作系统从 January 1, 1970 (UTC)的午夜以毫秒来计算绝对时间。时间一般以 CPU 周期或毫秒来计量的。

系统时间区域改变

如果因为性能原因正在缓存对时间敏感的对象，那么记住 BlackBerry 设备上的系统时间区域可能会改变。

当时间区域改变时，系统会发送一个全局的事件消息给应用程序。GlobalEventListener 的实现，包括 eventOccurred()，会接受这个事件。利用 invoking Application.addGlobalEventListener()注册你的实现。

```
public void eventOccurred( long guid, int data0, int data1, Object
```

```
object0,Object object1 ) {
        if( guid == DateTimeUtilities.GUID_TIMEZONE_CHANGED ) {
            _cal.setTimeZone( TimeZone.getDefault() );
            }
        }
```

决定手持设备上的网络时间

调用 `RadioInfo.GetNetworkTime(long deviceTime)`得到以毫秒计量的对应网络报告时间，然后调整本地时间。`deviceTime` 参数代表现在的毫秒级时间。

建议的实践

使用多线程

有效的利用 BlackBerry 操作系统多线程的能力。特殊地，为网络连接或长操作（大于 0.1 秒）创建线程。为监听者使用背后（Background）线程，或者当程序启动时使用在背后运行地其他进程。

最小化内存地使用

为了最小化运行时内存，使用下面地指南:

- 使用原始类型（如 int 或 Boolean）代替对象（如 String 或 Integer）。
- 不要全部依赖垃圾回收。避免快速地创建多个对象。当完成使用他们时，将对象引用设置为 null。尽可能重用对象。
- 将大地操作一到 Server 上，例如，在发送数据到 BlackBerry 设备之前，完成对数据地过滤或排序。

避免返回 null

如果你正在编写一个公共地方法返回一个对象，仅在下面地条件下它可以返回一个 null：

- 在正常地程序运行期间，null 对象时期望的。
- `Javadoc @return` 参数描述了 `null` 是一个可能的返回值。

如果一个 `null` 返回值不是正常期望的，那么程序将抛出一个合适的异常强迫调用者显式的处理这个问题。调用者不期望检验一个 `null` 的返回值，除非文档说明了。

避免传递 null 给方法

不要传递一个 `null` 参数给 `API` 方法，除非 `API` 引用显式说明了方法支持他们。

小心传递 null 参数给构造子

当传递 `null` 参数给构造子时，为了避免混淆，将 `null` 转化为合适的对象：

```
new someObject ( (someObject)null );
```

如果一个类有两个或多个构造子，传递 `null` 参数可能不会唯一识别哪一个构造子将会使用。结果编译器会报错。在 `API` 参考里，并不是所有的构造子都会出现，因为有些构造子仅供内部使用。

通过转化 `null` 为合适的对象，你可以明确指明编译器会使用哪一个构造子。如果后续的 `API` 发行版本增加了新的构造子，它也可向前兼容。

使用 long 标记唯一标志符

使用一个 long 的标志符代替 String 标志符来标记唯一的常数，如 GUID，哈希表键值，以及状态或上下文标志。
对于跨越第三方应用程序的标志符，为了保留其独立性，使用基于 string 生成的哈希生成的键值。在输入字符串里，包含了足够的信息提供唯一性。例如，使用一个完全信任的包名，如 com.rim.samples.docs.helloworld。

转化一个 string 为 long

1. 在 BlackBerry IDE 文本编辑器里，输入一个字符串。
2. 选择字符串。
3. 右击字符串。
4. 选择 **Convert"String" to Long.**

正确退出应用程序

在调用 System.exit(int status)之前，你的程序应该完成任何清理，例如移除在运行时存储的程序不在需要的对象。

打印栈跟踪（Stack trace）

当 VM 发现代码使用 catch(Exception e)捕获异常时，VM 优化为排除栈跟踪。如果捕获到 Throwable，它不会排除栈跟踪。
例如，下面的代码不会排除栈跟踪：

```
catch (IOException e) {
    e.printStackTrace()
}
```

为了打印栈跟踪，编写类似下面的代码：

```
catch (Throwable t) {
    t.printStackTrace();
}
```

当你调试时为了查看栈跟踪，捕获一个 Throwable 实例。

3

第 3 章 创建用户接口（UI）

UI API

当你为 BlackBerry 设备编写应用程序时，使用下面 2 组 UI API 的一组：

- MIDP UI API (javax.microedition.lcdui 包)
- BlackBerry UI API (net.rim.device.api.ui 包)

如果你正在编写一个在任何 MIDP 兼容设备上运行的应用程序，请使用 MIDP UI API. 如果你正在编写专门运行在 BlackBerry 设备上的应用程序，那就使用 BlackBerry UI API 吧。BlackBerry API 提供了访问 BlackBerry 设备的特定特性的功能，并且也允许更成熟的 UI 布局（layout）和交互。

注:不要在同一个程序里既使用 MIDP UI API，又使用 BlackBerry UI API，否则会抛出异常。在应用程序中，UI 框架支持一中类型的 UI 对象。

显示 UI 组件

显示屏幕(Screen)

UI 的主要结构是 Screen。一个应用程序一次只能显示一个屏幕。

注:不要使用 Screen 对象来输入文本。Screen 对象没有明确实现此功能，它需要复杂的输入方法，例如国际化的键盘和 7100 系列的设备。为实现无缝得集成不同输入方法，扩展 Field 或者其任一子类。参看 53 页“*创建定制的域*”得到更多信息。

显示栈（Stack）

Screen 对象在一个一组有序的 Screen 显示栈里得到维护。在栈顶的 Screen 对象是显示给用户的活动 Screen。当应用程序显示一个 Screen 时，它将这个 Screen 压入到栈顶。当关闭一

个 Screen，将这个 Screen 从栈里移出，然后显示栈里的下一个 Screen，如果必要会重绘它。


注:每个 Screen 在栈里只出现一次。如果同一个 Screen 压入到栈不止一次，VM 会抛出一个运行时异常。当用户完成和 Screen 交互，应用程序必须将 Screen 从栈里移出，以致内存不必再用。不要在同一时间里使用多个 Screen，因为每个 Screen 使用独立的线程。

Screen 的类型

在多数情况下，创建一个 Screen 最有效的方法是创建一个扩展 Screen 或其任一子类，FullScreen 或 MainScreen 的类。

类	描述
Screen	使用 Screen 类定义一个管理器布局 Screen 上的 UI 组件，并且使用在超类 Field 定义的常数的样式（Style）定义一明确的 Screen。
FullScreen	缺省的，一个 FullScreen 包含单个垂直[①]的域管理器（Field Manager）。使用一个 FullScreen 提供了一个空的 Screen，在这个空的 Screen 上，你可以增加 UI 组件到这个标准的垂直布局里。如果需要另外类型的布局，例如水平的或对角的，使用一个 Screen 类，并且在里面增加一个管理器。
MainScreen	MainScreen 类提供常见的标准 BlackBerry 应用程序常见特性。对你的应用程序的第一个 Screen，使用一个 MainScreen 对象来保持和其他 BlackBerry 应用程序的统一。MainScreen 提供一下的 UI 组件： ● Screen 标题的缺省位置，标题后的一个 SeperatorField ● 一个包含在 `VerticalManager` 里的滚动的主界面。 ● 有一个 **Close** 菜单项的菜单。 ● 当用户点击 **Close** 菜单项或者按 Escape 键时缺省的关闭操作。

响应用户交互

BlackBerry API 提供一个和 Java 标准版本类似的事件监听框架。特殊的，2 个监听接口使程序接收和响应用户交互：`TrackWheelListener` 和 `KeyboardListnener`。`Screen` 类和其子类都实现了这些方法。

提供 screen 导航（navigation）

BlackBerry 应用程序为用户提供一个菜单来完成操作。避免使用按钮(Button)或其他占据 Screen 空间的 UI 组件。


注:按滑轮访问菜单。

当创建一个 `FullScreen` 或 `Screen`，在构造子里指明 `DEFAULT_MENU` 和 `DEFAULT_CLOSE` 参数来提供缺省的导航。

```
FullScreen fullScreen = new FullScreen(DEFAULT_MENU | DEFAULT_CLOSE);
```

参数	描述
`DEFAULT_MENU`	这个参数增加一个缺省的菜单，它包含了不同的菜单项，这依赖域用户的上下文环境。例如，如果一个 `EditField` 获得焦点，将显示 **Cut**, **Copy** 和 **Paste** 菜单项。所有已选择的域提供 **Select** 和 **Cancel**

① 在 BlackBerry Screen 布局里，有水平，垂直，对话框以及流四种。译者注。

	Selection 菜单项。
DEFAULT_CLOSE	这个参数增加一个缺省行为的 **Close** 菜单项到菜单，当用户点击 **Close** 菜单项或者按 **Escapes** 按钮，如果 Screen 上的任何东西改变，一个确认的对话框将会出现。如果这个 Screen 是栈里的唯一一个 Screen，应用程序将关闭。

当创建一个 MainScreen 时，缺省的导航会自动提供。

增加菜单项

创建 MenuItem 对象。

```
private MenuItem viewItem = new MenuItem("View Message", 100, 10) {
    public void run() {
        Dialog.inform("This is today's message");
        }
    };
```

MenuItem 构造子接受下面的 3 个参数：

参数	描述
text	菜单项的名称
ordinal	菜单项的顺序；一个越大的值表明了这个菜单项越靠近菜单的底部。
priority	接收缺省焦点的菜单项优先级

run（）定义了当用户点击菜单项发生的操作的实现。如果你没有使用本地资源，重写 toString（）方法来指定菜单项的名字。

为了在应用程序加入上下文菜单给 field，调用 getLeafFieldWithFocus（），并且调用 getContextMenu（），其返回值决定哪一个 Field 接收 makeMenu（）里的客户化菜单项。为了得到更多信息，参看 60 页的“创建客户化的上下文菜单”。

当增加你自己的菜单项时，显式的定义一个 **Close** 菜单项。

为了增加菜单项到 Screen 里，重写 Screen.makeMenu()方法：

```
protected void makeMenu(Menu menu, int instance) {
    menu.add(viewItem);
    menu.add(closeItem);
}
```

如果你扩展 Screen 或其任一子类，那么当用户点击滑轮时，缺省的 TrackwheelListener 实现调用 makeMenu（）。

如果你没有扩展 Screen，那么实现 TrackwheelListener。特殊地，trackwheelClick（）的实现创建一个新的菜单，增加菜单项以及在 Screen 上显示菜单。

```
public boolean trackwheelClick(int status, int time) {
    Menu appMenu = new Menu();
    makeMenu(appMenu, 0); // Add menu items.
    appMenu.show(); // Display the menu on screen.
    return true;
  }
```

注:为了创建菜单项提供附加的功能，请扩展 MenuItem 类。为了得到更多信息，参

看 60 页“创建客户化的上下文菜单”。

显示对话框

PopupScreen 类通过使用它的子类，Dialog 和 Status，来提供创建对话框和状态 Screen 的特性。Popup screen 不会压入到显示栈中，为了显示一个 popup screen，调用 Dialog.ask(int)或 Status.show()。

为了控制对话框的布局，使用 DialogFieldManager 对象，为了得到更多的信息，参看 50 页的“为一个 PopupScreeen 指定布局”。

为了显示一个对话框，使用下面的一个参数来调用 Dialog.ask（）：

参数	描述
D_OK	显示一个字符串，并且提示用户点击 OK。
D_SAVE	实现一个字符串，并且提示用户点击 **Save**，**Discard**，或者 **Cancel**；按 **Escape** 取消。
D_DELETE	显示一个字符串，并且提示用户点击 **Delete** 或者 **Cancel**；按 **Escape** 撤销。
D_YES_NO	显示一个字符串，并且提示用户点击 **Yes** 或 **No**。

```
int response = Dialog.ask(Dialog.D_SAVE);
if (Dialog.SAVE == response || Dialog.CANCEL == response)
    return false;
if ( Dialog.DISCARD == response )

    _item.deleteItem(_itemIndex);
```

为了指定一个对话框的缺省的响应，使用一个接受 defaultChoice 作为参数的 Dialog.ask()版本。

```
int response = Dialog.ask(Dialog.D_YES_NO, "Are you sure?",
Dialog.NO);
```

显示状态消息

调用 Status.Show()显示一个状态消息。缺省的，状态屏幕保留其屏幕 2 秒钟。

```
Status.show("Status screen message");
```

参看 *API 参考*获取 Status.Show()的版本信息，它使你可以指定额外的参数，例如不同的图标或者保持状态对话框可见的时间长短。你可以创建模态的状态对话框（需要用户取消它们），也可以创建计时的状态对话框（在指定的时间后自动取消）。

显示域（Field）

所有 UI 组件以包含在管理器里的成矩形的 field 的形式表现。Field 的大小取决于它的布局需求。管理器为它们包含的 field 提供滚动（条）。

BlackBerry JDE 在 net.rim.device.api.ui.component 包里提供一个预创建接口控件和组件的库。多数情况下，你可以使用这些对象构建 UI 应用程序。

为了创建指定的 field 控件（如包含多个元素的文本 field），扩展 Field 类或者其任意子类来创建你自己定制的类型。为得到更多信息，参看 53 页的“创建定制的 field”。

注：参看 *API 参考*获取更多关于指定 field 类的有效、支持的格式的信息。如果使用一个不支持的格式实例化一个 Field，将抛出一个 IllegalArgumentException 异常。

Bitmap Field

一个 BitmapField 包含了位图。当使用 Graphics 对象绘图时使用 BitmapField。为了修改一个 field 的内容，调用 BitmapField 的绘图方法。为得到更多信息，参看 73 页的“使用 graphics 对象绘图”.

```
Bitmap myBitmap = Bitmap.getPredefinedBitmap(Bitmap.INFORMATION);
BitmapField myBitmapField = new
BitmapField(myBitmap.getPredefinedBitmap(myBitmap));
 ...

mainScreen.add(myBitmapField);
```

有 4 种预定义的位图：

- Bitmap.INFORMATION
- Bitmap.QUESTION
- Bitmap.EXCLAMATION
- Bitmap.HOURGLASS

为了使用原始的.gif 或.png 作为位图，调用 getBitmapResource().

注：一个二进制资源的大小，如一个.png 文件，不能超过 63，000 字节。

```
private static final Bitmap myBitmap =
    Bitmap.getBitmapResource("customBitmap.gif");
...
BitmapField bitmapField = new BitmapField(myBitmap);
mainScreen.add(bitmapField);
```

Button Field

ButtonField 包含了用户选择来完成操作的按钮。使用 ButtonField 可以创建超出菜单的扩展交互的界面。

```
ButtonField mySubmitButton = new ButtonField("Submit");
ButtonField myResetButton = new ButtonField("Reset");
mainScreen.add(mySubmitButton);

mainScreen.add(myResetButton);
```

为了给 button 增加功能，扩展 ButtonField 并且覆写 trackwheelClick（）方法，以让它能完成一个操作来代替调用菜单。当用户点击 button 后为了接受消息，使用一个 FieldChangeListener 对象。为得到更多信息，参看 78 页的“监听 UI 对象的改变”.

Choice Field

Choice field 类似于下拉列表。这里有 2 种 choice field：包含整数的和包含可以转化为字符串的对象。

你也可以显示一组选项作为 check box 或者 radio button。为了得到更多信息，参看 45 页的“option field”。

为了从 ChoiceField 里选择一个值，用户可以完成下面的操作：

- 点击 field，并且按 Space 键。
- 按住 Alt 键，滚动滑轮。
- 打开菜单，选择 Change Option。

类	描述
NumericChoiceField	NumericChoiceField 是一个包含了一组数字的 ChoiceField。NumericChoiceField 实例典型的用在较少的数字范围内（到 20 个之多）。 `NumericChoiceField myNumericChoice = new NumericChoiceField( "Select a number: ", 1, 20, 10); mainScreen.add(myNumericChoice);` 注：对于大数量的数字，使用 GaugeField，为得到更多信息，参看 47 页的“Gauge Field)。
ObjectChoiceField	ObjectChoiceField 是一个包含了对象的 ChoiceField。这个 Field 的所有对象必须实现 Object.toStirng() 以提供他们自己的字符串表现形式。

Option Field

OptionField 允许用户从列表种选择条目。为允许用户从选择列表中选择多个条目，使用 CheckBoxField。为允许用户从选择列表中仅选择一个条目，使用

RadioButtonField。

类	描述

CheckBoxField	每个 CheckBoxField 是一个独立的对象，与其他的可选框无关。
	`CheckboxField myCheckbox =`
	`  new CheckboxField("First checkbox", true);`
	`CheckboxField myCheckbox2 =`
	`  new CheckboxField("Second checkbox", false);`
	`...`
	`mainScreen.add(myCheckbox);`
	`mainScreen.add(myCheckbox2);`

RadioButtonField	多个 RadioButtonField 对象组合在一个 RadioButtonGroup 中，这样用户一次只能选择一个选项。
	`RadioButtonGroup rbGroup = new RadioButtonGroup();`
	`RadioButtonField rbField = new RadioButtonField("First field");`
	`RadioButtonField rbField2 = new RadioButtonField("Second field");`
	`...`
	`rbGroup.add(rbField);`
	`rbGroup.add(rbField2);`
	`...`
	`mainScreen.add(rbField);`
	`mainScreen.add(rbField2);`

Date Field

在你的应用程序中，一个 DateField 显示当前的日期和时间。

类型	描述
DATE	显示年月日
DATE_TIME	显示年月日，时分秒
TIME	显示时分秒

当创建一个 DateField 时，调用 System.currentTimeMillis()得到当前时间。

```
DateField dateField =
    new DateField("Date: ", System.currentTimeMillis(),
            DateField.DATE_TIME);
mainScreen.add(dateField);
```

Date Field 缺省为可编辑的。为了创建一个用户不能编辑的 Date Field，在其构造子中指定 Field.READONLY 参数。

将为可编辑的 Date Field 提供一个缺省的 **Change Options** 菜单项。

Edit Field

一个 EditField 允许用户在此 Field 里输入文本。AutoTextEditField, EditField, 和 PasswordEditField 都扩展了 BasicEditField.


注：net.rim.device.api.ui.component.TextField 类，扩展了 Field 类，并且是抽象的。实例化它的子类，例如 RichTextField 或 EditField,就是创建一个显示文本或允许用户输入文本的 UI Field。

你可以应用下面的过滤项（filter）到 Edit Field 中。

过滤项	描述
DEFAULT_MAXCHARS	限制 field 中字符的个数。对于 Edit Field，字符的最大个数缺省为 15。
FILTER_DEFAULT	这个是缺省的文本输入过滤项。当构造子需要此过滤项，但是你又不想应用任何特定的过滤项时，那么使用它。
FILTER_EMAIL	仅允许有效的 internet 消息地址字符（例如，用户可以仅输入一个@标记）.它会自动将文本格式化为 internet 消息地址格式（例如，当用户第一次按 Space 键时，一个@符号会出现，用户接着按 Space 键，每次.'s 也随之出现。
FILTER_HEXADECIMAL	仅允许数字和 A 到 F 的字母。
FILTER_INTEGER	仅允许数字和负号“－“。
FILTER_LOWERCASE	将字符转化为小写
FILTER_NUMERIC	仅允许输入数字
FILTER_PHONE	仅允许输入有效电话号码字符，数字，连接号(-),加号和减号，左括号和右括号，以及”x”.
FILTER_PIN_ADDRESS	仅接受在 PIN 地址上输入的有效字符。
FILTER_UPPERCASE	将字母转化为大写。
FILTER_URL	仅允许有效的 URL 字符。它也自动格式化 field。（当用户按 Space 键时它将插入一个节点）.
JUMP_FOCUS_AT_END	改变 field 的行为，以致当 field 获得焦点，并且用户试图滚动时，焦点移动到 field 的末端（代替移动到下一个 field）。
NO_NEWLINE	忽略文本中的换行和回车，例如用户从其他地方拷贝和粘贴的文本。

类	描述
RichTextField	RichTextField 类创建一个只读的 Field，它可以体格式化为各种不同的字体以及样式，RichTextField 虽然不可以编辑，但是可以获取焦点。 `mainScreen.add(new RichTextField(`

	"RichTextField"));
BasicEditField	BasicEditField 是 EditField, 和 PasswordEditField 的基类。 BasicEditField 是一个可编辑的文本 Field，它没有缺省的格式，但是却可以接受过滤。 BasicEditField bf = new BasicEditField("BasicEditField: ", "", 10, EditField.FILTER_UPPERCASE); mainScreen.add(bf);
EditField	EditField 是一个可以编辑的文本 Field，它扩展了 BasicEditField。EditField 允许用户当问特殊的字符。例如，用户按住 **A** 键，并且滚动滑轮来选择各种样式的 A 字符以及Æ字符。EditField 类接受样式，但是有些样式会让 EditField 失去其功能（例如 EditField.FILTER_PHONE）. mainScreen.add(new EditField("EditField: ", "", 10, EditField.FILTER_DEFAULT));
PasswordEditField	PasswordEditField 扩展了 BasicEditField，提供下面的功能： ● 让用户输入显示为星号（*）。 ● 自动文本（AutoText）（或其他自动格式化）不会应用。 ● 剪切或拷贝不支持 下面的样例使用了一个构造子，允许你为 PasswordEditField 提供了一个缺省的初始化值。 mainScreen.add(new PasswordEditField("PasswordEditField: ", ""));
AutoTextEditField	AutoTextEditField 应用了 AutoText 引擎指定的格式。在本 Field 中输入的任何文本都会根据 BlackBerry 设备上的 AutoText 数据库说明来进行格式化。 某些过滤让一些 AutoText 输入没有效果。例如，FILTTER_LOWERCASE render 一个包含大写无效的 AutoText 输入。 mainScreen.add(new AutoTextEditField("AutoTextEditFi

	eld: ", ""));"

Gauge Field

Gauge 允许你创建数值的可视表现。GaugeField 显示一个进度条或允许用户选择数字。你可以使用一个 Label 作为它的前缀，并显示 gauge 的当前值。例如，组合一个 GaugeField 和一个 NumericChoiceField 来创建一个用户制作的数字选择的图形化表现。

为了创建一个交互的 GaugeField，使用 Field.FOCUSABLE 和 Field.EDITABLE 样式实例化 field。

```
GaugeField staticGauge = new GaugeField("1: ", 1, 100, 20,
GaugeField.NO_TEXT);
GaugeField percentGauge = new GaugeField("Percent: ", 1, 100, 29,
GaugeField.PERCENT)
GaugeField interactiveGauge = new GaugeField("Gauge: ", 1, 100, 60,
Field.FOCUSABLE | Field.EDITABLE);
...
mainScreen.add(staticGauge);
mainScreen.add(percentGauge);
mainScreen.add(interactiveGauge);
```

Label（标签）和 Separator（分隔）Field

一个 LabelField 允许你增加文本标签到屏幕中。LabelField 是可读的。缺省的，它不能获得焦点。大部分应用程序在它们的第一个屏幕上使用 LabelField 来显示一个静态的标题。

一个 SeparatorField 是一个静态的水平线，它跨越屏幕的宽度。使用 SeparatorField 将屏幕上的相关内容和菜单分组。

MainScreen 缺省的在标题后显示一个分割线。

```
LabelField title = new LabelField(
       "UI Component Sample",
       LabelField.ELLIPSIS));
mainScreen.setTitle(title);
```

List Field

List 允许你创建子项的目录，通过此用户可以滚动并选择单个或多个条目。BlackBerry 地址簿就是 List 对象的一个例子。

你不可以直接将内容加入到 field 条目中。你的 ListField 的 ListFieldCallback 和 TreeField 的 TreeFieldCallback 的实现会绘图 field。

类	描述
ListField	ListField 包含了数行可选条目。为了显示 ListField 的内容，为了列表设置 ListFieldCallback。为了得到更多信息，参看 66 页的“创建一个回调对象”. String fieldOne = new String("Mark Guo");

	String fieldTwo = new String("Amy Krul");
	...
	ListField myList = new ListField();
	ListCallback myCallback = new ListCallback();
	myList.setCallback(myCallback);
	myCallback.add(myList, fieldOne);
	myCallback.add(myList, fieldTwo);
	...
	mainScreen.add(myList);
	
	注：为了使用户选择列表中的多个条目，指定 ListField 作为 MULTI_SELECT.ListFieldCallback.add() 加入列表元素到向量中，并且调用 List.insert()决定适当的位置。

ObjectListField	ObjectListField 是一个包含以对象作为子项的 list field。所有包含在 list 里的对象必须实现 Object.toString()，以提供他们自己的字符串表现。在接口上，ObjectListField 以和一个标准的 ListField 一样提供。

Tree Field

TreeField 包含父节点和子节点，并且显示一个折叠夹或它们（例如文档或信息折叠夹）之间的树关系。所有节点都是缺省可见的。为了指明一个折叠夹是否可以折叠，调用 TreeField 对象的 setExpand()方法。
图标显示在包含有子节点的每个节点边上以明确节点是打开的还是折叠的。

```
String fieldOne = new String("Main folder");
...
TreeCallback myCallback = new TreeCallback();
TreeField myTree = new TreeField(myCallback, Field.FOCUSABLE);
int node1 = myTree.addChildNode(0, fieldOne);
int node2 = myTree.addChildNode(0, fieldTwo);
int node3 = myTree.addChildNode(node2, fieldThree);
int node4 = myTree.addChildNode(node3, fieldFour);
...
int node10 = myTree.addChildNode(node1, fieldTen);
myTree.setExpanded(node4, false);
...
mainScreen.add(myTree);
```

TreeFieldCallback 的实现加入 field 到树中。为了获得更多关于回调的信息，参看 66 页的“创建一个回调对象”.

```
private class TreeCallback implements TreeFieldCallback {
    public void drawTreeItem(TreeField _tree, Graphics g,
            int node, int y, int width, int indent) {
        String text = (String)_tree.getCookie(node);
        g.drawText(text, indent, y);
    }
}
```

管理 UI 组件

管理布局

使用 BlackBerry API 布局管理器来安排屏幕上的组件。
下面四个类扩展了 Manager 类，以提供预定义的布局管理器：

- VerticalFieldManager
- HorizontalFieldManager
- FlowFieldManager
- DialogFieldManager

MainScreen 和 FullScreen 缺省的都使用了一个 VerticalFieldManager。仅为这些类定义一个布局管理器实例提供了不同的布局。

注：为了创建一个定制的布局管理器，请扩展 Manager。为了得到更多信息，参看 63 页的“创建定制的布局管理器”。

为一个指定的 Screen 实例定义布局管理器，完成下面的操作：

- 实例化合适的 Manager 子类。
- 加入 UI 组件到布局管理器中。
- 加入布局管理器到屏幕中。

```
VerticalFieldManager vfm = new
VerticalFieldManager(Manager.VERTICAL_SCROLL);
vfm.add(bitmapField);
vfm.add(bitmapField2);
...
mainScreen.add(vfm)
```

Manager 类定义了多个系统样式的常数，这些系统样式定义了如滚动和对齐的行为。当创建布局管理器时，使用这些样式作为参数。为得到更多信息，参看 *API 参考*的 net.rim.device.api.ui.Manager。

垂直组织 field

VerticalFieldManager 垂直地组织 field。所有 field 在一新地线（line）上开始。为了可以垂直滚动，提供 Manager.VERTICAL_SCROLL 参数。

```
VerticalFieldManager vfm = new VerticalFieldManager(Manager.VERTICAL_
```

```
SCROLL);
vfManager.add(bitmapField);
vfManager.add(bitmapField2);
...
mainScreen.add(vfManager);
```

缺省地，BitmapField 对象在 VerticalFieldManager 中是左对齐的。

水平组织 field

HorizonFieldManager 水平组织 field。为了可以水平滚动，提供 Manager.HORIZONTAL_SCROLL 样式。如果没有包含 HORIZONTAL_SCROLL 参数，field 水平排列他们自己，可能会超出屏幕宽度，但是用户不能滚动到超出屏幕右边的内容。

BlackBerry 设备没有显示水平滚动指示器或滚动条。

```
HorizontalFieldManager hfm = new
HorizontalFieldManager(Manager.HORIZONTAL_SCROLL);
```

水平垂直组织 Field

FlowFieldManager 先水平组织 field，然后再垂直组织。先水平组织 Field，直到没有足够空间放另外一个 field，然后管理器在下一行上水平的安排它们。首页屏幕(Home Screen)就是一个 FlowFieldManager 的例子。

```
FlowFieldManager flManager = new
FlowFieldManager(Manager.FIELD_HCENTER);
```

指定一个 PopupScreen 的布局

DialogFieldManager 指定了 PopupScreen 对象的布局。它管理了一个图标，一个消息，以及一列定制的 field 的布局。图标和消息相互靠近的出现在布局上方，定制的 field 出现在消息的下方。这个布局是 PopupScreen 对象的标准布局。为了创建定制的对话框，扩展 DialogFieldManager。

```
BitmapField bitmapField = new
BitmapField(Bitmap.getBitmapResource("x.gif"));
RichTextField message = new RichTextField("Dialog manager message",
Field.NON_FOCUSABLE);
LabelField dialogChoice = new LabelField("Choice one",
Field.FOCUSABLE);
...
DialogFieldManager dialogManager = new DialogFieldManager();
dialogManager.setMessage(message);
dialogManager.setIcon(bitmapField);
dialogManager.addCustomField(dialogChoice);
```

管理 UI 交互

一个时间只有一个线程（通常是事件调配线程）可以得到 UI 的访问权。通过下列方式，背

后（Background）线程也可从主事件处理或 UI 绘制代码的外部访问 UI：

- 获取并保持事件锁。
- 使用 invokeLater（）或 invokeAndWait()在事件调配线程上运行。

获取并保持事件锁

当它处理一个消息时，事件调配者在事件线程上设置一个事件锁。在没有打断事件调配者处理的情况下，背后线程（也就是，非事件调配线程）在短事件内通过获取这个锁可以访问 UI。

为了得到事件锁，调用 Application.getEventLock（）。和这个对象同步，序列化访问 UI。在短期内保持这个锁，因为锁会暂停事件调配者。一个应用程序应该永远不要在 EventLock 对象上调用 notify()或 wait()。

```
class MyTimerTask extends TimerTask {
    public void run() {
        synchronized(Application.getEventLock()) {
            _label.setText("new text " + System.currentTimeMillis());
            }
    }
}
```

在事件调配线程上运行

如果保持事件锁不合适，创建一个实现 Runnable 接口的类。在事件调配者上通过下面的 3 种方法之一调用它的 run()方法:

- 调用 invokeAndWait(Runnable)，以致在事件调配线程上立即调用 run()。这个调用会阻塞直到 run()完成为止。
- 调用 invokeLater(Runnable)，以致在所有等候的事件处理后，在事件调配线程上调用 run()。
- 调用 invokeLater(Runnable，long，boolean)以致在某一指定时间后，事件调配线程上调用 run()。在这里，在将 Runnable 加入到事件队列之前，时间指定了等待时间的长短。如果 repeat 为 true，每隔 time 毫秒后，Runnable 加入到事件队列中，

管理前台事件

系统调用 Application.activate()将应用程序带到前台。

大多数的应用程序不需要重写 activate()。应用程序应该完成应用程序构造子的任何初始化，包括任何必需的 UiApplicaiton.pushScreen()调用.因为对同一个应用程序，activate()能够调用多次，因此在在这个方法中，应用程序不应该完成一次初始化。

当带到前台时，应用程序可以覆写 activate()方法完成其他的附加处理。如果覆写了 activate(),在方法的定义里调用 super.activate（），以致应用程序能正确得重绘。

管理绘图区域

使用 XYRect 对象

Graphics 对象代表了应用程序可用的整个绘图表面。为了界定这个区域，将它分为多个 XYRect 对象。XYRect 在图形上下文（graphics context）的顶端创建一个矩形区域、一个 XYRect 对象有 2 个 XYPoint 对象组成。第一个 XYPoint 对象代表了 XYRect 左上方的坐标，第二个 XYPoint 对象代表了右下方的坐标。每个 XYPoint 代表了一个由 X，Y 坐标构成的屏幕的坐标。

```
XYPoint topLeft = new XYPoint(10, 10);
XYPoint bottomRight = new XYPoint(50, 50);
XYRect rectangle = new XYRect(topLeft, bottomRight);
```

Rectangle 对象将 XYRect 对象的上下文绘制区域界定为（10，10）与（50，50）之间的区域。

为了开始对 XYRect 对象进行绘图调用，调用 pushContext()或 pushRegion（）；

当开始用 pushContext()进行绘图调用时，指定区域原点不要调整绘图偏移（Drawing offset）。

```
graphics.pushContext(rectangle, 0, 0);
graphics.fillRect(10, 10, 30, 30);
graphics.drawRect(15, 15, 30, 30);
graphics.popContext();
```

当你首先调用 pushRegion()来调用绘图方法时，区域源（Region Origin）需调整绘图偏移，左上方的 XYPoint 对象代表了区域源。所有绘图都通过这个数来偏移。

在下面的例子中，pushContext()将 XYRect 对象的 10 个象素位放到右边，10 个放在下方。区域源调整了绘图偏移（XYPoint topLeft = new XYPoint(10, 10)).

```
graphics.pushRegion(rectangle);
graphics.fillRect(10, 10, 30, 30);
graphics.drawRect(15, 15, 30, 30);
graphics.popRegion();
```

旋转（Invert）个区域

旋转一个 Graphics 对象上的一个区域，它保留像素，只是转化像素值的位（也就是 0 变为1，1 变为 0）。大多数 field 使用旋转来表示焦点，尽管这样，你可以为定制的 field 创建你自己的焦点行为。

为了旋转 Graphics 对象的任何一个区域，提供坐标或者旋转一个指定的 XYRect 对象。指定 Graphics 对象的一个区域，并且压入栈中。在调用 pushContext()或 pushRegion()后，提供 Graphics 对象的一个区域来旋转。

```
graphics.pushContext(rectangle);
```

```
graphics.invert(rectangle); // invert the entire XYRect object
graphics.popContext();
```

转化(Translate)一个区域

为了将一个 Graphics 上下文上的区域移动到另外一个地方。调用 invoke()。

```
XYRect rectangle = new XYRect(1, 1, 100, 100);
XYPoint newLocation = new XYPoint(20, 20);
rectangle.translate(newLocation);
```

XYRect 将点（1，1）转化为（20，20，）。转化后，XYRect 的底部扩展了过去图形上下文的范围，并且重合了。

创建客户定制的 UI 组件

你仅能将定制的上下文菜单项和布局增加到一个定制的 field 中。

创建定制的 field

为覆写 field 的缺省行为，创建一个定制的 field。

注:不要使用 Screen 对象来输入文本。Screen 对象没有明确的实现此功能，它需要复杂的输入方法，例如国际化的键盘和 7100 系列的设备。为了实现不同输入方法的无缝集成，扩展 Field 或者其任一子类。参看 53 页“创建定制的域”得到更多信息。

DrawStyle 接口的实现允许在定制的 field 上绘制样式。为获得更多信息，参看 73 页的“创建一个与标准 BlackBerry UI 一致的接口”。
客户定制的 field 应该实现所有相关的系统样式。例如，USE_ALL_WIDTH 和 USE_ALL_HEIGHT 适用于许多 field。

