はじめに

本書は、「i モード Java プログラミング」シリーズの FOMA 対応版として、NTT ドコモが満を持して発売した FOMA 900i シリーズの最新機能とプログラム事例、そして API リファレンスから各種参考資料までを１冊にまとめたものだ。

ご存じのように、携帯電話を取り巻く状況や環境は、依然として活発に動いている。「パソコン」から「デジタル情報家電」に世の中のトレンドが大きく移り変わろうとしているいま、ケータイはこれまで以上にその存在感を増しつつある。NTT ドコモをはじめとするキャリア各社は、3G のインフラの上にケータイのマルチメディア化（パソコン化）をさらに進めるだけでなく、ケータイに決済機能を持たせる FeliCa 搭載やデジタル情報家電のコントローラとしての付加価値を加え、生活インフラとしての位置づけをより強固なものにしようと次々と新製品を投入している。

503i シリーズから搭載されている Java も着実にバージョンアップを重ね、FOMA 900i シリーズは、携帯端末の Java としてほぼ完成形に近づいたと言ってよいだろう。「ドラクエ」を動かすことを目標にしたというだけのことはあり、ファミコンを越えるスペックがケータイの中に入ったことは、503i シリーズのミニマムなスペックからは隔世の感がある。3G のインフラだから実現できた高速、大容量のパケット通信と相まって、本格的な Java プログラムの実行環境がポケットに入る時代になったのだ。さらに、「パケ・ホーダイ」というパケット定額制の料金オプションが追加されたことにより、通信機能を活用する i アプリが気軽に使えるようになったことにも注目したい。カメラ連携やバーコード認識などのネイティブアプリケーションと i アプリの連動機能はより強化されており、ゲームだけでなく実用アプリケーションの幅も大きく広がっている。

このように、ケータイの進化はとどまることを知らないが、国内のケータイ市場は飽和状態を迎えつつあり、大きく販売台数を伸ばしていくのは難しい状況にある。そういったなかでこれからは、生活ケータイとしてより個々人に密着したカスタマイズされた端末が求められるのではないだろうか。その意味で、Java の果たす役割は依然大きいと思われる。今後も、これまで以上にさまざまなジャンルの i アプリが続々登場することを期待したいところだ。

本書は、2002 年に刊行した 504i シリーズ対応の「i モード Java プログラミング」の構成をベースに、「FOMA 対応版」として大幅に内容の改変と追加を行ったものだ。DoJa のバージョンは 3.5 を基本に記述しているが、505i 系の DoJa3.0 やそれ以前の機種にも対応している。また、これから i アプリのプログラミングをはじめる読者のために「基礎編」として Java および i アプリの基本的な事項もまとめており、i アプリプログラミングの入門書としても活用できるように配慮した。本書が、ますます可能性を広げつつある携帯 Java プログラミングの一助になれば幸いだ。

2004 年 6 月　アスキー書籍編集部

本書の構成

本書は、「i モード Java プログラミング」シリーズの FOMA 対応版として、FOMA 900i シリーズの新機能を中心とした解説と API リファレンスや各種参考資料など、この 1 冊で FOMA 900i のすべてがわかる構成となっています。開発環境としては DoJa3.5 を対象にしていますが、505i/506i 系の DoJa3.0 やそれ以前の機種にも対応しているなど、FOMA 以外でも活用できるように配慮しました。

本書は、読者対象として Java のプログラミング経験があることを前提としていますが、本書だけでも i アプリのプログラミングを学ぶことができるように、はじめて i アプリおよび Java に触れる読者のための最小限必要な基礎編を設けています。ただし Java を基礎から学習したい方は、「入門 Java」（2002 年、株式会社アスキー）などの入門書を別途お読みになることをお薦めいたします。また、FOMA 900i シリーズ以前のバージョンに対応した i モード Java プログラミングに関しては、本書では詳しく取り上げていませんので、必要に応じて前書「i モード Java プログラミング」シリーズをご覧ください。

また、本書には開発環境は付属していませんので、本文を参照して該当の Web ページより開発環境をダウンロードする必要があります。あらかじめご了承ください。

- **NTT ドコモの Web ページより、「DoJa3.5 プロファイル向け i アプリ開発ツール」をダウンロードする（10.7MB）**
- **サンの Web ページより、「J2SE v1.3.1」をダウンロードする（31.78MB）**

■ 本書の内容

本書の各 Part の概要を以下に紹介します。なお本書は、11 名の著者が担当分野ごとに執筆しています。巻末の著者紹介をご覧ください。

▪ Part1　準備編

i アプリの位置づけや FOMA 900i 対応版 Java の新機能の概要と、開発環境のセットアップ方法や使い方、i アプリの開発手順を解説します。

▪ Part2　基礎編

はじめて i アプリおよび Java に触れる読者を対象とし、最小限必要な Java の基礎事項と i アプリのプログラミング手法について解説します。本 Part の解説は、一部を除き 503i 以降のすべての機種に対応しています。

▪ Part3　開発編

DoJa2.0 から最新版の DoJa3.5 まで順次追加されてきた主要な拡張 API を中心に、サンプルプログラムを交えながら、その機能を詳細に解説します。

▪ Part4　プログラム事例編

具体的なサンプルプログラム事例を紹介します。この Part で取り上げるサンプルプログラムはすべて弊社 Web ページよりダウンロードできますので、後述する URL を参照してください。

■ Part5　i モード Java 拡張 API リファレンス

i モード Java 拡張 API の詳細なリファレンスを掲載しています。DoJa3.0 をベースに解説し、DoJa3.5 で追加になった API は一目でわかるように配慮しました。なお、トラステッド i アプリ（NTT ドコモに登録が必要な i アプリ）に関連する項目に関しては本 Part では割愛しています。また、com.nttdocomo.opt.ui.j3d2 パッケージに関しても、現時点では D505iS ／ D505i ／ D504i のみの対応となっているため、本 Part では割愛しています。

■ Part6　参考資料

ADF ファイルのエントリ、機種実装依存一覧などプログラミングに役立つ各種の参考資料を掲載しています。

■ サンプルプログラムについて

本書で紹介するサンプルソースコードは、以下の Web ページよりダウンロードできます。なお、本書で掲載した各プログラムは、F900i、N900i、P900i、SH900i、D900i、および一部の 505i シリーズの機種で動作確認を行っています。それ以降の新機種ではチェックを行っていません。ただし、最新版へのアップデート情報などは、下記のサイトで随時公開します。

http://www.ascii.co.jp/pb/ant/foma/

また、Part4 に掲載した i アプリは下記より携帯電話にダウンロードしてすぐに試すことができます。下記 QR コードからも直接ダウンロードページにアクセスできます。

http://www.ascii.co.jp/pb/ant/foma/i/

なお、上記サイトは読者の便宜のためにあくまで動作確認のテスト用として公開しているものです。実用を保証するものではありません。一部 CGI 等を利用する i アプリに関してはサーバーに負荷がかかる可能性があるため、ユーザー数の制限を設けたり、利用を停止する場合があります。また**上記サイトは、予告なく内容の変更、中断、または中止する場合があります。**あらかじめご了承ください。

目次

Part 1 準備編

Part 2 基礎編

Part 5 i モード Java 拡張 API リファレンス

com.nttdocomo.ui

com.nttdocomo.util

com.nttdocomo.io

com.nttdocomo.net

com.nttdocomo.system

com.nttdocomo.lang

com.nttdocomo.device

例外

com.nttdocomo.opt.device

com.nttdocomo.opt.ui

com.nttdocomo.opt.ui.j3d

Part 6 参考資料

第1章 iモードJavaとは

本書は、iモードJavaすなわちiアプリの作成方法を、豊富な実例を示しながら解説する書籍です。本章では、まず、iモードやiアプリとは何であるかを解説します。その後、iアプリの進化の軌跡や、第三世代携帯電話であるFOMA 900iシリーズの特徴や以前のiアプリとの違いを解説します。

1.1 iモードとiアプリ

iモードは、1999年2月に株式会社NTTドコモ（以下、NTTドコモ）が開始した携帯電話上のインターネットサービスです。電話機にメール機能やWebブラウザが内蔵され、公式サイトのみならず一般サイトも多く生まれて、携帯電話機を単なる音声通話のための道具以上の新たなコミュニケーションツールへと生まれ変わらせることになりました。PCにおけるインターネットと異なり、プロバイダとの契約や面倒な設定が必要なく、手軽にいつでもどこでもネットワークにつながるということも手伝って、急速に契約者数を増加させ、2003年12月現在で4,033万人もの人がiモードを利用しています（最新の情報はhttp://www.nttdocomo.co.jp/corporate/report/contract/index.htmlにて入手できる）。NTTドコモ以外の携帯電話会社も同様のサービスを開始させ、携帯電話のインターネット機能は、仕事や生活において不可欠なインフラになったと言えるでしょう。

iモードサービスの開始以降、携帯電話機自体の機能も大きな進歩を遂げています。当初は狭い白黒の画面で、着信音も3和音程度であったものが、カラー化され、画面サイズが大きくなり、64和音が出せるようになりました。また、カメラ付き携帯電話が標準となり、100万画素を越えるカメラや、動画の撮影機能なども一般的になっています。その他、GPS機能やFlashの再生など、多くの新機能が次々と盛り込まれてきました。