扩展 Field 类

扩展 Field 类和任一其子类，指定定制 Field 的特征。

```
public class CustomButtonField
  extends Field implements DrawStyle {
    public static final int RECTANGLE = 1;
    public static final int TRIANGLE = 2;
    public static final int OCTAGON = 3;
    private String _label;
    private int _shape;
    private Font _font;
    private int _labelHeight;
    private int _labelWidth;
}
```

定义按钮的标签，图形，以及样式

你的构造子的实现定义了按钮的标签，图形，以及样式。

```
public CustomButtonField(String label) {
    this(label, RECTANGLE, 0);
}

public CustomButtonField(String label, int shape) {
    this(label, shape, 0);
    }

public CustomButtonField(String label, long style) {
    this(label, RECTANGLE, style);
}

public CustomButtonField(String label, int shape, long style) {
    super(style);
    _label = label;
    _shape = shape;
    _font = getFont();
    _labelHeight = _font.getHeight();
    _labelWidth = font.getWidth();
}
```

指定 field 中对象的安排

任何扩展 Field 的类必须实现 layout（）.Field 管理器调用了 layout()方法来决定 field 应该如何根据可用的控件安排它的内容。

```
protected void layout(int width, int height) {
     _font = getFont();
     _labelHeight = _font.getHeight();
     _labelWidth = _font.getAdvance(_label);
     width = Math.min( width, getPreferredWidth() );
     height = Math.min( height, getPreferredHeight() );
     setExtent( width, height );
}
```

定义需要的宽度

注：在大多数情况下，通过覆写 getPreferredWidth()，确保合适的布局出现在定制的布局管理器里。

getPreferredWidth()的实现计算出定制 Field 的宽度，这个定制 Field 是基于标签 Field 的相对尺寸的。使用相对尺寸来确保标签不会超出标签的尺寸。

```
public int getPreferredWidth() {
    switch(_shape) {
    case TRIANGLE:
       if (_labelWidth < _labelHeight) {
```

```
            return _labelHeight << 2;
            }
        else
        {
            return _labelWidth << 1;
        }
    case OCTAGON:
        if (_labelWidth < _labelHeight) {
            return _labelHeight + 4;
            }
        else {
            return _labelWidth + 8;
        }
    case RECTANGLE:
    default:
        return _labelWidth + 8;
    }
}
```

定义需要的高度

注：在大多数情况下，通过覆写 getPreferredHeight()，确保合适的布局出现在定制的布局管理器里。

getPreferredHeight()的实现计算出定制 Field 的高度，这个定制 Field 是基于标签 Field 的相对尺寸的。它确保了标签不会超出 field 的尺寸。

```
public int getPreferredHeight()
{
    switch(_shape){
     case TRIANGLE:
       if (_labelWidth < _labelHeight){
           return _labelHeight << 1;
             }
           else {
             return _labelWidth;
             }
      case RECTANGLE:
           return _labelHeight + 4;
     case OCTAGON:
           return getPreferredWidth();
           }
       return 0;
    }
```

定义定制 field 的外观

paint()的实现定义了 BlackBerry 设备屏幕上的定制 Field 的外观，不管什么时候 Field 的域标记为无效，Field 管理器都调用 paint()来重绘 Field。

技巧：验证 paint()是否是有效率的，因为不管什么时候 field 发生变化，UI 框架调用 paint()方法。对于大数量的 field，使用 Graphics.getClippingRect()并在可见的区域里绘图来保存绘制时间。

```
protected void paint(Graphics graphics) {
    int textX, textY, textWidth;
    int w = getWidth();
    switch(_shape) {
        case TRIANGLE:
            int h = (w>>1);
            int m = (w>>1)-1;
            graphics.drawLine(0, h-1, m, 0);
            graphics.drawLine(m, 0, w-1, h-1);
            graphics.drawLine(0, h-1, w-1, h-1);
            textWidth = Math.min(_labelWidth,h);
            textX = (w - textWidth) >> 1;
            textY = h >> 1;
            break;
        case OCTAGON:
            int x = 5*w/17;
            int x2 = w-x-1;
            int x3 = w-1;
            raphics.drawLine(0, x, 0, x2);
            graphics.drawLine(x3, x, x3, x2);
            graphics.drawLine(x, 0, x2, 0);
            graphics.drawLine(x, x3, x2, x3);
            graphics.drawLine(0, x, x, 0);
            graphics.drawLine(0, x2, x, x3);
            graphics.drawLine(x2, x3, x3, x2);
            textWidth = Math.min(_labelWidth, w - 6);
            textX = (w-textWidth) >> 1;
            textY = (w-_labelHeight) >> 1;
            break;
        case RECTANGLE:
        default:
            graphics.drawRect(0, 0, w, getHeight());
            textX = 4;
            textY = 2;
            textWidth = w - 6;
            break;
```

```
        }
    graphics.drawText(_label, textX, textY, (int)( getStyle() &
DrawStyle.ELLIPSIS | DrawStyle.HALIGN_MASK ), textWidth );
    }
```

处理焦点事件

为了支持焦点事件，使用 Field.FOCUSABLE 样式以及实现 Field.moveFocus().如果你想你的 Field 接收焦点，覆写 Field.isFocusable()返回 true。

当 Field 获得焦点时，UI 框架调用 onFocus()，当 Field 失去焦点时，调用 unFocus().如果你的 field 对于这些事件需要特定的行为，覆写这些方法。框架调用 moveFocus()来处理 field 的焦点移动事件。它对应 trackwheelRoll 事件，覆写 drawFocus（）。

实现 set 和 get 方法

Field 的 get 和 set 方法的实现，增加了 Field 的能力。

注：所有 get 和 set 方法应该在 field 加入到一个 Screen 的前后工作。例如，如果现在屏幕上的 field 合适的调用了 invalidate()或 updateLayout()setLabel()，应该使用一个新值来修改其显示。

```
public String getLabel() {
    return _label;
    }
public int getShape() {
    return _shape;
    }
public void setLabel(String label) {
    _label = label;
    _labelWidth = _font.getAdvance(_label);
    updateLayout();
    }
public void setShape(int shape) {
    _shape = shape;
    updateLayout();
    }
```

代码实例

CustomButtonField.java 创建了具有多个图形的 button field。

实例：CustomButtonField.java

```
/**
* CustomButtonField.java
* Copyright (C) 2001-2005 Research In Motion Limited. All rights
reserved.
*/
```

```
package com.rim.samples.docs.custombuttons;
import net.rim.device.api.ui.*;
import net.rim.device.api.system.*;

/**
* CustomButtonField is a class that creates button fields of various
* shapes. This sample demonstrates how to create custom UI fields.
*/
public class CustomButtonField extends Field implements DrawStyle {
    public static final int RECTANGLE = 1;
    public static final int TRIANGLE = 2;
    public static final int OCTAGON = 3;
    private String _label;
    private int _shape;
    private Font _font;
    private int _labelHeight;
    private int _labelWidth;

    /* Constructs a button with specified label, and the default
style and shape. */
    public CustomButtonField(String label) {
        this(label, RECTANGLE, 0);
    }

    /* Constructs a button with specified label and shape, and the
default style. */
    public CustomButtonField(String label, int shape) {
        this(label, shape, 0);
        }

    /* Constructs a button with specified label and style, and the
default shape. */
    public CustomButtonField(String label, long style) {
        this(label, RECTANGLE, style);
        }

    /* Constructs a button with specified label, shape, and style */
    public CustomButtonField(String label, int shape, long style) {
        super(style);
        _label = label;
        _shape = shape;
        _font = getFont();
        _labelHeight = _font.getHeight();
```

```
    _labelWidth = _font.getAdvance(_label);
    }


/* Method that draws the focus indicator for this button and
 * * inverts the inside region of the shape.
 *
 * */
protected void drawFocus(Graphics graphics, boolean on) {
    switch(_shape) {
    case TRIANGLE:
        int w = getWidth();
        int h = w >> 1;
        for (int i=h-1; i>=2; --i) {
            graphics.invert(i, h - i, w - (i << 1), 1);
            }
        break;
    case RECTANGLE:
            graphics.invert(1, 1, getWidth() - 2, getHeight() - 2);
            break;
    case OCTAGON:
        int x3 = getWidth();
        int x = 5 * x3 / 17;
        int x2 = x3 - x;
        x3 = x3 - 1;
        x2 = x2 - 1;
        graphics.invert(1, x, getWidth() - 2, x2 - x + 1);
        for (int i=1; i<x; ++i) {
            graphics.invert(1+i, x-i,getWidth() - ((i+1)<<1), 1);
            graphics.invert(1+i, x2+i,getWidth() - ((i+1)<<1), 1);
        }
        break;
        }
    }

/* Returns the label. */
public String getLabel() {
    return _label;
    }

/* Returns the shape. */
public int getShape() {
    return _shape;
    }
```

```
/* Sets the label. */
public void setLabel(String label) {
    _label = label;
    _labelWidth = _font.getAdvance(_label);
    updateLayout();
    }

/* Sets the shape. */
public void setShape(int shape) {
    _shape = shape;
    updateLayout();
    }

/* Retrieves the preferred width of the button. */
public int getPreferredWidth() {
    switch(_shape) {
    case TRIANGLE:
        if (_labelWidth < _labelHeight) {
            return _labelHeight << 2;
            }
        else {
            return _labelWidth << 1;
            }
    case OCTAGON:
        if (_labelWidth < _labelHeight) {
            return _labelHeight + 4;
            }
        else {
            return _labelWidth + 8;
            }

    case RECTANGLE: default:
        return _labelWidth + 8;
    }
    }


/* Retrieves the preferred height of the button. */
public int getPreferredHeight() {
    switch(_shape) {
    case TRIANGLE:
        if (_labelWidth < _labelHeight) {
            return _labelHeight << 1;
```

```
            }
        else {
            return _labelWidth;
            }

    case RECTANGLE:
        return _labelHeight + 4;
    case OCTAGON:
        return getPreferredWidth();
        }
    return 0;
    }

/* Lays out this button's contents.
 * This field's manager invokes this method during the layout
 * process to instruct this field to arrange its contents, given
an
 *  amount of available space.
**/
protected void layout(int width, int height) {
    //      Update the cached font in case it has been changed.
    _font = getFont();
    _labelHeight = _font.getHeight();
    _labelWidth = _font.getAdvance(_label);
    //      Calculate width.
    width = Math.min( width, getPreferredWidth() );
    //      Calculate height.
    height = Math.min( height, getPreferredHeight() );
    //      Set dimensions.
    setExtent( width, height );
    }
    /*
    * Redraws this button. The field's manager invokes this method
during the
    * repainting process to instruct this field to repaint itself.
    */
    protected void paint(Graphics graphics) {
    int textX, textY, textWidth;
    int w = getWidth();
    switch(_shape) {
    case TRIANGLE:
        int h = (w>>1);
        int m = (w>>1)-1;
        graphics.drawLine(0, h-1, m, 0);
```

```
            graphics.drawLine(m, 0, w-1, h-1);
            graphics.drawLine(0, h-1, w-1, h-1);
            textWidth = Math.min(_labelWidth,h);
            textX = (w - textWidth) >> 1;
            textY = h >> 1;
            break;
        case OCTAGON:
            int x = 5*w/17;
            int x2 = w-x-1;
            int x3 = w-1;
            graphics.drawLine(0, x, 0, x2);
            graphics.drawLine(x3, x, x3, x2);
            graphics.drawLine(x, 0, x2, 0);
            graphics.drawLine(x, x3, x2, x3);
            graphics.drawLine(0, x, x, 0);
            graphics.drawLine(0, x2, x, x3);
            graphics.drawLine(x2, 0, x3, x);
            graphics.drawLine(x2, x3, x3, x2);
            textWidth = Math.min(_labelWidth, w - 6);
            textX = (w-textWidth) >> 1;
            textY = (w-_labelHeight) >> 1;
            break;
        case RECTANGLE: default:
            graphics.drawRect(0, 0, w, getHeight());
            textX = 4;
            textY = 2;
            textWidth = w - 6;
            break;
            }
        graphics.drawText(_label, textX, textY, (int)( getStyle() &
DrawStyle.ELLIPSIS | DrawStyle.HALIGN_MASK ),textWidth );
    }
    }
```

创建定制的上下文菜单项

在 Field 类里，创建定制的上下文菜单项。为得到更多关于实现的菜单的信息，参看 39 页的“显示屏幕”.

```
private MenuItem myContextMenuItemA = new MenuItem( _resources,
MENUITEM_ONE, 200000, 10)
    {
```

```
        public void run() {
            onMyMenuItemA();
            }
    };

    private MenuItem myContextMenuItemB = new MenuItem( _resources,
MENUITEM_ONE, 200000, 10)
    {
        public void run() {
            onMyMenuItemB();
        }
    };
```

提供一个上下文菜单

在主应用程序类里，覆写 makeContextMenu()方法提供一个上下文菜单。

```
protected void makeContextMenu(ContextMenu contextMenu) {
    contextMenu.addItem(myContextMenuItemA);
    contextMenu.addItem(myContextMenuItemB);
    }
```

创建应用程序菜单

在主应用程序类里，覆写 makeMenu()方法创建应用程序菜单，并且无论合十，当特定的 field 获取焦点时，更新上下文菜单。

```
protected void makeMenu(Menu menu) {
    Field focus = UiApplication.getUiApplication().getActiveScreen()
     .getLeafFieldWithFocus();
    if (focus != null) {
        ContextMenu contextMenu = focus.getContextMenu();
        if (!contextMenu.isEmpty()) {
            menu.add(contextMenu);
            menu.addSeparator();
            }
        }
    }
```

代码实例

实例：ContextMenuSample.java

```
/**
* ContextMenuSample.java
* Copyright (C) 2001-2005 Research In Motion Limited. All rights
reserved.
*/
```

```
package com.rim.samples.docs.contextmenus;
import net.rim.device.api.i18n.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import com.rim.samples.docs.baseapp.*;

public class ContextMenuSample
extends BaseApp implements ContextMenuSampleResource {
    private MyContextField myContextField;
    private static ResourceBundle _resources
    = ResourceBundle.getBundle(
            ContextMenuSampleResource.BUNDLE_ID,
            ContextMenuSampleResource.BUNDLE_NAME);

    public static void main(String[] args) {
        ContextMenuSample app = new ContextMenuSample();
        app.enterEventDispatcher();
        }

    // Inner class to define a new field.
    private class MyContextField extends RichTextField {
        private MenuItem myContextMenuItemA = new MenuItem(
                _resources, MENUITEM_ONE, 200000, 10) {
            public void run() {
                onMyMenuItemA();
                }
            };

    private MenuItem myContextMenuItemB = new MenuItem(
            _resources, MENUITEM_TWO, 200000, 10) {
        public void run() {
            onMyMenuItemB();
            }
        };

    private void onMyMenuItemA() {
        // Perform an action when user selects menu item.
        }

    private void onMyMenuItemB() {
        // Perform an action when user selects menu item.
        }
```

```
    protected void makeContextMenu(ContextMenu contextMenu) {
        contextMenu.addItem(myContextMenuItemA);
        contextMenu.addItem(myContextMenuItemB);
        }

    MyContextField(String text) {
        super(text);
        }
    }

    protected void makeMenu(Menu menu) {
        super.makeMenu(menu, 0); // Implemented by BaseApp.
        }

    public ContextMenuSample() {
        MainScreen mainScreen = new MainScreen();
        MyContextField myContextField = new MyContextField("Field
label: ");
        mainScreen.add(myContextField);
        mainScreen.addKeyListener(this);
        mainScreen.addTrackwheelListener(this);
        pushScreen(mainScreen);
        }

    public void onExit() {
        // Perform action when application closes.
        }
}
```

创建定制的布局管理器

Manager 对象管理 UI 组件的位置以及决定屏幕上的 field 如何安排。

创建一个定制的布局管理器

扩展 Manager 类或其任一子类

```
class DiagonalManager extends Manager {
    public DiagonalManager(long style){
    super(style);
    }
    ...
}
```

返回一个优先的 Field 宽度

覆写 getPreferredWidth()，以致它能为管理器返回一个优先的 Field 宽度。

getPreferredWidth()的实现可以返回不同的值，取决于布局管理器的目的。例如，如果管

理器扩展了 HorizontalFieldManager,getPreferredWidth()返回所有 field 宽度的总和。如果扩展了 VerticalFieldManager，getPreferredWidth()返回最宽 field 的宽度。

```
public int getPreferredWidth() {
    int width = 0;
    int numberOfFields = getFieldCount();
    for (int i=0; i<numberOfFields; ++i) {
        width += getField(i).getPreferredWidth();
        }
    return width;
    }
```

注：TextField 和 Manger 使用了指派给他们的整个宽度。为组织 2 个或更多的水平上的对象，分别覆写它们各自的 getPreferredWidth()方法。为了组织多个水平上的 TextField，覆写 layout().

返回一个优先 Field 高度

覆写 getPreferredHeight()，以致它能为管理器返回一个优先的 Field 高度。

```
public int getPreferredHeight() {
    int height = 0;
    int numberOfFields = getFieldCount();
    for (int i=0; i<numberOfFields; ++i) {
        height += getField(i).getPreferredHeight();
    }
    return height;
    }
```

指定子 Field 如何安排

subLayout()方法指定了管理器如何在屏幕上组织 field。它得到管理器中 field 的个数，然后为子 field 设置合适的位置以及布局。

layout()调用了 subLayout()方法，subLayout()方法通过调用每个管理器包含的 field 的 setPositionChild()以及 LayoutChild()，控制每个子 field 如何加到屏幕上。

```
protected void sublayout(int width, int height) {
    int x = 0;
    int y = 0;
    Field field;
    int numberOfFields = getFieldCount();
    for (int i=0; i<numberOfFields; ++i) {
        field = getField(i);
        layoutChild(field, width, height);
        setPositionChild(field, x, y);
        field.setPosition(x,y);
        x += field.getPreferredWidth();
        y += field.getPreferredHeight();
```

```
        }
    setExtent(width,height);
    }
```

注：为设置 field 需要的大小，在 subLayout()方法里调用 setExtend()。如果你不调用 setExtend()，则不会绘制每个 field，并且也不抛出一个异常。

处理焦点

当用户滚动滑轮时，为了指定 field 该如何得到焦点，覆写 nextFocus()方法。direction 参数描述了焦点移动的方向(一般来说，当滑轮向下滚动，焦点向下并且向右方向。当滑轮向上滚动，焦点向上并且向左)。

```
protected int nextFocus(int direction, boolean alt) {
    int index = this.getFieldWithFocusIndex();
    if(alt) {
        if(direction > 0) {
            //  action to perform if trackwheel is rolled up
            }
        else {
            //  action to perform if trackwheel is rolled down
          }
    }
    if (index == this.getFieldWithFocusIndex())
        return super.nextFocus(direction, alt);
    else
        return index;
    }
```

为了将焦点转移到下一个 field，而以管理器的顺序，这个 field 不是下一个 field，那么覆写 nextFocus().例如，如果你想为你的管理器实现 Page-up 和 Page-down 的功能，那么 nextFocus()就有用了。

当可见区域改变时重绘 field

缺省的，定制的管理器在不考虑剪辑区域下调用 paint()重绘所有 field。如果这导致了不必要的重绘，当可见区域改变时，仅 subpaint()的实现重绘所有 field。

代码实例

例: DiagonalManager.java

```
/**
* DiagonalManager.java
* Copyright (C) 2001-2005 Research In Motion Limited. All rights
reserved.
*/
package com.rim.samples.docs.custommenu;
import net.rim.device.api.system.*;
```

```
import net.rim.device.api.ui.container.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;

class DiagonalManager extends Manager {
    public DiagonalManager(long style) {
        super(style);
        }

    public int getPreferredWidth() {
        int width = 0;
        int numberOfFields = getFieldCount();
        for (int i=0; i<numberOfFields; ++i) {
            width += getField(i).getPreferredWidth();
            }
        return width;
        }

    public int getPreferredHeight() {
        int height = 0;
        int numberOfFields = getFieldCount();
        for (int i=0; i<numberOfFields; ++i) {
            height += getField(i).getPreferredHeight();
            }
        return height;
        }

    protected void sublayout(int width, int height) {
        int x = 0;
        int y = 0;
        Field field;
        int numberOfFields = getFieldCount();
        for (int i=0; i<numberOfFields; ++i) {
            field = getField(i);
            layoutChild( field, width, height );
            setPositionChild(field, x, y);
            x += field.getPreferredWidth();
            y += field.getPreferredHeight();
            }
        setExtent(width,height);
        }

    protected int nextFocus(int direction, boolean alt) {
        int index = this.getFieldWithFocusIndex();
```

```
        if(alt)
        {
            if(direction > 0)
            {
                //  Action to perform if trackwheel is rolled up.
            }
            else
            {
                //  Action to perform if trackwheel is rolled down.
            }
        }

        if (index == this.getFieldWithFocusIndex())
            return super.nextFocus(direction, alt);
        else
            return index;
        }
    }
```

创建列表

一个 ListField 包含了多列可选项。为了使用户可以选择列表中多项，声明列表为 MULTI_SELECT.

创建一个回调对象

ListFieldCallback 对象为列表控制所有重绘任务。每次要求 Field 显示列表中的一个条目。必要的方法也会在回调对象中调用。

ListFieldCallback 接口的实现创建了一个回调对象。系统调用这个接口的方法绘制列表的行，获得一个指定的列表元素，或决定列表的宽度。

```
private class ListCallback implements ListFieldCallback {
    // The listElements vector contain the entries in the list.
    private Vector listElements = new Vector();
    ...
}
```

允许 Field 重绘一行

drawListRow()的实现允许 Field 重绘一行。传递到 drawListRow()的图形上下文代表整个列表。相应地，drawText()必须指明绘制哪一行。

```
public void drawListRow(ListField list, Graphics g, int index, int y,
int w) {
    String text = (String)listElements.elementAt(index);
```

```
    g.drawText(text, 0, y, 0, w);
}
```

允许 Field 从列表中得到一个条目（Entry）

get()的实现允许 field 从列表中得到一个条目。本方法返回一个包含在有明确索引行中的对象。

```
public Object get(ListField list, int index)
{
    return listElements.elementAt(index);
}
```

为列表返回一个优先的宽度

getPreferredWidth()的实现为列表返回一个优先的宽度。在下面的实现中，getPreferredWidth()返回整个屏幕的绘制宽度。

getPreferredWidth()的实现返回一个不同的值，这依赖 field 管理器的类型。例如，如果管理器扩展了 HorizontalFieldManager,getPreferredWidth()返回所有 field 宽度的总和。如果扩展了 VerticalFieldManager，getPreferredWidth()返回最宽 field 的宽度。

```
public int getPreferredWidth(ListField list) {
    return Graphics.getScreenWidth();
}
```

指派回调以及加入条目到列表中

创建列表对象，并且将回调指派这个对象。

创建列表对象

为了列表创建 ListField 对象以及 ListCallback 对象。

注：ListCallback 是一个定制的 ListFieldCallback 类，这个类在 66 页的“创建一个回调对象”中创建。

```
ListField myList = new ListField();
ListCallback myCallback = new ListCallback();
```

设置回调

调用 setCallback()将 ListFieldCallback 与 ListField 关联。这个关联允许回调增加列表项到列表中。

```
myList.setCallback(myCallback);
```

增加列表条目

为了将条目增加到列表中，创建条目，并指定一个索引，并在这个索引上插入每个条目到 ListField 对象中。然后每个 ListField 对象到 ListFieldCallback 中。

```
String fieldOne = new String("Field one label");
String fieldTwo = new String("Field two label");
```

```
String fieldThree = new String("Field three label");
myList.insert(0);
myList.insert(1);
myList.insert(2);
myCallback.insert(fieldOne, 0);
myCallback.insert(fieldTwo, 1);
myCallback.insert(fieldThree, 2);
mainScreen.add(myList);
```

代码实例

例：SampleListFieldCallback.java

```
/**
* SampleListFieldCallback.java
* Copyright (C) 2001-2005 Research In Motion Limited. All rights
reserved.
*/
package com.rim.samples.docs.listfields;
import java.util.*;
import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;

public class SampleListFieldCallback
   extends UiApplication {

    private ListField myList;

    public static void main(String[] args) {
        SampleListFieldCallback app = new SampleListFieldCallback();
        app.enterEventDispatcher();
        }

    private static class ListCallback implements ListFieldCallback {

        private Vector listElements = new Vector();

        public void drawListRow(ListField list, Graphics g, int index,
int y, int w)
        {
            String text = (String)listElements.elementAt(index);
            g.drawText(text, 0, y, 0, w);
```

```
        }


        public Object get(ListField list, int index) {
            return listElements.elementAt(index);
            }

        public int indexOfList(ListField list, String p, int s) {
            return listElements.indexOf(p, s);
            }

        public int getPreferredWidth(ListField list) {
            return Graphics.getScreenWidth();
            }

        public void insert(String toInsert, int index) {
            listElements.addElement(toInsert);
            }

        public void erase() {
            listElements.removeAllElements();
            }
        }

    public SampleListFieldCallback() {
        MainScreen mainScreen = new MainScreen();
        myList = new ListField();
        ListCallback myCallback = new ListCallback();
        myList.setCallback(myCallback);
        String fieldOne = "ListField one";
        String fieldTwo = "ListField two";
        String fieldThree = "ListField three";
        myList.insert(0);
        myCallback.insert(fieldOne, 0);
        myList.insert(1);
        myCallback.insert(fieldTwo, 1);
        myList.insert(2);
        myCallback.insert(fieldThree, 2);
        mainScreen.add(myList);
        pushScreen(mainScreen);
        }
    }
}
```

操作图片

使用未处理（raw）的图像数据

为了从图像的特定区域获取未处理的图像数据，并存储在一个整数数组中，调用 Bitmap.getARGB().应用程序然后可以直接对未处理的图像数据进行操作。

```
void getARGB(int[] argbData, int offset, int scanLength,
                             int x, int y, int width, int height);
```


注：getARGB()方法只在彩屏的 BlackBerry 设备适用。

参数	描述
argbData	整数数组，用来存储 ARGB 数据；每个象素都以 0xAARRGGBB 的格式存储。
offset	数据的偏移，从这里开始读取。
scanLength	数据数组内每一个扫描行的宽。
x	矩形的左边，从这里开始读取图像数据。
y	矩形的上边，从这里开始读取图像数据。
width	矩形的宽，从这里开始读取图像数据。
height	矩形的高，从这里开始读取图像数据。

设备模拟器显示图像数据时每一个象素作为一个整数，每个象素中，每个字符（不透明）有 8 个位，红，绿以及蓝的值。颜色由 8 个整数以 0xAARRGGBB 的形式组成。

获取图像数据

初始化一个整型数组，然后调用 Bitmap.getARGB()将新的或预定义的位图的未处理的图像数据存储到整型数组中。

```
Bitmap original = Bitmap.getPredefinedBitmap(Bitmap.INFORMATION);
int[] argb = new int[original.getWidth() * original.getHeight()];
original.getARGB(argb, 0, original.getWidth(), 0, 0,
                         original.getWidth(), original.getHeight());
```

比较 2 个图像

调用 Bitmap.equals()决定 2 个位图是否相同。

```
if(restored.equals(original))
{
    System.out.println("Success! Bitmap renders correctly with RGB
data.");
}
else if(!restored.equals(original))
{
    System.out.println("Bitmap rendered incorrectly with RGB data.");
}
```

使用编码的图像

net.rim.device.api.system.EncodedImage 类封装了各种格式的编码图像。BlackBerry 设备支持下面的图像格式：.gif,.png,.wbmp,以及.jpeg.只有彩屏的 BlackBerry 设备才支持.jpeg 图像。

注：JPEGEncodedImage 类需要一个不可用的签名。

使用 EncodedImage 的子类，PNGEncodedImage 和 WBMPEncodedImage，来分别访问.png 和.wbmp 图像的特定属性。例如，PNGEncodedImage 提供方法来获得图像的色彩深度（Bit Depth），alpha 通道（alpha channel[①]），以及颜色类型。

在 BlackBerry IDE 中，一个应用程序能够直接访问加到工程或者依赖的类库工程中的图像。

访问一个图像

在 BlackBerry IDE 中，保存一个图像到你的项目文件夹或者子文件夹，然后增加图像到工程中。调用 Class.getResourceAsStream()获取图像作为一个字节的输入流。

```
private InputStream input;
...
try
{
    input = Class.forName("com.rim.samples.docs.imagedemo.ImageDemo").
    getResourceAsStream("/images/example.png");
}
catch (ClassNotFoundException e)
{
     System.out.println("Class not found");
}
```

解码一个图像

为了编码一个图像，调用 EncodedImage.createEncodedImage()。这个方法使用字节数组里的未处理的图像数据来创建了一个 EncodedImage 的实例。如果作为参数的字节数组布包汉一个可以识别的图像格式，它将抛出一个 IllegalArgumentException 异常。

```
private byte[] data = new byte[2430];// Store the contents of the
image file.
try {
    input.read(data); // Read the image data into the byte array.
}
catch (IOException e)
{
    // Handle exception.
}
try {
```

[①]用于指定像素的透明度，译者注

```
    EncodedImage image = EncodedImage.createEncodedImage(data, 0,
data.length);
}
catch (IllegalArgumentException iae)
{
    System.out.println("Image format not recognized.");
}
```

注：缺省地，BlackBerry 设备软件监测基于图像格式的 MIME 类型的图像。如果正确的 MIME 类型未能自动的监测到，使用下面 EncodedImage.createEncodedImage()的形式指定一个特定的 MIME 类型：

```
createEncodedImage(byte[] data, createEncodedImage(byte[] data, int
offset, int length, String mimeType)
```

如果图像格式预指定的 MIME 类型不匹配，这个方法抛出一个 IllegalArgumentException 异常。支持的 MIME 类型包括：image/gif, image/png, image/vnd.wap.wbmp, 以及 image/jpeg.

显示一个编码的图像

调用 BitmapField.setImage()指定一个编码的图像到一个 BitmapField，然后调用 add()将 BitmaoField 加入到屏幕中。

```
BitmapField field = new BitmapField();
field.setImage(image);
add(field);
```

设置解码模式

调用 EncodedImage.setDecodeMode()来设置图像的解码模式。提供下面模式之一作为方法的一个参数：

解码模式	描述
DECODE_ALPHA	解码一个 alpha 通道，如果一个存在（这是缺省的模式）。
DECODE_NATIVE	强制将位图解码未手持设备软件的原生位图类型。
DECODE_READONLY	将解码的位图标记为只读。

设置缩放因子（scaling factor）

当解码时，为了设置用在缩减一个图像的整数因子，调用 EncodedImage.setScale()。图像通过作为 scale 参数的整型来缩放。例如，如果你设置缩放因子为 2，图像将缩小到原大小的 50%。

代码实例

ImageDemo.java 实例从一个包含在项目中的图像获得未处理的数据，然后使用这个未处理的数据来重新创建一个 EncodedImage。

例：ImageDemo.java

```
/**
* ImageDemo.java
*/
package com.rim.samples.docs.imagedemo;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import java.io.*;

/*
 * The ImageDemo.java sample retrieves raw data from an image that
 * is included in its project, and then uses that raw data to
 * recreate an EncodedImage.
 */
public class ImageDemo extends UiApplication {
    public static void main(String[] args) {
        ImageDemo app = new ImageDemo();
        app.enterEventDispatcher();
        }

    public ImageDemo() {
        pushScreen(new ImageDemoScreen());
        }
}

final class ImageDemoScreen extends MainScreen {
    private static final int IMAGE_SIZE = 2430;
    private InputStream input;
    private byte[] data = new byte[IMAGE_SIZE];
    public ImageDemoScreen() {
        super();
        setTitle(new LabelField("Image Demo Sample"));
        try {
            input =
Class.forName("com.rim.samples.docs.imagedemo.ImageDemo")
            .getResourceAsStream("/images/hellokitty.png");
            }
        catch (ClassNotFoundException e) {
            System.out.println("Class not found");
        }

        if(input == null) {
            System.out.println("Error: input stream is not
```

```
initialized.");
            }
        else if (input != null) {
            System.out.println("OK: input stream is initialized.");

        try {
            int code = input.read(data);
            System.out.println("Total number of bytes read into buffer:
 " + code + " .");
            }
        catch (IOException e) {
            // Handle exception.
        }

        try {
            EncodedImage image = EncodedImage.createEncodedImage(data,
0, data.length);
            add(new BitmapField(image.getBitmap()));
            }
        catch (IllegalArgumentException iae) {
            System.out.println("Image format not recognized.");
            }
        }
    }
}
```

使用图像对象画图

Graphics 对象允许应用程序完成绘图功能和描绘（rendering）操作。使用 Graphics 类绘制整个屏幕或一个 BitmapField。如果你的应用程序不包含任何 field，调用 Screen.getGraphics（）来获得整个屏幕的绘图上下文。

为了绘制一个指定的 BitmapField，应用程序通过传入一个 field 到 Graphics 的 构造子中，来为一个指定的 field 获得一个绘图上下文.当绘制一个 BitmapField 时，field 管理器在 field 重绘时传递一个绘图上下文给 field。为了完成绘制一个定制的 field，当你扩展 Field 类时覆写 Graphics.paint()方法。

Graphics 类允许你绘制图形，例如弧线，直线，矩形以及圆。

使用图形上下文

为了利用 Graphics 类绘制，为每各自的 field 或整个屏幕获得一个图形上下文。
为了为各自的 Field 获取一个图形上下文，调用 Graphics 的构造子。

```
Bitmap surface = new Bitmap(100, 100);
BitmapField surfaceField = new BitmapField (surface);
Graphics graphics = new Graphics(surface);
```

为了为整个屏幕获得一个图形上下文，调用 Screen.getGraphics（）。

```
Graphics graphics = Screen.getGraphics();
```

为使用任何图形上下文绘图，你的方法务必要在 field 或屏幕的界限内完成它们的绘制功能。

```
graphics.fillRect(10, 10, 30, 30);
graphics.drawRect(15, 15, 30, 30);
```

如果你的图形上下文没有应用到整个屏幕，加入 BitmapField 到屏幕中。

```
mainScreen.add(surfaceField);
```

创建一个与标准的 BlackBerry UI 一致的界面

DrawStyle 接口提供了 Graphics 和 Field 对象使用的接口。DrawStyle 的实现允许你创建一个与标准 BlackBerry UI 一致的接口。如果你正扩展 Field 类来创建一个定制的 Field，你的代码需要接受合适的样式，这样它与标准的 BlackBerry 应用程序类似。

DrawStyle 作为 style 参数应用在 field 上，如下面的例子：

```
ButtonField buttonField = new ButtonField(DrawStyle.ELLIPSIS);
```

你可以在下面的对象中使用 DrawStyle 元素:

- BitmapField
- ButtonField
- DateField
- Graphics
- LabelField
- ObjectListField

用颜色绘制

利用彩色绘制只在彩屏的 BlackBerry 设备上适用。为了判断 BlackBerry 设备是否支持彩色显示，调用 Graphics.isColor(). 为了决定 BlackBerry 设备支持的颜色象素，调用 Graphics.numColors（）。

设置 alpha 值

全局 alpha 值决定他和绘制区域中象素的透明度，0（0x0000）是完全透明（不可见），255（0x00FF）是完全不透明。为了设置或得到全局 alpha 值，调用 Graphics.setGlobalAlpha（）或 Graphics.getGlobalAlpha（）.