しかしながら、これらはいずれも携帯電話機に組み込みの機能です。従って、自分のWebサイトを作成してそれを携帯電話上に表示させることはできても、自分の好きなソフトウェアをダウンロードしてインストールしたり、自ら作成したソフトウェアを携帯電話機上で動作させることはできませんでした。いわば機能が固定された専用機であったわけです。PCでは、自分の好きなソフトウェアをインストールして利用するのが一般的ですが、携帯電話機はワープロ専用機のように機能が固定されていたというわけです。

これに対し、2001年1月に発売された503iシリーズの携帯電話機では、自由にソフトウェアを作成して配布し、ユーザーがダウンロードして使用できる機能が搭載されました。これが「iアプリ」です。iアプリの登場によって、携帯電話機はようやくPCのように自由にソフトウェアを開発したりダウンロードしたりできる時代になったわけです[*1]。iアプリでは、配布にあたって必要なものは、通常のWebサイトを

＊1　ちなみに、iアプリとは無関係だが、2003年10月以降、携帯電話機に組み込みの機能を無線通信でダウンロー

公開するサーバーだけであり、NTT ドコモの許可を得たり、特定のサーバー上でしか配布できないといった制約もありません（**図 1-1** 参照）。このため、数多くの無料の i アプリが作成されており、2004 年 4 月現在、その総数は 6,000 個を越えています。

しかしながら、電話番号などの個人情報を有する携帯電話機上で、ソフトウェアが PC 上と同じようにどんなデータでも取得できてしまうと、個人情報の流出や悪意を持ったウィルスなどといった問題が生じることになりかねません。そこで、i アプリでは、開発のためのプログラミング言語として、セキュリティ面での配慮のしやすい Java が選択されました。Java はネットワークとも親和性が高く、無線通信機器である携帯電話機の特性に適していたとも言えます。

503i シリーズ発売当初は、配布するプログラムのサイズが 10 キロバイト（1 キロバイトとは 1,024 バイト。以下 KB と表記）以下と制限されており、また、機能的にも基本的な機能に限られていましたが、その後、504i、504iS、505i と新しい端末機が発売されるに伴い、配布ファイルのサイズが 30KB まで拡張され、待ち受けアプリや、赤外線通信、組み込みのブラウザなど（以下ネイティブアプリと言う）との連携機能、三次元 CG、カメラ制御、コードリーダーなど拡張が施されてきました。画面サイズの拡大や、液晶の精細化も伴い、i アプリの表現力はどんどん向上しています。

こういった機能を、NTT ドコモでは、開発者向けに Docomo Java（**DoJa** と略記される）という形で整理しています。503i シリーズでの機能を DoJa1.0、504i シリーズを DoJa2.0、505i シリーズを DoJa3.0 と呼び、DoJa2.1 など細かなバージョンも存在します。**表 1-1** に DoJa の主要なバージョンと対応する電話機の型名、および主な機能をまとめてあります。

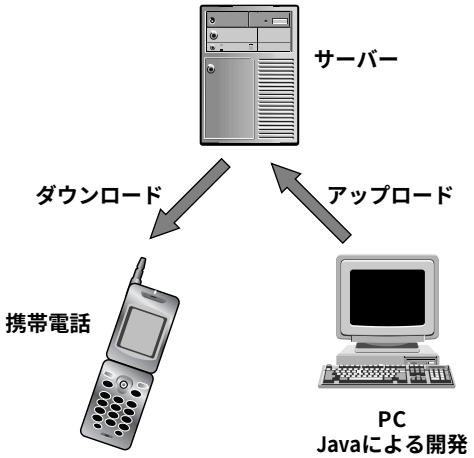


図 1-1　i アプリの開発と実行

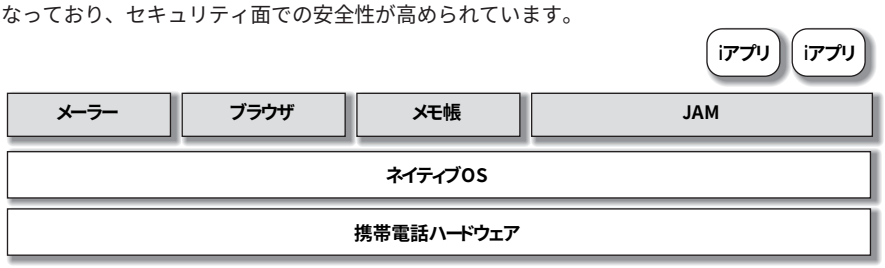
ドしてバージョンアップできるソフトウェア更新機能が搭載されている。ただしこの機能は、電話機組み込みのソフトウェアに不具合があったときに差し替えるためのもので、一般の開発者が自分の作成したソフトウェアを配布する目的には利用できない。

表 1-1　DoJa のバージョンと対応する電話機・主な機能

バージョン	機種名	主要な機能
DoJa1.0	503i	基本機能のみ。配布ファイルのサイズ 10KB 以内
DoJa2.0	504i/504iS	待ち受け、赤外線通信、カメラ制御、ネイティブ機能との連携、三次元 CG、配布ファイルのサイズ 30KB 以内
DoJa3.0	505i/505iS	バーコード、QR コードリーダー、ネイティブ連携機能強化、赤外線リモコン
DoJa3.5	900i	配布ファイルサイズ 100KB 以内

ネイティブアプリと i アプリ

携帯電話機でも PC と同様に、OS が存在します。そして OS の上で、メーラ、Web ブラウザ、メモ帳などといった各種の組み込みアプリが動作しています。これらの OS の上で直接動作するアプリをネイティブアプリと言います。ネイティブアプリは、電話機の開発者以外は作成できません。これに対し、i アプリは、OS の上で直接動くのではなく、JAM（Java Application Manager）という、i アプリを制御する特殊なネイティブアプリの上で動作しています。i アプリの動作はすべて JAM が監視しているので、不正な操作を行うことができなくなっており、セキュリティ面での安全性が高められています。



1.2 第三世代携帯と 900i

こうして、携帯電話機の高機能化と i アプリの機能強化が進むにつれて、通信されるデータ量も増えていきました。しかしながら、データの量が増えると通信が完了するまで時間がかかり、利用料金もそれに比例して増大することになります。

こういった流れの中で、従来の携帯電話機よりも高速度で通信できる新しい通信環境が求められるようになってきました。そこで登場したのが第三世代携帯電話である FOMA です。従来型の携帯電話通信は第二世代と呼ばれていましたが、それとまったく異なる通信方法を採用することによって、通信速度が 28.8Kbps から最大 384Kbps へと大きく向上し、これに伴ってパケットあたりの通信料金も値下げされました。**表 1-2** に第二世代の主要端末における通信速度と、FOMA の通信速度を比較してあります。

表 1-2　主な i アプリ機能搭載機の最高通信速度

機種名	速度
503i（第二世代）	9.6Kbps
504i（第二世代）	28.8kbps
900i（第三世代）	384Kbps

第 5 章
i アプリ プログラミング入門

ここでは、Java の基礎的な知識を持つことを前提に i アプリの作成方法を順に解説していきます。基本的に、本章で述べる内容は 503i で動作するものです。503i 以降の機種で実装された内容も一部含みますが、その場合には［504i 以降］、［505i 以降］などと明記します。

また本章では、ソースコードを示す際に、各節で解説した内容をアミ掛けで表し、特に本文中で番号を列挙する形式で解説した場合には、対応する番号をソースコード内の該当箇所に付します。本文中で列挙しない形式の説明がある場合には丸数字またはアルファベットで記し、ソースコード内で対応する箇所に表示します。

5.1 キャンバス型 i アプリとパネル型 i アプリ

i アプリは、必ず **IApplication** クラスを継承して作成します。この継承元すなわちスーパークラスである IApplication クラスに、i アプリが持つべきさまざまな機能が詰め込まれているので、開発者はそれを使いまわせばよいというわけです。

i アプリを実行すると、最初に **start() メソッド**が呼び出されると決まっています。この start() メソッド内で、画面表示などの基本的な処理を行うことになります。従って、i アプリを開発するには、以下の手順に従います。