注：BlackBerry 为特定的光栅操作使用 alpha 值。文本和绘制操作不会用到。

决定光栅操作的支持

为决定一个 Graphics 对象是否支持一个特定的光栅操作，调用 Graphics.isRopSupported(int), 使用下面提供的常数之一作为参数。

常数	光栅操作
ROP_CONST_GLOBALALPHA	将一个使用一个全局 alpha 常量值的前台常量颜色和目标象素混合。
ROP_SRC_GLOBALALPHA	将一个使用一个全局 alpha 常量值的源位图和目标象素混合。

绘制一个路径(Path)

为了绘制一组阴影填充的路径，调用 Graphics.drawShadedFilledPath():

```
public void drawShadedFilledPath(
        int[] xPts,
        int[] yPts,
        byte[] pointTypes,
        int[] colors,
         int[] offsets)
```

参数	描述
xPts	有序的列表定义了每个在路径里的顶点的 x 值。
yPts	有序的列表定义了每个在路径里的顶点的 y 值。
pointTypes	为每个定义的(x,y)点指定一个下面的常数。如果 pointTypes 为 null，所有点缺省为 Graphics.CURVEDPATH_END_POINT. ● Graphics.CURVEDPATH_END_POINT ● Graphics.CURVEDPATH_QUADRATIC_BEZIER_CONTROL_POINT ● Graphics.CURVEDPATH_CUBIC_BEZIER_CONTROL_POINT
colors	有序的列表为每个顶点以 0x00RRGGBB 格式定义颜色值。如果是 null，将绘制一个以当前前景颜色的路径。
offsets	列表在 xPts 和 yPts 数组里，定义了每个路径的起点。null 描述了单个路径，这个路径的起点在(xPts[offsets[i]],yPts[offsets[i]])，终点在(xPts[offsets[i+1]]-1,yPts[offsets[i+1]]-1)。

下面的例子绘制了一个从蓝色到红色混合的路径。

```
Bitmap surface = new Bitmap(240, 160);
BitmapField surfaceField = new BitmapField(surface);
add(surfaceField);
Graphics graphics = new Graphics(surface);
int[] X_PTS = { 0, 0, 240, 240 };
```

```
int[] Y_PTS = { 20, 50, 50, 20 };
int[] drawColors = { 0x0000CC, 0x0000CC, 0xCC0000, 0xCC0000 };
try
{
    graphics.drawShadedFilledPath(X_PTS, Y_PTS, null, drawColors,
null);
}
catch (IllegalArgumentException iae)
{
    System.out.println("Bad arguments.");
}
```

使用绘制格式

为了将绘制格式打开或关闭，调用 Graphics.setDrawingStyle(int drawStyle, Boolean on),在这里，on 指定是否打开（true）或关闭(false)绘制格式.为了判断一个绘制格式是否已经设置，调用 Graphics.isDrawingStyleSet(int drawStyle).

常数	描述
DRAWSTYLE_AALINES	直线的访混淆图像的格式，通过 setDrawingStyle() 和 isDrawingStyleSet() 使用。
DRAWSTYLE_AAPOLYGONS	多边形的访混淆图像的格式，通过 setDrawingStyle() 和 isDrawingStyleSet() 使用。
DRAWSTYLE_FOCUS	当绘制已完成，系统为焦点绘制设置的格式。
DRAWSTYLE_SELECT	当绘制已完成，系统为选择绘制设置的格式。

像印花一样使用单色位图 field

通过用颜色提交不透明的区域，STAMP_MONOCHROME 选项允许应用程序使用单色位图，如同印花一样。这个选项用于位图，这个位图是 1 位的，并且有定义的 alpha。

```
BitmapField field = new BitmapField(original,
BitmapField.STAMP_MONOCHROME);
```

从未处理的数据绘制一图像

1. 创建一空的位图。在本例中，类型和大小都复制到一个已经存在的位图。
2. 使用新建的位图创建一个 Graphics 对象作为绘图表面。
3. 调用 Graphics.rawRGB()，并使用从源处来的未处理的数据绘制一个新的图像。

```
Bitmap restored = new Bitmap(original.getType(), original.getWidth(),
original.getHeight());
Graphics graphics = new Graphics(restored);
try {
    graphics.drawRGB(argb, 0, restored.getWidth(), 0,
            0, restored.getWidth(),
            restored.getHeight());
}
```

```
catch(Exception e)
{
    System.out.println("Error occurred during drawing: " + e);
}
}
```

使用位图类型

注：下面关于位图类型的详情只提供类型信息。应用程序应不要依赖位图的实际位格式作为格式，因在手持设备的软件的未来版本可能会变化。

为了决定 Bitmap 类型，调用 Bitmap.getType()。这个方法返回下面常数中的一个：

位图类型	描述
COLUMNWISE_MONOCHROME	数据存储在列中，每个象素一个位：0 是白色，1 是黑色。在一个字节里，最上方的象素在低的有效字节里。低有限的字节包含了一列里的最上方的象素。
ROWWISE_MONOCHROME	数据存储在行里，一个象素一个位：0 代表黑色，1 代表白色。在宽度上，每行是 4 个字节的倍数。在一个字节里，最左边的象素在低的有效字节里。低有限的字节包含了一行里的最左边的象素。
ROWWISE_16BIT_COLOR	据存储在行里，一个象素有 2 个字节：0 是黑色，0xffff(65535)是白色。在宽度上，每行是 4 字节的倍数。

- 在黑白屏幕的 BlackBerry 设备上，数据存储在列中，因此，Bitmap.getType()返回 COLUMNWISE_MONOCHROME。头 2 个字节代表了位图第一个列里的头 16 个象素。
- 在彩屏的 BlackBerry 设备上，数据保存在行里，因此 itmap.getType()为黑白图片返回 ROWWISE_MONOCHROME，为彩色图片返回 ROWWISE_16BIT_COLOR。在黑白图片里，头 2 个字节代表了位图的第一行的头 16 个象素，从左到右。在彩色图片里，头2 个字节代表了第一个象素。

下面 2 个 Bitmap 的构造子允许你指定一个 type 参数：

- Bitmap(int type, int width, int height)
- Bitmap(int type, int width, int height, byte[] data)

为了获取 BlackBerry 设备的缺省位图类型，调用静态方法：
Bitmap.getDefaultType()

代码实例

DrawDemo.java 从预定义的位图里获取未处理的数据，然后使用这些数据绘制一个新的位图。最后显示原始的和恢复的图像。

例：DrawDemo.java

```
/*
* DrawDemo.java
* Copyright (C) 2002-2005 Research In Motion Limited.
*/

package com.rim.samples.docs.drawing;
import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;

/* The DrawDemo.java sample retrieves raw data from a predefined
bitmap
image, and then draws a new bitmap using the data. It then displays
the original and restored images. */
public class DrawDemo extends UiApplication {
    public static void main(String[] args) {
    }
    DrawDemo app = new DrawDemo();
    app.enterEventDispatcher();
}

public DrawDemo()
{
    pushScreen(new DrawDemoScreen());
}

final class DrawDemoScreen extends MainScreen {

    public DrawDemoScreen()
    {
        super();
        LabelField title = new LabelField("UI Demo",
LabelField.USE_ALL_WIDTH);
        setTitle(title);
        Bitmap original =
Bitmap.getPredefinedBitmap(Bitmap.INFORMATION);
        Bitmap restored = new Bitmap(original.getType(),
original.getWidth(),
                original.getHeight());
        Graphics graphics = new Graphics(restored);
```

```
        // Retrieve raw data from original image.
        int[] argb = new int[original.getWidth() *
original.getHeight()];
        original.getARGB(argb, 0, original.getWidth(), 0, 0,
                original.getWidth(),original.getHeight());

        // Draw new image using raw data retrieved from original image.
        try
        {
            graphics.drawRGB(argb, 0, restored.getWidth(), 0, 0,
                    restored.getWidth(),restored.getHeight());
        }
        catch(Exception e) {
            System.out.println("Error occurred during drawing: " + e);
            }

        if(restored.equals(original))
        {
            System.out.println("Success! Bitmap renders correctly with
RGB data.");
            }
        else if(!restored.equals(original))
        {
            System.out.println("Bitmap rendered incorrectly with RGB
data.");
        }

        BitmapField field1 = new BitmapField(original,
BitmapField.STAMP_MONOCHROME);
        BitmapField field2 = new BitmapField(restored);
        add(new LabelField("Original bitmap: "));
        add(field1);
        add(new LabelField("Restored bitmap: "));
        add(field2);
        }
    }
```

监听 UI 对象的改变

UI EventListeners 允许应用程序响应一个 UI 对象的改变。这里有 3 种类型的 UI 事件监听者：

监听者	描述
FieldChangeListener	当 field 的属性改变时发出事件通知

FocusChangeListener	当 Field 获取或失去焦点时发出事件通知。
ScrollChangeListener	当一个管理器的水平或者垂直滚动值变化时发出事件通知。

监听 field 属性的变化

为了监测 field 的变化，实现 FieldChangeListener 接口。调用 setChangeListener()来把你的实现指派给一个 field。

```
private class FieldListener implements FieldChangeListener {
    public void fieldChanged(Field field, int context) {
        if (context != FieldChangeListener.PROGRAMMATIC) {
            // Perform action if user changed field.
            }
        else
        {
            // Perform action if application changed field.
        }
    }
}

// ...
FieldListener myFieldChangeListener = new FieldListener()
myField.setChangeListener(myFieldChangeListener);
```

监听焦点的改变

为了监测 field 之间焦点的改变，指派给他们一个 FocusChangeListener。实现这个 FocusChangeListener，然后通过调用 setChangeListener()把你的实现指派给一个 Field。一个 FocusChangeListener 关心一个与之相关的明确的 Field 的焦点的获取，失去或改变。

当 field 通过实现 focusChanged()获取，失去或改变焦点时， FocusChangeListener 的实现应该指明 field 将采取什么样的动作。

```
private class FocusListener implements FocusChangeListener {
    public void focusChanged(Field field, int eventType) {
        if (eventType == FOCUS_GAINED) {
            // Perform action when this field gains the focus.
            }
        if (eventType == FOCUS_CHANGED) {
            // Perform action when the focus changes for this field.
           }
        if (eventType == FOCUS_LOST) {
            // Perform action when this field loses focus.
```

```
            }
        }
    }

FocusListener myFocusChangeListener = new FocusListener();
myField.setChangeListener(myFocusChangeListener);
```

监听滚动事件

ScrollChangeListener 接口的实现允许你的 field 管理器管理滚动事件，调用 setScrollListener() 将你的实现给一个 Manager。当水平或垂直的（或都有）滚动值发生变化时，scrollChanged() 方法传递一个新的值。

注：典型地，监听滚动变化没有必要，因为你的应用程序可以监听 field 的焦点变化；尽管这样，ScrollChangeListener 在游戏实现中可能有用。

为将监听者指派给一个 field，调用 field 管理器上的 setScrollListener().

```
private class ScrollListener implements ScrollChangeListener {
    scrollChanged(Manager manager, int newHoriztonalScroll, int
newVerticalScroll){
        // Perform action.
        }
}

ScrollListener myScrollChangeListener = new ScrollListener();
myManager.setScrollListener(myScrollChangeListener);
```

第 4 章 使用音频

播放一个支持的音频格式的曲调

在支持标准音频格式的 BlackBerry 设备上，你可以播放下列支持的格式之一的音频文件：

- audio/MPEG-1 Layer 3
- audio/midi
- audio/x-midi
- audio/mid

BlackBerry 设备使用 Mobile Media API（javax.microedition.media）包来支持标准的音频文件格式。

为了在运行时确定支持的音频格式，调用 Manager.getSupportedContentTypes().为得到信息，参看 API 参考里的 javax.microedition.media 包。

语音记事 API

在 net.rim.device.api.system 包里，语音记事 API 由下面的三个方法组成：

- `Audio.playFile(int audioCodec, int fs, String fileName)`
- `Audio.recordFile(int audioCodec, int fs, String fileName)`
- `Audio.stopFile(int audioCodec, int fs, String fileName)`

每个方法都接受一个编码，一个文件系统以及一个文件名。语音记事编码由 Audio.AUDIO_CODEC_VOICENOTE 表现。iDEN™ 文件系统由 net.rim.device.api.io.FILESYSTEM_PATRIOT 来表现。文件系统是普通的文件系统，因此，文件名参数由一个没有路径名的文件名组成。

当录音，播放或停止操作失败或完成时，应用程序应该注册一个音频监听者来接收这些消息。为了注册一个监听者，实现 net.rim.device.api.system.AudioFileListener. 通过调用 Audio.addListener(Application, AudioListener)来注册。

注：文件系统的大小，目前大约是 250KB，制约者录音的长短，大约是 8 到 9 秒的语音记事录音。如果录音超过文件系统大小，录音停止并保存文件。

为获得更多信息，参看 API 参考里的 net.rim.device.api.system audio 类。

5

第 5 章 支持的媒体内容（Media Content）

PME 内容

BlackBerry 设备支持 PME 格式的富（rich）媒体内容。
开发者可以使用 Plazmic Content Developer's Kit for BlackBerry 来创建 PME 内容。这个工具，以及附带的文档可以在 Plazmic 网站（www.plazmic.com）找到。
Media Engine API（在 net.rim.plazmic.mediaengine 和 net.rim.plazmic.mediaengine.io 包中）允许应用程序获取和播放存储在 BlackBerry 设备上或网络上的 PME 内容。

注：Media Engine API 支持媒体格式 application/x-vnd.rim.pme. Web 服务器必须为 application/x-vnd.rim.pme 设置 MIME 类型。

PME API 概览

下面 3 个主要类（在 net.rim.plazmic.mediaengine 包里）提供了加载和播放 PME 媒体内容的能力。

类	描述
MediaManager	提供从本地或网络上加载媒体内容的方法。
MediaPlayer	提供播放 PME 媒体的方法。
MediaException	为获取或播放媒体的错误提供异常代码。

媒体加载

Media Engine API 允许应用程序使用下面 4 种协议种的一种加载媒体内容：

协议	描述
http://	http 协议从一个使用 HTTP 连接网络 Web 服务器下载内容。这个协议需要一个带有 BlackBerry MDS 服务的 BES（BlackBerry Enterprise Server，BlackBerry 企业服务器）.

https://	https 协议从一个使用 HTTPS 连接网络 Web 服务器下载内容。这个协议需要一个带有 BlackBerry MDS 服务的 BES（BlackBerry Enterprise Server，BlackBerry 企业服务器）.
Jar:///<pme_file>	jar 协议加载存储在本地 BlackBerry 设备上的 jar 文件。 jar:///sample.pme **注意**：开始的斜线（/）是需要的。 在 BlackBerry IDE 中，.jar 文件必须加入到调用应用程序或应用程序依赖的库的相同项目中。
cod:///<module><pme_file>	cod 协议加载存储在本地 BlackBerry 设备上的 cod 文件。 cod://mediasample/sample.pme

为使用其他协议，实现定制的 Connector。为获得更多信息，参看 91 页的“创建定制的 Connector”.

播放状态（Playback states）

为了获取 MediaPlayer 的当前状态，调用 MediaPlayer.getState().

状态	描述
UNREALIZED	MediaPlayer 未准备播放媒体。为了转到 REALIZED 状态，调用 MediaPlayer.setMedia（）.
REALIZED	MediaPlayer 准备好播放媒体。为了开始播放，并转到 STARTED 状态，调用 MediaPlayer.start().
STARTED	MediaPlayer 正在播放媒体。为了停止播放和返回到 REALIZED 状态，调用 MediaPlayer.stop().

异常

MediaEngine 和 MediaManager 类的方法抛出一个 MediaException 异常，这个异常包含了一个标准的 HTTP 响应代码或者下面异常代码之一。为了获取与异常相联系的错误代码，调用 MediaException.getCode().

异常代码	描述
INVALID_HEADER	媒体格式无效。
REQUEST_TIMED_OUT	请求超时。
INTERRUPTED_DOWNLOAD	应用程序调用 MediaManager.cancel()来取消下载。
UNSUPPORTED_TYPE	媒体类型（MIME 类型）不支持。
UPGRADE_PALYER	媒体引擎的版本和请求的内容不兼容。
UPGRADE_MEDIA	媒体引擎的版本不在支持请求的内容。
CHECKSUM_MISMACTH	求和校验失败，因此媒体内容不能读取。

OUT_OF_BOUNDS	数组出界，或应用程序试图访问一个文件结尾后的输入流。

事件

MediaListener 接口允许应用程序接受或响应下面的事件：

事件	描述
MEDIA_REQUEST	媒体已请求加载，当 animation 自动请求新内容或当用户点击媒体内容的超连接时，事件发生。
MEDIA_REALIZED	媒体已经创建播放了。当 MediaManager.createMediaManager（）已经调用时发生。
MEDIA_COMPLETE	媒体已经加载，并成功播放。
MEDIA_TO	媒体正在加载。

为获得更多信息，参考 85 页的“监听 Media Engine 事件”.

播放媒体内容

为了获取 BlackBerry 设备或网络上的 PME 内容，使用 MediaManager 的方法。为了播放已经下载到 BlackBerry 设备的 PME 内容，使用 MediaPlayer 类的方法。

下载内容

为下载 PME 内容，创建一个 MediaManager 对象，然后调用 MediaManager.createMedia().

```
try
 {
    Object media = manager.createMedia("http://webserver/sample.pme");
 }
 catch (IOException ioe)
 {
     System.out.println("Error: requested content was not
downloaded.");
 }
 catch (MediaException me)
 {
     System.out.println("Error: " + me.getCode());
 }
```

注：下面缺省的协议会被支持:http://,https://.jar://,和 cod://.为获得更多信息，参看 81 页的“媒体加载”。

第一次调用 MediaManager.createMedia()，URL 必须是绝对路径，除非首先调用

MediaManager.setProperty(“URI_BASE”,<base_url>)设置基 URL 路径。当你之后调用 createMedia（）时，前面的 URL 作为基 URL。

播放 PME 内容

为播放设置 PME 对象

调用 MedialPlayer.setMedia().

```
MediaPlayer player = new MediaPlayer();
try
{
 player.setMedia(media);
}
catch (MediaException me)
{
  System.out.println("Error: requested content type is not
supported.”);
}
```

获取一个显示 PME 内容的 UI 对象

调用 MediaPlayer.getUI()。转化 getUI()返回的一个作为 Field 的对象，然后将之加入到屏幕来显示。

```
add((Field)player.getUI());
```

开始播放下载的 PME 内容

调用 MediaPlayer.start()。

```
if(player.getState() == MediaPlayer.REALIZED)
{
 try
 {
  player.start();
 }
 catch(MediaException me) {
     System.out.println("Error occurred during media playback: " +
            me.getCode() + me.getMessage());
 }
}
```

注：在调用 MediaPlayer.start()前检查 MediaPlayer 的状态，如果媒体播放器不是 REALIZED 状态，start()方法抛出一个异常。

代码实例

MediaSample.java 实例从一个 Web 服务器获取一个 PME 文件，然后显示它。

例：MediaSample.java

```
/**
* MediaSample.java
* Copyright (C) 2001-2005 Research In Motion Limited. All rights
reserved.
*/
package com.rim.samples.docs.mediasample;
import java.io.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.plazmic.mediaengine.*;

public class MediaSample extends UiApplication {
    public static void main(String[] args) {
        MediaSample app = new MediaSample();
        app.enterEventDispatcher();
    }
    public MediaSample() {
        pushScreen(new MediaSampleScreen());
    }

final static class MediaSampleScreen extends MainScreen {
    public MediaSampleScreen() {
        super();
        LabelField title = new LabelField("Media Sample",
LabelField.ELLIPSIS| LabelField.USE_ALL_WIDTH);
        setTitle(title);
        MediaPlayer player = new MediaPlayer();
        MediaManager manager = new MediaManager();
        try {
            Object media =
manager.createMedia("http://webserver/SVGFILE.pme");
            player.setMedia(media);
        }
        catch (IOException ioe) {
        }
        catch (MediaException me) {
            System.out.println("Error during media loading: ");
```

```
            System.out.println(me.getCode());
            System.out.println(me.getMessage());
        }
        add((Field)player.getUI());
        try {
            player.start();
        }
        catch(MediaException me) {
            System.out.println("Error occured during media playback:
");
            System.out.println(me.getCode());
            System.out.println(me.getMessage());
            }
        }
    }
}
```

监听媒体引擎事件

MediaListener 接口允许应用程序注册接收媒体引擎事件。应用程序可以在注册 MediaPlayer 和 MediaEngine 对象上注册监听者。

当应用程序实现监听者时，它可以完成以下的动作：

- 提供内容下载状态的信息。
- 在后台下载内容，当完成时播放它。
- 下载一个 animation 自动请求的内容。

MediaListener 接口包含一个方法，listen 方法。

```
public void mediaEvent(Object sender,
                       int event,
                       int eventParam,
                       Object data);
```

参数	描述
sender	本参数引用了发送事件的对象，如 MediaPlayer 或 MediaManager 对象。
event	参数可以是下列事件之一： ● MEDIA_REQUESTED:当新的内容请求时发送事件。 ● MEDIA_COMPLETE:当所有计划好的媒体动作完成时触发事件。 ● MEDIA_REALIZED：由 MediaManager 发送，返回下载的媒体。 ● MEDIA_IO:由 MediaPlayer 发送，提

	供现在进度或状态的信息。
eventParam	不要使用这个参数，因为它可能接收一个任意值。它存在是为了为额外的事件提供一个一致的接口。
data	当 data 参数是 MEDIA_REQUESTED，data 把请求的 URL 作为一个 String 对象。 当 data 参数是 MEDIA_REALIZED,data 引用了创建的媒体对象。 当 data 参数是 MEDIA_IO,data 引用了一个 net.rim.plazmic.mediaengine.io.LoadingStatus 对象。

监听媒体引擎事件

MediaListener 接口的实现允许你的应用程序监听一个媒体引擎事件。mediaEvent（）的实现应该处理所有可能的媒体事件。下面的例子使用了一个 switch 语句来处理可能媒体事件。

```
public final class MediaListenerImpl implements MediaListener {
    public void mediaEvent(Object sender, int event, int eventParam,
Object data) {
        switch(event) {
          case MEDIA_REQUESTED:
              // Perform action.
              break;
          case MEDIA_COMPLETE:
              // Perform action.
              break;
          case MEDIA_REALIZED:
              // Perform action.
             break;
          case MEDIA_IO:
              // Perform action.
              break;
             }
        }
}
```

注册监听者

为了注册你的监听者，调用 MediaPlayer 和 MediaManager 对象上的 addMediaListener()方法。

```
private MediaListenerImpl _listener = new MediaListenerImpl();
private MediaPlayer player = new MediaPlayer();
private MediaManager manager = new MediaManager();
player.addMediaListener(_listener);
```

```
manager.addMediaListener(_listener);
```

在后台加载内容

当实现 MediaListener 时，你可以在背后下载 PME 内容，并且当下载完成后播放内容。调用 MediaManager.createMediaListener（）为将来的播放下载内容。

注：和 createMedia()不一样，createMediaLater()不返回一个媒体内容的对象。

在 MediaListener.mediaEvent()中，当请求的内容下载时，加入代码来处理 MEDIA_REALIZED 事件。为了注册在 data 参数里指定的内容，调用 MediaPlayer.setMedia(data)。为了开始播放，调用 MediaPlayer.start（）。

```
manager.createMediaLater("http://webserver/sample.pme");

public void mediaEvent(Object sender, int event,
                   int eventParam, Object data) {
    switch(event) {
    ...
    case MEDIA_REALIZED:
     try {
     player.setMedia(data);
     player.start();
     }
     catch(MediaException me) {
     System.out.println("Error playing media" + me.getCode() +" +
           "   me.getMessage());
     }
   break;
 }
}
```

跟踪下载进度

为得到下载进度的信息，使用 net.rim.plazmic.mediaengine.io.LoadingStatus 类。这个类包含了一些方法来允许你获得媒体内容类型，字节总数，字节读取数，以及内容的源 URL。

状态	描述
LOADING_STARTED	加载开始。
LOADING_READING	数据流正在解析。
LOADING_FINISHED	加载媒体成功。
LOADING_FAILED	媒体记载失败. ● 为获取详细的错误代码，调用 getCode(). 参看 82 页获得更多详情。

	● 为得到异常信息，调用 getMessage().

在 mediaEvent()的实现里，当 MEDIA_IO 事件发生时，将 data 参数里的 Object 转化为一个 LoadingStatus 对象。

调用 LoadingStatus.getStatus()来获取下载的状态，然后处理每个状态。

对每个正常的状态，打印一个消息到控制台。

对 LOADING_FAILED 状态，完成下面的动作:

- 调用 LoadingStatus.getCode()获得错误代码。
- 调用 LoadingStatus.getMessage()获得详细的消息。
- 调用 LoadingStatus.getSource()获得内容的 URL 字符串。

```
public void mediaEvent(Object sender, int event,
                    int eventParam, Object data) {
    switch(event) {
       ...
      case MEDIA_IO: {
      }
      ...
      break;
    }
    break;
    ...

switch(s.getStatus()) {
    case LoadingStatus.LOADING_STARTED:
       System.out.println("Loading in progress");
       break;
    case LoadingStatus.LOADING_READING:
       System.out.println("Parsing in progress");
       break;
    case LoadingStatus.LOADING_FINISHED:
       System.out.println("Loading completed");
       break;
    case LoadingStatus.LOADING_FAILED:
      String errorName = null;
      int code = s.getCode();
      switch (code) {
         case MediaException.INVALID_HEADER:
           errorName = "Invalid header" + "\n" + s.getSource();
           break;
         case MediaException.REQUEST_TIMED_OUT:
           errorName = "Request timed out" + "\n" +
           s.getSource();
           break;
         case MediaException.INTERRUPTED_DOWNLOAD:
           break;
```

```
        case MediaException.UNSUPPORTED_TYPE:
          errorName = "Unsupported type" + s.getMessage() + "\n" +
s.getSource();
          break;
        default: {
          if (code > 200) {
            // A code > 200 indicates an HTTP error
            errorName = "URL not found";
          }
          else {
           // default unidentified error
            errorName = "Loading Failed";
           }
        errorName += "\n" + s.getSource() + "\n" + s.getCode()+ ": "
+ s.getMessage();
        break;
      }
    }
    System.out.println(errorName);
    break;
  } // End switch s.getStatus().
 break;
}
```

代码实例

MediaSample2.java 实例实现了一个监听者在后台下载媒体内容，并显示下载的状态到控制台。

例：MediaSample2.java

```
/**
* MediaSample2.java
* Copyright (C) 2001-2005 Research In Motion Limited. All rights
reserved.
*/
package com.rim.samples.docs.mediasample;
import java.io.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;

import net.rim.plazmic.mediaengine.*;
import net.rim.plazmic.mediaengine.io.*;
```

```
public class MediaSample2 extends UiApplication {
    private MediaPlayer player = new MediaPlayer();
    private MediaManager manager = new MediaManager();
    private MediaListenerImpl _listener = new MediaListenerImpl();
    private MediaSample2Screen _screen;

    public static void main(String[] args) {
        MediaSample2 app = new MediaSample2();
        app.enterEventDispatcher();
        }

    public MediaSample2() {
        _screen = new MediaSample2Screen();
        pushScreen(_screen);
        }


    public final class MediaListenerImpl implements MediaListener {
        public void mediaEvent(Object sender, int event,
                int eventParam, Object data) {
            switch(event) {
            case MEDIA_REQUESTED:
                System.out.println("Media requested");
                break;
            case MEDIA_COMPLETE:
                System.out.println("Media completed");
                break;
            case MEDIA_REALIZED:
                try {
                    player.setMedia(data);
                    player.start();
                    }
                catch(MediaException me) {
                    System.out.println("Error during media loading: " +
                            me.getCode() + me.getMessage());
                    }
                break;
            case MEDIA_IO: {
                LoadingStatus s = (LoadingStatus)data;
                switch(s.getStatus()) {
                case LoadingStatus.LOADING_STARTED:
                    System.out.println("Loading in progress");
                    break;
```

```
                case LoadingStatus.LOADING_READING:
                    System.out.println("Parsing in progress");
                    break;
                case LoadingStatus.LOADING_FINISHED:
                    System.out.println("Loading completed");
                    break;
                case LoadingStatus.LOADING_FAILED:
                    String errorName = null;
                    int code = s.getCode();
                    switch (code) {
                    case MediaException.INVALID_HEADER:
                        errorName = "Invalid header" + "\n" +
s.getSource();
                        break;
                    case MediaException.REQUEST_TIMED_OUT:
                        errorName = "Request timed out" + "\n" +
s.getSource();
                        break;
                    case MediaException.INTERRUPTED_DOWNLOAD:
                        break;
                    case MediaException.UNSUPPORTED_TYPE:
                        errorName = "Unsupported type" + s.getMessage()
                        + "\n" + s.getSource();
                        break;
                    default: {
                        if (code > 200) {
                            // A code > 200 indicates an HTTP error.
                            errorName = "URL not found";
                            }
                        else {
                            // Default unidentified error.
                            errorName = "Loading Failed";
                            }
                    errorName += "\n" + s.getSource() + "\n"+
                    s.getCode() + ": " + s.getMessage();
                    break;
                }
            }
        System.out.println(errorName);
        break;
    } // End switch s.getStatus().
    break;
    }
   }
```

```
  }
}

final class MediaSample2Screen extends MainScreen {
   public MediaSample2Screen() {
     super();
     LabelField title = new LabelField("Media Sample",
LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
     setTitle(title);
     manager.addMediaListener(_listener);
    // Change this to the location of a test .pme file.
    manager.createMediaLater("http://test.rim.com/SVGBS0001.pme");
    add((Field)player.getUI());
}
}
}
```

创建一个定制的连接

MediaManager 使用一个 Connector 对象加载媒体，并打开输入流。缺省的 Connector 支持下列协议:http://.https://,jar://,以及 cod://。为了增加支持一个定制的协议或者为了覆写缺省的行为，通过实现 net.rim.plazmic.mediaengine.io.Connector 接口创建一个定制的 Connector

方法签名	实现
InputStream getInputStream(String, ConnectionInfo)	实现本方法返回一个输入流从指定 URI 读取内容。
void releaseConnection(ConnectionInfo)	实现本方法释放连接。MediaManager 调用本方法来通知 Connector 可以释放连接了。
void setProperty(String, String)	实现本方法设置连接指定的属性。

实现一个定制的 connector

为了完成处理一个定制的协议，实现 Connector 接口，包含 getInputStream（）。为了处理一个标准的协议，调用缺省的 Connector。

setProperty(String name, String value)的实现设置了指定的属性。在本例中，connector 不必设置任何指定的属性，因此 setProperty（）的实现调用了 Connector 上的 setProperty（）。

```
public class SampleConnector implements Connector {
    Connector delegate; // The default Connector.
    SampleConnector(Connector delegate) {
        this.delegate = delegate;
        }

    public InputStream getInputStream(String uri, ConnectionInfo info)
```

```
throws IOException, MediaException {
    InputStream input = null;
    if (uri.startsWith("myprotocol://")) {
        // Perform special tasks.
        info.setConnection(new MyProtocolConnection());
        info.setContentType("application/x-vnd.rim.pme");

        // openMyInputStream() is a custom method that opens
        //stream for "myprotocol://".
        input = openMyInputStream(uri);
        }
    else {
        input = delegate.getInputStream(uri, info);
        }
    return input;
}

public void releaseConnection(ConnectionInfo info)
throws IOException, MediaException {
    Object o = info.getConnection();
    if (o instanceof MyProtocolConnection) {
        ((MyProtocolConnection)o).close(); // Perform cleanup.
        }
    else
    {
        delegate.releaseConnection(info);
    }
}

public void setProperty(String property, String value) {
    delegate.setProperty(property, value);
    }
}
```

注册一个定制的连接器

在你的主要方法里，调用 MediaManager.setConnector()注册你的定制的连接器。

```
MediaManager manager = new MediaManager();
manager.setConnector(new
CustomPMEConnector(manager.getDefaultConnector()));
```

代码实例

CustomPMEConnector.java 实例为实现一个定制的连接器提供了一个框架。

例：CustomPMEConnector.java

```
/*
* CustomPMEConnector.java
* Copyright (C) 2003-2005 Research In Motion Limited. All rights
reserved.
*/
package com.rim.samples.docs.mediasample;
import java.io.*;
import net.rim.plazmic.mediaengine.*;
import net.rim.plazmic.mediaengine.io.*;

public class CustomPMEConnector implements Connector {
    private Connector delegate;
    private InputStream input;

    CustomPMEConnector(Connector delegate)
    {
        this.delegate = delegate;
    }


    public InputStream getInputStream(String uri, ConnectionInfo info)
            throws IOException, MediaException
    {
        if (uri.startsWith("myprotocol://"))
        {
            // Perform special tasks.
            info.setConnection(new MyProtocolConnection());
            info.setContentType("application/x-vnd.rim.pme");

            // OpenMyInputStream() is a custom method that opens
            //stream for "myprotocol://"
            input = openMyInputStream(uri);
        }
        else
        {
            input = delegate.getInputStream(uri, info);
            return input;
        }
```

```
        private InputStream openMyInputStream(String uri)
        {
            InputStream input = null;
        }


        // @todo: open stream here
        return input;
        }

    public void releaseConnection(ConnectionInfo info)
    throws IOException, MediaException
    {
        Object o = info.getConnection();
        if (o instanceof MyProtocolConnection)
        {
            ((MyProtocolConnection)o).close(); // Perform cleanup.
        }
        else
        {
            delegate.releaseConnection(info);
        }
    }


    public void setProperty(String property, String value) {
        delegate.setProperty(property, value);
        }


    // Inner class that defines the connection class.
    public static class MyProtocolConnection {
        public MyProtocolConnection()
        {
            // ...
        }

        public void close()
        {
            // ...
        }
    }
}
```

6

第 6 章 连接网络

HTTP 和 Socket 连接

尽管你可以通过 socket 连接实现 HTTP，但是最好使用 HTTP 连接，因为 socket 连接不支持 BlackBerry MDS 服务特性，例如 push。也最好使用 HTTP 连接，因为比起那些使用 HTTP 连接的应用程序，使用 socket 连接的应用程序明显需要更多的带宽。


注：如果你使用 socket 连接，将你的应用程序设计为适应断断续续的无线网络连接。例如，如果你的程序发生错误时，它会重新打开连接。

使用 HTTP 连接


注：使用 BlackBerry Internet Service Browser 的 java 程序不会启动 HTTP，HTTPS 和 TCP 连接。

打开一个 HTTP 连接

为了打开一个 HTTP 连接，调用 Connector.open(),指定 http 为协议。将返回的对象转化为一个 HTTPConnection 或者 StreamConnection 对象。HttpConnection 是一个 StreamConnection，它提供访问指定 HTTP 功能，包括 HTTP 头和其他 HTTP 资源。

```
HttpConnection conn = null;
String URL = "http://www.myServer.com/myContent";
conn = (HttpConnection)Connector.open(URL);
```