1. i アプリのアプリケーションクラスとして、IApplication クラスのサブクラスを作成
2. start() メソッドを実装

実際にプログラムを作成する場合には、画面上にさまざまな描画を行う必要があります。しかし、IApplication クラスには描画の機能はいっさい用意されていません。実は IApplication クラスは絵画の額縁のようなもので、それ自体には描画機能はなく、別に画用紙のようなものを用意してそこに描画する必要があります（**図 5–1** 参照）。

この画用紙の機能を持つのが、「Canvas クラス」と「Panel クラス」です。Canvas クラスは実際の画用紙のようにいろいろな描画ができますが、Panel クラスでは描画機能は限られており、代わりに文字の入力欄や一覧表といった便利な部品が用意されていて、それらをパネル上に並べて使うようになっています。Canvas クラスを用いて作成した i アプリを**キャンバス型**と言い、Panel クラスを用いて作成したアプリを**パネル型**といいます。キャンバス型とパネル型の例を**図 5–2** に示します。

Panel と Canvas の機能を同一の画面で実現することはできません。たとえば、入力欄があって、そこに文字や数値を入力すると入力欄の真下に円グラフが表示されるといった i アプリを作ることはできません。ただし、Panel と Canvas を切り替えて表示する i アプリを作成することは可能です。入力欄があって数値や文字を入力すると、画面が切り替わって円グラフだけが表示されるといった i アプリを作成することは

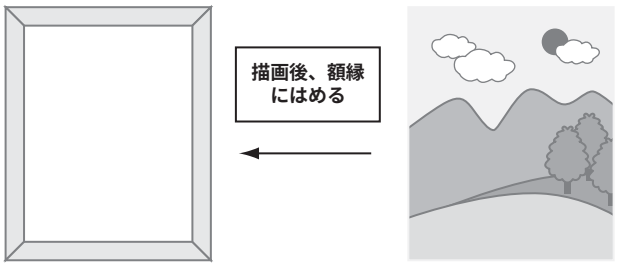


図 5–1　IApplication クラスと Canvas クラスの役割

図 5–2　キャンバス型アプリとパネル型アプリの例

できます。

もしくは、キャンバス型アプリとして作成し、入力欄なども自分で描画することも可能です。504i までは、キャンバス型アプリでは日本語の入力ができずあまり実用的ではなかったのですが、505i 以降では imeOn というメソッドで日本語の入力もできるようになっています（5.2.5 を参照）。

5.2 キャンバス型 i アプリ

それでは、「キャンバス型アプリ」から解説していきましょう。

5.2.1 キャンバス型 i アプリの基本

キャンバス型 i アプリは、以下の手順で作成します。

1. 額縁である IApplication クラスとは別に、画用紙である Canvas クラスを継承させたクラスを作成
2. それを額縁である IApplication クラスにはめ込む

リスト 5–1 がその例です。プログラムのポイントを解説します。

① import 文により、Canvas クラスを使用できるようにする

② Canvas クラスを継承して、MyCanvas クラスを作成。Canvas クラスは、paint メソッドをオーバーライドしなければならない（Canvas クラスは描画するためのものなので、描画をしないことはありえないため）。ここでは、中身は何もないままでオーバーライドだけしておく

③ CanvasSample クラスの start メソッド内で、 MyCanvas クラスをインスタンス化して myCanvas というインスタンスを作成

④ CanvasSample クラスの start メソッド内に、Display.setCurrent(myCanvas) と記述し、myCanvas を携帯電話の画面上に表示する。これが額縁に画用紙を入れる処理に相当する

リスト 5-1 を実行すると、**図 5-3** のようにエミュレータの液晶画面部分が白で表示されます。上部と下部の一部だけ青が残りますが、ここは電波状態を表すアイコンや後述するソフトキーの表示がされる部分で額縁の一部分といえます。画用紙に相当する myCanvas が、額縁に相当する IApplication クラスを継承した CanvasSample にはめ込まれていますが、実際には何も描画していないため真っ白な画面になっているわけです。とりあえず **オンフック** キーをクリックして強制終了しましょう。

リスト 5-1　CanvasSample.java ／ Canvas を用いる最も簡単な例

```
import com.nttdocomo.ui.*;//---①

//↓額縁の部分
public class CanvasSample extends IApplication {

  public void start() {
    System.out.println("Start IApplication");
    MyCanvas myCanvas = new MyCanvas();//----③
    Display.setCurrent(myCanvas);//----④
  }
                          ↑
                       ②で作成した画用紙部分である MyCanvas を
                       インスタンス化し、額縁にはめ込んでいる
}
//↑額縁の部分

//↓画用紙の部分
class  MyCanvas  extends   Canvas {
  public void paint(Graphics g) {
  }
}
//↑画用紙の部分
```

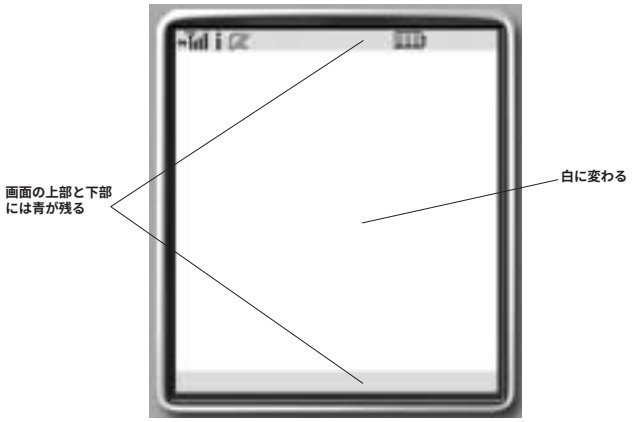


図 5-3　リスト 5-1 の実行画面

次に、Canvas 上に描画をします。描画の処理には paint メソッドを用います。この paint メソッドは、以下のような構造をしています。

```
public void paint(Graphics g)
```

引数 g：これに、表 5-1 で定義されているメソッドを作用させて描画する

引数である g に対して描画処理を行うメソッドを作用させることで描画が実現します。ここでは、線を描くメソッドである drawLine を使います。drawLine メソッドの書式は、次のとおりです。

```
public void drawLine(int x1, int y1, int x2, int y2)
```

第一引数 *x1*：描き始めの点の X 座標
第二引数 *y1*：描き始めの点の Y 座標
第三引数 *x2*：描き終わりの点の X 座標
第四引数 *y2*：描き終わりの点の Y 座標

リスト 5-1 に次の修正を施して**リスト 5-2** を作成すると、**図 5-4** のように線が表示されます。

1. paint メソッド内で「g.drawLine(10,10,100,100);」を記述
2. CanvasSample クラスの start メソッド内の「System.out.println("Start IApplication");」を削除

リスト 5-2　CanvasSample.java ／線を描く機能を追加

```
import com.nttdocomo.ui.*;


public class CanvasSample extends IApplication {

  public void start() {
    MyCanvas myCanvas = new MyCanvas();
    Display.setCurrent(myCanvas);
  }

}
```

```
class  MyCanvas  extends   Canvas {
  public void paint(Graphics g) {
    g.drawLine(10,10,100,100); //  -------  1
  }
}
```

図 5-4　リスト 5-2 の実行画面

drawLine 以外のよく使われる描画関係のメソッドを**表 5-1** に示します。各メソッドの引数については、「Part5 i モード Java 拡張 API リファレンス」を参照してください。

表 5-1　Graphics オブジェクトの描画関連の主なメソッド

Graphics クラスのメソッド	機能
clearRect	矩形領域の背景色での塗りつぶし
drawImage	画像の描画
drawLine	直線の描画
drawRect	矩形の描画
drawArc	円弧の描画［505i 以降］
drawString	文字列の描画
fillPolygon	多角形の塗りつぶし
fillRect	矩形領域の塗りつぶし
fillArc	円弧の塗りつぶし［505i 以降］
getColorOfName	色名指定によるカラー整数値の取得
getColorOfRGB	RGB 値指定によるカラー整数値の取得
setColor	描画色の指定
setFont	描画フォントの指定
lock	画面の固定
unlock	画面の固定解除

なお、複雑な描画を行うと処理に時間がかかり、描画の途中経過が見えてしまうことがあります。そういった状況を防ぐため、次の処理が一般的に行われます。

1. 描画の前に「g.lock();」として画面を固定
2. すべての描画処理が終了してから、「g.unlock(true);」で描画内容を一気に反映させる

リスト 5-2 の paint メソッド内を書き換えると、**リスト 5-3** のようになります。

リスト 5-3　画面をロックして描画の途中経過を見えなくする

```
class  MyCanvas  extends   Canvas {
  public void paint(Graphics g) {
    g.lock();// -------1
    g.drawLine(10,10,100,100);
    g.unlock(true);// -------2
  }
}
```