设置 HTTP 请求方式

为设置 HTTP 请求方式（GET 或 POST），调用 HttpConnection.setRequestMethod().

```
conn.setRequestMethod(HttpConnection.POST);
```

设置或获取头字段

为 HTTP 请求或 HTTP 响应消息设置或获取头字段，调用 HttpConnection 上的 getRequestProperty() 或 setRequestProperty()。

```
conn.setRequestProperty("User-Agent", "BlackBerry/3.2.1");
String lang = conn.getRequestProperty("Content-Language");
```

发送和接受数据

为发送和接受数据，调用 HTTPConnection 的 openInputStream()和 openOutputStream()获得输入和输出流。

```
InputStream in = conn.openInputStream();
OutputStream out = conn.openOutputStream();
```

代码实例

HttpFetch.java 实例使用了一个 HTTP 连接来获取数据。它遵循下列步骤：

1. 创建一个连接线程。
2. 定义一个方法获取数据。
3. 定义一个方法将数据显示给用户。
4. 定义一个方法退出应用程序。
5. 定义应用程序构造子。

注：HTTPFetch.java 实例需要你在应用程序工程里创建资源文件，并且定义需要的资源键值。参看 125 页“本地化应用程序”获得更多信息。

例：HTTPFetch.java

```
/**
* HTTPFetch.java
* Copyright (C) 2001-2005 Research In Motion Limited. All rights
reserved.
*/
package com.rim.samples.docs.httpfetch;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
```

```
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import javax.microedition.io.*;
import java.io.*;
import com.rim.samples.docs.baseapp.*;
import com.rim.samples.docs.resource.*;

public class HTTPFetch extends BaseApp implements HTTPFetchResource
{
    // Constants.
    private static final String SAMPLE_PAGE =
"http://localhost/testpage/sample.txt";
    private static final String[] HTTP_PROTOCOL =    {"http://",
"http:\\"};
    private MainScreen _mainScreen;
    private RichTextField _content;

    /**
     *  Send and receive data over the network on a
     *  separate thread from the main thread of your application.
   */
    ConnectionThread _connectionThread = new ConnectionThread();

    //statics
    private static ResourceBundle _resources =
ResourceBundle.getBundle(
        HTTPFetchResource.BUNDLE_ID, HTTPFetchResource.BUNDLE_NAME);
        public static void main(String[] args)
        {
            HTTPFetch theApp = new HTTPFetch();
            theApp.enterEventDispatcher();
        }

    /**
     * The ConnectionThread class manages the HTTP connection.
     * Fetch operations are not queued, but if a second fetch request
     * is made while a previous request is still active,
     * the second request stalls until the previous request completes.
     */

    private class ConnectionThread extends Thread
    {
        private static final int TIMEOUT = 500; //ms
        private String _theUrl;
```

```
        /* The volatile keyword indicates that because the data is
shared,
         * the value of each variable must always be read and written
from memory,
         * instead of cached by the VM. This technique is equivalent
to wrapping
         * the shared data in a synchronized block, but produces less
overhead.
         */
        private volatile boolean _start = false;
        private volatile boolean _stop = false;
        /**
         * Retrieve the URL. The synchronized keyword makes sure that
only one
         * thread at a time can call this method on a ConnectionThread
object.
         */
        public synchronized String getUrl()
        {
            return _theUrl;
        }
        /**
         * Fetch a page. This method is invoked on the connection
thread by
         * fetchPage(), which is invoked in the application
constructor when
         * the user selects the Fetch menu item.
         */
        public void fetch(String url)
        {
            _start = true;
            _theUrl = url;
        }
        /**
         * Close the thread. Invoked when the application exits.
         */
        public void stop()
        {
            _stop = true;
        }
        /**
         * Open an input stream and extract data. Invoked when the
thread
         * is started.
```

```
 */
public void run()
{
    for(;;)
    {
        // Thread control.
        while( !_start && !_stop)
        {
            // No connections are open for fetch requests,
            // but the thread has not been stopped.
            try
            {
                sleep(TIMEOUT);
            }
            catch (InterruptedException e)
            {
                System.err.println(e.toString());
            }
        }
        // Exit condition.
        if ( _stop )
        {
            return;
        }
        /* Ensure that fetch requests are not missed
         * while received data is processed.
         */
        synchronized(this)
        {
            // Open the connection and extract the data.
            StreamConnection s = null;
            try
            {
                s = (StreamConnection)Connector.open(getUrl());
                InputStream input = s.openInputStream();
                // Extract data in 256 byte chunks.
                byte[] data = new byte[256];
                int len = 0;
                StringBuffer raw = new StringBuffer();
                while ( -1 != (len = input.read(data)) )
                {
                    raw.append(new String(data, 0, len));
                }
                String text = raw.toString();
```

```
                    updateContent(text);
                    input.close();
                    s.close();
                }
                catch (IOException e)
                {
                    System.err.println(e.toString());
                    // Display the text on the screen.
                    updateContent(e.toString());
                }
                // Reset the start state.
                _start = false;
            }
        }
    }
    // Constructor.
    public HTTPFetch()
    {
        _mainScreen = new MainScreen();
        _mainScreen.setTitle(new LabelField(
            _resources.getString(APPLICATION_TITLE),
LabelField.ELLIPSIS
            | LabelField.USE_ALL_WIDTH));
        _mainScreen.add(new SeparatorField());
        _content = new RichTextField(
            _resources.getString(HTTPDEMO_CONTENT_DEFAULT));
        _mainScreen.add(_content);
        _mainScreen.addKeyListener(this);
        _mainScreen.addTrackwheelListener(this);
        // Start the helper thread.
        _connectionThread.start();
        pushScreen(_mainScreen);
        fetchPage(SAMPLE_PAGE);
    }
    // Retrieve web content.
    private void fetchPage(String url)
    {
        // Perform basic validation (set characters to lowercase and
add http:// or https://).
        String lcase = url.toLowerCase();
        boolean validHeader = false;
        int i = 0;
        for (i = HTTP_PROTOCOL.length - 1; i >= 0; --i)
```

```
        {
            if ( -1 != lcase.indexOf(HTTP_PROTOCOL[i]) )
            {
                validHeader = true;
                break;
            }
        }
        if ( !validHeader )
        {
            // Prepend the protocol specifier if it is missing.
            url = HTTP_PROTOCOL[0] + url;
        }
        // Create a new thread for connection operations.
        _connectionThread.fetch(url);
    }
    // Display the content.
    private void updateContent(final String text)
    {
        /* This technique creates several short-lived objects but
avoids
         * the threading issues involved in creating a static Runnable
and
         * setting the text.
         */
        UiApplication.getUiApplication().invokeLater(new Runnable()
        {
            public void run()
            {
                _content.setText(text);
            }
        });
    }
    // Close the connection thread when the user closes the
application.
    protected void onExit()
    {
        _connectionThread.stop();
    }
}
```

使用 HTTPS 连接

注: BlackBerry Internet Service Browser 不允许 Java 应用程序启动 HTTP, HTTPS 和 TCP 连接。

打开一个 HTTPS 连接

为打开一个 HTTPS 连接，调用 Connector.open()，指定 https 作为协议。将返回的对象转化为一个 HttpsConnection;

```
HttpsConnection stream
= (HttpsConnection)Connector.open("https://host:443/");
```

指定代理或终端到终端（end_to_end）模型

缺省的，连接在代理模型中使用 HTTPS。用户也可以设置一个 BlackBerry 设备选项来使用缺省的 end_to_end 模型。为获得更多信息，参看 189 页的“HTTPS 支持“。

为了在 end_to_end 模型里打开一个 HTTPS 连接，将下列参数中的一个增加到传递给 Connector.open()的连接字符串中：

参数	描述
EndToEndRequired	这个参数指定了一个 end_to_end HTTPS 连接必须从 BlackBerry 设备到目标服务器中使用。如果一个 end_to_end 的 HTTPS 连接不能建立，这个连接将会关闭。
EndToEndDesired	这个参数指定一个 end_to_end HTTPS 连接应该从 BlackBerry 设备到目标服务器中被使用，如果 BlackBerry 设备支持它的话。如果 BlackBerry 设备不支持 end_to_end TLS，并且用户许可代理 TLS 连接，那么一个代理连接将被使用。

```
HttpsConnection stream =
    (HttpsConnection)Connector.open("https://host:443/;EndToEndDesire
d");
```

注：BlackBerry 设备缺省没有安装 end_to_end 模块。尽管这样，BlackBerry 桌面版软件 3.6.0 及后续版本中包含了它。当应用程序加载到 BlackBerry 设备时，为了加载模块，为你的应用程序在.alx 文件中加入下面的标记：

```
  <requires id="net.rim.blackberry.crypto1"/>
  <requires id="net.rim.blackberry.crypto2"/>
```

为获得更多信息，参看 183 页的".alx 文件"。

使用 socket 连接

注: BlackBerry Internet Service Browser 不允许 Java 应用程序启动 HTTP, HTTPS 和 TCP 连接。

指定 TCP 的设置

应用程序可以在下面的一种模式下建立一个 TCP socket 连接或一个在 TCP 上的 HTTP 连接。

模式	描述
代理模式	BES 的 MDS 服务特征为 BlackBerry 设备建立与 Web 服务器的 TCP 连接
直接模式	BlackBerry 设备建立一个直接与 Web 服务器的 TCP 连接。

注：使用直接 TCP 模式需要你和服务提供商一起紧密工作。联系你的服务提供商确保支持直接 TCP socket 连接。

为了通过编程指定 TCP 设置，增加可选的 deviceside 参数到传递给 Connector.open()的连接字符串。
在 GSM 网络里，为指定 BlackBerry 设备商的 TCP 设置，用户点击 BlackBerry 设备选项的 TCP。

注：如果 IT 策略设置允许 TCP 连接，TCP 才在 BlackBerry 设备选项里显示。
如果 TCP 设置没有指定，下将使用下面缺省的。

网络	缺省的 TCP 设置	可选的 TCP 设置
GSM	代理模式	直接模式
iDEN	直接模式	代理模式

参看 API 参考的 Connector 获得更多信息。

打开一个 socket 连接

为打开一个 socket 连接，调用 Connector.open(),指定 socket 为其协议。

注：应用程序必须显式的输入他们的本地机器 IP，因为 localhost 不被支持。

```
private static String URL = "socket://<local machine IP>:4444";
StreamConnection conn = null;
conn = (StreamConnection)Connector.open(URL);
```

在 socket 连接上发送和接收数据

使用 openInputStream() 和 openOutputStream()获得输入和输出流。

```
OutputStreamWriter _out = new
OutputStreamWriter(conn.openOutputStream());
String data = "This is a test";
int length = data.length();
_out.write(data, 0, length);
InputStreamReader _in = new InputStreamReader(conn.openInputStream());
char[] input = new char[length];
for ( int i = 0; i < length; ++i ) {
    input[i] = (char)_in.read();
}
```

关闭连接

调用输入和输出流，以及 socket 连接上的 close()方法。

注：每个 close()方法抛出一个 IOException。应用程序必须实现这个异常的处理。

```
_in.close();
_out.close();
conn.close();
```

使用端口连接

注：当你的应用程序首先访问 net.rim.device.api.system.SerialPort 类或 net.rim.device.api.system.USBPort 类时，检查 NoClassDefFoundError。如果系统管理员使用应用程序管理限制访问序列端口和 USB 接口，这个错误就会抛出。参看 16 页的“应用程序管理”获得更多信息。

当它们使用一个序列端口或 USB 接口连上一台计算机式，应用程序可以使用一个序列端口或 USB 接口和桌面的应用程序进行通信。连接类型也可以使用来和一个插到序列端口或 USB 接口的外围设备通信。

注：如果你正在使用端口连接和桌面应用程序通信，你不必让所有其他正在使用序列端口或 USB 接口的应用程序运行。

打开一个 USB 接口或序列端口连接

调用 Connector.open(),指定 comm 作为协议，COM1 或 USB 作为端口。

```
private StreamConnection _conn =
    (StreamConnection)Connector.open(
    "comm:COM1;baudrate=9600;bitsperchar=8;parity=none;stopbits=1");
```

在端口连接上发送数据

调用 openDataOnputStream() 和 openOutputStream()获得输出流。

```
DataOutputStream _dout = _conn.openDataOutputStream();
```

使用输出流上的写方法来写数据。

```
private String data = "This is a test";
_dout.writeChars(test);
```

在端口连接上接收数据

调用 openDataInputStream() 和 openIutputStream()获得输入流。

```
DataInputStream _din = _conn.openInputStream();
```

使用输入流上的读方法来写数据。

```
   String contents = _din.readUTF();
```

注：你不能从在主事件线程上的输入流读取，因为这个操作会阻塞直至数据接收完成。创建一个独立的线程，在此线程上接收数据。

关闭端口连接

调用输入和输出流，以及端口连接上的 close()方法。

注：每个 close()方法可能抛出一个 IOException 异常。应用程序必须实现异常的处理。

```
   _din.close();
   _dout.close();
   conn.close();
```

使用蓝牙序列端口连接

蓝牙 API（net.rim.device.api.bluetooth）允许应用程序访问蓝牙序列端口配置（Profile）以及允许启动一个服务器或者客户端蓝牙序列端口连接到一台计算机或其他蓝牙无线技术支持的设备。

注：当你的应用程序首先访问蓝牙 API 时，会检查 NoClassDefFoundError。如果系统

管理员使用应用程序管理限制访问序列端口和 USB 接口，这个错误就会抛出。参看 16 页的“应用程序管理”获得更多信息。

BlackBerry 模拟器不支持蓝牙。

打开一个蓝牙序列端口连接

为了打开一个蓝牙序列端口连接，调用 Connector.open(), 它提供由 BluetoothSerialPort.getSerialPortInfo()返回的序列端口信息作为参数。

由这个方法返回的连接字符串指定了作为协议的 btspp:// 以及下面条目之一：

- 如果你正在打开一个连接作为客户端，由 getSerialPortInfo().toString()返回的连接字符串包含了设备号（device ID）以及 Server 设备正在监听的端口。
- 如果你正在打开一个连接作为服务器，由 getSerialPortInfo().toString()返回的连接字符串包含了你的 BlackBerry 设备正在监听的端口。

```
BluetoothSerialPortInfo[] info =
BluetoothSerialPort.getSerialPortInfo();
StreamConnection _conn =
(StreamConnection)Connector.open( info.toString(),
       Connector.READ_WRITE );
```

在蓝牙序列端口连接上发送数据

调用 openDataOutputStream()或 openOutputStream()获得一个输出流。

注：直到连接建立，这个调用会阻塞。

```
DataOutputStream _dout = _conn.openDataOutputStream();
```

在输出流上使用写方法来写数据

```
private String data = "This is a test";
_dout.writeChars(test);
```

在蓝牙序列端口连接上接收数据

调用 openDataInputStream()或 openInputStream()获得一个输入流。

```
   DataInputStream _din = _conn.openInputStream();
```

在输入流上使用读方法来读数据

```
   String contents = _din.readUTF();
```

注：你不能在主事件线程上读取输入流数据，因为这个操作会阻塞直到数据接收完毕。创建一个独立的线程来接收数据。

关闭一个端口连接

在输入和输出流以及蓝牙序列端口连接上调用 close()方法。

```
if (_bluetoothConnection != null) {
    try {
        _bluetoothConnection.close();
        }
    catch(IOException ioe)
    {

    }
}

if (_din != null) {
   try {
     _din.close();
   }
   catch(IOException ioe) {
   }
}
if (_dout != null) {
   try {
    _dout.close();
  }
   catch(IOException ioe) {
 }
}
_bluetoothConnection = null;
_din = null;
_dout = null;
```

代码实例

BluetoothSerialPortDemo.java 实例使一个简单蓝牙序列端口应用程序的客户端。这个应用程序监听在序列端口上的数据，并且当数据到达时提交数据。

例：BluetoothSerialPortDemo.java

```
/**
* BluetoothSerialPortDemo.java
* Copyright (C) 2004-2005 Research In Motion Limited.
*/
/* The client side of a simple serial port demonstration application.
* This application listens for text on the serial port and
```

```
* renders the data when it arrives.
*/
package com.rim.samples.docs.bluetoothserialportdemo;
import java.io.*;
import javax.microedition.io.*;
import net.rim.device.api.bluetooth.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;
import com.rim.samples.docs.baseapp.*;
import com.rim.samples.docs.resource.*;

public class BluetoothSerialPortDemo extends BaseApp implements
BluetoothSerialPortDemoResResource {

    //statics ------------------------------------------------------
-----------
    private static ResourceBundle _resources;
    private static final int INSERT = 1;
    private static final int REMOVE = 2;
    private static final int CHANGE = 3;
    private static final int JUST_OPEN = 4;
    private static final int CONTENTS = 5;
    private static final int NO_CONTENTS = 6;

    static {
        _resources =
ResourceBundle.getBundle(BluetoothSerialPortDemoResResource.BUNDLE_ID,
                BluetoothSerialPortDemoResResource.BUNDLE_NAME);
        }

    private EditField _infoField;
    private StreamConnection _bluetoothConnection;
    private DataInputStream _din;
    rivate DataOutputStream _dout;

    public static void main(String[] args)
    {
        BluetoothSerialPortDemo theApp = new BluetoothSerialPortDemo();
        theApp.enterEventDispatcher();
    }
```

```
    //constructor ----------------------------------------------------
-----------
    public BluetoothSerialPortDemo()
    {
        MainScreen mainScreen = new MainScreen();
        mainScreen.setTitle(new LabelField(_resources.getString(TITLE),
                LabelField.USE_ALL_WIDTH));
        _infoField = new EditField(Field.READONLY);
        mainScreen.add(_infoField);
        mainScreen.addKeyListener(this);
        mainScreen.addTrackwheelListener(this);
        pushScreen(mainScreen);
        invokeLater(new Runnable()
        {
            public void run() {
                openPort();
                }
        });
    }

    protected void onExit() {
        closePort();
    }

    // Close the serial port.
    private void closePort() {
        if (_bluetoothConnection != null)
        {
            try {
                _bluetoothConnection.close();
                }
            catch(IOException ioe) {
                }
            if (_din != null)
            {
                try {
                    _din.close();
                    }
                catch(IOException ioe) {
                }
            }
```

```
            if (_dout != null) {
                try {
                    _dout.close();
                    }
                catch(IOException ioe)
                {
                }
            }
            _bluetoothConnection = null;
            _din = null;
            _dout = null;
            }
    }


    // Open the serial port.
    private void openPort() {
        if (_bluetoothConnection != null) {
            closePort();
            }
        new InputThread().start();
        }

    private class InputThread extends Thread {
        public void run() {
            try {
                BluetoothSerialPortInfo[] info =
                    BluetoothSerialPort.getSerialPortInfo();
                if( info == null || info.length == 0 ) {
                    invokeAndWait( new Runnable() {
                        public void run() {
                            Dialog.alert( "No bluetooth serial ports
available for connection.");
                            onExit();
                            System.exit(1);
                            }
                        });

                _bluetoothConnection =
(StreamConnection)Connector.open( info[0].toString(),
                        Connector.READ_WRITE);
                _din = _bluetoothConnection.openDataInputStream();
                _dout = _bluetoothConnection.openDataOutputStream();
                }
```

```
            catch(IOException e) {
                invokeAndWait( new Runnable() {
                public void run() {
Dialog.alert("Unable to open serial port");
onExit();
System.exit(1);
}
});
} catch( UnsupportedOperationException e ) {
invokeAndWait( new Runnable() {
public void run() {
Dialog.alert("This handheld or simulator does not support
bluetooth.");
onExit();
System.exit(1);
}
});
}
try {
int type, offset, count;
String value;
_dout.writeInt(JUST_OPEN);
_dout.flush();
for (;;) {
type = _din.readInt();
if (type == INSERT) {
  offset = _din.readInt();
  value = _din.readUTF();
  insert(value, offset);
}
 else if (type == REMOVE) {
  offset = _din.readInt();
  count = _din.readInt();
  remove(offset, count);
}
else if (type == JUST_OPEN) {
  // Send contents to desktop.
  value = _infoField.getText();
  if (value == null || value.equals("")) {
  _dout.writeInt(NO_CONTENTS);
  _dout.flush();
}
else {
  _dout.writeInt(CONTENTS);
```

```
  _dout.writeUTF(_infoField.getText());
  _dout.flush();
}
}
else if (type == CONTENTS) {
  String contents = _din.readUTF();
  synchronized(Application.getEventLock()) {
  _infoField.setText(contents);
}
}
else if (type == NO_CONTENTS) {
}
else {
   throw new RuntimeException();
 }
}
}
catch(IOException ioe) {
  invokeLater(new Runnable() {
    public void run() {
        Dialog.alert("Problems reading from or writing to serial
        port.");
        onExit();
        System.exit(1);
       }
    });
  }
 }
}

private void insert(final String msg, final int offset) {
  invokeLater(new Runnable() {
    public void run() {
      _infoField.setCursorPosition(offset);
      _infoField.insert(msg);
      }
   });
}
private void remove(final int offset, final int count) {
invokeLater(new Runnable() {
    public void run() {
      _infoField.setCursorPosition(offset+count);
      _infoField.backspace(count);
    }
```

```
    });
}
/**
* Override makeMenu to add custom menu items.
*/
protected void makeMenu(Menu menu, int instance)
{
if (_infoField.getTextLength() > 0) {
menu.add(new MenuItem(_resources, MENUITEM_COPY, 100000, 10) {
public void run() {
Clipboard.getClipboard().put(_infoField.getText());
}
});
}
super.makeMenu(menu, instance);
}
}
```

7

第 7 章 使用数据报（Datagram）连接

数据报连接

通过利用 UDP（User Datagram Protocal，用户数据报协议），BlackBerry 设备支持数据报连接。应用程序使用 UDP 和标准的网络服务通信。
数据报是应用程序发送到网络的独立数据包。对于 Datagram 的负载字节数组来说，一个 Datagram 对象是一个包装器。为获得这个字节数组的一个引用，调用 getData()方法。

和 HTTP 连接不一样，数据报连接不稳定：数据包以任意的顺序到达，并且传输得不到保证。应用程序的责任是确保请求数据报的数据负载根据网络服务的标准来格式化，这个标准是在数据报传播上的。应用程序也必须能解析从服务器返回发送的数据报。

使用数据报连接来发送短消息。为获得更多信息，参看 121 页的“发送和接受 SMS”。

使用 UDP 连接

为使用UDP连接，你必须有一个你自己的基础设施来连接无线网络，包括一个GPRS（General Packet Radio Service，通用分组无线业务）网络的 APN（Access Point Name）。

注：数据报连接没有使用 BlackBerry 的基础设施，因此连接没有加密。模拟器的 APN 是 net.rim.gprs.

javax.microedition.io.DatagramConnection 接口，扩展了 Connection 类，它定义了发送和接受数据报的连接。Datagram 接口定义了在数据报连接上发送和接受的数据包。

注：使用 UDP 连接需要你和服务商紧密联系。联系你的服务商确认 UDP 连接是否支持。

如果你的服务商不支持多个 PDP 上下文，那么你可能没有建立一个 UDP 连接。一个 PDP 上下文为发送消息的 BlackBerry.net.APN 保留。尽管如此，你可以为 UDP 使用 blackberry.net 当作 APN。联系你的服务商以获得更多信息。

获得一个数据报连接

使用下面的格式调用 Connector.open().指定 udp 为你的协议。将返回的对象转化为一个 DatagramConnection。

```
(DatagramConnection)Connector.
open("udp://host:dest_port[;src_port]/apn");
```

参数	描述
host	指定点阵 ASCII 十进制格式的主机地址
dest_port	指定了主机地址的目标端口（接受信息时是可选的）。
src_port	指定本地源端口（可选）。
apn	指定字符串形式的网络 APN。

注：可以在相同的端口上收发 UDP 数据报。

为了在 UDP 连接上发送数据，在连接字符串里指定目标端口。为了在 UDP 连接上接收数据，在连接字符串里指定源端口。为了接收指定主机的所有端口上的数据报，省略目标端口。

注：为了在一个非 GPRS 的网络里打开一个连接，不要指定 APN。在源端口后包含斜线”/”.例如 CDMA 网络连接的地址应该是 udp://121.0.0.0:6343/.

创建一个数据报

调用 DatagramConnection.newDatagram().

```
Datagram outDatagram = conn.newDatagram(buf, buf.length);
```

将数据加入到数据报中

调用 Datagram.setData().数据的格式由接收它的服务决定。

```
byte[] buf = new byte[256];
outDatagram.setData(buf, buf.length);
```

在 UDP 连接上发送数据

在数据报连接山调用 send()。

```
conn.send(outDatagram);
```

注：如果应用程序试图在一个 UDP 连接上发送一个数据报，并且接收者没有监听指定的源端口，一个 IOException 会抛出.

接收 UDP 连接上的数据

调用数据报连接上的 receive().

```
byte[] buf = new byte[256];
Datagram inDatagram = conn.newDatagram(buf, buf.length);
conn.receive(inDatagram);
```

注: receive()方法会阻塞其他操作, 直至它接收完一个数据包. 如果你知道正在发送的数据格式, 转化他们为合适的格式.

从数据报提取数据

在数据报连接上调用 getData()方法。如果你知道正在接收的数据类型，将数据转化为合适的格式。

```
String received = new String(inDatagram.getData());
```

关闭 UDP 连接

和 MIDP 框架中所有连接一样，调用输入和输出流以及数据报上的 close（）方法，

使用 Mobitex 网络

DatagramConnectionBase 类提供了方法来处理 Mobitex 网络上的 BlackBerry 数据报连接以及传输操作。

打开一个 Mobitex 数据报连接

调用 Connector.open(), 然后将返回的对象转化为一个 DatagramConnectionBase 。DatagramConnectionBase 类实现了 DatagramConnection，并且提供了额外的方法，对注册一个数据报状态监听者来说，这些方法是必要的。

为提供一个参数给 Connector.open(),连接字符串使用下面的格式：

mobitex:<type>:<MAN>

参数	描述
<type>	接受下列值:“TEXT”,”DATA”,”STATUS”,或”HPDATAHPID（在这些值中，JPID 的格式是 ASCII 十进制）
<MAN>	Mobitex 访问号码（Mobitex Access Number），接受 ASCII 十进制格式。

注：如果你打开一个服务器连接（一个监听者），MAN 留为空白。

```
// The datagram connection <type> is DATA and the MAN is left blank
for an incoming
// connection.
DatagramConnection dc =
(DatagramConnection)Connector.open("mobitex:DATA:");
DatagramConnectionBase dcb = (DatagramConnectionBase)dc;
```

监听数据报状态事件

如果你想注册一个数据报状态事件监听者，使用一个 DatagramBase，而不是 Datagram 来抓住（hold）数据。DatagramBase 实现了 Datagram 接口，并且提供额外的方法，这些方法对注册数据报状态事件监听者是有用的。

```
// dc is a DatagramConnection.
Datagram d = dc.newDatagram(dc.getMaximumLength());
d.setAddress(address);
d.setData(raw, 0, raw.length);
DatagramBase db = (DatagramBase)d; // An error if this fails.
```

注册一个数据报状态监听者

你的 DatagramStatusListener 接口的实现监听事件，例如一个数据报的接收。参看 *API 参考* 的 DatagramStatusListener 以获得完整的数据报状态事件列表。

为了分配一个数据报 ID，并将之显式指派给 DatagramStatusListener，调用 DatagramConnectionBase.allocateDatagramId().

```
 int id = dcb.allocateDatagramId(d);
```

以此种方式预先分配数据报 ID，可以确保你的监听者代码知道与此 ID 相关联的数据报。

获得数据报信息

MobitexAddress 类封装了 Mobitex 地址信息，例如 MAN，消息的类型以及消息的状态。MobitexPacketHeader 类提供对无线头字段（radio header field）的底层访问。为了对所有地址操作使用 MobitexPacketHeader 并忽略其他的 MobitexAddress 字段，调用 MobitexAddress.setPacketHeader().

获取无线和网络信息

MobitexInfo 类提供对象存储普通的无线（radio）状态信息。Mobitex.MobitexCellInfo 类提供对象存储 Mobitex 蜂窝信息。

代码实例

MobitexDemo.java 代码实例描述了 Mobitex 无线称 API 的使用。

例：MobitexDemo.java

```
/*
* MobitexDemo.java
*
```

```
* Copyright (C) 2003-2005 Research In Motion Limited
*/
package com.rim.docs.samples.mobitexdemo;
import javax.microedition.io.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import java.util.*;
import net.rim.device.api.io.*;
import net.rim.device.api.system.*;
import java.io.*;
import com.rim.docs.samples.baseapp.*;

/**
* A simple mobitex layer sample.
*/
public final class MobitexDemo extends BaseApp
    implements MobitexDemoResource {

    private MainScreen _mainScreen;
    private EditField _pin;
    private EditField _data;
    private RichTextField _messages;
    private SenderThread _sendThread;
    private ReceiverThread _receiverThread;

    // statics ------------------------------------------------------
------------
    private static ResourceBundle _resources =
        ResourceBundle.getBundle(MobitexDemoResource.BUNDLE_ID,
                                 MobitexDemoResource.BUNDLE_NAME);

    static public void main(String[] args)
    {
        new MobitexDemo().enterEventDispatcher();
    }

    // menu items ---------------------------------------------------
-------------
    // Cache the send menu item for reuse.
    private MenuItem _sendMenuItem = new MenuItem(_resources,
MOBITEXDEMO_MENUITEM_SEND,
            100, 10) {
```

```
        public void run()
        {
            // Don't execute on a blank address.
            String pin = _pin.getText();
            String data = _data.getText();
            if ( pin.length() > 0 )
            {
                send(pin, data);
            }
        }
    };

    // Cache the clear messages menu item for reuse.
    private MenuItem _clearMessages = new MenuItem(_resources,
            MOBITEXDEMO_MENUITEM_CLEARMESSAGES, 105, 10) {
        public void run() {
            _messages.setText("");
            }
    };

    public MobitexDemo()
    {
        _mainScreen = new MainScreen();
        _mainScreen.setTitle( new
LabelField(_resources.getString(MOBITEXDEMO_TITLE),
                LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH));
        _pin = new
EditField(_resources.getString(MOBITEXDEMO_LABEL_PIN), null,
                Integer.MAX_VALUE, EditField.FILTER_PIN_ADDRESS);

        _mainScreen.add(_pin);
        _data = new
EditField(_resources.getString(MOBITEXDEMO_LABEL_DATA), null);
        _mainScreen.add(_data);
        _mainScreen.add(new SeparatorField());
        _messages = new
RichTextField(_resources.getString(MOBITEXDEMO_CONTENT_DEFAULT));
        _mainScreen.add(_messages);
        _mainScreen.addKeyListener(this); //implemented by the super
        _mainScreen.addTrackwheelListener(this); //implemented by the
super

        //start the helper threads
        _sendThread = new SenderThread();
```

```
        _sendThread.start();
        _receiverThread = new ReceiverThread();
        _receiverThread.start();

        //push the main screen - a method from the UiApplication class
        pushScreen(_mainScreen);
        }

    // methods ------------------------------------------------------
------------
    /*public boolean keyChar(char key, int status, int time)
     {
if
( UiApplication.getUiApplication().getActiveScreen().getLeafFieldWith
Focus() ==
pin && key == Characters.ENTER )
{
_sendMenuItem.run();
return true; // I've absorbed this event, so return true.
}
else
{
return super.keyChar(key, status, time);
}
}*/


    protected void makeMenu(Menu menu, int instance)
    {
        menu.add(_sendMenuItem);
        menu.add(_clearMessages);
        menu.addSeparator();
        super.makeMenu(menu, instance);
    }

    private void send(String pin, String data)
    {
        _sendThread.send(pin, data);
    }

    private void message(String msg)
    {
        System.out.println(msg);
        _messages.setText(_messages.getText() + msg + "\n");
```

```
    }


    // inner classes ------------------------------------------------
-------------
    private class ReceiverThread extends Thread
    {
        private DatagramConnection _dc;

        // Shut down the thread.
        public void stop()
        {
            try {
                _dc.close();
                }
            catch(IOException e) {
                MobitexDemo.this.message(e.toString());
                }
        }

        public void run()
        {
            try {
                // Incoming data connection - leave the MAN blank.
                _dc =
(DatagramConnection)Connector.open("mobitex:DATA:");
                for(;;)
                {
                    Datagram d =
_dc.newDatagram(_dc.getMaximumLength());
                    _dc.receive(d);
                    DatagramBase db = (DatagramBase)d;
                    MobitexAddress ma =
(MobitexAddress)db.getAddressBase();
                    MobitexPacketHeader mph = ma.getPacketHeader();
                    StringBuffer sb = new StringBuffer();
                    sb.append("Recieved packet");
                    sb.append("\nFROM:");
                    sb.append(mph.getSourceAddress());
                    sb.append("\nTO:");
                    sb.append(mph.getDestinationAddress());
                    sb.append("\nFLAGS:");

    sb.append(Integer.toHexString(mph.getPacketFlags()));
```

```
                    sb.append("\nDATA:");
                    sb.append(new String(db.getData()));
                    MobitexDemo.this.message(sb.toString());
                    }
                }
            catch (IOException e) {
                MobitexDemo.this.message(e.toString());
                }
            }
    }

    /**
     * The ConnectionThread class manages the datagram connection
     */
    private class SenderThread extends Thread implements
DatagramStatusListener {
        private static final int TIMEOUT = 500; //ms
        private Vector _sendQueue = new Vector(5);
        private volatile boolean _start = false;
        private volatile boolean _stop = false;

        // Queue something for sending
        public void send(String pin, String data)
        {
            synchronized(_sendQueue) {
                _sendQueue.addElement(new String[] { pin, data });
                _start = true;
            }
        }

        // Shut down the thread.
        public void stop() {
            _stop = true;
            }

        public void run()
        {
            for(;;) {
                String pin = null;
                String data = null;

                // Thread control.
                synchronized(_sendQueue) {
                    while( !_start && !_stop)
```

```
                {
                    // Sleep for a bit so we don't spin.
                    try {
                        _sendQueue.wait(TIMEOUT);
                        }
                    catch (InterruptedException e) {
                        System.err.println(e.toString());
                        }
                    }

                if (_start) {
                    String[] a = (String[])_sendQueue.firstElement();
                    if ( a != null ) {
                        pin = a[0];
                        data = a[1];
                        }
                    _start = false;
                    }
                else if ( _stop ) { // Exit condition
                    return;
                    }
                }

            // Open the connection and extract the data.
            DatagramConnection dc = null;
            try {
                String address = "DATA:" + pin;
                dc = (DatagramConnection)Connector.open("mobitex:"
+ address);

                //an error if this fails
                DatagramConnectionBase dcb =
(DatagramConnectionBase)dc;
                Datagram d = dc.newDatagram(dc.getMaximumLength());
                byte[] raw = data.getBytes();
                d.setAddress(address);
                d.setData(raw, 0, raw.length);

                // An error if this fails.
                DatagramBase db = (DatagramBase)d;

                // Allocate a datagram ID - if you want to know
about status.
                // For this particular datagram, then we can
```

```
allocate the id
                    // here and log it for later follow up

                    dcb.allocateDatagramId(d);
                    // Set up a status listener.
                    db.setDatagramStatusListener(this);
                    dcb.send(d);
                    dc.close();
                    }
                catch (IOException e) {
                    MobitexDemo.this.message(e.toString());
                    }

                // We're done one connection so reset the start state.
                _start = false;
                }
            }


        public void updateDatagramStatus(int dgId, int code, Object
context) {
            String msg = "Datagram: " + Integer.toHexString(dgId) +
            "\nStatus: " +
DatagramStatusListenerUtil.getStatusMessage(code);
            MobitexDemo.this.message(msg);
            }
        }

    protected void onExit() {
        _sendThread.stop();
        _receiverThread.stop();
        }
    }
```

发送和接收短消息

使用 UDP 在数据报包上发送和接收短消息（SMS）。包含了 BlackBerry 消息头信息的短消息数据报包，其大小是固定的 160 个字节。

注：短消息不是所有的网络都支持。和你的服务商确认是否相关的网络全部和部分支持 SMS。大多数情况下，GPRS 和支持 CDMA 的 BlackBerry 设备支持 SMS。如果服务商提供 SMS 服务，系统管理员也可以使用 IT 策略来控制公司用户对 SMS 的使用。管理员可以将 ENABLE_SMS 项设置为 TRUE 或 FALSE。缺省值是 TRUE（SMS 可用）。

发送 SMS

打开一个数据报连接来发送 SMS

调用 Connector.open().使用下面格式提供一个连接字符串:

```
DatagramConnection _dc = Connector.open("sms://<peer_address>");
```

在这里，<peer_address>是接收者的电话号码－MSISDN（MobileStation ISDN Number）。你也可以忽略 peer_address，代替它的是调用 Datagram.setAddress（）来设置消息的目的地地址。

创建一个 SMS

调用 DatagramConnection.newDatagram().

```
Datagram smsMessage = conn.newDatagram(buf, buf.length);
```

设置 SMS 内容

调用 setData（）。

```
private String _msg = "This is a test message";
byte[] data = _msg.getBytes();
smsMessage.setData(data, 0, data.length);
```

发送一个 SMS 消息

注：在一个与主应用程序线程独立的线程上打开网络连接，这样 UI 就不会停止。

如果你没有在连接字符串中指定 peer_address，那么调用 Datagram.setAddress（）设置 SMS 地址。为了发送 SMS，调用 DatagramConnection.send()。

```
smsMessage.setAddress("sms://+15555551234");
_dc.send(datagram);
```

代码实例

SendSms.java 代码实例描述了如何在独立的线程上发送一条 SMS 消息。

在实例的工作空间中，SendSms.java 实例需要一个服务器端的应用程序来与 BlackBerry 设备模拟器交互，来模拟发送和接收 SMS 消息。你不能从 BlackBerry 设备模拟器发送一个实际的 SMS。

样例（SMSServer.java）的服务器端程序包含在 BlackBerry JDE 下。为了运行服务器端程序，运行 run.bat，它在你的 BlackBerry JDE 实例的子目录下。例如:

\samples\com\rim\samples\server\smsdemo\.