5.2.2 キャンバス型 i アプリにおけるキー操作

実際のゲームでは、キー操作によってキャラクタを移動させるといったことが一般的に行われます。キー操作が行われると、Canvas クラスの processEvent メソッドが呼び出されます。このメソッドは、次の構造を持ちます。第一引数は**表 5-2** に示される値をとります。

```
public void processEvent( int type, int param)
```

第一引数 *type*：キーが押されたか離されたかなどのタイミングを表す

第二引数 *param*：どのキーが押されたかなど詳細内容を表す

表 5-2　processEvent メソッドの第一引数の取り得る値

type の値	意味
Display.KEY_PRESSED_EVENT	キーが押されたとき
Display.KEY_RELEASED_EVENT	キーが離されたとき
Display.RESUME_VM_EVENT	レジュームイベント
Display.TIMER_EXPIRED_EVENT	タイマーイベント
Display.UPDATE_VM_EVENT	アップデートイベント
Display.MEDIA_EVENT	メディアイベント（505i 以降では発生しない）
Display.RESET_VM_EVENT	リセットイベント（505i 以降では発生しない）

第二引数の意味は、第一引数である type の内容によって変わりますが、Display.KEY_PRESSED_EVENT および Display.KEY_RELEASED_EVENT の場合には、**表 5-3** に示された値をとります。

表 5-3　processEvent メソッドの第一引数が Display.KEY_PRESSED_EVENT または Display.KEY_RELEASED_EVENT のときの第二引数の取り得る値

param の値	意味
Display. KEY_SOFT1	ソフトキー1（左側のソフトキー）
Display. KEY_SOFT2	ソフトキー2（右側のソフトキー）
Display.KEY_UP	↑ キー
Display.KEY_DOWN	↓ キー
Display.KEY_RIGHT	→ キー
Display.KEY_LEFT	← キー
Display.KEY_SELECT	セレクト キー
Display.KEY_POUND	# キー
Display.KEY_ASTERISK	* キー
Display. KEY_0～Display. KEY_9	数字 キー

ソフトキー2を放した時に終了するiアプリのコードを、**リスト 5-4** に示します。A の部分が終了させるコードです。終了させる機能は Canvas クラスにはなく、額縁である CanvasSample クラスが有する terminate メソッドを呼び出して行います。

しかし、ソフトキー2が押されたときの処理は Canvas クラスを継承してできた MyCanvas クラスが持ちますので、なんらかの方法で額縁である CanvasSample クラス（正確にはそのインスタンス）を取得しなければなりません。これには、IApplication クラスの静的メソッドである getCurrentApp を用います。これは現在動作中の i アプリ、すなわち CanvasSample クラスを得るメソッドです。そして、IApplication.getCurrentApp で得たインスタンスの terminate メソッドを呼び出して終了させています。

リスト 5-4　CanvasSample.java ／ ソフトキー2 で終了

```
import com.nttdocomo.ui.*;

public class CanvasSample extends IApplication {

  public void start() {
    MyCanvas myCanvas = new MyCanvas();
    myCanvas.setSoftLabel(Frame.SOFT_KEY_2);//----B
    Display.setCurrent(myCanvas);
  }

}


class  MyCanvas  extends   Canvas {
  public void paint(Graphics g) {
    g.lock();
g.drawLine(10,10,100,100);
g.unlock(true);
  }

  public void  processEvent( int type, int param) {
```

```
    if (type==Display.KEY_RELEASED_EVENT) {
      if  (param==Display.KEY_SOFT2) {
        IApplication.getCurrentApp().terminate(); // ----A
        }
    }
  }
}
```

ソフトキー2による終了が可能になったので、ソフトキー2に「終了」と表示させたほうがわかりやすいでしょう（**図 5-5**）。これには、Canvas クラスの setSoftLabel メソッドを用います。setSoftLabel メソッドは、次の構造をしています。

```
public void setSoftLabel(int key, String label)
```

第一引数 *key*：Frame.SOFT_KEY_1 または Frame.SOFT_KEY_2 を指定して、ソフトキー1、ソフトキー2のどちらに文字を設定するかを指定

第二引数 *label*：設定する文字を指定

第一引数には、Canvas クラスの継承元クラスである Frame クラスで定義されている Frame.SOFT_KEY_2 を使用します。processEvent メソッドのときのように、Display.KEY_SOFT2 を用いるのではないことに注意してください（**リスト 5-4** の B を参照）。

図 5-5　リスト 5-4 の実行画面

5.2.3 ShortTimer の利用

キャンバス型の i アプリを作成する際に、一定時間ごとになんらかの動作をさせたいことがよくあります。たとえば、ゲームの場合には一定時間ごとにキャラクタを動かします。こういった場合、Thread クラスを用いてもよいのですが、キャンバス型 i アプリでは ShortTimer を用いることもできます。

ShortTimer を用いるには、次の手順をとります。

1. ShortTimer クラスの静的メソッドである、getShortTimer メソッドを呼び出して、ShortTimer クラスのインスタンスを取得する

2. 1. で取得した ShortTimer クラスの start メソッドを呼び出して動作開始

getShortTimer メソッドは、次の構造をしています。

---

getShortTimer(Canvas *c*, int *id*, int *interval*, boolean *repeat*)
第一引数 *c*：一定時間ごと処理を行わせたい Canvas クラスのインスタンス
第二引数 *id*：ShortTimer の識別番号。複数の ShortTimer を利用する際に使用
第三引数 *interval*：一定時間に動作させるときの時間間隔。単位はミリ秒
第四引数 *repeat*：繰り返して動作させるときには true、最初の 1 回の動作で終えるときには false を指定

一定の時間になると、Canvas クラスの processEvent メソッドが呼び出されます。前節で見たように、processEvent メソッドは第一引数 type にイベントの種類が与えられ、第二引数の param に詳細内容が与えられますが、ShortTimer を用いた場合は、次のようになります。

---

type：Display.TIMER_EXPIRED_EVENT
param：getShortTimer メソッドの第二引数に指定した id

---

i アプリが動作中に電話がかかってきた場合、i アプリは動作を中断し ShortTimer は動きを停止します。通話終了後に i アプリが動作を再開しても、ShortTimer は動作を再開しません。i アプリの動作が再開したときには IApplication クラスの resume メソッドが呼び出されるので、resume メソッド内で ShortTimer の start メソッドを呼び出すようにプログラムを書かなければなりません。

また、getShortTimer の第一引数に設定した Canvas が表示されていない場合には ShortTimer は動作しません。

getShortTimer の第一引数に設定した Canvas が表示されていれば、start メソッドを呼び出すことにより ShortTimer は動作を開始しますが、表示を別の Canvas や Panel に切り替えると、ShortTimer も動作を停止し、そのあと元の Canvas が表示されても ShortTimer は再起動しません。従って、切り替え操作を行ったあとで、表示した Canvas に対応する ShortTimer の start メソッドを呼び出す必要があります。

リスト 5-5 に ShortTimer を使って、**図 5-6** のように 1 秒ごとに数字を増やして表示させるタイマーのような機能を持つ i アプリの例を示します。一定時間ごとに変数 nCount の値を増加させていますが、nCount の値を増やしても、画面を再描画しない限り表示される数は増えません。画面を再描画させる命令が Canvas クラスの repaint メソッドです（A の部分）。再描画されると paint メソッドが呼び出されますが、その際に一度画面を消去しないと文字が重なって表示されてしまいます。この消去を行っているのが B の部分です。

リスト 5-5　ShortTimerSample.java ／ ShortTimer を用いる

```
import com.nttdocomo.ui.*;

public  class  ShortTimerSample  extends  IApplication {

  ShortTimer st;

  public void start() {
    MyCanvas myCanvas = new MyCanvas();
    myCanvas.setSoftLabel(Frame.SOFT_KEY_2,"終了");
```

```
    st=ShortTimer.getShortTimer(myCanvas, 1, 1000, true);//------1
    Display.setCurrent(myCanvas);
    st.start();//-----2
  }

  public void resume() {
    st.start();
  }
}


class  MyCanvas  extends  Canvas {

  int nCount=0;

  public void paint(Graphics g) {
    g.clearRect(0, 0, getWidth(), getHeight());//------B
    g.drawString("nCount="+nCount, 30, 60);
  }

  public void  processEvent(int type, int param) {
    if (type==Display.KEY_RELEASED_EVENT) {
      if  (param==Display.KEY_SOFT2) {
        IApplication.getCurrentApp().terminate();
      }
    } else if (type==Display.TIMER_EXPIRED_EVENT) {
      nCount++;
      repaint();        //------A
    }
  }

}
```

図 5-6　リスト 5-5 の実行画面

5.2.4 色とフォント

iアプリでは、色を整数型の数値として扱います。色から数値を得るには、次の2通りがあります。

- **表 5-4** に示した Graphics クラスで定義されている色の名称を、Graphics クラスの getColorOfName メソッドの引数に指定する

  例）`int color_yellow= Graphics.getColorOfName(Graphics.YELLOW);`

- 赤・緑・青の光の強さを 0 から 255 までの数値で指定して、Graphics クラスのメソッドの引数に指定する

  例）`int color2= Graphics.getColorOfRGB(255, 200, 100);`

どの色にどういった数値が割り当てられるかは、機種によって異なる可能性があるので、上で得られた color_yellow や color2 などの値を読み取って、プログラム中に直接記述するということは推奨されていません。たとえば、黒はほとんどの機種で 0 で表されますが、黒を 255 × 255 × 255 で表す機種が存在する可能性もあります。従って、Graphics.getColorOfName もしくは Graphics.getColorOfRGB を用いて数値を得る必要があります。

表 5-4　Graphics **クラスで定義されている色**

名称	色	名称	色
Graphics.AQUA	水色	Graphics.BLACK	黒色
Graphics.BLUE	青色	Graphics.FUCHSIA	紫色
Graphics.GRAY	灰色	Graphics.GREEN	暗い緑色
Graphics.LIME	緑色	Graphics.MAROON	暗い赤色
Graphics.NAVY	暗い青色	Graphics.OLIVE	暗い黄色
Graphics.PURPLE	暗い紫色	Graphics.RED	赤色
Graphics.SILVER	銀色	Graphics.TEAL	暗い水色
Graphics.WHITE	白色	Graphics.YELLOW	黄色

キャンバス型 i アプリにおいては、Canvas の背景色・描画色の指定を Canvas クラスの setBackground メソッドと Graphics クラスの setColor メソッドの引数に、上で得られた色を表す数値を指定して行います。なお、Canvas の背景色は、Canvas クラスの setBackground メソッドを呼び出しただけでは変更されず、Canvas の clearRect メソッドを呼び出して背景を再描画する必要があります。

リスト 5-6 に、背景と線の色を指定して描画する例を挙げます。

リスト 5-6　ColorSample.java ／**色を指定して描画**

```
import com.nttdocomo.ui.*;

public class ColorSample extends IApplication {

  public void start() {
    MyCanvas myCanvas = new MyCanvas();
    myCanvas.setSoftLabel(Frame.SOFT_KEY_2,"終了");
    Display.setCurrent(myCanvas);
```

リスト 5-9　PanelTest.java／**パネル上に「終了」と書いたボタンを乗せる**

```
import com.nttdocomo.ui.*;

public class PanelTest extends IApplication {

  public void start() {
    Panel panel= new Panel();  //    -----1
    Button button= new Button("終了"); //   -----2
    panel.add(button); //  ------3
    Display.setCurrent(panel); //  ------4
  }
}
```

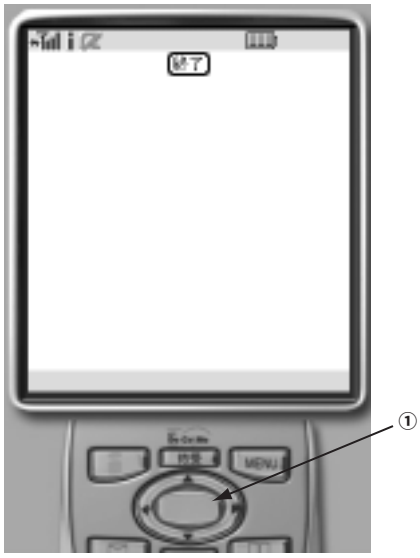


図 5-9　**リスト** 5-9 **の実行画面**

リスト 5-9 を実行すると**図 5-9** のようになります。キーボードの Enter キーか**図 5-9** の①で示された **セレクト** キーをクリックすると、［終了］ボタンが反転し、ボタンが押されたことが確認できます。例によってこのサンプルは終了機能がありませんので、 **オンフック** キーをクリックして強制終了してください。

次に、ボタンが押されたときに終了するように変更しましょう。ボタンなどのパネル上に配置した部品を用いたキー操作は、次の手順で行います。

1. ComponentListener を実装する
2. componentAction メソッド内に、部品上でキー操作があったときの処理を記述
3. Panel クラスの setComponentListener メソッドで、componentAction メソッドのあるクラスを指定

ここで、componentAction メソッドは次の構造を持ちます。

```
public void componentAction(Component source, int type, int param)
```

第一引数 *source*：キー操作の生じた部品

第二引数 *type*：キー操作の内容（表 5-8 に記述）

第三引数 *param*：第一・第二引数の内容によっては必要となる詳細情報

表 5-8　componentAction の第二引数のとり得る値

名称	内容
ComponentListener.BUTTON_PRESSED	ボタンが押された
ComponentListener.SELECTION_CHANGED	リストでの選択が変化した
ComponentListener.TEXT_CHANGED	入力欄の文字列入力が確定した

リスト 5-10 に、ボタンを押したときに終了するプログラムを示します。この場合、パネル上に乗っている部品はボタンしかないので、componentAction メソッドの第一引数を用いていませんが、複数のボタンがある場合には、componentAction メソッドの第一引数を用いて、どのボタンが押されたのかを知ることができます。

リスト 5-10　PanelTest.java ／パネル上のボタンを押すと終了する

```
import com.nttdocomo.ui.*;

public class PanelTest extends IApplication implements ComponentListener {  //  ---1
//  ↓---2
  public void componentAction(Component source, int type, int param) {
    if (type==ComponentListener.BUTTON_PRESSED) {
      terminate();
    }
  }
//  ↑---2

  public void start() {
    Panel panel= new Panel();
    Button button= new Button("終了");
    panel.add(button);
    panel.setComponentListener(this);//----3
    Display.setCurrent(panel);
  }

}
```

ソフトキーに関しても、ほぼ同様に次の手順で行います。

1. SoftKeyListener を実装する
2. softKeyPressed(int softKey)、softKeyReleased(int softKey) のメソッド内に、ソフトキーが押され

第6章 カメラとコードリーダー

504iS 以降、NTT ドコモの携帯電話機にカメラが内蔵され i アプリからの制御機能も用意されました。505i では機種によってバーコードリーダー機能も追加され、FOMA 900i ではバーコードリーダーの機能も全機種で標準機能となりました。本章では、これらのカメラを利用した i アプリの利用方法を解説します。

6.1 カメラ制御の基本

カメラを制御するには Camera クラスを利用します。これは com.nttdocomo.device パッケージに含まれているので、プログラムの先頭で以下のように import 文を記述します。

```
import   com.nttdocomo.device.*;
```

なお 504i では、Camera クラスがオプション API として com.nttdocomo.opt.ui パッケージに含まれているので、次のようにする必要があります。

```
import   com.nttdocomo.opt.ui.*;
```

Camera クラスを用いて画像を撮影するには、次の手順で行います。

1. Camera クラスの getCamera メソッドを用いて Camera クラスのインスタンスを取得
2. 1. で取得した Camera クラスのインスタンスに対して、takePicture メソッドを作用させて画像を撮影

ここで、Camera クラスの getCamera メソッドと takePicture メソッドの書式は次のとおりです。

`public static Camera` **`getCamera`**`(int` *id*`)`

引数　*id*：カメラの識別番号。どの番号がどのカメラに対応するかは機種依存であるが、900i では複数のカメラを持つ全機種で、背面側に付いているカメラを使う時には 0 を、前面のカメラを使うときには 1 を指定するようになっている。カメラが 1 つしかない場合には 0 しか指定できない

`public void` **`takePicture`**`()`

カメラ機能が中断した場合には、`InterruptedOperationException` 例外が発生する

実際には、takePicture メソッドを呼び出すと即座に画像が撮影されるのではなく、ネイティブのカメラ機能が起動して撮影待ちの状態になり、その後ユーザーが撮影操作をして初めて画像撮影が行われます。撮影された画像は、次の手順で取得します。

3. 1. で取得した Camera クラスのインスタンスに対して、getImage メソッドを作用させて MediaImage クラスのインスタンスを取得
4. 3. で取得した MediaImage クラスのインスタンスの use メソッドを呼び出し、さらに getImage メソッドを呼び出して Image クラスのインスタンスを取得