例：SendSms.java

```
/**
* SendSms.java
* Copyright (C) 2002-2005 Research In Motion Limited. All rights
reserved.
*/
```

```
package com.rim.samples.docs.smsdemo;
import net.rim.device.api.io.*;
import net.rim.device.api.system.*;
import javax.microedition.io.*;
import java.util.*;
import java.io.*;

public class SendSms extends Application {
    private static final int MAX_PHONE_NUMBER_LENGTH = 32;

    // Members.
    private String addr = "15195551234";
    private String msg = "This is a test message.";
    private DatagramConnection _dc = null;
    private static String _openString = "sms://";

    public static void main(String[] args) {
        new SendSms().enterEventDispatcher();
        }

    public SendSms() {
        try {
            _dc = (DatagramConnection)Connector.open(_openString);
            byte[] data = msg.getBytes();
            Datagram d = _dc.newDatagram(_dc.getMaximumLength());
            d.setAddress("//" + addr);
            _dc.send(d);
            }
        catch ( IOException e) {

        }
        System.exit(0);
        }
}
```

接收 SMS 消息

创建一个独立的监听线程

在一个与主应用程序线程独立的线程上监听消息，以致 UI 不会停止。

打开数据报连接

调用 Connector.open().使用下面的格式提供一个字符串连接：

```
_dc = (DatagramConnection)Connector.
```

```
open("sms://<peer_address><port>");
```

在这里：

- <peer_address>是接收者的电话号码－MSISDN（MobileStation ISDN Number）
- <port>是应用程序接收 SMS 消息的端口号。

获取数据报

创建一个 Datagram 对象存储数据报。为获取 SMS 消息数据报，调用数据报连接上的 receive()。这个操作会阻塞直至数据接收完毕。

```
Datagram d = _dc.newDatagram(160); // SMS messages have a fixed size
of 160 bytes
_dc.receive(d);
```

从数据报提取数据

为了从 SMS 消息提取地址，调用 Datagram.getAddress().为了从 SMS 消息中提取数据，调用 Datagram.getData().

```
String address = d.getAddress();
String msg = new String(d.getData());
```

代码实例

ReceiveSms.java 描述了如何在一个独立的线程上接收一个 SMS 消息。

例：ReceiveSms.java

```
/**
* ReceiveSms.java
* Copyright (C) 2002-2005 Research In Motion Limited. All rights
reserved.
*/
package com.rim.samples.docs.smsdemo;
import net.rim.device.api.io.*;
import net.rim.device.api.system.*;
import javax.microedition.io.*;
import java.util.*;
import java.io.*;

public class ReceiveSms extends Application {
    private ListeningThread _listener;
    // Additional code required for complete sample.
    public static void main(String[] args) {
        new ReceiveSms().enterEventDispatcher();
        }

    ReceiveSms() {
        _listener = new ListeningThread();
        _listener.start();
```

```
        }

    private static class ListeningThread extends Thread {
        private boolean _stop = false;
        private DatagramConnection _dc;
        public synchronized void stop() {
            _stop = true;
            try {_
                dc.close(); // Close the connection so the thread
returns.
            }
            catch (IOException e) {
                System.err.println(e.toString());
                }
            }

        public void run() {
            try {
                _dc = (DatagramConnection)Connector.open("sms://");
                for(;;) {
                    if ( _stop ) {
                        return;
                        }
                    Datagram d =
_dc.newDatagram(_dc.getMaximumLength());
                    _dc.receive(d);
                    String address = new String(d.getAddress());
                    String msg = new String(d.getData());
                    System.out.println("Message received:" + msg);
                    System.out.println("From:" + address);
                    System.exit(0);
                    }
                }
            catch (IOException e) {
                System.err.println(e.toString());
                }
            }
        }
    }
```

8

第 8 章 本地化应用程序

资源文件

将应用程序设计为在代码不改变的情况下它们可以本地化（适合指定的语言以及区域）。为了代替在你的源代码中包含原文元素，将文本字符串存储到一个独立的资源文件中。在你的源代码例中，使用唯一的标志符映射到合适的资源。

在独立的资源文件中存储文本字符串有 2 个好处：

- 文本翻译有更高的效率，因为一个给定位置（locale）的所有文本字符串都存储在一个单独的文件中，这个文件在你的源代码之外。
- 基于用户的位置，应用程序可以动态的获得合适的文本显示给用户。

BlackBerry JDE 包含了一个内置的资源机制来创建字符串资源。本地化（Localization）API 包含在 net.rim.device.api.i18n 包里。

注：MIDP 应用程序不支持本地化。

一个给定的位置的资源存贮在一个 ResourceBundle 对象中。一个 ResourceBundleFamily 对象包括了一个 ResourceBundle 的集合，它将应用程序的资源进行分组。在不需要新的资源包下，应用程序可以切换语言，这依赖于用户的位置。

BlackBerry JDE 将每个资源包编译为一个独立已编译的.cod 文件。你可以为应用程序将其他.cod 文件和适合的.cod 文件一起加载到 BlackBerry 设备上。

本地化需要的文件	描述	例子
资源头文件	这个文件为每个本地字符串定义具有描述性的键。 当 BlackBerry 编译一个项目时，它创建一个以 Resource 加到.rrh 文件名末作为文件名的资源接口。例如，如果你创建 AppName.rhh，接口就是 AppNameResouce。	AppName.rrh
资源内容文件（根位置）	文件为根（全局）位置将资源	AppName.rrc

	键映射到字符串值。它和资源头文件有相同的名字。	
资源内容文件（指定的位置）	文件为指定的位置（语言和国家）将资源键映射到字符串值。文件和资源头文件有相同的名字，跟在后面的是下划线（_）和语言代码，然后是可选的，一个下划线和国家代码。 2 个字母的语言代码和国家代码分别在 ISO-639 以及 ISO－3166 有描述。	AppName_en.rrc AppName_en_GB_rrc AppName_fr.rrc
初始化文件	文件初始化资源包机制。仅当资源作为一个独立的工程编译时本文件为才需要	Init.java

资源继承

在一个基于继承的层次里组织资源。若一个字符串在一个位置里没有定义，下一个最靠近的字符串会被使用。

为应用程序加入本地化支持

增加资源头文件

1. 在 BlackBerry IDE，打开 **File** 菜单，点击 **New**。
2. 在 **Source file name** 里，键入一个文件名。
3. 点击 **Browse**。
4. 选择一个包含文件的文件夹。
5. 点击 **OK**。
6. 在域里，键入包的名字，例如：**com.rim.samples.docs.countryinfo**.
7. 点击 **OK**。
8. 点击 **Yes**。
9. 除了包文件的描述，保留出现在文本编辑器的文件。
10. 在右边的区域里，右击加入文件到项目中，然后点击 **Insert into project**。

加入资源内容文件

在相同的文件夹下创建 3 个资源内容文件，在这里，ContryInfo.java 有：CountryInfo.rrc（根位置），CountryInfo_en.rrc(English),以及 CountryInfo_fr.rrc(French).

1. 在 **File** 菜单，点击 **New**。

2. 键入文件名以及位置。
3. 点击 **OK**。
4. 点击 **Yes**。
5. 保留文件为空。
6. 为加入.rrc 文件到你的应用程序项目中，在右边右击文件。
7. 点击 **Insert into project**。

加入资源

1. 在 BlackBerry IDE，双击一个资源头文件。
2. 在 **Root** 标签，键入资源键和应用程序的每个字符串或字符串数组的值。
   每行定义了单个资源。**Keys** 列为资源显示了一个具有描述性的名字。它是在你代码中使用它获得本地化文本的名字。**Values** 列为某个指定位置的资源显示文本。

   注：为单个资源键增加一组值，在资源编辑器中，右击一个资源，电解 Convert to Multiple Values。加入一个或多个值到数组。
3. 为指定一个其他位置里的不同文本字符串，选择位置的标签，例如法语是 **fr**.
4. 在资源的 **Value** 单元格里，为 locale 输入文本字符串。如果你没有为一个特定 locale 的资源定义一个值，root 的值将会使用。

   注：直接输入 unicode 字符到 Value 单元格中。访问 http://ww.unicode.org 获得更多信息。
5. 设置应用程序标题。

你可以提供一个本地化的应用程序标题显示在主屏幕（Home Screen）上。如果你没有为应用程序标题提供一个资源，将会使用项目属性窗口上的 **Application** 标签里的 **Title** 域的输入值。

1. 在资源编辑器中，为应用程序标题增加一个资源，例如 APPLICATION_TITLE.
2. 在你支持的每个 locale，为这个资源输入一个值。

注：为了为一个应用程序创建一个快捷键，在你用来作为快捷键的字符后加入 unicode 下划线字符(\u0332)。一个快捷键就是一个在主屏幕上你可以按此键启动应用程序的键。

3. 在 BlackBerry IDE 中，右击应用程序项目，然后点击 **Properties**。
4. 点击 **Resource** 标签。
5. 选择 **Title Resource Available** 选项。
6. 在 **Resource Bundle** 下拉列表中，为此应用程序选择资源头文件名。
7. 在 **Title Id** 下拉列表中，为应用程序的标题选择资源，例如 APPLICATION_TITLE.
8. 在 **Description ID** 下拉列表中，选择一个描述性的 ID。

代码实例

为了指定的 locale，而不是在代码里直接提供文本字符串，CountryInfo.java 实例描述了如何

在单独的资源文件中存储文本字符串。在你的源代码中，从资源里获取字符串为用户 locale 显示合适的文本。

例，CountryInfo.java

```
/**
* CountryInfo.java
* Copyright (C) 2001-2005 Research In Motion Limited. All rights
reserved.
*/
package com.rim.samples.docs.countryinfo;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.i18n.*;
import com.rim.samples.docs.resource.*;

/* This sample demonstrates how to store text strings in separate
resource
files for specific locales rather than providing text strings
directly
in the code. In your source code, you retrieve the string from the
resource
to display the appropriate text for the user locale. */

public class CountryInfo extends UiApplication
{
    public static void main(String[] args) {
        CountryInfo theApp = new CountryInfo();
        theApp.enterEventDispatcher();
        }

    public CountryInfo() {
        pushScreen(new HelloWorldScreen());
        }
}


final class HelloWorldScreen extends MainScreen implements
CountryInfoResource {
    private InfoScreen _infoScreen;
    private ObjectChoiceField choiceField;
    private int select;
    private static ResourceBundle _resources =
```

```
ResourceBundle.getBundle(
          CountryInfoResource.BUNDLE_ID,
CountryInfoResource.BUNDLE_NAME);

    public HelloWorldScreen() {
        super();
        LabelField title = new
LabelField(_resources.getString(APPLICATION_TITLE),
              LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
        setTitle(title);
        add(new RichTextField(_resources.getString(FIELD_TITLE)));
        String choices[] = _resources.getStringArray(FIELD_COUNTRIES);
        choiceField = new
ObjectChoiceField(_resources.getString(FIELD_CHOICE), choices);
        add(choiceField);
        }

    public boolean onClose() {
           Dialog.alert(_resources.getString(CLOSE));
           System.exit(0);
           return true;
           }

    private MenuItem _viewItem = new MenuItem(_resources,
MENUITEM_VIEW, 110, 10) {
        public void run() {
           select = choiceField.getSelectedIndex();
           _infoScreen = new InfoScreen();
           UiApplication.getUiApplication().pushScreen(_infoScreen);
           }
        };

    private MenuItem _closeItem = new MenuItem(_resources,
MENUITEM_CLOSE,200000, 10)
    {
        public void run() {
           onClose();
           }
        };

    protected void makeMenu( Menu menu, int instance ) {
           menu.add(_viewItem);
           menu.add(_closeItem);
           }
```

```
    private class InfoScreen extends MainScreen {
        public InfoScreen() {
            super();
            LabelField lf = new LabelField();
            BasicEditField popField = new BasicEditField(
                    _resources.getString(FIELD_POP), null, 20,
Field.READONLY);
            BasicEditField langField = new BasicEditField(
                    _resources.getString(FIELD_LANG), null, 20,
Field.READONLY);
            BasicEditField citiesField = new BasicEditField(
                    _resources.getString(FIELD_CITIES), null, 50,
Field.READONLY);
            add(lf);
            add(new SeparatorField());
            add(popField);
            add(langField);
            add(citiesField);
            if (select == 0) {
                lf.setText(_resources.getString(FIELD_US));
                popField.setText(_resources.getString(FIELD_US_POP));
                langField.setText(_resources.getString(FIELD_US_LANG));

    citiesField.setText(_resources.getString(FIELD_US_CITIES));
                }
            else if (select == 1) {
                lf.setText(_resources.getString(FIELD_CHINA));
                popField.setText(_resources.getString(FIELD_CHINA_POP));

    langField.setText(_resources.getString(FIELD_CHINA_LANG));

    citiesField.setText(_resources.getString(FIELD_CHINA_CITIES));
                }
            else if (select == 2) {
                lf.setText(_resources.getString(FIELD_GERMANY));

    popField.setText(_resources.getString(FIELD_GERMANY_POP));

    langField.setText(_resources.getString(FIELD_GERMANY_LANG));
                citiesField.setText(
                        _resources.getString(FIELD_GERMANY_CITIES));
                }
```

```
            }
        }
    }
```

从资源文件中获取字符串

实现资源接口

为支持国际化，实现合适的资源接口。BlackBerry IDE 自动编译这个资源头文件(.rhh)的接口。接口的名字是加入 Resource 的.rrh 文件名称

```
public class HelloWorldScreen extends MainScreen
                                implements CountryInfoResource
{
  ...
 }
```

获取资源包（bundle）

为此应用程序声明一个类变量来保持资源包。一个 ResourceBundle 对象为应用程序包含了所有本地资源，例如字符串。一个应用程序可以在运行时根据它的 locale 选择合适的包。

```
  private static ResourceBundle _resources = ResourceBundle.getBundle(BUNDLE_ID,
   BUNDLE_NAME);
```

为获取合适的包族（bundle family），调用 getBundle（）。当它创建资源接口作为项目的一部分时，BlackBerry IDE 创建 BUNDLE_ID 以及 BUNDLE_NAME 常数。

使用资源创建菜单项

为了使用资源创建 MenuItem 对象，使用 MenuItem 的构造子，这个构造子为了菜单项的名字，接受一个资源包以及一个资源来代替字符串。不要实现 toString（），因为菜单项的文本由资源提供。

```
private MenuItem _viewItem = new MenuItem(_resources, MENUITEM_VIEW,
110, 10)
{
    public void run() {
        select = choiceField.getSelectedIndex();
        _infoScreen = new InfoScreen();
        UiApplication.getUiApplication().pushScreen(_infoScreen);
        }
```

```
    }
```

用合适的资源替代文本字符串

为每个出现在屏幕上的 field，用合适的资源替代文本字符串。调用 getString()，或者 getStringArray()为适合的语言来获取字符串。

```
LabelField title = new
LabelField(_resources.getString(APPLICATION_TITLE),
       LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
add(new RichTextField(_resources.getString(FIELD_TITLE)));
String choices[] = _resources.getStringArray(FIELD_COUNTRIES);
choiceField                                =                                new
ObjectChoiceField(_resources.getString(FIELD_CHOICE), choices);
```

代码实例

下面的例子修改 CountryInfo.java 实例，以从资源中获取字符串。

例：CountryInfo.java（支持本地化）

```
/**
* CountryInfo.java
* Copyright (C) 2001-2005 Research In Motion Limited. All rights
reserved.
*/
package com.rim.samples.docs.localization;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.i18n.*;
import com.rim.samples.docs.resource.*;

public class CountryInfo extends UiApplication {
    public static void main(String[] args) {
        CountryInfo theApp = new CountryInfo();
        theApp.enterEventDispatcher();
        }

    public CountryInfo() {
        pushScreen(new HelloWorldScreen());
        }
}
```

```
final class HelloWorldScreen extends MainScreen implements
CountryInfoResource {
    private InfoScreen _infoScreen;
    private ObjectChoiceField choiceField;
    private int select;
    private static ResourceBundle _resources =
        ResourceBundle.getBundle( BUNDLE_ID, BUNDLE_NAME );

    public HelloWorldScreen() {
        super();
        LabelField title = new
LabelField(_resources.getString(APPLICATION_TITLE),
                LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);

        setTitle(title);
        add(new RichTextField(_resources.getString(FIELD_TITLE)));
        String choices[] = _resources.getStringArray(FIELD_COUNTRIES);
        choiceField = new
ObjectChoiceField(_resources.getString(FIELD_CHOICE), choices);
        add(choiceField);
        }

    public boolean onClose() {
        Dialog.alert(_resources.getString(CLOSE));
        System.exit(0);
        return true;
        }

    private MenuItem _viewItem = new MenuItem(_resources,
MENUITEM_VIEW, 110, 10) {
        public void run() {
            select = choiceField.getSelectedIndex();
            _infoScreen = new InfoScreen();
            UiApplication.getUiApplication().pushScreen(_infoScreen);
            }
        };

    private MenuItem _closeItem = new MenuItem(_resources,
MENUITEM_CLOSE, 200000, 10) {
        public void run() {
            onClose();
            }
        };
```

```
    protected void makeMenu( Menu menu, int instance ) {
        menu.add(_viewItem);
        menu.add(_closeItem);
        }

    private class InfoScreen extends MainScreen {
        public InfoScreen() {
            super();
            LabelField lf = new LabelField();
            BasicEditField popField = new BasicEditField(
                    _resources.getString(FIELD_POP), null, 20,
Field.READONLY);
            BasicEditField langField = new BasicEditField(
                    _resources.getString(FIELD_LANG), null, 20,
Field.READONLY);
            BasicEditField citiesField = new BasicEditField(
                    _resources.getString(FIELD_CITIES), null, 50,
Field.READONLY);
            add(lf);
            add(new SeparatorField());
            add(popField);
            add(langField);
            add(citiesField);
            if (select == 0) {
                lf.setText(_resources.getString(FIELD_US));
                popField.setText(_resources.getString(FIELD_US_POP));
                langField.setText(_resources.getString(FIELD_US_LANG));

    citiesField.setText(_resources.getString(FIELD_US_CITIES));
                }
            else if (select == 1) {
                lf.setText(_resources.getString(FIELD_CHINA));
                popField.setText(_resources.getString(FIELD_CHINA_POP));

    langField.setText(_resources.getString(FIELD_CHINA_LANG));

    citiesField.setText(_resources.getString(FIELD_CHINA_CITIES));
                }
            else if (select == 2) {
                lf.setText(_resources.getString(FIELD_GERMANY));

    popField.setText(_resources.getString(FIELD_GERMANY_POP));
```

```
        langField.setText(_resources.getString(FIELD_GERMANY_LANG));

        citiesField.setText(_resources.getString(FIELD_GERMANY_CITIES));
                }
            }
        }
    }
```

为应用程序组（suite）管理资源文件

你可以将资源加入到你的应用程序项目中，或者如果你正在创建一组应用程序，对每个 locale 将资源组织为单独的工程。

创建资源项目

为每个资源包（locale）创建一个项目，包含 root locale。
为每个 locale 给项目一个和 root locale 项目相同的名称，紧跟随一个双下划线（__）,语言编码，以及一个可选的在国家编码后的下划线。例如，，如果 root locale 项目命名为 com_company_app, 那么每个 locale 的项目可以明明为 com_company_app__en, com_company_app__en_GB, com_company_app__fr.

指定输出文件名

1. 右击项目，点击 **Properties**。
2. 点击 **Build** 标签。
3. 在 Output file name 域，为编译文件输入一个没有文件扩展名的名字。

> **注**：所有资源 locale 项目的输出文件名，必须和 root locale 一样，跟随一个双下划线，合适的语言以及国家编码。例如，如果 root locale 的输出名为 com_company_app，法语 locale 的输出文件名必须为 com_company_app__fr.

创建初始化文件

当在一个独立项目中编译资源时，创建一个初始化文件（例如，init.java）.BlackBerry IDE 提供一个内置的初始化机制，一时你仅需要创建一个空 main()函数的空初始化类。

```
package com.rim.samples.device.resource;
import net.rim.device.api.i18n.*;
public class init {
public static void main (String[] args) { }
}
```

增加文件到合适的资源项目中

为每个应用程序创建一个头文件，为每个应用程序，每个支持的 locale 创建一个资源文件。组织资源文件到项目中。

1. 为增加资源头文件到每个应用程序的项目中以及每个资源项目中。定义应用程序项目以及它他的资源项目之间的依赖性是有必要的。
2. 在每资源项目中，右击每个.rrh 文件，然后点击 **Properties**。
3. 选择 **Dependency Only，Do not build**。
4. 增加资源内容文件（.rrc）到合适 locale 的项目中。

注：如果你支持大量的 locale，为所有资源头文件(.rrh)创建一个单库,并且设置项目类型为 Library.对于每个项目中资源 locale,定义项目之间的依赖。

9

第 9 章 IT 策略（Policy）

IT 策略

BlackBerry IT 策略 API（net.rim.device.api.itpolicy）允许应用程序访问 BlackBerry 设备上的 IT 策略数据库。应用程序可以获取定制的 IT 策略设置相应的改变他们的行为以及功能。


注：管理员使用应用程序控制控制 BlackBerry 设备上的第三方应用程序的出现以及功能。为获取更多关于应用程序控制的信息，参看 *BES 手持设备管理指南*

每个 IT 策略项由一个描述性的键以及一个值组成。这个值可以为字符串，整型或者 Boolean 型。例如，AllowPhone 策略可以设置一个 true 或 false 的值。
在 Microsoft Exchange 的 BES 3.5 以及后续版本中，手持设备策略设置会无线同步与更新。在早期的手持设备软件的版本中，当用户把 BlackBerry 设备和桌面同步时，手持设备策略设置会得到更新。
参看 *Microsoft Exchange BES 手持设备管理指南*获得更多信息。

获取客户策略


注：IT 策略 API 仅允许应用程序为客户（第三方）IT 策略项获取值。应用程序不能获取标准 TT 策略项的值。

为根据名称获取一个第三方 IT 策略项的值，使用每个接收一个 String 参数的方法。

```
public static String getString( String name );
public static boolean getBoolean( String name, boolean defaultValue );
public static int getInteger( String name, int defaultValue );
```

参数 defaultValue 指定了如果参数没有设置时的返回值。

监听策略的改变

当 BlackBerry 设备上 IT 策略数据库得到更新时，一个全局事件会触发。
为使用 IT 策略，应用程序实现了 GlobalEventListener 接口。注册你的实现来接收全局事件。

当一个全局事件，例如一个 IT 策略的改变，发生时，GlobalEventListener.eventOccurred()将会被调用。在 eventOccurred()的实现里，获取 IT 策略项的值来决定值是否已经改变。

代码实例

ITPolicyDemo.java 实例实现了 IT 策略控制。

例：ITPolicyDemo.java

```
/**
* ITPolicyDemo.java
* Copyright (C) 2002-2005 Research In Motion Limited.
*/
package com.rim.samples.docs.itpolicy;
import net.rim.device.api.system.*;
import net.rim.device.api.itpolicy.*;
public class ITPolicyDemo extends Application implements
GlobalEventListener {
    public static void main(String[] args) {
        ITPolicyDemo app = new ITPolicyDemo();
        app.enterEventDispatcher();
        }

    ITPolicyDemo() {
        this.addGlobalEventListener(this);
        boolean appEnabled = ITPolicy.getBoolean("DemoAppEnabled",
true);
        System.out.println("App Enabled:" + appEnabled);
        System.exit(0);
        }
    public void eventOccurred(long guid, int data0, int data1, Object
obj0, Object obj1)
    {
        if (guid == ITPolicy.GUID_IT_POLICY_CHANGED )
        {
            String security = ITPolicy.getString("DemoSecurityLevel");
            boolean appEnabled = ITPolicy.getBoolean("DemoAppEnabled",
true);
            int retries = ITPolicy.getInteger("DemoAppRetries", 10);
        }
    }
}
```

10

第 10 章 创建 Client/Server Push 应用程序

Push 应用程序

注：Push 应用程序需要 3.5 以及后续版本的 Microsoft Exchange BES，或 2.0 以及后续版本的 IBM Lotus Domino BES，它们需启用 BlackBerry MDS 服务。

Push 应用程序将新的 web 内容和 alert 发送到指定的用户。用户不必请求下载数据，因为当信息可用时 push 应用程序递送这个信息。

有 2 种 push 应用程序：

1. **浏览器 push 应用程序**：将 Web 内容发送到 BlackBerry 设备上。BlackBerry 浏览器配置支持 MDS 服务 push 应用程序。WAP 浏览器配置支持 WAP push 应用程序。Internet 浏览器配置不支持 push 应用程序。参看 *BlackBerry 浏览器开发指南*获取更多关于编写一个浏览器 push 应用程序的信息。
2. **Client/Server push 应用程序**：将数据 push 到一个 BlackBerry 设备上的客户 Java 应用程序。Client/Server push 应用程序由一个 BlackBerry 设备上的客户 Client 应用程序和一个 push 内容给它的服务器端应用程序组成。和浏览器 push 应用程序相比，这种方法对这种你可以发送出去的内容以及数据是如何处理并且显示在 BlackBerry 设备上提供了更多的控制。

Client/Server push 请求

应用程序可以使用下面的 2 种方法将内容 push 到 BlackBerry 设备：

1. Push Access Protocol（PAP，push 访问协议），它是 WAP 2.0 里的一部分。
2. RIM push。

注：MDS 服务仅支持 1000 个 push 请求，包括了 RIM 和 PAP push 请求。如果它接收超过 1000 个请求，MDS 服务回应服务器一个错误。

这 2 种 push 服务的实现都支持下面的任务：

- 发送一个服务器端 push 提交（submission）。
- 为 push 提交指定一个信任的模式。
- 为 push 提交指定一个传递前（deliver-before）的时间戳。
- 请求一个 push 提交的结果通知。

PAP 的实现还支持下面额外的任务：

- 为 push 提交指定一个传递后（deliver-after）时间戳。
- 取消一个 push 请求提交。
- 查询一个 push 请求提交的状态。

存储 push

PAP push 存储在数据库中，但是 RIM push 则存储在 RAM 中。如果服务器重启，没有递送的 RIM push 可能会丢失。

代码转换（Transcoding）

如果可用，BlackBerry MDS Data Optimization（BlackBerry MDS 数据优化）根据它的代码转换器规则将一个代码转换应用到 push 请求。

Push 请求可以覆写这些规则并使用 transfer-encoding 头来请求一个指定的代码转换器转化。例如，如果设置了 HTTP 头 transfer-encoding：vnd.wap.wml，那么在它把数据 push 到 BlackBerry 设备之前，MDS 数据优化服务运行 wml 代码转换器。

参看 190 页的“代码转换器”获得更多信息。

信任模式

信任模式	描述
透明层信任模式	当 push 到达 BlackBerry 设备时，BlackBerry MDS 连接服务启动一个 push 请求指定的 URL 连接来通知传送的服务器。手持设备软件 3.6 或以后版本支持透明层确认。
应用程序级信任模式	当 push 到达 BlackBerry 设备时，应用程序确认内容。MDS 连接服务启动一个 push 请求时指定的 URL 连接来通知传送的服务器。如果遇到错误，MDN 连接服务发送一个错误消息到服务器。手持设备软件 4.0 或以后版本支持应用程序级确认 RIM push 提供一个优先应用程序（Application-preferred）的选项,在手持设备软件 4.0 或以后版本中，它使用应用程序级确认，否则就是透明层确认。 注:3.8 以及更早版本的 MDS 服务不可以为 BlackBerry 设备特征查询 BES。为了得到 Blac

	kBerry 设备的特征，MDS 连接服务必须在它接收到一个 push 请求之前接收到一个 HTTP 请求。 为了提供必要的请求，使用 BlackBerry 浏览器和一个 MDS 服务浏览器配置浏览一个 web 页面。

发送一个 RIM push 请求

为了使用 RIM push 将数据 push 到 BlackBerry 设备上，使用下面的格式发送一个 HTTP POST 请求，在这里***<destination>***是目的 PIN 或者 internet 消息地址，***<port>***是目的端口，***<uri>***是发送到 BlackBerry 的 URI，***<content>***是字节流：

/push?DESTINATION=<destination>&PORT=<port>&REQUESTURI=<uri><headers><content>

对于 RIM push 请求，下列的头是有效的：

HTTP 头	描述
X-RIM-Push-ID	这个头指定了一个唯一的消息 ID，它可以用来取消或检查消息的状态。典型地，以一个值组合指定 URL，例如 123@blackberry.com. 如果忽略这个头，MDS 服务生成一个唯一的消息 ID. 注:Push 标识符不得以@ppg.rim.com 结束。
X-RIM-Push-NotifyURL	这个头指定一个 URL 来发送一个结果通知。这个结果通知包含了指定的消息 ID 的 X-RIM-Push-ID 头，以及指定 HTTP 响应代码的 X-RIM-Push-Status 头。
X-RIM-Push-Reliability-Mode	这个头指定了内容透明级(TRANSPORT),应用程序级(APPLICATION)或优先应用程序（APPLICATION-PREFERRED）的传送信任模式。
X-RIM-Push-Deliver-Before	这个头指定将内容传送给 BlackBerry 设备的日期和时间。在这个时间之前没有传送的内容不会被传送。
X-RIM-Push-Priority	这个头指定了频道 push 消息的优先级。允许的字符串有 none(缺省)，low,medium，以及 high.如果优先级为 low,medium 或 high,用户接收频道更新的通知。如果优先级为 high，一个状态对话框会伴随着通知。

发送一个 PAP push 请求

为了使用 PAP 将数据 push 到 BlackBerry 设备，使用下面的格式发送一个 HTTP POST 请求:

/pap

这个请求是一个 MIME 多部分（multipart）的消息，它由下面的项组成：

- 一个指定控制实体（entity）XML 文档。
- push 内容。

例如，控制实体可能包含 BlackBerry 设备地址，消息 ID，以及传送时间戳信息。

使用 PAP DTD(Document Type Definition)指定下面的属性：

XML 控制实体属性	描述	实例
X-Wap-Application-Id	这个实体属性指定了 RIM push 的 REQUEST URI HTTP 同等体。	"/"
push-id	指定唯一的消息 ID.另外，这个控制实体属性可以用来取消或检查消息的状态。建议你在一个值的组合里使用一个 URL，例如，123@BlackBerry.com.	123@wapforum.org
ppg-notify-requested-to	指定发送结果通知的 URL.	http://wapforum:8080/ ReceivePAPNotification.
deliver-before-timestamp	指定日期和时间，通过它将内容传送到 BlackBerry 设备。这个时间之前的没有发送的内容将会丢失。 以 UTC 格式显示时间: YYYY-MM-DDThh:mm:ssZ 在这里： ● YYYY 是 4 个数字的年份。 ● MM 是 2 个数字的月份。 ● DD 是 2 个数字的日期。 ● hh 是 2 个数字的 24 小时制式的小时。 ● mm 是 2 个数字的分 ● ss 是 2 个数字的秒 ● Z 描述了时间是 UTC 格式。	2004-01-20T22:35:00Z
deliver-after-timestamp	指定日期和时间，在这个时间之后的内容发送到 BlackBerry 设备。这个时间前的内容不会被传送。 以 UTC 格式显示日期与时间.	2004-01-20T21:35:00Z

address-value	指定将 PUSH 内容发送到 BlackBerry 设备的地址。*destination* 是目的 internet 消息地址或 PIN.	WAPPUSH=*destionation%3Aport1*/TYPE=USER@blackberry.com
Delivery-method	指定内容，透明级，或应用程序级的传送信任模式	Confirmed;uncomfirmed

参看 *Push Access Protocol(WAP-247-PAP-20010429-a)*文档获取更多关于使用 PAP 编写服务器端 push 用用程序。参看 PAP 2.0 DTD 得到关于 WAP Push DTD 的信息。

例: PAP push request

```
Content-Type: multipart/related; type="application/xml";
boundary=asdlfkjiurwghasf
X-Wap-Application-Id: /
--asdlfkjiurwghasf
Content-Type: application/xml
<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"
"http://www.wapforum.org/DTD/pap_2.0.dtd">
<pap>
  <push-message push-id="a_push_id" ppg-notify-requested-
to="http://foo.rim.net/ReceiveNotify">
    <address address-
value="WAPPUSH=aisha.wahl%40blackberry.com%3A7874/TYPE=USER@rim.net"/
>
    <quality-of-service delivery-method="unconfirmed"/>
  </push-message>
</pap>
--asdlfkjiurwghasf
Content-Type: text/html
<html><body>Hello, PAP world!</body></html>
--asdlfkjiurwghasf--
```

发送 PAP push 取消请求

使用下面的头取消一个已经发送到 MDS 服务的 push 提交。

头	描述	实例
cancel-message push-id	取消前面提交的 push 消息。	<cancel-message push-id=123@wapforum.org>
address address-value	指定 push 消息提交的地址。这个标记是	<address address-value=WAPPUSH=aisha.wah1%40bl1ackberry.com%3A7874/TYPE=UER@rim.net/>

	必需的。	

例：PAP push 取消请求

```
Content-Type: application/xml
<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"
"http://www.wapforum.org/DTD/pap_2.0.dtd">
<pap>
<cancel-message push-id="a_push_id">
<address address-value=
 "WAPPUSH=aisha.wahl%40blackberry.com%3A7874/TYPE=USER@rim.net "/>
</cancel-message>
</pap>
```

发送一个 PAP push 查询请求

为了查询已经发送到 MDS 服务的 push 提交的状态，使用下面的头：

XML 控制实体属性	描述	实例
statusquery-message push-id	指定 push 消息的哪个状态是需要的。 返回的响应是下面消息状态之一：delivered,pending,undelivered,expired,rejected,timeout,cancelled,aborted 或 unknown. 你必须在请求里包含地址属性。	<statusquery-message push-id=123@wapforum.org>
address ddress-vue	指定一个提交的 push 消息地址。这个标记是必需的。	<address address-value="WAPPUSH=aisha.wahl%40blackberry.com%3A7874/TYPE=USER@rim>net"/>

```
Content-Type: application/xml
<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"
"http://www.wapforum.org/DTD/pap_2.0.dtd">
<pap>
  <statusquery-message push-id="a_push_id">
     <address address-
value="WAPPUSH=aisha.wahl%40blackberry.com%3A7874/TYPE=USER@rim.net"/
>
  </statusquery-message>
</pap>
```

决定 BlackBerry 设备是否在 push 覆盖范围内

为了从 MDS 服务接收某个特定 BlackBerry 设备的网络覆盖信息，指定下面的头：

XML 控制实体属性	描述	实例
x-rim-use-coverage	设置 true 来接收网络覆盖信息，push 消息的状态是必要的。 **注**：这个头典型用来决定网络覆盖，优先于发送一个 push 请求。	<rim-push-use-coverage=”true”>
address address-value	指定 BlackBerry 设备的地址来决定网络覆盖。	<address address-value=”WAPPUSH=aisha.wahl%40blackberry.com %3A7874/TYPE=USER@rim>net”/>

例:RIM 网络状态查询请求

```
Content-Type: application/xml
<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"
"http://www.wapforum.org/DTD/pap_2.0.dtd">
<pap>
   <rim-push-use-coverage="true" push-id="a_push_id" \>
</pap>
```

例:RIM 网络状态查询响应

```
Content-Type: application/xml
<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"
"http://www.wapforum.org/DTD/pap_2.0.dtd">
<pap>
<x-rim-device-state="true">
   <!-- a response of true means the device is in network coverage --
>
    <address address-
value="WAPPUSH=aisha.wahl%40blackberry.com%3A7874/TYPE=USER@rim.net
"/>
</rim-push-use-coverage>
</pap>
```

编写一个客户端 push 应用程序

创建一个监听线程

在独立线程上发送和接收数据，这样你不会在主事件线程上阻塞。

打开一个输入连接

调用 Connector.open(String),并让 http://作为协议，选择大端口号。将返回的对象转化为一个 StreamConnectionNotifier。

打开连接一次，并保持其连接状态为打开。仅当 IOException 发生时，重新打开连接。每次 push 的数据接收到时，不要关闭和重新打开连接，因为在前一个 push 后，如果再调用 Connector.open()之前数据就到达了，push 的数据会丢失。

为避免和其他程序冲突，选择一个大的端口号，端口号必须是 1 到 65535。端口 7874 为 BlackBerry 浏览器保留。

```
StreamConnectionNotifier _notify =
    (StreamConnectionNotifier)Connector.open("http://:6234");
// open a server-side socket connection
StreamConnection stream = _notify.acceptAndOpen();
// open an input stream for the connection
InputStream input = stream.openInputStream();
```

关闭流连接通知

调用流连接通知上的 close()方法.

```
_notify.close();
```

代码实例

HTTPPushDemo.java 实例描述了如何编写一个应用程序监听来自服务器的进入数据。创建一个监听线程监听指定端口的图像数据，当数据到达时，然后显示它。

例: HTTPPushDemo.java

```
/**
* HTTPPushDemo.java
* Copyright (C) 2001-2005 Research In Motion Limited. All rights
reserved.
*/
package com.rim.samples.docs.httppush;
```

```
import java.io.*;
import javax.microedition.io.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;
import com.rim.samples.docs.baseapp.*;

public class HTTPPushDemo extends BaseApp {
    // Constants.
    private static final String URL = "http://:6234";
    private static final int CHUNK_SIZE = 256;

    // Fields.
    private ListeningThread _listeningThread;
    private MainScreen _mainScreen;
    private RichTextField _infoField;
    private BitmapField _imageField;
    public static void main(String[] args) {
        HTTPPushDemo theApp = new HTTPPushDemo();
        theApp.enterEventDispatcher();
        }

    /**
     * * Create a separate listening thread so that you do not
     * * block the application's main event thread.
     * */
    private class ListeningThread extends Thread {
        private boolean _stop = false;
        private StreamConnectionNotifier _notify;
        public synchronized void stop() {
            _stop = true;
            try {
                _notify.close(); // Close the connection so thread
returns.
                }
            catch (IOException e) {
                System.err.println(e.toString());
                }
            catch (NullPointerException e) {
                // The notify object likely failed to open, due to an
IOException.
```

```
            }
        }

    public void run() {
        StreamConnection stream = null;
        InputStream input = null;
        try {
            synchronized(this)
            {
                // Open the connection once or re-open after an
IOException.
                _notify =
(StreamConnectionNotifier)Connector.open(URL);
            }
            while (!_stop) {
                // NOTE: This method blocks until data is received.
                stream = _notify.acceptAndOpen();
                input = stream.openInputStream();
                // Extract the data from the input stream.
                DataBuffer db = new DataBuffer();
                byte[] data = new byte[CHUNK_SIZE];
                int chunk = 0;
                while ( -1 != (chunk = input.read(data)) ) {
                    db.write(data, 0, chunk);
                    }
                input.close();
                data = db.getArray();
                updateBitmap(data);
                }
            }
        catch (IOException e) {
            System.err.println(e.toString()); // It is likely the
stream was closed.
            }
        }
    }

    // Constructor.
    public HTTPPushDemo() {
        _mainScreen = new MainScreen();
        _mainScreen.setTitle(new LabelField("Latest Logos",
LabelField.USE_ALL_WIDTH));
        _infoField = new RichTextField();
        _mainScreen.add(_infoField);
```

```
        _mainScreen.add(new SeparatorField());
        _imageField = new BitmapField(null,
BitmapField.HCENTER|BitmapField.TOP);
        _mainScreen.add(_imageField);
        _mainScreen.addKeyListener(this);
        _mainScreen.addTrackwheelListener(this);
        _listeningThread = new ListeningThread();
        _listeningThread.start();
        _infoField.setText("Application is listening...");
        pushScreen(_mainScreen);
        }

    private void updateBitmap(final byte[] data) {
        Application.getApplication().invokeLater(new Runnable() {
            public void run() {
                // Query the user to load the received image.
                String[] choices = {"OK", "CANCEL"};
                if ( 0 != Dialog.ask("Do you want to display latest
logo?",choices, 0) )
                {
                    return;
                }
                _infoField.setText("Image received. Size:"+
data.length);
                _imageField.setBitmap(Bitmap.createBitmapFromPNG(data,
0,data.length));
                }
            });
        }

    protected void onExit() {
        // Stop the listening thread.
        _listeningThread.stop();
        try {
            _listeningThread.join();
            }
        catch (InterruptedException e) {
            System.err.println(e.toString());
            }
        }
    }
```

编写一个服务器端 push 应用程序

任何能够建立 HTTP 连接的编程语言都可以用来创建一个 push 应用程序。下面章节使用标准 Java 来描述一个服务器端的 push 应用程序。

构造 push URL

像下面格式化 RIM push 请求:

/push?DESTINATION=<destination>&PORT=<port>&REQUESTURI=<uri><headers><content>

参看 139 页的“发送一个 RIM push 请求”获得更多关于构造 RIM push 请求的 URL.

如下构造 PAP push 请求:

/pap

参看 139 页的“发送一个 RIM push 请求”获得更多关于构造 PAP push 请求的 URL.

连接 BES

在 push URL 上调用 openConnection()，然后将返回的对象转化为一个 HttpURLConnection. 一个 HttpURLConnection 代表了一个远程对象的连接。

```
HttpURLConnection conn =(HttpURLConnection)url.openConnection();
```

为 HTTP POST 请求设置属性

服务器端 push 应用程序使用一个 POST 请求方法。

```
conn.setRequestMethod("POST"); // Post to the BlackBerry Enterprise Server.
```

为了接收确认，设置 doInput(Boolean)为 true，这描述了应用程序打算从 URL 连接读取数据 。

conn.setDoInput(true).

为了发送数据，设置 doOutput(Boolean)为 true，这描述了应用程序打算发送数据到 URL 连接。

conn.setDoOutput(true).

写数据到服务器连接

调用 getOutputStream()来访问一个输出流。写数据到输出流上，然后关闭它。

```
OutputStream out = conn.getOutputStream();
out.write(data);
out.close();
```

读取服务器响应

调用 getInputStream()来访问一个输入流。决定内容的大小，如果它是非零，打开一个数据输入流，然后读取内容。

```
InputStream ins = conn.getInputStream();
int contentLength = conn.getContentLength();
if (contentLength > 0) {
    byte[] someArray = new byte [contentLength];
    DataInputStream dins = new DataInputStream(ins);
    dins.readFully(someArray);
    System.out.println(new String(someArray));
}
ins.close();
```

断开连接

调用 disconnect()，它描述了应用程序对服务器不会再有请求。
conn.disconnect().

代码实例

HTTPPush.java 实例，它由标准 Java 编写，发送单个.png 图像到一个 BlackBerry 设备上的监听客户端程序。应用程序 push 基于一个 internet 消息地址的数据。为了用模拟器测试 push 应用程序，定义一个 internet 消息地址和模拟器 PIN（2100000A）之间的映射。
下面的代码使用 J2SE 1..4.2 编译:

例: HTTPPush.java

```
/*
* HttpPushServer.java
* Copyright (C) 2001-2004 Research In Motion Limited. All rights
reserved.
*/
package com.rim.docs.samples.httppush;
import java.io.*;
import java.net.*;
import java.util.*;

public class HTTPPushServer {
    //constants
    private static final String HANDHELD_EMAIL =
"scott.tooke@rim.com";
    private static final String HANDHELD_PORT = "6234";
```

```
    private static final String BES_HOST = "localhost";
    private static final int BES_PORT = 8080;
    private static final String CONTENT =
"com/rim/docs/samples/httppush/logo.png";

    //constructor
    public HTTPPushServer() {

    }

    private static URL getPushURL(String HandheldEmail) {
        URL _pushURL = null;
        try {
            if ((HandheldEmail == null) || (HandheldEmail.length() ==
0)) {
                HandheldEmail = HANDHELD_EMAIL;
                }
            _pushURL = new URL("http", BES_HOST, BES_PORT,
                    "/push?DESTINATION="+ HandheldEmail+"&PORT="
                    +HANDHELD_PORT+"&REQUESTURI=/");
            }
        catch (MalformedURLException e) {
            System.err.println(e.toString());
            }
        return _pushURL;
        }

    public static void postData(byte[] data) {
        try {
            URL url = getPushURL(HANDHELD_EMAIL);
            System.out.println("Sending to" + url.toString());
            HttpURLConnection conn =
(HttpURLConnection)url.openConnection();
            conn.setDoInput(true); //for receiving the confirmation
            conn.setDoOutput(true); //for sending the data
            conn.setRequestMethod("POST"); //post the data to the BES
            OutputStream out = conn.getOutputStream();
            out.write(data); //write the data
            out.close();
            InputStream ins = conn.getInputStream();
            int contentLength = conn.getContentLength();
            System.out.println("Content length:"+ contentLength);
            if (contentLength > 0) {
                byte[] someArray = new byte [contentLength];
```

```
            DataInputStream dins = new DataInputStream(ins);
            dins.readFully(someArray);
            System.out.println(new String(someArray));
            }
        ins.close();
        conn.disconnect();
        }
    catch (IOException e) {
        System.err.println(e);
        }
    }
public static void main (String args[]) {
    try {
        File f = new File(CONTENT);
        if ( f == null ) {
            throw new RuntimeException("Unable to Open File");
            }
        FileInputStream fi = new FileInputStream(f);
        if ( null == fi ) {
            throw new RuntimeException("Unable to open file");
            }
        int size = fi.available();
        byte[] imageData = new byte[size];
        int bytesRead = fi.read(imageData);
        fi.close();
        postData(imageData);
        }
    catch (IOException e) {
        System.err.println(e.toString());
        }
    }
}
```

Push 应用程序疑难解答

注：下面的情况适用于 Microsoft Exchange BES.

Push 应用程序对基于 internet 消息地址的 BlackBerry 设备进行标识，当用户换到另外一台不同的 BlackBerry 设备时，如果用户停止接收一个 push 应用程序的数据，
可能意味着用户 internet 消息地址和 BlackBerry 设备 PIN 之间的映射已经过期了。验证 BES 是否正确运行。

Microsoft Exchange BES 的 MDS 服务特征使用了一个数据库一致工具（Database Consistency tool）,dbconsistency.exe，进行维护 internet 消息地址和 BlackBerry 设备 PIN 之间的映射。管

理员可以配置这个工具运行的频率，并且也可以手动运行它。为获得更多信息，参看 *Microsoft Exchange BES 维护指南*。

11

第 11 章 使用位置信息

位置 API

位置 API(javax.microedition.location)允许应用程序获取 BlackBerry 设备的全球定位系统(GPS)位置。GPS 位置是 BlackBerry 设备中的地理坐标（经度和纬度）。根据使用的位置方法，应用程序也可以获得 BlackBerry 设备的速度，方向，以及路线。

获得 GPS 位置的方法

方法	常数	描述
Cellsite	GPS_AID_MODE_CELLSITE	本方法使用活动的蜂窝（cellsite） 塔的 GPS 位置来提供第一个有序 GPS 信息。它提供最不精确的位置信息；虽说如此，但它是最快的位置模式。 **注**：如果使用本模式，将得不到方向，路径，以及速度获。这个位置方法需要网络连接以及运营商的支持。
Assisted	GPS_AID_MODE_ASSIST	本方法使用网络为设备的芯片提供天文卫星数据。它比自动模式提供 GPS 位置更快些，并且比蜂窝模式更精确。 **注**：本位置方法需要网络连接以及运营商的支持。
Autonomous	GPS_AID_MODE_AUTONOMOUS	本方法在没有网络的协助下使用 BlackBerry 上的 GPS 芯片。自动模式提供第一个 GPS 位置最慢。

为选择 GPS 位置方法指定原则

通过创建一个 javax.microedition.location.Criteria 类，调用合适的 set 方法，然后传递这个 LocationProvider.getInstance()实例来指定需要的原则.

注：为创建一个缺省规则的 LocationProvider 实例，调用 LocationProvider.getInstance(null).

```
Criteria criteria = new Criteria();
// Allow cost.
criteria.setCostAllowed(true);
// Require a horizontal accuracy of 50 metres.
criteria.setHorizontalAccuracy(50);
// Require a vertical accuracy of 50 metres.
criteria.setVerticalAccuracy(50);
LocationProvider provider = LocationProvider.getInstance(criteria);
```

选择 GPS 位置方法的原则

建议的 GPS 位置方法	水平精确度	垂直精确度	费用	耗电量
自动	需要	需要	不允许	不可用
自动	需要	需要	允许	低，一般或没有需求
第一次修正：协助 后续修正：自动	需要	需要	允许	高
自动	不需要	不需要	不允许	一般，高或没有需求
协助	不需要	不需要	允许	一般，或没有需求
第一次修正：协助 后续修正：自动	不需要	不需要	允许	高
蜂窝	不需要	不需要	允许	低

注：如果 BlackBerry 无线设备有一个卫星的障碍视图，GPS 可能不可用。当设备在室内或被建筑物，树，以及密云围绕，这是有可能发生的。

获取 BlackBerry 设备的位置

第一次获取 BlackBerry 设备位置所花的时间依赖多方面的因素，例如选择的 GPS 模式，GPS 的信号强度。在自动模式中，典型的至少需要 2 分钟，在协助模式。典型的是至少需要 30 秒。
如果 GPS 修正在 10 秒的请求内发生后，连续请求的平均响应时间为 1 到 2 秒，这取决位置条件。

注：如果你使用了一个可能花费用户资费的位置方法，不要经常查询 BlackBerry 设备的位置。

为了指定一个需要的响应事件，调用 Criteaia.setPreferredReponseTime（），以毫秒为单位提供需要的时间。

获取 BlackBerry 设备的位置

调用 LocationProvider.getLocation(int),提供一个以毫秒为单位的超时时间。

注：如果 LocationProvider.getLocation(int)早事件线程调用，LocationExcept 会抛出。

```
try {
    // Specify -1 to have the implementation use its default timeout
value
    // for this provider.
    Location location = provider.getLocation(-1);
}
catch (Exception e) {
    // handle LocationException, InterruptedException,
SecurityException
    // and IllegalArgumentException
}
```

获取位置信息

Location 类提供方法来获取位置信息，例如 GPS 坐标，以及路径。

注：RIM 的实现不支持文本地址信息。结果，getAddressInfo（）方法会空。

```
QualifiedCoordinates coordinates = location.getQualifiedCoordinates;
float speed = location.getSpeed();
float course = location.getCourse();
```

注册一个位置监听者

实现 LocationListener 方法。调用 LocationProvider.setLocationListener()注册你的实现。

注：一个位置监听者可以与一个指定的位置提供者项关联。应用程序在一个独立的线程上典型用来监听更新。

```
import javax.microedition.LocationProvider.*;

public class SampleLocationApp {
    public static void main (string[] Args) {
        // ...
        provider.setLocationListener(new SampleLocationListener(), 0,
60, 60);
        }
    }

class SampleLocationListener implements LocationListener {
    void locationUpdated(LocationProvider provider, Location location)
    {
```

```
    // Respond to the updated location.
    // If the application registered the location listener with an
interval of
    // 0, the location provider does not provide location updates.
    }

    void providerStateChanged(LocationProvider provider, int newState)
 {
        switch (newState) {
          case LocationProvider.AVAILABLE :
              // The location provider is available.
              break;
          case LocationProvider.OUT_OF_SERVICE :
              // The location provider is permanently unavailable.
              // Consider cancelling the location listener by calling
              // provider.setLocationListener() with null as the
listener.
              break;
         case LocationProvider.TEMPORARILY_UNAVAILABLE :
             // The location provider is temporarily unavailable.
             break;
             }
        }
}
```

代码实例

例：GPSDemo.java

```
/**
* A GPS sample application using the JSR 179 APIs.
*
* Copyright (C) 2005 Research In Motion Limited.
*/
package com.rim.samples.docs.gpsdemo;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import com.rim.samples.docs.baseapp.*;
import net.rim.device.api.io.*;
import net.rim.device.api.system.*;
import net.rim.device.api.i18n.*;
import javax.microedition.io.*;
```

```
import java.util.*;
import java.io.*;
import javax.microedition.location.*;
import net.rim.device.api.util.*;
import com.rim.samples.docs.resource.*;

/* This application acts as a simple travel computer, recording route
coordinates,
* speed, and altitude.
* Recording begins as soon as the application is invoked.
*/
public class GPSDemo extends BaseApp implements GPSDemoResResource
{
    // Constants. ------------------------------------------------------
-------------
    // The number of updates in seconds over which the altitude is
calculated.
    private static final int GRADE_INTERVAL=5;
    // com.rim.samples.docs.gpsdemo.GPSDemo.ID
    private static final long ID = 0x4e94d9bc9c54fed3L;
    private static final int CAPTURE_INTERVAL=10;
    // Statics. --------------------------------------------------------
-------------
    private static ResourceBundle _resources =
ResourceBundle.getBundle(GPSDemoResResource.BUNDLE_ID,
GPSDemoResResource.BUNDLE_NAME);
    // The period of the position query in seconds.
    private static int _interval = 1;
    private static Vector _previousPoints;
    private static float[] _altitudes;
    private static float[] _horizontalDistances;
    private static PersistentObject _store;
    // Initialize or reload the persistent store.
    static
    {
        _store = PersistentStore.getPersistentObject(ID);
        if(_store.getContents()==null)
        {
            _previousPoints= new Vector();
            _store.setContents(_previousPoints);
        }
        _previousPoints=(Vector)_store.getContents();
    }
```

```
    private long _startTime;
    private float _wayHorizontalDistance;
    private float _horizontalDistance;
    private float _verticalDistance;
    private ListField _listField;
    private EditField _status;
    private StringBuffer _messageString;
    private String _oldmessageString;
    private LocationProvider _locationProvider;
    /* Instantiate the new application object and enter the event
loop.
     * @param args unsupported. no args are supported for this
application
     */
    public static void main(String[] args)
    {
        new GPSDemo().enterEventDispatcher();
    }
    // Constructors. ------------------------------------------------
-------------
    public GPSDemo()
    {
        // Used by waypoints; represents the time since the last
waypoint.
        _startTime = System.currentTimeMillis();
        _altitudes=new float[GRADE_INTERVAL];
        _horizontalDistances=new float[GRADE_INTERVAL];
        _messageString= new StringBuffer();
        MainScreen screen = new MainScreen();
        screen.setTitle(new
LabelField(_resources.getString(GPSDEMO_TITLE),
LabelField.USE_ALL_WIDTH));
        _status = new EditField();
        screen.add(_status);
        screen.addKeyListener(this);
        screen.addTrackwheelListener(this);
        // Start the GPS thread that listens for updates.
        startLocationUpdate();
        // Render our screen.
        pushScreen(screen);
    }

    /* Update the GUI with the data just received.
     */
```

```
    private void updateLocationScreen(final String msg)
    {
        invokeLater(new Runnable()
        {
            public void run()
            {
                _status.setText(msg);
            }
        });
    }

    // Menu items. ------------------------------------------------
-------------
    // Cache the markwaypoint menu item for reuse.
    private MenuItem _markWayPoint = new MenuItem(_resources,
GPSDEMO_MENUITEM_MARKWAYPOINT, 110, 10)
    {
        public void run()
        {
            GPSDemo.this.markPoint();
        }
    };
    // Cache the view waypoints menu item for reuse.
    private MenuItem _viewWayPoints = new MenuItem(_resources,
GPSDEMO_MENUITEM_VIEWWAYPOINTS, 110, 10)
    {
        public void run()
        {
            GPSDemo.this.viewPreviousPoints();
        }
    };



    // Cache the close menu item for reuse.
    private MenuItem _close = new MenuItem(_resources,
GPSDEMO_MENUITEM_CLOSE, 110, 10)
    {
        public void run()
        {
            System.exit(0);
        }
    };
    protected void makeMenu(Menu menu, int instance)
```

```
    {
        menu.add( _markWayPoint );
        menu.add( _viewWayPoints );
        menu.add( _close );
        menu.addSeparator();
        super.makeMenu(menu, instance);
    }


    /* Invokes the Location API with the default criteria.
     */
    private void startLocationUpdate()
    {
        try
        {
            _locationProvider = LocationProvider.getInstance(null);
            if ( _locationProvider == null )
            {
                Dialog.alert("GPS is not supported on this platform,
exiting...");
                System.exit(0);
            }
            // A single listener can be associated with a provider,
            // and unsetting it involves the same call but with null,
            // so there is no need to cache the listener instance.
            // Request an update every second.
            _locationProvider.setLocationListener(new
LocationListenerImpl(), _interval, 1, 1);
        }
        catch (LocationException le)
        {
            System.err.println("Failed to add a location listener.
Exiting...");
            System.err.println(le);
            System.exit(0);
        }
    }

    /* Marks a point in the persistent store. Calculations are based
on
     * all data collected since the previous way point, or from the
start
     * of the application if no previous waypoints exist.
     */
```

```
    private void markPoint()
    {
        long current = System.currentTimeMillis();
        WayPoint p= new WayPoint(_startTime, current,
_wayHorizontalDistance, _verticalDistance);
        addWayPoint(p);
        // Reset the waypoint variables.
        _startTime = current;
        _wayHorizontalDistance = 0;
        _verticalDistance = 0;
    }

    // View the saved waypoints.
    private void viewPreviousPoints()
    {
        PointScreen pointScreen = new PointScreen(_previousPoints,
_resources);
        pushScreen(pointScreen);
    }

    // Called by the framework when this application is losing focus.
    protected void onExit()
    {
        if ( _locationProvider != null )
        {
            _locationProvider.reset();
            _locationProvider.setLocationListener(null, -1, -1, -1);
        }
    }

    /* Adds a new WayPoint and commits the set of saved waypoints
     * to flash memory.
     * @param p The point to add.
     */
    /*package*/
    synchronized static void addWayPoint(WayPoint p)
    {
        _previousPoints.addElement(p);
        commit();
    }

    /* Removes a waypoint from the set of saved points and
     * commits the modifed set to flash memory.
     * @param p the point to remove
```

```
     */
    /*package*/ synchronized static void removeWayPoint(WayPoint p)
                {
                    _previousPoints.removeElement(p);
                    commit();
                }

    // Commit the waypoint set to flash memory.
    private static void commit()
    {
        _store.setContents(_previousPoints);
        _store.commit();
    }

    /*
     * Implementation of the LocationListener interface.
     */
    private class LocationListenerImpl implements LocationListener
    {
        // Members. ------------------------------------------------
------------
        private int captureCount;
        // Methods. ------------------------------------------------
------------
        public void locationUpdated(LocationProvider provider,
Location location)
        {
            if(location.isValid())
            {
                float heading = location.getCourse();
                double longitude =
location.getQualifiedCoordinates().getLongitude();
                double latitude =
location.getQualifiedCoordinates().getLatitude();
                float altitude =
location.getQualifiedCoordinates().getAltitude();
                float speed = location.getSpeed();
                // Horizontal distance.
                float horizontalDistance = speed * _interval;
                _horizontalDistance += horizontalDistance;
                // Horizontal distance for this waypoint.
                _wayHorizontalDistance += horizontalDistance;
                // Distance over the current interval.
                float totalDist = 0;
```

```
                // Moving average grade.
                for(int i = 0; i < GRADE_INTERVAL - 1; ++i)
                {
                    _altitudes[i] = _altitudes[i+1];
                    _horizontalDistances[i] = _horizontalDistances[i+1];
                    totalDist = totalDist + _horizontalDistances[i];
                }
                _altitudes[GRADE_INTERVAL-1] = altitude;
                _horizontalDistances[GRADE_INTERVAL-1] =
speed*_interval;
                totalDist= totalDist +
_horizontalDistances[GRADE_INTERVAL-1];
                float grade = (_altitudes[4] - _altitudes[0]) *
100/totalDist;
                // Running total of the vertical distance gain.
                float altGain = _altitudes[GRADE_INTERVAL-1] -
_altitudes[GRADE_INTERVAL-2];
                if (altGain > 0) _verticalDistance = _verticalDistance
+ altGain;
                captureCount += _interval;
                // If we're mod zero then it's time to record this data.
                captureCount %= CAPTURE_INTERVAL;
                // Information to display on the device.
                StringBuffer sb = new StringBuffer();
                sb.append("Longitude: ");
                sb.append(longitude);
                sb.append("\n");
                sb.append("Latitude: ");
                sb.append(latitude);
                sb.append("\n");
                sb.append("Altitude: ");
                sb.append(altitude);
                sb.append(" m");
                sb.append("\n");
                sb.append("Heading relative to true north: ");
                sb.append(heading);
                sb.append("\n");
                sb.append("Speed : ");
                sb.append(speed);
                sb.append(" m/s");
                sb.append("\n");
                sb.append("Grade : ");
                if(Float.isNaN(grade))sb.append(" Not available");
                else sb.append(grade+" %");
```

```
                GPSDemo.this.updateLocationScreen(sb.toString());
            }
        }
        public void providerStateChanged(LocationProvider provider,
int newState)
        {
            // No operation defined.
        }
    }
    /* WayPoint describes a way point, a marker on a journey or point
of interest.
     * WayPoints are persistable.
     * package
     */
    static class WayPoint implements Persistable
    {
        public long _startTime;
        public long _endTime;
        public float _distance;
        public float _verticalDistance;
        public WayPoint(long startTime,long endTime,float
distance,float verticalDistance)
        {
            _startTime=startTime;
            _endTime=endTime;
            _distance=distance;
            _verticalDistance=verticalDistance;
        }
    }
}

public float _verticalDistance;
public WayPoint(long startTime,long endTime,float distance,float
verticalDistance)
{
    _startTime=startTime;
    _endTime=endTime;
    _distance=distance;
    _verticalDistance=verticalDistance;
}
}


/*PointScreen.java
```

```
Copyright (C) 2005 Research In Motion Limited.
*/
package com.rim.samples.docs.gpsdemo;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import com.rim.samples.docs.baseapp.*;
import net.rim.device.api.io.*;
import net.rim.device.api.system.*;
import com.rim.samples.docs.resource.*;
import net.rim.device.api.i18n.*;
import javax.microedition.io.*;
import java.util.*;
import java.io.*;
import javax.microedition.location.*;
import com.rim.samples.docs.gpsdemo.GPSDemo.WayPoint;
import com.rim.samples.docs.resource.*;

/*
 * PointScreen is a screen derivative that renders the saved
WayPoints.
 * */
public class PointScreen extends MainScreen implements
ListFieldCallback, GPSDemoResResource
{
    private Vector _points;
    private ListField _listField;
    private ResourceBundle _resources;
    public float _verticalDistance;
    public WayPoint(long startTime,long endTime,float distance,float
verticalDistance)
    {
        _startTime=startTime;
        _endTime=endTime;
        _distance=distance;
        _verticalDistance=verticalDistance;
    }
}

/*
 * PointScreen.java
Copyright (C) 2005 Research In Motion Limited.
*/
package com.rim.samples.docs.gpsdemo;
```

```
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import com.rim.samples.docs.baseapp.*;
import net.rim.device.api.io.*;
import net.rim.device.api.system.*;
import com.rim.samples.docs.resource.*;
import net.rim.device.api.i18n.*;
import javax.microedition.io.*;
import java.util.*;
import java.io.*;
import javax.microedition.location.*;
import com.rim.samples.docs.gpsdemo.GPSDemo.WayPoint;
import com.rim.samples.docs.resource.*;

/*
 * PointScreen is a screen derivative that renders the saved
WayPoints.
 */
public class PointScreen extends MainScreen implements
ListFieldCallback, GPSDemoResResource
{
    private Vector _points;
    private ListField _listField;
    private ResourceBundle _resources;
}
    // Menu items. ------------------------------------------------
-------------
private class ViewPointAction extends MenuItem
{
    private int _index;
    public ViewPointAction( int index )
    {

    super(PointScreen.this._resources.getString(GPSDEMO_POINTSCREEN_M
ENUITEM_VIEW), 100000, 10);
        _index = index;
    }
    public void run()
    {
        ViewScreen screen = new
ViewScreen( (WayPoint)_points.elementAt(_index),
            index, _resources );
        UiApplication.getUiApplication().pushModalScreen( screen );
```

```
    }
}
private class DeletePointAction extends MenuItem
{
    private int _index;
    public DeletePointAction( int index )
    {

    super(PointScreen.this._resources.getString(GPSDEMO_POINTSCREEN_M
ENUITEM_DELETE), 100000, 10);
        _index = index;
    }
    public void run()
    {
        GPSDemo.removeWayPoint((WayPoint)_points.elementAt(_index));
    }
}
protected void makeMenu(Menu menu, int instance)
{
    if( _points.size() > 0 )
    {
        ViewPointAction viewPointAction = new ViewPointAction(
               listField.getSelectedIndex() );
        menu.add( viewPointAction );
        menu.addSeparator();
        DeletePointAction deletePointAction =
            new DeletePointAction(
                   listField.getSelectedIndex() );
        menu.add( deletePointAction );
        }
    super.makeMenu(menu, instance);
    }
/*
 * Renders a particular Waypoint.
 */
private static class ViewScreen extends MainScreen
{
    private ResourceBundle _resources;
    private MenuItem _cancel;
    public ViewScreen(WayPoint point, int count, ResourceBundle
resources)
    {
        super();
        _resources = resources;
```

```
        LabelField title = new
LabelField(resources.getString(GPSDEMO_VIEWSCREEN_TITLE) + count,
            LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
        setTitle(title);
        Date date = new Date(point._startTime);
        String startTime = date.toString();
        date = new Date(point._endTime);
        String endTime = date.toString();
        float avgSpeed = point._distance/(point._endTime -
point._startTime);
        add(new
BasicEditField(resources.getString(GPSDEMO_VIEWSCREEN_STARTFIELD),
startTime, 30, Field.READONLY));
        add(new
BasicEditField(resources.getString(GPSDEMO_VIEWSCREEN_ENDFIELD),
endTime, 30, Field.READONLY));
        add(new
BasicEditField(resources.getString(GPSDEMO_VIEWSCREEN_HORIZONTALDISTA
NCEFIELD), Float.toString(point._distance), 30, Field.READONLY));
        add(new
BasicEditField(resources.getString(GPSDEMO_VIEWSCREEN_VERTICALDISTANC
EFIELD), Float.toString(point._verticalDistance), 30,
Field.READONLY));
        add(new
BasicEditField(resources.getString(GPSDEMO_VIEWSCREEN_AVESPEEDFIELD),
Float.toString(avgSpeed), 30, Field.READONLY));
    }
        private class CancelMenuItem extends MenuItem
    {
        public CancelMenuItem()
        {
            // Reuse an identical resource below.
            super(ViewScreen.this._resources,
GPSDEMO_OPTIONSSCREEN_MENUITEM_CANCEL, 300000, 10);
        }
        public void run()
        {
            UiApplication uiapp = UiApplication.getUiApplication();
            uiapp.popScreen(ViewScreen.this);
        }
    };

    protected void makeMenu( Menu menu, int instance )
    {
```

```
        if ( _cancel == null ) _cancel = new CancelMenuItem(); //
Create on demand.
        menu.add(_cancel);
        super.makeMenu(menu, instance);
    }
}
}
```

12

第 12 章 打包和部署

使用 BlackBerry 桌面软件部署应用程序

应用程序加载工具是 BlackBerry 桌面软件的一部分,它使用一个应用程序加载文件(.alx)将新的应用程序装载到 BlackBerry 设备上.
为每个应用程序创建一个应用程序加载文件(.alx),然后把.alx 和.cod 分发给用户.为获得更多信息,*参看应用程序加载在线帮助.*

创建一个应用程序加载文件

1. 在 BlackBerry IDE 中,选择一个项目.
2. 在 **Project** 菜单上,单击 **Generate .alx file**.