ここで用いる getImage メソッドの書式は次のとおりです。

```
public MediaImage getImage(int index)
```

引数　*index*：Camera クラスのインスタンス内に蓄えられている画像の何番目を用いるかを指定。0 から始まる数字になるので、1 枚目の画像を用いる場合は 0 を指定する。後述する連写モードで撮影しない限り 0 を指定するが、撮影作業をキャンセルした場合に対応するため、Camera クラスの `getNumberOfImages()` メソッドを呼び出して、何枚の画像が蓄えられているかを確認する必要がある。撮影をしなおすと、以前に蓄えられていた画像は一度破棄されるため、連写モードでない限り 0 以外の値を指定することはない

カメラでの撮影と撮影した画像の表示を行うプログラム例を**リスト 6-1** に示します。なお、エミュレータでは実際に画像を撮影することはできませんので、エミュレータ上に表示させる画像ファイルを選択する画面が開きます。**図 6-1** に実行結果を示します。

図 6-1　リスト 6-1 の実行結果

リスト 6-1　CameraSample.java ／**カメラを用いて画像を撮影する例**

```
import com.nttdocomo.ui.*;
import com.nttdocomo.device.*;

public class CameraSample extends IApplication {
  public void start() {
    CameraCanvas cv=new CameraCanvas();
    Display.setCurrent(cv);
  }

}
```

```
class  CameraCanvas extends Canvas {
  Image im=null;
  MediaImage mi=null;

  public CameraCanvas() {
    setSoftLabel(Frame.SOFT_KEY_1,"撮影");
    setSoftLabel(Frame.SOFT_KEY_2,"終了");
  }

  public void paint(Graphics g) {
    if (im!=null) g.drawImage(im,10,60);
  }

  public void processEvent(int type, int param) {
    if (type==Display.KEY_RELEASED_EVENT) {
      if (param==Display.KEY_SOFT1) {
        try {
          Camera c=Camera.getCamera(0);//----1
          c.takePicture();//-----2
          if  (c.getNumberOfImages()>=1) mi=c.getImage(0);//----3
          //↓----4
          mi.use();
          im=mi.getImage();
          //↑----4
          repaint();
        }catch(Exception e) {}
      } else if (param==Display.KEY_SOFT2) {
        IApplication.getCurrentApp().terminate();
      }
    }
  }
}
```

リスト 6-1 では通常の液晶画面に描画していますが、N900i と SH900i の場合は背面液晶への描画が可能です。背面液晶に描画するには、次のように import 文を記述します（com.nttdocomo.opt.ui パッケージには 504i 対応の Camera クラスがあるので、import com.nttdocomo.opt.ui.* とすると Camera クラスが重複してしまうので注意）。

```
import  com.nttdocomo.opt.ui.SubDisplay;
```

背面液晶への描画は、以下のように行います。

```
SubDisplay.setImage(im);
```

これは paint メソッド内に記述する必要はありません。画像を背面液晶に表示させるのは、N900i では電話機側で「i アプリからのサブディスプレイへの描画を可能にする」を設定したときのみ可能であり、SH900i では電話機を閉じた場合のみ可能です。背面液晶への表示機能を持たない機種の場合には、SubDisplay クラスのメソッドを呼び出すと、UnsupportedOperationException 例外が生じます。

エミュレータ上で表示するときには、device2 を選択して電話機の折りたたみ状態を閉じた状態にする必要があります。**図 6-2** にエミュレータでの実行結果を示します。

図 6-2　背面液晶への表示

撮影した画像をすぐに表示させるのではなく、スクラッチパッドに保存することによって、後で再利用したいといった場合には、一度バイナリデータとして取り出す必要があります。これは、次の手順で行います。

1. Camera クラスの getImageLength メソッドを用いて、画像データのバイナリデータのサイズを取得
2. 1. で得たデータサイズと同じ長さの byte 型配列を用意
3. Camera クラスの getInputStream メソッドを用いて、InputStream クラスのインスタンスを取得
4. 3. で得た InputStream クラスのインスタンスの read メソッドを呼び出して、2. で用意した byte 型配列に画像データをコピー

ここで使われる getImageLength メソッドと getInputStream メソッド、および InputStream クラスの read メソッドの書式は次のとおりです。

```
public long getImageLength(int index)
public java.io.InputStream getInputStream(int index)
```

引数 *index*：カメラの識別番号。getCamera の引数と同じ

```
public int read(byte data[], offset, length)
```

第一引数 *data[]*：取得するデータを入れるバイナリ型の配列
第二引数 *offset*：読み込んだ最初のデータを data[offset] から入れていく
第三引数 *length*：読み込むデータの長さ

また、InputStream を用いるため、ソースコードの先頭に以下の記述が必要です。

```
import     java.io.*;
```

カメラで撮影した画像をバイナリデータとして取得する例を**リスト 6-2** に示します。

リスト 6-2　CameraBinary.java ／撮影した画像をバイナリデータとして保存する例

```
import com.nttdocomo.ui.*;
import com.nttdocomo.device.*;
import java.io.*;

public class CameraBinary extends IApplication {
  public void start() {
```

```
    CameraCanvas cv=new CameraCanvas();
    Display.setCurrent(cv);
  }

}

class  CameraCanvas extends Canvas {
  Image im=null;
  MediaImage mi=null;

  public CameraCanvas() {
    setSoftLabel(Frame.SOFT_KEY_1,"撮影");
    setSoftLabel(Frame.SOFT_KEY_2,"終了");
  }

  public void paint(Graphics g) {
    if (im!=null) g.drawImage(im,10,60);
  }

  public void processEvent(int type, int param) {
    if (type==Display.KEY_RELEASED_EVENT) {
      if (param==Display.KEY_SOFT1) {
        try {
          Camera c=Camera.getCamera(0);
          c.takePicture();
          if  (c.getNumberOfImages()>=1) {
            int imageLength=(int) c.getImageLength(0);//-----1
            byte binaryData[]=new byte[imageLength];//-----2
            InputStream binaryDataIS=c.getInputStream(0);//-----3
            binaryDataIS.read(binaryData,0,imageLength);//-----4
            //以下にスクラッチパッドへの保存等の処理がくる
          }
        }catch(Exception e) {}
      } else if (param==Display.KEY_SOFT2) {
        IApplication.getCurrentApp().terminate();
      }
    }
  }
}
```

6.2 連写・画質の設定

Camera クラスの setAttribute メソッドを用いて、連写モードや画質の指定が行えます。setAttribute メソッドの書式は次のとおりです。

`public void` **`setAttribute`**`(int` *attr*`, int` *value*`)`

第一引数　*attr*：連写モードの設定をするときには Camera.DEV_CONTINUOUS_SHOT を、画質の設定をするときには、Camera.DEV_QUALITY を指定

第二引数 *value*：第一引数の値に応じて、以下を設定

連写モードの設定のとき：
Camera.ATTR_CONTINUOUS_SHOT_ON：連写する
Camera.ATTR_CONTINUOUS_SHOT_OFF：連写しない
画質の設定のとき：
Camera.ATTR_QUALITY_HIGH：高画質
Camera.ATTR_QUALITY_STANDARD：標準
Camera.ATTR_QUALITY_LOW：低画質

リスト 6-3 に、低画質で連写するプログラム例を示します。実機では連写モードに対応しているかどうかは機種依存であるため（F900i と SH900i のみが対応）、Camera クラスの isAvailable メソッドを用いて連写モード等が設定可能かどうかを**リスト 6-3** 内の A の行でチェックしています。また実機では、低画質を「ノーマル」、標準を「ファイン」、高画質を「スーパーファイン」等のように表現していることがあります。なお、エミュレータには連写や画質設定を実現する機能はありません。