把应用程序的.alx 和.cod 分发给用户.当用户将 BlackBerry 设备和他们的计算机连接起来时,他们可以使用 BlackBerrry 桌面软件将应用程序装载到 BlackBerry 设备.

注：缺省的, 应用程序的. cod 文件与它的. alx 文件放在同一目录下. 如果你改变了和. alx 文件相关的. cod 位置, 编辑. alx 文件, 加入一个<directory>元素来指定文件的位置. 参看 183 页的”附录:. alx 文件的格式”获得更多信息.

无线部署应用程序

手持设备软件允许用户利用 BlackBerry 浏览器无线下载应用程序.用户可以下载标准的 MIDlet 和 BlackBerry 应用程序.为了让用户无线下载应用程序,你必须提供一个恰当的应用程序描述符 (.jad),以及一个应用程序的.cod 或.jar 文件.在 BlackBerry 浏览器中,用户选择一个.jad 文件进行下载应用程序.

系统管理员可以设置应用程序控制策略来控制第三方应用程序的使用.参看 16 页的”应用程序控制”获得更多信息.

采用下列方式让用户可以无线下载 BlackBerry 或 MIDlet 应用程序:

- 使用 BlackBerry MDS 服务,它将.jar 文件转化为.cod 文件.
- 使用 BlackBerry JDE 创建你的工程,它生成一个.cod 文件.

部署.jar 文件

BES 的 BlackBerry MDS 优化服务提供了一个内置的编码转化器,它将.jar 文件转化为.cod 文件,它允许用户下载标准的 MIDlet.例如,公司的管理员可以维护局域网内一系列已许可的 MIDlet.用户可以浏览 web 页面,并且为应用程序选择一个.jad 文件下载.在 BES 将它们发送到 BlackBerry 设备之前,它会将.jar 文件转化为.cod 文件.

注：Web 服务器必须为.cod 文件和.jad 文件设置 MIME 类型.对于.cod 文件,MIME 类型是 application/vnd.rim.cod.对于.jad 文件,MIME 类型是 text/vnd.sun.j2me.app-descriptor.对于.jar 文件,MIME 类型是 application/java-archive.
下列版本的 BES 支持将.jar 文件转化为.cod 文件.

- Microsoft Exchange BES 3.6 以及以后的版本
- IBM Lotus Domino BES 2.2 以及以后版本.

注：如果用户使用带有 MDS 服务的 BES 访问网络,他们只能下载.jar 文件.MDS 服务将.jar 文件转化为 BlackBerry 设备需要的.cod 文件格式.如果用户使用 WAP 网关访问网络,用户只能下载.cod 文件.

MIDlet 应用程序描述符属性

应用程序描述符文件有一个.jad 的扩展.一个标准的 MIDlet .jad 文件包括了下列预定义的属性,可能也会包含应用程序定义的附加属性.

必需的 MIDlet 属性	描述
MIDlet-Jar-Size	.jar 文件的字节大小.
MIDlet-Jar-URL	可以加载.jar 文件的 URL.
MIDlet-Name	MIDlet 包的名称.
MIDlet-Vendor	提供 MIDlet 包的组织.
MIDlet-Version	MIDlet 包的版本,形式为 *<major><minor><micro>*.

可选的 MIDlet 属性	描述
MIDlet-Data-Size	MIDlet 包需要的持久数据的字节大小.缺省为 0.
MIDlet-Delete-Confirm	当用户确定删除 MIDlet 包时,文本消息弹出.
MIDlet-Description	MIDlet 的描述.
MIDlet-Icon	在.jar 文件里,用来代表 MIDlet 包的.png 图像的文件名.
MIDlet-Info-URL	描述 MIDlet 包的未来信息的 URL.
MIDlet-Install-Notify	发送一个 POST 请求以确认 MIDlet 成功安装的 URL.

部署.cod 文件

当你编译项目时,BlackBerry JDE 创建一个需要的.jad 文件.你也可以使用 BlackBerry JDE 将

MIDlet .jar 文件转化为.cod 文件格式.

使 web 服务器上的.cod 和.jar 文件供用户下载.通过使.cod 文件可用,你可以将应用程序部署到那些使用 BES 但不能访问网络的用户.

注: Web 服务器必须为. cod 文件和. jad 文件设置 MIME 类型. 对于. cod 文件, MIME 类型是 application/vnd. rim. cod. 对于. jad 文件, MIME 类型是 text/vnd. sun. j2me. app-descriptor. 对于. jar 文件, MIME 类型是 application/java-archive.

BlackBerry 应用程序描述符属性

除了 MIDlet 应用程序属性外,下面的属性也应用到 BlackBerry .jad 文件中.

必需的 RIM 属性	描述
RIM-COD-Creation-Time	.cod 文件创建的时间.
RIM-COD-Module-Dependencies	.cod 文件需要的模块列.
RIM-COD-Module-Name	包含在.cod 文件中的模块名称.
RIM-COD-SHA1	.cod 文件的 SHA1 哈希
RIM-COD-Size	.cod 文件的字节大小.
RIM-COD-URL	可以加载.cod 文件的 URL.

可选的 RIM 属性	描述
RIM-Library-Flags	RIM 保留使用
RIM-MIDlet-Flags	RIM 保留使用
RIM-MIDlet-NameResourceBundle	应用程序依赖的资源包名
RIM-MIDlet_Position	应用程序图标在主页面上建议的位置. 注:这个位置可能不是应用程序图标在主页面上的实际位置.

BlackBerry 允许你创建一个双目的的.jad 文件,支持将 MIDlet 文件下载到 BlackBerry 设备和其他无线设备中.为了实现它,创建一个既包括 RIM-COD-URL 和 RIM-COD-Size 属性,又包括 MIDlet-Jar-URL 和 MIDlet-Jar-Size 属性的.jad 文件.在 BlackBerry 设备上,你可以下载.cod 文件,在其他设备上,你可以下载.jar 文件.

设置.cod 文件之间的依赖

.jad 包含了一个 RIM-COD-Module-Dependencies 属性,它指定了应用程序需要的模块,但是没有提供它.如果没有任何需要的模块,BlackBerry 浏览器将阻止应用程序的无线安装,并且为用户列出丢失的模块.RIM-COD-Module-Dependencies 属性使用户避免下载一个不能运行的应用程序.

RIM-COD-Module-Dependencies 属性将以逗号分隔的模块名为参数.例如,一个需要 RIM XML 库的应用程序使用下面的应用程序描述符:

```
RIM-COD-Module-Dependencies:net_rim_cldc,net_rim_xml
```

部署兄弟.cod 文件

BlackBerry 为应用程序创建单个.cod 文件和.jad 文件.如果应用程序包含了大于 64K 字节的代码或源数据,BlackBerry IDE 将创建一个包含兄弟文件的.cod 文件.仅 BlackBerry 浏览器支持

包含兄弟文件的.cod 文件安装.

注：如果.cod 文件需要验证，在.jad 文件中更新已验证的.cod 文件大小.参看*BlackBerry IDE 帮助*获得更多信息.

决定.cod 文件是否包含兄弟.cod 文件

1. 提取.cod 文件的内容.

任何在原始.cod 文件的.cod 文件都是兄弟文件.

使用 MDS 服务

为了使用 BlackBerry 浏览器将一个应用程序部署到一个 BlackBerry 设备,.此设备已和 3.6.4 或更高版本的带有 MDS 服务的 BES 连接,你需要将.cod 文件和.jad 文件放到 web 服务器上.为了将一个应用程序部署到 BlackBerry 设备,.jad 文件和.cod 文件必须放到 web 服务器上.MDS 服务使用浏览器下载每个兄弟.cod 文件,一次一个.

BlackBerry 首先加载.jad 文件.MDS 服务的 MDS Provisioning Service(MDS 供应服务)为每个原始.cod 文件里的兄弟文件重复此过程.

使用 BlackBerry Internet 服务或 WAP 浏览器

为了使用 BlackBerry 浏览器将应用程序部署到一个没有使用 BES 的设备上,修改.jad 文件单独列出每个兄弟文件.你必须从原始的.cod 文件提取每个兄弟文件,并且将它们放在 web 服务器上.BlackBerry 浏览器将按照.jad 文件列出的顺序一次只下载一个兄弟.cod 文件.

注：为了避免覆盖原始的.cod 文件，提取兄弟.cod 文件到一个不同的目录下，而不是原始文件所在的目录.

为了提取兄弟.cod 文件,完成下面的操作:

1. 将原始的.cod 文件解压缩,提取兄弟.cod 文件.
2. 将每个兄弟.cod 文件放在 web 服务器上.
3. 在.jad 文件中,单独列出兄弟.cod 文件.对每个兄弟文件,创建 RIM-COD-URL-<#>, RIM-COD-Size-<#>参数.

- RIM-COD-URL-<#>:为每个兄弟.cod 文件创建一个 RIM-COD-URL-<#>,并将兄弟文件名放在参数的右边.#一个对每个兄弟文件都增加 1 的数字.每个兄弟.cod 文件的名字和原始的.cod 文件的名字一样,后面紧跟-<#>.
- RIM-COD-Size-<#>:为每个兄弟.cod 文件创建一个 RIM-COD-Size-<#>参数,并将每个文件的大小放在参数右边,#和加到兄弟文件名后面的数字一样.将 RIM-COD-Size-<#>参数放在紧靠 RIM-COD-URL-<#>的下面.

在下面的例子中,有 2 个兄弟文件.在原始的.cod 文件 myApp 后,开发者将兄弟文件命名为 myApp-1.cod 和 myApp-2.cod.开发者为每个兄弟文件名加入'.cod'扩展.并为每个兄弟文件创建一个 RIM-COD-Size-<#>的参数.

```
Manifest-Version: 1.0
MIDlet-Version: 1.0.0
MIDlet-1: ,,
RIM-COD-Module-Dependencies: net_rim_cldc
MicroEdition-Configuration: CLDC-1.0
RIM-COD-Module-Name: MyApp
```

```
MIDlet-Name: My Application
RIM-COD-URL: myApp.cod
RIM-COD-Size: 55000
RIM-COD-URL-1: myApp-1.cod
RIM-COD-Size-1: 50000
RIM-COD-URL-2: myApp-2.cod
RIM-COD-Size-2: 25000
MicroEdition-Profile: MIDP-1.0
```

注：为兄弟.cod 文件使用下面的命名规则:<原始.cod 文件名>-<序列号>.你必须为每个兄弟文件分配一个数字.这个数字从 1 开始,并且逐次加 1.

4. 在你打算部署应用程序的设备上,从 BlackBery 浏览器下载每个单独的兄弟文件.

13

第 13 章 测试和调试

测试应用程序

通过在 BlackBerry 设备模拟器或在一个已连接的 BlackBerry 设备上运行应用程序来测试它们。

1. 在 BlackBerry IDE 的 **Debug** 菜单中，点击 **Go**.
2. 将应用程序部署到设备。为获得更多信息，参看 166 页的”使用 BlackBerry 桌面软件部署应用程序”。
3. 在模拟器或一个 BlackBerry 使用应用程序。
4. 在 BlackBerry IDE 的 **Debug** 菜单中，点击 **Break Now**。
5. 进行下列任何操作：

操作	过程
重新运行应用程序	在 **Debug** 菜单，单击 **Continue**。
完成调试	在 **Debug** 菜单，单击 **Stop Debugging**。

使用设备模拟器测试应用程序

在 BlackBerry IDE 中，当运行应用程序时，设备模拟器会自动启动。

在访问 BES 下,模拟器设计为模拟一个在 BlackBerry 设备上运行应用程序的各个方面,包括 email 状况,浏览器状况,HTTP/TCP 连接,以及 push 功能.

在没有访问 BES 的情况下,你需要使用 BlackBerry MDS 模拟器来模拟浏览器状况,第三方应用程序的 http/tcp 连接,以及 push 功能.为获得更多信息,参看 176 页的”测试 HTTP 网络连接”.

你需要一个 BlackBerry 邮件服务器模拟器(邮件模拟器)来发送和接收 BlackBerry 设备模拟器以及一个计算机邮件应用程序之间的消息.为获得更多信息,参看 173 页的”使用邮件服务器模拟器”.

动作	鼠标过程	键盘过程
滚动滑轮	滚动鼠标的滑轮.	在键盘上按 **UP ARROW** 和 **DOWN ARROW** 键
点击滑轮	点击鼠标的滑轮按钮.	按**回车键**.

运行一个应用程序	选择适合的图标,点击鼠标滑轮按钮.	按**回车键**.
按键	-	按键盘上的键.
分配 Escape 按钮给滑轮	1.在设备模拟器中的 Edit 菜单,点击 **Map Cursor Key To Escape**. 2.完成下面的一个操作: ● 为分配 **Left** 给滑轮,要么选择 **Right** ● 为分配 **Right** 给滑轮,要么选择 **Left**.	

测试 BES API 和 IT 策略

注：设备模拟器需要的版本为 4.0 以及更高的 BES.

设备模拟器可以连接到 BES 上的一个帐户.如果你可以访问一个 BES,可以把一个模拟的 BlackBerry 设备连接到 BES.使用选项来模拟 BES API 以及应用的 IT 策略.这个选项需要有 BES 管理员的知识,并且它为测试应用了网络依赖.你不需要一个 BlackBerry MDS 模拟器或一个邮件服务器模拟器.你可以模拟 BES 扩展 API 的使用,应用的 IT 策略,邮件和浏览器消息,HTTP/TCP 连接以及 push 功能.

1. 启动设备模拟器.
2. 在主屏幕上,单击"Turn Wireless Off".
3. 在微软窗口的任务栏,点击**开始>程序>BlackBerry>Desktop Manager**.
4. 在模拟器菜单,完成下面任一操作:

动作	过程
模拟 USB 连接.	点击 Simulate>USBConnected
模拟序列端口连接	点击 Simulate>Serial Connected

5. 单击 **Yes**.
6. 完成执导操作生成一个新的加密键.
7. 在初始的插件中:Verifying Application 对话框,点击 **Cancel**.

当和计算机的初始同步完成时,企业激活就启动了.

使用邮件服务器模拟器(ESS)

ESS 允许你发送和接收设备模拟器和任意一个计算机邮件程序,例如 Microsoft Outlook Express 或者邮件服务器如 POP3 和 SMTP,之间的消息.使用 ESS 代替连接一个设备模拟器到 BES 来进行测试单机上的本地应用程序..

1. 在任务栏,单击**开始>程序>Research In Motion>BlackBerry JDE 4.1.0>ESS**.
2. 完成下面任一操作:
   - **Standalone mode(单机模式)**:存储消息到本地文件系统,和一个计算机邮件程序直接通信.你不需要一个 POP3 或 SMTP 服务器.
   - 打开计算机邮件程序,
   - 设置 POP3 服务器到本地的 110 端口号.

- 设置 SMTP 服务器到本地的 25 端口号.
- **Connected mode**.邮件模拟器为接收消息 poll 用户 POP3 邮件服务器,并且使用 SMTP 服务器发送消息.邮件模拟器需要有效的 POP3 和 SMTP 服务器.

3. 为了从本地文件系统移除邮件模拟器，单击 **Clean FS**。
   - 在 **Outgoing** 域，输入你的帐户使用的 SMTP 服务器的主机名。
   - 在 **Incoming** 域，输入你的帐户使用的 POP3 服务器的主机名。
   - 在 **User name** 域，输入用户名连接到你的帐户。
   - 在 **Password** 域，输入密码连接你的消息帐户。
   - 在 **Poll inbox** 域，指定邮件模拟器检查收件箱新消息的频率（以秒为单位）。
   - 在 **Name** 域，输入一个名字，它显示在设备模拟器发出的消息中。
   - 在 **Email** 域，输入消息帐户地址，它显示在 BlackBerry 设备模拟器发出的消息中。
   - 在 PIN 域，输入设备模拟器使用的 PIN（Personal Identification Number，个人验证码）（缺省值为 0x2100000A）。
4 点击 **Launch**。
   如果你在 ESS 窗口中改变了参数值，一个对话框将提示你保存修改。
5 检查命令行窗口中的启动信息，它也包含了任何错误日志。
   当邮件模拟器启动时，使用模拟器里的消息列表来发送和接收带有一个帐户的消息。


注：如果你从命令行启动设备模拟器，指定/rport=0x4d4e 参数和 ESS 通讯。

在模拟器中使用同步来测试一个应用程序

1. 退出 BlackBerry 桌面软件。
2. 在你的计算机的 COM1 和 COM2 之间连接一个空的 modem 线。
3. 在 BlackBerry IDE 的 **Edit** 菜单，单击 **Preferences**。
4. 在首选项窗口，挡架 **Basic** 标签。
5. 选择 **Set Serial port for device(s)**选项，输入 **1**。
6. 单击 **OK**。
7. 在 BlackBerry IDE 中，编译和允许应用程序。
8. 在模拟器启动后，启动 BlackBerry 桌面软件。
9. 在 BlackBerry 桌面管理窗口的 **Options** 菜单里，单击 **Connection Settings**。
10. 点击 **Detect** 检测模拟器。


注：如果 BlackBerry 桌面软件没有检测到模拟器，重启计算机。重复步骤 7 到 10。

使用一个已连接的 BlackBerry 设备测试应用程序

当你将一个 BlackBerry 设备连接到一台计算机时，在 BlackBerry 设备上运行应用程序，并且利用 BlackBerry IDE 调试工具来完成测试以及优化。


注：为了将 BlackBerry IDE 附加到一个序列化端口连接的 BlackBerry 设备，需要安装 Java 通信 API v2.0，在 http://java.sun.com/products/javacomm/可得到它。当 BlackBerry 设备连接到 USB 端口时此 API 则不需要。

安装.debug 文件

为了使用 BlackBerry 设备调试应用程序，BlackBerry IDE 中的.debug 文件必须和 BlackBerry 设备的版本号相匹配。

1. 从 BlackBerry 开发区 http://blackBerry.com/developers 为 BlackBerry 设备软件版本号下载.debug 文件。
2. 在 BlackBerry IDE 的 **Edit** 菜单，单击 **Preferences**。
3. 单击 **Debug** 标签。
4. 单击 **Other** 标签。
5. 在 **Handheld debug file locatio**n 域，输入下载的.debug 文件的路径。

加载一个应用程序进行测试

注：在加载一个应用程序进行测试之前, 请备份你的 BlackBerry 设备应用程序信息. JavaLoader. exe 工具允许你使用命令行增加或更新 BlackBerry 设备上的应用程序. 仅当以开发和测试目的时使用本工具. 对于产品应用程序, 使用 BlackBerry 桌面软件.

注：你必须按照依赖关系的正确顺序加载应用程序. 如果项目 A 依赖项目 B,在加载项目 A 之前先加载项目 B.

1.退出桌面软件.

2.将 BlackBerry 设备连接到计算机.

3.在命令行里,找到 BlackBerry JDE 安装目录下的 **Bin** 文件夹.

4.输入下面的命令:

JavaLoader [-usb] [-p<port> [-b<bps>] [-w<password>] load <files>

选项	描述
port	BlackBerry 设备连接的 COM 端口(缺省为 1). 如果 BlackBerry 使用 USB 端口连接, 为 BlackBerry 设备的 PIN(**-usb** 选项必须指定).
bps	到序列端口的位速率(缺省为 115200).
password	BlackBerry 的设备密码,如果你设置了一个的话.
files	加载到 BlackBerry 设备中的一个或多个.cod 文件名,它们分别以空格分开.

从 BlackBerry 删除应用程序

在命令行,输入下面的命令行:

JavaLoader [-usb] [-p<port>] [-b<bps>] [-w<password>] erase [-f] <files>

在这里,-f 选项删除应用程序,甚至此应用程序正在使用.

将 BlackBerry IDE 调试器和 BlackBerry 设备相连

1. 为了将 BlackBerry IDE 调试器与使用 USB 端口连接的 BlackBerry 设备连接,启动 **BBDevMgr.exe**.BlackBerry 桌面软件 3.5.1 以及后续版本安装 BBdevMgr.exe 工具.
C:\program files\Common Files\Research In Motion\USB Drivers.

2. 完成下面任一操作:

操作	过程

BlackBerry IDE 调试器连接一个使用序列端口的 BlackBerry 设备	单击 **Attach to**>**Handheld**>**COM n**,在这里 n 是你的 BlackBerry 设备连接的序列端口.
BlackBerry IDE 调试器连接一个使用 USB 端口的 BlackBerry 设备	单击 **Attach to**>**Handheld**>**USB(PIN)**,在这里 PIN 是已连接的 BlackBerry 设备的 PIN..

- 对于一个连接到序列端口的 BlackBerry 设备, 单击 **Attach to**>**Handheld**>**COM n**, 在这里 n 是你的 BlackBerry 设备连接的序列端口.
- 对于一个连接到 USB 端口的 BlackBerry 设备, 单击 **Attach to**>**Handheld**>**USB(PIN)**,在这里 PIN 是已连接的 BlackBerry 设备的 PIN.

测试 HTTP 网络连接

为了测试一个需要 HTTP 网络连接的应用程序,使用 BlackBerry MDS 模拟器,它是一个 BlackBerry JDE 的组件.

> 在任务栏里,单击**开始**>**程序**>**Research In Motion**>**BlackBerry JDE 4.1.0**>**MDS**.

注:当设备模拟器启动时, 为了启动 MDS 服务模拟器, 在 BlackBerry IDE 的 Eidt 菜单, 单击 Preferences. 点击 Simulator 标签, 选择 Launch Mobile Data Service(MDS) with simulator 选项.

使用一个 WAP 网关

你可以使用服务商提供的一个 WAP 网关来配置 HTTP 连接.BlackBerry 设备支持 WAP 1.1.

注: WAP 服务仅在你所选的无线网络可以得到. 在你开始开发之前, 联系你的服务商获得更多关于 WAP 网关的信息.

> 为了使用 WAP 建立一个 HTTP 连接,在 Connector.open() 的 URL 后的末尾包含 WAPGatewayIP 以及 WAPGetwayAPN 参数.

```
Connector.open("http://host;WAPGatewayIP=127.0.0.1;
WAPGatewayAPN=rim.net.gprs");
```

使用分号(;)分隔 WAP 参数,验证 WAP 参数没有空格.

参数	描述	缺省
WapGatewayIP	网关的 IP 地址	-
WapGatewayAPN	APN(仅为 GPRS),为了测试,使用 net.rim.gprs	-
WapGatewayPort	网关端口值	9201
WAP_GETWAY_PORT_DEFAULT		
WapSourceIP	源 IP 地址	127.0.0.1
WAP_SOURCE_IP _DEFAULT		
WapSourcePort	源端口值	8205
WAP_SOURCE_PORT_DEFAULT		
TunnelAuthUserName	APN 对话或 PAP 或 CHAP 验证的用户名.	无
TunnelAuthPassword	APN 对话或 PAP 或 CHAP	无

	验证的密码.	
WapEnableWTLS	打开或关闭 WTLS(如果你没有指定这个参数,那么到端口 8203 的连接将使用 WTLS).BlackBerry 设备支持 WTLS 级别 1(仅加密,没有认证)和级别 2(加密和服务器验证).	无

注: 在 BlackBerry 设备模拟器中,当你测试一个需要 APN 连接的应用程序,增加命令行选项/rport=<wap_source_port>,一般为/rport=8205.模拟器的 APN 是 net.rim.gprs.

增加模拟器命令行选项

在 BlackBerry IDE 的 **Edit** 菜单里,单击 **Preferences**.
单击 **Simulator** 标签.
点击 **Advanced** 标签
在 **Simulator Command Line** 域里,增加命令行选项.

配置 BlackBerry MDS 模拟器

1. 在文本编辑器中,打开 rimpublic.property 文件(在 MDS\Config 文件夹下).
2. 编辑参数,配置下列特性:
   - 参看 178 页的"日志级别参数"获得更多信息.
   - 参看 178 页的"HTTP 支持参数"获得更多信息.
   - 参看 178 页的"HTTPS 支持参数"获得更多信息.
   - 参看 179 页的"Push 支持参数"获得更多信息.
   - 参看 179 页的"Internet 消息的地址-PIN 映射"获得更多信息.
3. 重起 BlackBerry MDS 模拟器.

注: 在产品环境中,BES 系统管理员通过使用 BlackBerry 管理器来配置 MDS 服务参数.联系你的管理员获得更多信息.

日志级别参数

参数	描述	缺省
Logging.level	如果启用日志,这个参数指定记录写到日志中的信息类型. ● 1:仅写入事件信息,例如 MDS 服务的启动和停止. ● 2:写事件和错误 ● 3:写事件,错误和警告. ● 4:写事件,错误,警告以及调试信息.	4
Logging.console.log.level	如果启用日志,这个参数指定	4

	在控制台输出的信息类型.参看 Logging.level 参数的描述.	

HTTP 支持参数

参数	描述	缺省
application.handler.http. logging	此参数打开(TRUE)或关闭(FALSE)HTTP 标准日志(仅 HTTP 头)	TRUE
application.handler.http. logging.verbose	此参数打开(TRUE)或关闭(FALSE)HTTP 调试日志(HTTP 数据和头).当需要调试一个特殊的问题时,此参数设置为 TRUE.	FALSE
application.handler.http. cookieSupport	此参数打开（TRUE）或关闭（FALSE）cookie 的存储。如果选择 TRUE，BlackBerry MDS 服务代替 BlackBerry 设备来管理 cookie 存储。这明显减少了 BlackBerry 上的加载。	TRUE
application.handler.http .AuthenticationSupport	此参数打开（TRUE）或关闭（FLASE）用户验证信息的存储。	TRUE
application.handler.http. AuthenticationTimeout	如果 HTTP 验证设为 TRUE，在验证信息无效之前此参数决定时间的长短。无论何时，当用户发布一个为某一域调用验证信息的时候，这个计时器都会重置。	3600000
application.handler.http. device.connection.timeout	在一个连接试图访问 BlackBerry 设备时，如果 BlackBerry 不可用，此参数设定过期时间的长短（以毫秒计）	140000
application.handler.http. server.connection.time	在一个连接试图访	150000

out	问服务器时，如果服务器不可用，此参数设定过期时间的长短（以毫秒计）	

HTTP 支持参数

参数	描述	缺省
application.handler.https.logging	此参数打开（TRUE）或关闭（FALSE）测试目的的 HTTPS 日志	TRUE
application.handler.https. allowUntrustedServer	此参数允许 MDS 服务连接到一个不信任的服务器（TRUE），或者仅受限访问一个可信任的服务器（FALSE），如果服务器的证书安装在 MDS 服务主机上，那么服务器是信任的。参看 190 页的“使用 keytool 安装一个证书”获得更多信息。	FALSE

PUSH 支持参数

不要改变这些参数。

参数	描述	缺省
WebServer.listen.host	此参数定义了 MDS 服务监听发送 HTTP POST 请求的 PUSH 应用程序所在的计算机。	localhost
WebServer.listen.port	此参数定义了 MDS 服务监听发送 HTTP POST 请求的 PUSH 应用程序所在的端口。	8080

Internet 消息的地址-PIN 映射

在一个产品环境里，BES 自动将用户的 internet 消息地址映射到 BlackBerry 设备的 PIN 上。在 BlackBerry JDE，你可以模拟 internet 消息地址和 PIN 之间的映射。

注：如果你正在测试一个 push 应用程序，你仅需要配置 internet 消息地址到 PIN 之间的映射。为获得更多信息，参看 137 页的“创建一个 client/server push 应用程序。

在 rimpublic.property 文件里，在你的 BlackBerry JDE 安装目录下的 MDS/config 子目录下，增加或修改【simulator】部分的入口。入口使用下面的格式:

Simulator. <PIN>=<host> :< port>, <messaging_address>

例如：

Simulator.2100000a=<local machine IP>:81, user2100000a@blackberry.com

改变 internet 消息地址，以致你可以测试使用实际的 internet 消息地址的应用程序。发送 push

的数据到一明确的 BlackBerry 设备模拟器。

模拟器缺省的 PIN 是 2100000a。
为了改变模拟器 PIN，设置/rsim 选项。在 BlackBerry IDE 的 **Edit** 菜单，单击 **Preferences**。单击 **Simulator** 标签，然后单击 **Advanced** 标签。在 **Simuator Command Line** 域里，修改为 **/rsim-0x2100000A**.

端口必须和 IPPP.push.listen.tcp.port 参数里的值相匹配。缺省是 81。

使用调试工具

注：本部分提供 BlackBerry IDE 中一些可用调试工具的概括。参看 *BlackBerry IDE 在线帮助*获得更多关于使用 BlackBerrry IDE 的详情。

分析代码覆盖

为了显示已经运行代码的概括，使用覆盖工具。当你设计和运行测试案例时，一个概括是有用的，因为你可以看到什么已经精确的测试了。

1. 设置 2 个或多个断点。
2. 在 blackBerry IDE 的 **View** 菜单中，点击 **Coverage**。
3. 为将信息重置为 0，在覆盖（coverage）区域，点击 **Clear**。
4. 运行你的程序到下一个断点。
5. 为了显示你单击 **Clear** 后运行的代码的覆盖率，在覆盖区域，单击 **Refresh**。

使用 profiler

使用 BlackBerry IDE profiler 工具优化你的代码。Profiler 工具显示了每个代码区所花费的时间百分比，一直到当前运行点。

注：当你运行 profiler 时，为了改善结果的可信度，退出其他的 Microsoft windows 应用程序。

运行 profiler

1. 在代码段的开始，按 **F9** 设置一个断点。
2. 在代码段的末尾，按 **F9** 设置一个断点。
3. 在 **Debug** 菜单里，单击 **Go**。
4. 在模拟器里使用应用程序运行合适的代码，直到它达到断点。
5. 在 **View** 菜单，单击 **Profile**。
6. 在 profile 区域，单击 **Options**。
7. 选择方法属性的类型，一个有序的方法，以及信息的类型给 Profile。为获得更多信息，参看 181 页的“设置 profle 选项”。

8. 单击 **OK**。
9. 为删除 profiler 数据，并且重设运行时时间，在 profile 区域，单击 **Clear**。
10. 在 **Debug** 菜单，单击 **Go**。
11. 在模拟器里使用应用程序运行合适的代码，直到它达到断点。
12. 如果 profile 区域不可见，在 **View** 菜单里单击 **Profile**。
13. 为了获得所有精确的 profile 数据，在 profile 区域里，单击 **Refresh**。
14. 单击 Save 将 profile 区的内容保存到. 以逗号分割的 csv 文件中。

Profile 视图	**描述**
Summary	Summary 视图显示关于系统和垃圾回收的普通统计。它显示了 Java VM 空闲时间，运行代码，以及完成快速和完全垃圾回收的时间的比例。Percent 列显示了总的 VM 运行时间，包括空闲和回收时间的比例。
Methods	Methods 视图显示了一列模块，他们要么根据你正在 profile 的信息排序，要么根据每项运行的次数排序，
Source	Source 视图显示了单个方法的源代码行。它允许你通过调用的或被调用的方法来找到那个方法。单击 Back 和 Forward 按钮来跟踪你在 Source 视图里访问过的方法的历史记录。在这个视图里，ercent 列显示了总的 VM 运行时间，不包括空闲和回收时间的比例。

设置 Profile 选项

1. 单击 Options 标签

1 在 Method attribution 下拉列表中，选择下列任一选项
   - 为了计算执行方法和它调用的方法里字节代码所花费的时间，选择 Cumulative。
   - 为了只计算执行方法里字节代码所花费的时间，选择 In Method only。当调用另外的方法时，计时器停止。

2 在 Sort method by 下拉列表中，选择 Count 根据项运行的次数将 Profile 视图里的方法进行排序，或者选择其他选项根据 profile 的数据将方法排序。

3 从 What to profile 下列列表中，选择 profile 的数据类型。

查找内存泄漏

使用内存统计和对象来查找和修正内存泄漏.

使用内存统计工具

内存统计工具显示了对象数以及对象处理,RAM,以及闪存用到的字节数的统计.

1 在 **View** 菜单,单击 **Memory statistics**.

2 在你的代码中设置 2 个或多个断点.

3 运行你的程序到第一个断点.

4 单击 **Refresh** 刷新内存统计.

5 单击 **Snapshot** 得到一个快照.

6 将程序运行到下一个断点.
7 单击 **Refresh**.
8 单击 **Compare to Snapshot**.
9 为了将内存统计的内容存储到一个逗号分隔的.csv 文件中,单击 **Save**.

使用对象工具

对象工具显示了内存中所有的对象,它帮助你定位到发生泄漏的对象.对象泄漏可能导致 VM 在闪存的外部运行,它将重新设置你的 BlackBerry 设备.

1 在 BlackBerry IDE 的 **Debug** 菜单,单击 **Go**.
2 在 **Debug** 菜单中,单击 **Break Now**.
3 在 **View** 菜单中,单击 **Objects**
4 单击 **GC**.
5 单击 **Snapshot**.
6 在 **Debug** 菜单,单击 **Continue**.
7 在程序里完成的任务不应增加可达到的对象的数量;例如,创建一个新的联系人,然后再删除它.
8 在 **Debug** 菜单,单击 **Continue**.
9 在对象域里,单击 **GC**.

单击 **Compare to Snapshot**.对象域,它将显示自上一次快照以来删除和增加的对象数.如果增加的对象数和删除的对象数不一样,你可能有一个对象泄漏.使用 **Type** 和 **Process** 过滤项查看特定的对象.

10 为了将内存统计的内容存储到一个逗号分隔的.csv 文件中,单击 **Save**.

A

附录:.alx 文件的格式

.alx 文件

.alx 文件

应用程序加载工具是 BlackBerry 桌面软件的一部分,它使用了一个应用程序加载文件(.alx)将应用程序加载到 BlackBerry 设备中去.利用 BlackBerry IDE 为你的工程生成一个.alx 文件.下面提供的信息只作为辅助参考.大多数情况下,你没有必要编辑 BlackBerry IDE 生成的.alx 文件.①

在文本编辑器里,你可以编辑 BlackBerry IDE 生成的.alx 文件..alx 文件使用 XML 格式:

例:.alx 文件样例

```
<?xml version="1.0" encoding="UTF-8"?>
<loader version="1.0">
    <application id="com.rim.samples.device.httpdemo">
        <name>Sample Network Application</name>
        <description>Retrieves a sample page over HTTP
connection.</description>
        <version>1.0</version>
        <vendor>Research In Motion</vendor>
        <copyright>Copyright 1998-2003 Research In Motion</copyright>
        <language langid="0x000c">
            <name>Application D'閏hantillon</name>
            <description>Obtenir une page du r閟eau
            </description>
        </language>
        <fileset Java="1.0">
            <directory>samples/httpdemo</directory>
            <files>
              net_rim_httpdemo.cod
              net_rim_resource.cod
              net_rim_resource__en.cod
              net_rim_resource__fr.cod
</files>
```

① 按照个人经验，JDE 生成的 ALX 并不是很理想，特别是比较复杂的工程。所以对于稍微复杂的工程来说，应该手工编辑。译者注。

```
        </fileset>
    </application>
</loader>
```

嵌套模块

在.alx 文件中创建一个嵌套的结构为应用程序提供可选的组件.一般来说,嵌套模块提供的可选特性并不适合所有用户.用户可以选择是否安装这些可选模块.