リスト 6-3　CameraSettings.java ／**連写と画質の設定を行い撮影を実行する例**

```
import com.nttdocomo.ui.*;
import com.nttdocomo.device.*;

public class CameraSettings extends IApplication {
  public void start() {
    CameraCanvas cv=new CameraCanvas();
    Display.setCurrent(cv);
  }

}

class  CameraCanvas extends Canvas {
  Image im=null;
  MediaImage mi=null;
  int nMax=0; boolean bCShot=false;//連写した枚数と連写等が利用可能かどうかを入れる変数

  public CameraCanvas() {
    setSoftLabel(Frame.SOFT_KEY_1,"撮影");
    setSoftLabel(Frame.SOFT_KEY_2,"終了");
  }

  public void paint(Graphics g) {
    g.clearRect(0,0,getWidth(),getHeight());
    if (im!=null) g.drawImage(im,10,60);
    g.drawString("撮影枚数:"+nMax,10,10);
    if (bCShot) g.drawString("連写OK",100,10);
  }

  public void processEvent(int type, int param) {
    if (type==Display.KEY_RELEASED_EVENT) {
      if (param==Display.KEY_SOFT1) {
        try {
          Camera c=Camera.getCamera(0);
          if ( c.isAvailable(Camera.DEV_QUALITY)
          && c.isAvailable(Camera.DEV_CONTINUOUS_SHOT) ) {//----A
```

```
            c.setAttribute(Camera.DEV_CONTINUOUS_SHOT, Camera.ATTR_CONTINUOUS_SHOT_ON);
            c.setAttribute(Camera.DEV_QUALITY, Camera.ATTR_QUALITY_LOW);
            bCShot=true;
            c.takePicture();
          }
          if  (c.getNumberOfImages()>=1) {
            mi=c.getImage(c.getNumberOfImages()-1);
            nMax=c.getNumberOfImages();
            mi.use();
            im=mi.getImage();
            repaint();
          }
        }catch(Exception e) {}
      } else if (param==Display.KEY_SOFT2) {
        IApplication.getCurrentApp().terminate();
      }
    }
  }
}
```

6.3 画像サイズとフレームの設定

カメラで撮影する画像のサイズを設定したり、撮影後に写真の周囲にフレームを設定することもできます。画像サイズの設定は、Camera クラスの setImageSize メソッドを用います。書式は次のとおりです。

```
public void setImageSize(int width, int height)
```

第一引数 *width*：画像の幅

第二引数 *height*：画像の高さ

設定可能なサイズは、Camera クラスの getAvailablePictureSizes メソッドを用いて取得可能です。このメソッドは次の構造をもち、戻り値として設定可能なサイズが得られます。たとえば、200 × 200、400 × 400 の 2 種類が可能であるときには、getAvailablePictureSizes メソッドの戻り値は {{200, 200}, {400, 400}} になります。

```
public int[][] getAvailablePictureSizes()
```

getAvailablePictureSizes メソッドで得られる値以外の値を setImageSize メソッドで指定した場合には、設定可能なサイズのうち指定した画像サイズが収まる最も小さな画像サイズに設定されます。また、まったく設定を行わないときには、機種依存のデフォルトサイズが使用されます。

リスト 6-4 に、設定可能なサイズの一覧を表示し、その中から数字キーを押して撮影する画像のサイズを設定する例を示します（実行結果は、**図 6-3** を参照）。

リスト 6-4　CameraSizes.java／**撮影画像のサイズを設定する例**

```
import com.nttdocomo.ui.*;
import com.nttdocomo.device.*;

public class CameraSizes extends IApplication {
  public void start() {
    CameraCanvas cv=new CameraCanvas();
    Display.setCurrent(cv);
  }

}

class  CameraCanvas extends Canvas {
  Image im=null;
  Camera c;
  int[][] sizes;

  public CameraCanvas() {
    c=Camera.getCamera(0);
    sizes=c.getAvailablePictureSizes();
    setSoftLabel(Frame.SOFT_KEY_1,"撮影");
    setSoftLabel(Frame.SOFT_KEY_2,"終了");
  }

  public void paint(Graphics g) {
    g.clearRect(0, 0, getWidth(), getHeight());
    int nPictNo=sizes.length;
    if (im==null) {
      g.drawString("選択肢:"+nPictNo+"通り", 10, 20);
      for (int i=0; i<nPictNo; i++) {
        g.drawString((i+1)+":("+sizes[i][0]+","+sizes[i][0]+")" , 10, 20*(i+2));
      }
    } else  g.drawImage(im,10,60);
  }

  public void processEvent(int type, int param) {
    if (type==Display.KEY_RELEASED_EVENT) {
      switch (param) {
        case Display.KEY_SOFT1:
          try {
            c.takePicture();
            MediaImage mi=c.getImage(c.getNumberOfImages()-1);
            mi.use();
            im=mi.getImage();
            repaint();
          }catch(Exception e) {}
          break;
        case Display.KEY_SOFT2:
          IApplication.getCurrentApp().terminate();
          break;
        case Display.KEY_1:
          if (sizes.length>=1) c.setImageSize(sizes[0][0], sizes[0][1]);
          break;
        case Display.KEY_2:
```

```
            if (sizes.length>=2) c.setImageSize(sizes[1][0], sizes[1][1]);
            break;
          //以下同様なので省略
        }
      }
    }
  }
}
```

図 6-3　リスト 6-4 を実行した様子

カメラ機能を用いて撮影した画像には、フレームをつけることが可能です。フレームに設定する画像を Camera クラスの setFrameImage メソッドを用いて指定します。このメソッドは、引数に透過 GIF 画像を持つ MediaImage を指定します。指定可能な画像のサイズは、Camera クラスの getAvailableFrameSizes メソッドにより取得可能で、このメソッドは前述の getAvailablePictureSizes メソッドと同じ方法で利用します。フレームのサイズと撮影する画像のサイズは同じでなければなりません。従って、Camera クラスの setImageSize メソッドを呼ぶのは setFrameImage メソッドでフレーム画像を設定する前に行う必要があります。

フレーム機能が利用可能かどうかは機種によるので、Camera クラスの getAttribute メソッドの引数に Camera.DEV_FRAME_SHOT を指定して、戻り値が Camera.ATTR_FRAME_ON であるかどうかで、フレーム設定機能が有効かどうかを調べます。**表 6-1** に機種ごとのフレーム機能の有無を示します。背面液晶への描画機能と連写機能の有無もあわせて示しています。なお、エミュレータには、フレームを設定して実際に画面上に表示させる機能はありません。

表 6-1　900i（FOMA）シリーズでの機種ごとの各カメラ機能の有無

機種名	背面液晶への描画	連写	フレーム
P900i	×	×	×
N900i	○	×	×
F900i	×	○	○
SH900i	○	○	○
D900i	×	×	○

※たとえネイティブ機能に該当する機能があっても、i アプリから操作できないときには×と記述している。

リスト 6-5 に、フレームを用いたプログラム例を示します。ただし、**リスト 6-5** では、実際の携帯電話機でフレームとして利用可能な QCIF サイズ（176 × 144 ピクセル）の画像をあらかじめ準備しておく必要があります。

リスト 6-5　CameraFrame.java ／**撮影した画像にフレームを設定する例**

```
import com.nttdocomo.ui.*;
import com.nttdocomo.device.*;

public class CameraFrame extends IApplication {
  public void start() {
    CameraCanvas cv=new CameraCanvas();
    Display.setCurrent(cv);
  }

}

class  CameraCanvas extends Canvas {
  Image im=null;
  MediaImage mif;//フレーム画像を保持
  boolean bFrameNA=false;//フレーム機能が利用可能でないときtrueになる
  Camera c;
  int[][] sizes;

  public CameraCanvas() {
    c=Camera.getCamera(0);
    mif=MediaManager.getImage("resource:///frame.gif");
    try { mif.use(); } catch (Exception e) {}
    if (c.getAttribute(Camera.DEV_FRAME_SHOT)==Camera.ATTR_FRAME_ON) {
      c.setImageSize(176, 144);
      c.setFrameImage(mif);
    } else bFrameNA=true;
    setSoftLabel(Frame.SOFT_KEY_1,"撮影");
    setSoftLabel(Frame.SOFT_KEY_2,"終了");
  }

  public void paint(Graphics g) {
    g.clearRect(0, 0, getWidth(), getHeight());
    if (bFrameNA) g.drawString("フレーム機能不可",10,20);
    else g.drawString("フレーム機能可",10,20);
    if (im!=null)  g.drawImage(im,10,60);
  }

  public void processEvent(int type, int param) {
    if (type==Display.KEY_RELEASED_EVENT) {
      if (param==Display.KEY_SOFT1) {
        try {
          c.takePicture();
          MediaImage mi=c.getImage(c.getNumberOfImages()-1);
          mi.use();
          im=mi.getImage();
          repaint();
        }catch(Exception e) {}
      } else {
```

```
            IApplication.getCurrentApp().terminate();
        }
    }
  }
}
```

6.4 バーコードリーダー

バーコードリーダーを使うには、CodeReader クラスを用います。読み取れるバーコードは、JAN8、JAN13、QR コードであり、ORC としてテキストコードも読み取れます。JAN8、JAN13 は数字しか扱えませんが、QR コードは文字も扱えます。JAN13 は本書の裏表紙などについており、商品管理に一般的に使われます。JAN8、JAN13、QR の例を**図 6-4** に示します。ただし、各コードが読み取り可能かどうかは機種に依存します。

図 6-4　左から順に JAN8、JAN13、QR コードの例。JAN はバーコードの下に表示してある数字を表し、QR コードは「iMode Java Programming」という文字列を表す