嵌套，为基本应用程序的嵌套模块创建一个隐含的依赖.为了定义一个对其他应用程序或库的显式依赖,使用<requires>标记.为获得更多信息,参看 185 页的".alx 文件的元素"

例:.带有一个嵌套模块的应用程序.alx 文件样例

```
<loader version="1.0">
    <application id="net.rim.sample.contacts">
        <name>Sample Contacts Application</name>
        <description>Provides the ability to store a list of contacts.
</description>
        <version>1.0</version>
        <vendor>Research In Motion</vendor>
        <copyright>Copyright 1998-2001 Research In Motion</copyright>
        <fileset Java="1.0">
            <directory>samples/contacts</directory>
                    <files>
                        net_rim_contacts.cod
                        net_rim_resource.cod
                        net_rim_resource__en.cod
                        net_rim_resource__fr.cod
                    </files>
        </fileset>
        <application id="net.rim.sample.contacts.mail">
            <name>Sample Module for Contacts E-Mail Integration</name>
            <description>Provides the ability to access the messaging
application</description>
            <version>1.0</version>
            <vendor>Research In Motion</vendor>
            <copyright>Copyright 1998-2001 Research In
Motion</copyright>
            <fileset Java="1.0">
                <directory>samples/contacts</directory>
                <files>
                    net_rim_contacts_mail.cod
                </files>
```

```
            </fileset>
        </application>
    </application>
</loader>
```

指定一个 BlackBerry 设备版本

使用了指定版本的手持设备软件中的 API 的应用程序,应该使用_blackBerryVersion 属性来指定支持的 BlackBerry 设备版本.
使用下面的规则指定一个范围:

- 方括号[]描述闭合范围匹配
- 圆括号()描述开合范围匹配.
- 缺少低围代表 0.
- 缺少高围代表无限值.

例如,[4.0,)代表 4.0 和无限大之间的任何版本.
下面的例子阻止模块加载 4.0 版本以前的手持设备软件.

```
<application id="<application_id>" _blackberryVersion="[4.0,)">
...
</application>
```

下面的例子为不同版本的手持设备软件提供了可选的模块.

```
<application id="<application_id>">
...
<fileset _blackBerryVersion="(,4.0)">
... modules for handheld software versions earlier than 4.0
</fileset>
<fileset _blackBerryVersion="[4.0,)">
... modules for handheld software versions 4.0 and later
</fileset>
</application>
```

.alx 文件元素

元素	属性	描述
loader	version	loader 包含了一个或多个 applicaiton 元素 version 属性指定了应用程序加载器的版本.
applicaiton	id	application 元素为单个应用程序包含了元素. application 元素也可以包含附加嵌套的 application 元素.嵌套使你需要:当一个应用程序加载时,它的先决条件模块也被加载. id 属性为应用程序指定了一个唯一标记符.为了唯一性,使用一个包含你公司域名的 ID 例如,com.rim.samples.docs.helloworld.
library	id	Library 代替 application 标签

name	-	name 为应用程序提供了一个描述性的名称.这个名称在应用程序加载器中显示.它不会在 BlackBerry 设备中显示.
description	-	description 元素为应用程序提供了一个简短的描述.本描述在应用程序加载器中显示.它不会在 BlackBerry 设备中显示.
version	-	version 元素为应用程序提供了一个版本号.它在应用程序加载器中显示.版本号仅为显示信息而用.
vendor	-	vendor 元素提供创建应用程序的公司名,它在应用程序加载器中显示.
copyright	-	copyright 元素提供版权信息, 它在应用程序加载器中显示..
required	-	required 允许你强制加载一个应用程序.在应用程序加载器里,选择一个应用程序安装.加入下面的行: < required>true< required> required 标记仅由企业系统管理员使用.它不为第三方应用程序的用户使用. **注**:BlackBerry 桌面软件 3.6 或后期版本支持本元素.
hidden	-	hidden 元素隐藏包,以至在应用程序加载器里对用户不可见.增加下面的行: <hidden>true</hidden> 缺省的,它和 required 元素联合起来加载应用程序.或如果一个应用程序已加载,设置 required 标记来加载此包. hidden 标记仅由企业系统管理员使用.它不为第三方应用程序的用户使用. **注**:BlackBerry 桌面软件 3.6 或后期版本支持本元素.
language	langid	当应用程序加载器以 langid 属性指定的语言运行时,language 标记允许你覆写在应用程序显示的文本. 为了支持多语言,指定多个 language 标记.为了指定每种语言的 name(名称),version(版本),vendor(发行商)以及 copyright(版权),在 language 标签里嵌套他们.如果不没有嵌套一个标记,将以缺省的语言显示. langid 属性为此信息应用的语言指定 Win32 langid 语言代码.例如,一些 Win32 langid 代码是:0x0009(英语),0x0007(德语),0x000a(西班牙语),0x00c(法语).
requires	id	requires 元素可选,它指定了应用程序依赖的包的 id.如果一应用程序依赖不止一其他的应用程序,id 可以不止一次出现. 当一个应用程序加载到 BlackBerry 设备时,所有<requires>指定的包也被加载. **注**:BlackBerry 桌面软件 3.6 或后期版本支持本元素.

fileset	java radio langid color	fileset 元素包含了一个可选的 directory 元素以及一个或多个 files 元素.它在一个单独的目录中指定一组.cod 文件加载到 BlackBerry 设备中.为了加载多个目录下的文件,在.alx 文件中包含一个或多个 fileset 元素. java 属性指定.cod 文件兼容的 BlackBerry Java VM 最小版本,当前的 VM 版本是版本 1.0.Java 属性是必需的. radio 属性允许你加载不同的应用程序或模块.这些程序或模块依赖 BlackBerry 设备的网络类型.可能的值有 Mobitex,DataTAC,GPRS,CDMA 和 IDEN.radio 属性是可选的. langid 属性允许你加载不同的应用程序或模块.这些程序或模块依赖用户增加到 BlackBerry 设备的支持的语言. langid 属性为此信息应用的语言指定 Win32 langid 语言代码.例如,一些 Win32 langid 代码是 :0x0009(英语),0x0007(德语),0x000a(西班牙语),0x00c(法语). color 属性允许你为彩色或黑白屏幕加载不同的应用程序或模块.它是一个 Boolean 值;true 代表彩色,false 代表黑白.
directory	-	directory 元素提供了文件的位置. directory 元素是可选的.如果你没有指定 directory 元素.文件必须和.alx 文件的目录相同.指定.alx 文件位置的目录.
files	-	files 为应用程序提供单个目录下的一个或多个.cod 文件列表,加载它们到 BlackBerry 设备.

B

附录:MDS 服务参考

HTTP 请求①

一个客户端建立一个连接,并且发送一个 HTTP 请求消息到服务器.服务器然后发送一个响应消息,这个消息通常包含了请求的资源.

```
<method> <resource_path><version>
   Header1: value1
   Header2: value2
   Header3: value3
<optional message>
```

HTTP 请求变量	描述
method	方法名,指定了一个动作,例如 GET,HEAD 或 POST.常用的方法是 GET,它从服务器请求一个资源.
resource_path	指向请求资源的路径,它是 URL 中的一部分,在主机名后面出现.它也称为*请求 URL(Request URL).*
version	你正在运行的 HTTP 版本,标记为"*HTTP /x.x*".BES 支持 1.0 和 1.1 版本.
Header	提供了关于请求的或在消息体里发送的对象的信息.
optional message	HTTP 消息可以包含数据.在一个请求里,它是发送到服务器的用户类型的数据,或上传的文件.当一个对象伴随着此消息时,请求通常也包含定义它属性的消息头.

① 其实就是 HTTP 协议。译者注。

HTTP 响应

在 HTTP 请求消息的接收之上,服务器发送一个响应消息,它通常包含了请求的资源.

```
<HTTP version><status_code><reason>
  Header1: value1
  Header2: value2
  Header3: value3
<message>
```

HTTP 响应变量	描述
HTTP_version	正在运行的 HTTP 版本,标记为"HTTP /x.x".BES 支持 1.0 和 1.1 版本.
status_code	状态码的数值，它反映了客户端提出的请求的结果。例如，200（OK）说明了传输成功，404（Not Found）说明了请求的 URL 没有找到。
reason	reason 是和状态码相关的文本消息。
Header	消息头提供了响应的信息，也提供了消息体中正在发送的对象的信息。
message	HTTP 消息必须包含数据。在一个响应消息里，它提供了客户端请求的的内容。此响应也包含了定义它的属性的消息头。

注：应用程序应该检查 HTTP 响应消息的状态码。任何不是 200（OK）的状态码都说明当建立 HTTP 连接时发生了一个错误。

Push 请求响应状态码

为 push 请求连接服务，BlackBerry MDS 返回 2 种响应码：

- PAP（Push Access Protocal）响应码。
- RIM Push 响应码。

注：访问 www.wapforum.org 获得更多关于 PAP 响应码的信息。

RIM Push 请求响应码

RIM Push 请求响应码	描述	解释
200	OK	请求成功完成。
400	任何其他的错误。	
403	访问控制错误，或者未知的邮件地址或特定的 BlackBerry PIN。	服务器接收了请求，但是不能响应。
503	服务器忙，服务不可用。	当前，服务器不能管理请求，因

		为暂时的负载或服务器正在维护。

HTTPS 支持

如果你的应用程序通过 Internet 访问服务器，为了提过其他的验证和安全，在 TLS 上建立一个安全的 HTTP（HTTPS）连接。

HTTPS 认证（Certificate）管理

当 BlackBerry 设备以代理模式请求一个 HTTPS 连接时，BlackBerry MDS 服务为 BlackBerry 设备建立一个 SSL 连接。系统管理员配置 MDS 服务，要么允许连接到未信任的服务器，要么是限制访问信任的服务器，这个配置仅适用代理模式下的连接。在 end-to-end 模式里，BlackBerry 设备建立一个 SSL 连接。
在 BlackBerry 管理器里，系统管理员编辑 MDS 服务属性，并且设置 TLS 和 HTTP 选项，为获得更多信息，*参看 BES 管理指南*。

注：在 MDS 服务模拟器里，通过设置 rimpublic.property 文件来允许或拒绝访问未信任的服务器。设置 application.handler.https.allowUntrustedServer 参数为 true 或 false。参看 177 页的“配置 BlackBerry MDS 模拟器”获得更多信息。

如果 MDS 服务包含认证，MDS 服务将信任一个服务器。

使用 keytool 安装一个认证

1. 保存认证。
2. 将认证文件拷贝到 MDS 服务所在计算机上的 **C:\Java\j2re1.4.2\lib\security** 文件夹。
3. 为了将认证导入到键存储里，请使用 keytool，它可以在 JRE bin 文件夹下找到，例如 C:\Java\j2re1.4.2\bin.例如，输入：

keytool -import -file <cert_filename> -keystore cacerts

访问 http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html 获得更多关于 keytool 的信息。

编码转化器

BlackBerry MDS 支持插件的 Java 应用程序，称为编码转化器，它处理 BlackBerry 设备发送和接收的数据。
MDS 服务提供下列缺省的编码转化：

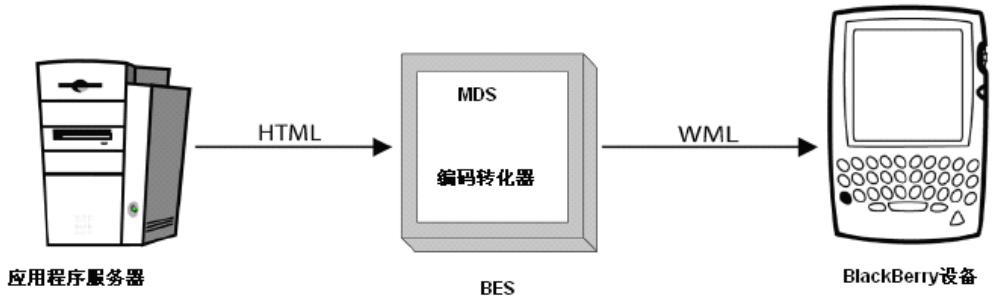
编码转化名	描述
WML>WMLC	将.wml(Wireless Markup Lanuage)转化为一个压缩的格式。
WMLScript>WMLScriptC	将 MIDlet 应用程序为 BlackBerry 格式。
JAD>COD :XML>WBXML	将 xml 文件转化为下面的 WAP 二进制 XML(.

	wbxml) MIME 类型:application/vnd.wap.wbxml..
SVG >PME	将可 svg(Scalable Vector Graphics)转化为 BlackBerry 设备支持的 pme(Plazmic Media Engine)二进制格式.
图像编码转化	将下列图像的文件类型转化为 png 文件: ● Jpeg ● Gif ● Tiff ● Pgm ● Ppm ● pnm:包含 ASCII 和二进制 pbm,pgm 和 ppm 文件 ● ico ● wbmp ● pbm ● wbmp ● bmp

注： 图像编码转化器将先前的图像格式转化为下面的.PNG MIME 类型:image/vnd.rim.png.

你也可以编写自己的编码转化器完成客户化数据的处理.例如,如果你的应用程序正在和服务器交换数据,而服务器不是为 BlackBerry 设备设计的,并且发送到服务器端的数据对 BlackBerry 设备来说没有合适的格式.那么你可能编写一个编码转化器.

一个编码器可能改变数据格式或删除冗余信息来降低网络流量,并且支持 BlackBerry 设备上的简单应用程序.例如,你可以编写一个编码转化器将 HTML 内容转化为 WML.



数据转化处理

如果你专门编写一个服务器端的应用程序支持一个 BlackBerry 设备上客户应用程序,编码转化器就不需要了.你可以设计一个服务器应用程序一合适的格式输出数据.
在发送数据到 BES 之前,你也可以将数据处理作为一单独的服务器端进程的一部分.

编码转化器 API

主要的编码转化器类	描述
HttpContentTranscoder	此类提供方法来控制 HTTP 请求和响应的内容和属性.
HttpContentTranscoderException	本异常的抛出说明编码转化工程不成功. BlackBerry 设备将把本异常作为 IOException 抛出.
HttpHeader	本类提供方法对 HTTP 请求和响应的消息头进行操作.
HttpParameter	本类代表了 HTTP 查询参数.它提供获取和设置参数名和值.
HttpRequest	本类扩展了 HttpTransmission 类,它代表了一个包含头,参数和内容的 HTTP 请求.
HttpResponse	本类扩展了 HttpTransmission 类,它代表了一个包含头,参数和内容的 HTTP 响应.
HttpTransmission	本类提供获取和设置内容,头,以及 HTTP 请求和响应的参数.

参看 *API 参考*获得更多信息.

创建一个 HTTP 内容编码转化器

为了创建一个 HTTP 内容编码转化器来全面控制 HTTP 请求与响应内容,扩展 HttpContentTranscoder 类.本编码转化器也可以修改,增加,或删除 HTTP 请求与响应属性.

编码转化器必须在 net.rim.protocal.http.content.tanscoder.<name>包里定义,在这里,<name>是映射文件中使用的编码转化器的标志符.它的类名必须是 Transcoder.

编码转化器的位置要么在 httpcontenttranscoder 文件指定,要么由应用程序指定.

处理 HTTP 内容编码转化器异常

任何由编码转化器抛出的异常将作为 IOException 被发送到应用程序.相关具体的消息也拷贝到 IOException 中.

注：为代替抛出一个异常到 HTTP 连接处理器, 一个编码转化器也可能发送一个 HTTP 响应到一个应用程序, 这表明发生了一个内部的服务器错误.

选择编码转化器

当 BlackBerry 设备请求内容时,MDS 服务决定客户端请求的内容类型和服务器响应的内容类型的比较是否需要一个编码转化器.如果类型不一样,MDS 服务检查 httpcontenttranscoder.property 文件来决定一个编码转化器是否可用来转化内容.

BlackBerry 设备发送的 HTTP 请求里,x-rim-transcode-content 头指定了 MDS 优化服务转化内容的输入格式.如果内容遇见以下情形,MDS 数据优化服务将转化内容:

- 由 x-rim-transcode-content 头指定的输入格式也 HTTP 请求的 Accept 头中指定,此 HTTP 请求是 MDS 服务发送到服务器的请求.
- Web 服务器响应一个内容类型,或者输出类型,它和 BlackBerry 设备接受的类型不同.
- httpcontenttranscoder.property 文件包含了一个编码转化器,此编码转化器可以将 Web 服务器发送到 MDS 服务的内容转化为 BlackBerry 设备可接受的类型.

为了判断当 push 内容时一个编码转化器是否需要,MDS 服务检查内容类型(content-type)和服务器端应用程序的 x-rim-transcode-content 头.如果 2 者相同,MDS 数据优化服务将头文件指定的内容转为为 BlackBerry 可以显示的格式.

httpcontenttranscoder.property 文件将输入的格式转化为一个给出的编码转化器.在下面的例子中, httpcontenttranscoder.property 文件将.wml 输入格式转化为.wbxml 或.wmlc.

```
content-type: text/vnd.wap.wml
x-rim-transcode-content: text/vnd.wap.wml
```

你也可以创建定制的编码转化器来重新格式化或者转化应用程序请求的数据.例如,有愧乐意编写一个编码转化器将 HTML 内容转为为 WML.参看 196 也的"创建编码转化器"获得更多信息.

HTTP 请求

MDS 服务使用 MIME 类型决定附加到一个服务器 HTTP 响应的内容格式是否和 BlackBerry 设备请求的内容格式相匹配.

MDS 服务测试一列可用的编码转化器,并且扩展 Accept 头.例如,如果一个请求接受 WML 的 MIME 类型和一个编码转化器存在的 HTML-to-WML,将 HTML 加到 Accept 头.参看 195 页的"映射编码转化器"获得更多信息.

在 HTTP 请求里,应用程序可以包含 2 个头来控制它接收的内容的类型:

- Accept 头
- Content-Transcoder 头.

头	描述
Accept	在 HTTP 请求的 Accept 头里,应用程序列出它从内容服务器接收的 MIME 类型.例如,如果应用程序可以接收 WML 或 XML 格式的内容,它在 HTTP 请求里发送如下的消息头: Accept:text/wml,text/xml 缺省的,当 Accept 头列出了不止一个 MIME 类型,MDS 服务将从左到右分配引用,在这里列出的第一个 MIME 类型假设为优先的. 应用程序页可以将质量因子(Quality factor)赋值给 MIME 类型来表明优先权.质量因子范围时 0(最低优先权)到 1(最高优先权),例如,下面的头表明了编译的 WML(WMLC)是优先的,但是 WML 是可接受的. Accept:text/vnd.wap.wml;q=0.5, application/vnd.wap.wmlc;q=1
Content-Transcoder	Content-Transcoder 头是一个 BlackBerry 头,在内容返回到 BlackBerry 设备之前,它允许应用程序指定一个应用到任何内容的特定编码转化器.使用 Content-Transcoder 头覆写 MDS 服务运用的缺省处理来选择一个编码转化器.

HTTP 响应

内容服务器产生一个包含一个 Content-Type 头的 HTTP 响应.. Content-Type 头描述了一个特定数据包的 MIME 类型.例如,当内容服务器一 HTML 格式返回内容时,HTTP 响应包含一个

值为 text/html 的 Content-Type 头.

MDS 服务将应用程序的请求和内容服务器的响应进行比较.下面的部分将描述 MDS 服务如何决定什么时候转化内容:

响应类型	描述
不需要编码转化	应用程序发送一个请求获取 WML 格式的内容.请求包含以下的头: Accept:text/vnd.wap.wml 在它的响应里,内容服务器发送以下的头: Content-Type:text/vnd.wap.wml 因为服务器返回的内容类型与 BlackBerry 请求类型相符,MDS 服务不需要任何转化.它将内容毫无变化的转发给 BlackBerry 设备.
需要编码转化	应用程序发动一个请求获取内容.请求包含一下内容: Accept: text/vnd.wap.wml; q=0.5, application/vnd.wap.wmlc; q=1 此消息头表明应用程序接受 WML 或者编译的 WML 格式(WMLC)的内容. 在它的响应中,内容服务器发送下面的头: Content-Type: text/vnd.wap.wml 因为服务器返回的内容类型与 BlackBerry 请求类型不符,MDS 服务搜索一个可以将 MIME 类型(WML)转化为优先的 MIME 类型(WMLC)的可用编码转化器. 如果一个适合的编码转化器不可用,MDS 服务将 WML 内容毫无变化的转发给 BlackBerry 设备,因为初始请求表明 BlackBerry 设备应用程序接受 WML 内容.

每个编码转化器都实现了一个方法创建一个哈希表,此哈希表映射作为来自内容服务器的输入,编码转化器接受的格式,以及作为输出编码转化器创建的格式.
在启动时,将自 BlackBerry 设备的 HTTP 请求转发之前,MDS 服务使用哈希表修改 Accept 头域.例如,一个 BlackBerry 设备应用程序使用下面的 Accept 头发送一个 HTTP 请求:

```
Accept: application/vnd.wap.wmlc, text/vnd.wap.wmlscriptc
```

在审查编码转化器映射表之后,MDS 数据优化服务加入下面的项到 Accept 头里:

```
application/vnd.wap.wmlscript, text/wml, and text/vnd.wap.wml
Accept: application/vnd.wap.wmlc, text/vnd.wap.wmlscriptc,
application/vnd.wap.wmlscript, text/wml, text/vnd.wap.wml
```

这个扩展的 Accept 头现在列出了所有内容服务器可以提供的 MIME 类型.

映射编码转化器

httpcontenttranscoder.property 文件在你的 BlackBerry JDE 安装目录的 MDS\config 子目录下.它指定了 MDS 数据优化服务如何管理应用程序与内容服务器之间的多种内容类型的交换.
下面是 httpcontenttranscoder.property 的一个实例:

```
text/vnd.wap.wml->application/vnd.wap.wmlc:vnd.wap.wml
text/vnd.wap.wml->application/vnd.wap.wbxml:vnd.wap.wml
text/vnd.wap.wmlscript->application/vnd.wap.wmlscriptc:wmls
```

```
text/html->application/vnd.rim.html.filtered:html
text/vnd.wap.wml:vnd.wap.wml
text/vnd.sun.j2me.app-descriptor->application/vnd.rim.cod:vnd.rim.cod
default:pass
```

每个入口使用了下面的格式:

```
<InputType> [-> <OutputType>]: <Action>
```

在这里:

- <*InputType*>是内容服务器可用的 MIME 类型,或者 default.
- <*OutputType*>是 BlackBerry 设备请求的 MIME 类型.
- <*Action*>是下列值之一:
  - 编码转化器包名,如 vnd.wap.wml
  - Pass:在没有改变下发送数据.
  - Discard:放弃数据,不发送它.

下面几节解释在 httpcontenttranscoder.property 文件里可能的输入格式:

格式 1

当服务器端可用的格式和 BlackBerry 设备请求的格式不同时,下面格式的入口指定了 MDS 完成的动作.

```
InputType -> OutputType:Transcoder OR RSV
```

例如,应用程序请求 text/wml,但是内容服务器只有 text/xml.MDS 服务找到下面的 MIME 类型入口:

```
Text/xml -> Text/wml : vnd.wap.wml
```

根据本入口,MDS 服务必须使用 vnd.wap.wml 编码转化器将 XML 内容转化为 WML.

格式 2

当服务器发送一个给定类型的内容,不考虑 BlackBerry 设备请求的内容,,下面格式的入口指定了 MDS 完成的动作.

```
InputType:Transcoder OR RSV
```

例如,如果服务器仅有 WML 内容(text/vnd.wap.xml), .MDS 服务找到下面的 MIME 类型入口:

```
text/vnd.wap.wml: vnd.wap.wml
```

根据本入口,MDS 服务必须使用 vnd.wap.wml 编码转化器将 XML 内容转化为 WML.

缺省的入口

如果 MDS 服务不能为一个特定的 MIME 类型:

```
default : Transcoder or RSV
```

找到一个入口,一个缺省的入口指定了 MDS 服务完成的缺省动作.

例如,如果 MDS 服务不能为 defualt:pass 缺省入口的内容找到一个入口,MDS 服务将内容毫无变化的转发给 BlackBerry 设备.

创建编码转化器

将 HTML 标记和内容转化为大写

遵循编码转化器包层次结构(hierarchy)

在 net.rim.protocol.http.content.transcoder.<transcoder name>包里定义 HTTP 内容转化器.所有 HTTP 内容编码转化器必须在 Transcoder 包层次结构里定义.

```
package net.rim.protocol.http.content.transcoder.uppercasehtml;
```

扩展 HTTPContentTranscoder

一个编码转化器的类名必须为 Transcoder.它扩展了 HttpContentTranscoder 类.

在本例中, 一个公共类 Transcoder 定义了 HTTP 内容编码转化器.

```
public class Transcoder extends HttpContentTranscoder { ... }
```

定义 HTTP 头

为 HTTP 头定义常数以及编码转化器加到这些头的字符串.

```
private static final String CONTENTTYPE_HEADER_NAME = "Content-Type";
private static final String CONTENTLENGTH_HEADER_NAME = "Content-
Length";
private static final String ACCEPT_HEADER_NAME = "Accept";
// Add this line to the Accept header field if it exists when
// the BlackBerry device issues an HTTP request.
private static final String ACCEPTLINE= "text/html";
// This line identifies the output content type that this transcoder
produces.
private static final String OUTPUT_TYPE= "text/UPPERCASEHTML";
```

创建一个输入和输出类型的映射

getMapOfOutputToAcceptLine()的实现创建了一个编码转化器输入和输出类型之间的映射.

随着启动,MDS 服务使用映射修改 Accept 消息头域.MDS 服务发送一个 HTTP 请求中修改的头到一个 HTTP 服务器.

```
public HashMap getMapOfOutputToAcceptLine() {
    HashMap mapping = new HashMap();
    mapping.put(OUTPUT_TYPE, ACCEPTLINE);
    return mapping;
    }
```

设置连接 URL

当 BlackBerr 设备请求一个打开的 HTTP 连接时,定义一个方法设置连接 URL.

```
public void setURL(URL newURL) {
    url = newURL;
```

```
}
```

定义 BlackBerry 设备的请求处理

在 MDS 服务将 BlackBerry 设备请求转发到目标服务器之前,transcodeDevice(HttpRequest)的实现定义了任何对这些请求的处理.下面的例子不需要处理:

一个应用程序可以使用一个称为 Content-Transcoder 的 HTTP 头请求一个指定的编码转化器,

```
public void transcodeDevice(HttpRequest request)
    throws HttpContentTranscoderException {
   // Implementation.
}
```

定义 BlackBerry 设备的响应处理

在 MDS 服务将 BlackBerry 设备响应转发到目标服务器之前, transcodeDevice(HttpResponse)的实现定义了任何对这些响应的处理.下面的例子不需要处理:

```
public void transcodeDevice(HttpResponse response)
   throws HttpContentTranscoderException {
    // Implementation.
}
```

定义服务器的请求处理

在 MDS 服务将它转发到 BlackBerry 设备之前, transcodeServer(HttpRequest)的实现定义了任何对内容服务器请求的处理需求.

```
public void transcodeServer(HttpRequest request)
  throws HttpContentTranscoderException {
    try {
        // Retrieve the request content, which, in this case, is in
HTML.
        byte[] requestContent = request.getContent();
        if (requestContent != null) {
            // Convert the content to String object.
            String requestContentAsString = new
            String(requestContent).toUpperCase();
            // Convert the requestContentAsString to bytes again.
            requestContent = requestContentAsString.getBytes();
        }
        else {
            // Send an HTML message to indicate that the server
            // is not responding with appropriate content.
            StringBuffer sb = new StringBuffer();
            sb.append("<HTML>\n");
            sb.append("<HEAD>\n");
            sb.append("<TITLE> UPPERCASEHTML TRANSCODER</title>\n");
            sb.append("</HEAD>\n");
            sb.append("<BODY>\n");
```

```
            sb.append("SERVER IS NOT PUSHING APPROPRIATE CONTENT\n");
            sb.append("</BODY\n");
            sb.append("</HTML>\n");
            requestContent = sb.toString().getBytes();
            }

        request.setContent(requestContent);
        // Update the Content-Length
        HttpHeader contentLengthHeader =
            request.getHeader(CONTENTLENGTH_HEADER_NAME);
        if (contentLengthHeader != null) {
            contentLengthHeader.setValue("" + requestContent.length);
            }
        else {
            // The server did not send the Content-Length.
            // No update is needed.
            }

        // Update the Content-Type.
        HttpHeader contentTypeHeader =
            request.getHeader(CONTENTTYPE_HEADER_NAME);
        if (contentTypeHeader != null) {
            contentTypeHeader.setValue(OUTPUT_TYPE);
            }
        else {
            // Add the Content Type here if the server does not
specify one.
            request.putHeader(new HttpHeader(
                    CONTENTTYPE_HEADER_NAME, OUTPUT_TYPE));
            }
        }
    catch (Throwable t) {
        throw new HttpContentTranscoderException(t.toString());
        }
    }
```

定义服务器的响应处理

在 MDS 服务将它转发到 BlackBerry 设备之前, transcodeServer(HttpResponse)的实现定义了包含来自于内容服务器的内容服务器响应的任何处理需求.如果响应不包含任何内容,MDS 服务将响应毫无变化的转发给 BlackBerry 设备.

```
public void transcodeServer(HttpResponse response)
   throws HttpContentTranscoderException {
    try {
        // Retrieve the response content, which in this case is in HTM
```

```
L.
        byte[] responseContent = response.getContent();
        if (responseContent != null) {
            // Convert the content to String object.
            String responseContentAsString = new
            String(responseContent).toUpperCase();
            // Convert the responseContentAsString to bytes again.
            responseContent = responseContentAsString.getBytes();
            }
        else {
            // No response is received from the server.
            StringBuffer sb = new StringBuffer();
            sb.append("<HTML>\n");
            sb.append("<HEAD>\n");
            sb.append("<TITLE> UPPERCASEHTML TRANSCODER </title>\n");
            sb.append("</HEAD>\n");
            sb.append("<BODY>\n");
            sb.append("SERVER IS NOT RESPONDING\n");
            sb.append("</BODY\n");
            sb.append("</HTML>\n");
            responseContent = sb.toString().getBytes();
            }
        response.setContent(responseContent);
        // Update the Content-Length.
        HttpHeader contentLengthHeader =
            response.getHeader(CONTENTLENGTH_HEADER_NAME);
        if (contentLengthHeader != null) {
            contentLengthHeader.setValue("" + responseContent.length);
            }
        else {
            // Server did not send Content-Length so no update is
required.
            }

        // Update the Content-Type.
        HttpHeader contentTypeHeader =
            response.getHeader(CONTENTTYPE_HEADER_NAME);
        if (contentTypeHeader != null) {
            contentTypeHeader.setValue(OUTPUT_TYPE);
            }
        else {
            // Add the Content Type here.
            response.putHeader(new
                    HttpHeader(CONTENTTYPE_HEADER_NAME, OUTPUT_TYPE));
```

```
            }
        }
    catch (Throwable t) {
        throw new HttpContentTranscoderException(t.toString());
        }
    }
```

编译和安装编码转化器

1. 编译编码转化器的类文件,在类路径里包含 **bmds.jar**.
2. 为编译编码转化器创建一个 jar 文件.
3. 以下面的任一方式安装编码转化器.jar 文件
   - **BES**:将一个编码转化器.jar 文件加到 JRE 的 **lib\jar** 目录下.此.jar 文件必须可访问 Java VM.
   - **模拟器**:将一个编码转化器.jar 文件加到 BlackBerry JDE 的 **MDS\classpath** 目录下.
4. 打开 **httpcontenttranscoderslist.property** 文件,此文件在 BlackBerry JDE 安装目录的 **MDS\Config** 子目录下.
5. 为了指定何时使用编码转化器,加入一个或多个入口,.例如,为了使用 uppercasehtml 编码转化器将 HTML 内容转化为大写的 HTML,加入下面的入口:

   text/html -> text/UPPERCASEHTML : uppercasehtml
6. 保存属性文件.
7. 重启 MDS 服务或 BlackBerry MDS 模拟器.

术语表

A

ALX
: Application Loader XML

APDUs
: Application Protocol Data Units

APN
: Access Point Name

ATR
: Answer to reset

C

CA
: Certificate Authority

cHTML
: Compact Hypertext Markup Language

CLDC
: Connected Limited Device Configuration

D

DSA
: Digital Signature Algorithm

G

GUI
: graphical user interface

GUID
: globally unique identifier

I

i18n
: internationalization

IP
: Internet Protocol

IPPP
: IP Proxy Protocol

ISDN
: Integrated Services Digital Network

J

JAD
: Java Application Descriptor

JAR
: Java Archive

JDWP
: Java Debug Wire Protocol

JRE
: Java Runtime Environment

JTWI
: Java Technology for the Wireless Industry

K

KVM
: Kilobyte virtual machine

L

LDAP
: Lightweight Directory Access Protocol

LTPA
: Lightweight Third-Party Authentication

M

MB
: Megabyte

MHz
Megahertz

MIDlet
MIDP application

MIDP
Mobile Information Device Profile

MIME
Multipurpose Internet Mail Extensions

O

OCSP
Online Certificate Status Protocol

P

PAP
Password Authentication Protocol

PIM
personal information management

PIN
personal identification number

PNG
Portable Network Graphics

R

RRC
Radio Resource Control

RSA
Public-key Encryption algorithm

RTC
real-time clock

S

SDK
software development kit

SIM
Subscriber Identity Module

Smart Card
A device used to securely store and transfer data

Smart Card Reader
A wireless smart card reader for Bluetooth enabled BlackBerry devices

SMS
Short Message Service

SRAM
static random access memory

SRP
Service Relay Protocol

SSL
Secure Sockets Layer

S/MIME
Secure Multipurpose Internet Mail Extensions

T

TCP
Transmission Control Protocol

TIFF
Tag Image File Format

TLS
Transport Layer Security

U

UDP
User Datagram Protocol

URI
Uniform Resource Identifier

URL
: Uniform Resource Locator

UTC
: Universal Time Coordinate

W

WAP
: Wireless Application Protocol

WBMP
: wireless bitmap

WML
: Wireless Markup Language

WMLC
: Wireless Markup Language Compiled

WTLS
: Wireless Transport Layer Security

索引

C

D

E

F

G

N

O

P

Q

R

S

T

U

V

W

X

Z