バーコードリーダーを使用するには、次の手順で行います。

1. CodeReader クラスの getCodeReader メソッドを用いて CodeReader クラスのインスタンスを取得
2. 1. で取得したインスタンスの setCode メソッドを呼び出して読み取るコードの種別を指定
3. 1. で取得したインスタンスの read メソッドを呼び出してカメラで読み取る
4. 1. で取得したインスタンスの getResultType メソッドを呼び出して、戻り値が CodeReader.TYPE_UNKNOWN でなければ 5. に進む
5. 1. で取得したインスタンスの getString メソッドを呼び出して結果を取得

ここで使われる各メソッドの書式は、次のとおりです。

`public static CodeReader` **`getCodeReader`**`(int` *id*`)`

引数 *id*：カメラの識別番号。Camera クラスの `getCamera` メソッドの引数と同じ

`public void` **`setCode`**`(int` *code*`)`

引数 *code*：読み取るコードの種別。サポートしていないコード種別が指定されたときには、`IllegalArgumentException` 例外が発生

RSS Viewer RSSViewer.jar 実用

対応機種：P900i ／ F900i ／ N900i ／ SH900i ／ D900i　© MURAOKA Taro

RSS Viewer は、インターネット上のニュースサイトやウェブログ（通称 blog、ウェブ日記）などの記事情報を取得・表示・閲覧する i アプリです。簡易ではありますが独自のテキスト Web ブラウザを搭載しているので、i モードでは読むことのできない通常の HTML で書かれた記事本文を閲覧することもできます。

特に FOMA では、「パケ・ホーダイ」というパケット定額制のプランが追加され、本アプリのようにネットワークアクセスを頻繁に行うソフトウェアでも安心して使うことができるようになりました。

■ RSS とは

以前から Web サイトの更新チェックを行うソフトウェアはいくつかありました。はじめは単に更新されたことがわかる程度でしたが、すぐに更新内容（内容＝記事）をチェックできるソフトウェアも開発されました。ところが、更新内容を知るためには以下のいずれかが必要になります。

①ページに書かれた内容を解釈するサイトごとのモジュールを作る
②ページ内に内容を示す特別なタグを記述する

前者は、サイトごとに異なったプログラムを作ったり、サイトデザインの変更に伴いプログラムを変更する必要があります。後者は、ソフトウェアごとに特別なタグの書式が異なり、どのソフトウェアへ対応するかによっては、他のソフトウェアを切り捨てる結果になります。いずれにせよ、内容を記述する統一された方法がなかったために、それを実現するには必要以上の困難を伴いました。

RSS とは、Web サイトの更新情報を記述するために統一された文章書式の規格で、XML をベースにしています。本来は RDF（Resource Description Framework）が規格の名前で、その応用方法の 1 つとしての RSS（RDF Site Summary）というのが正確ですが、ここでは RSS と言ってしまっています。

RSS の登場で、サイト提供者は更新情報をソフトウェアに依存することなく提供でき、ソフトウェア作成者はサイトに依存することなく更新内容を取得・表示できるようになりました。PC 用にはいくつかの RSS ビューワーがすでに存在し、サイトが提供する RSS を利用することで、どのソフトウェアを使っても同じように更新内容にアクセスできます。ここで紹介する RSS Viewer は、この RSS を FOMA 端末で取得し表示するとともに、更新内容を閲覧してしまおうという i アプリです。

■ 使い方

RSS Viewer を起動すると、RSS 一覧画面が表示されます。この画面で ソフトキー2 を押すと、i アプリを終了することができます。サイトの登録や編集は、 ソフトキー1 でメニューを表示して行います。 ↑ ↓ キーを使ってサイトを選択して セレクト キーを押すと、インターネットへアクセスしてそのサイトの RSS を取得して見出し一覧画面を表示します。興味のある見出しを同様に選択して セレクト キーを押すと記事情報詳細画面が表示されます。画面上で［本文を読む］ボタンを押すと、インターネットアクセスすることの確認ダイアログを表示後、簡易 Web テキストブラウザによる記事本文画面で記事の全文を閲覧する

ことができます。各画面では[ソフトキー2]を押すと、1つ前の画面に戻ることができます。

RSS Viewerは階層的に、以下の画面から構成されています。

①RSS一覧
②見出し一覧
③記事情報詳細
④記事本文（簡易Webテキストブラウザ）

RSS一覧（起動画面）　見出し一覧　記事本文　記事情報詳細

上位から下位へは基本的に[セレクト]キーで進むことができ、逆へは[ソフトキー2]で戻ることができます。最上位のRSS一覧画面では[ソフトキー2]は終了に割り当てられています。また、各画面では[ソフトキー1]でメニューにアクセスすることができます。ただしメニュー内の項目が存在しない画面もあります。

画面に表示しきれないほど多くの項目が存在する場合は、[↑][↓]キーを使って画面を1行ずつスクロールさせることができます。また、[←][→]キーを使ってページ単位でスクロールすることも可能です。これらのキーは押し続けることでキーリピートが働き、連続的にスクロールします。

■ プログラムの解説

RSS Viewerのソースコードの構成について解説します。技術的なポイントは、大まかに分けて以下の4点です。ソースコードを読む際の基点としてください。

① RSSを読み込むためのクラス群「RDF*.java」
② エンコードフリーなStreamとReader、「AutoDecodeStream.java」と「UtfReader.java」
③ 独自UIコンポーネント「TextWindow.java」とその派生によるリストビュー「*ListView.java」
④ HTMLの簡易整形「WebTextBrowser.java」

RSSはXMLですから、Javaで書かれた既成のXMLパーサを使えれば読み込みは簡単です。しかしDoJa3.5でJARサイズ制限が緩くなったとはいえ、そこまでは難しいと判断し簡易なパーサを作成しました。実装は単純ですから「厳密に定義されているXMLをこんなインチキな方法で読んじゃうのもありなんだな」くらいに見てください。

文字エンコードの自動判別と変換処理は、普段あまり語られることのないノウハウの塊です。これは厳密を期すると、いくら調べても把握しきれないモノです。ですからRSS Viewerでは相当にコアな部分だけをサポートすることにしました。現実の利用では、このくらいで大半をカバーできる、と思ってください。

独自UIコンポーネントTextWindowを始めとして、RSS Viewerでは継承やインターフェイスを用いてリッチな、でもJavaのソフトウェアとしては普通の設計を採用しています。過去のiアプリでは、なるべくクラスを減らし、メソッドを減らし、メンバー変数や一時変数を減らすことが定石でした。本書で紹

介している FOMA 向け i アプリでは、JAR ファイルのサイズ制限が緩和されていますから、そうそう気にする必要がないとの判断です。

HTML の整形アルゴリズムも文字エンコード技術と同様に、あまり語られないところです。設計にあたっては、とにかく文章が読めればよいというコンセプトで、まずはタグを消去することから始めました。次に画面が小さいため、余分な空白、改行を行わないこと。そして最後に読みやすさを確保するため、適度な改行と空白を追加するように組み立てました。実装には Perl でプロトタイプを作成し、Java へ移植するという手順を踏んでいます。複雑なものについては、このような手法も有効です。

■ その他

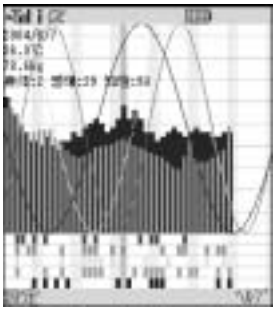
RSS と HTML の読み込みには、簡単なプロキシ CGI（proxy.cgi）を作成して稼動させることで、i アプリの配布サイト以外への HTTP 接続を可能にしています。プロキシ CGI の URL は自動的に設定していますが、ダウンロード時のパラメータ設定で変更することも可能なように設計してあります。

DoJa3.5 において、JAR サイズの制限が大幅に緩和されたのは先に触れたとおりです。このことは単純に多くの機能や画像等のリソースを盛り込めるということだけではなく、以前よりも設計にゆとりを持てることをも意味しています。ですからカツカツに切り詰めて設計し、開発に必要以上に時間をかけたり、メンテナンス性の低いアプリを作成するよりは、なるべく普通に設計して手早く動くものを作成した方がよいと言えます。それが後々は i アプリ利用者の利益にも繋がるでしょう。FOMA の恩恵は、まず初めに i アプリ作成者に与えられていると言ってもよいようです。

DoJa3.5 の大きな特徴の 1 つとして、HTTP による通信サイズが、以前の上り 10K バイト/下り 20K バイトから、それぞれ 80K と 150KB にまで大幅に引き上げられたことが挙げられます。これにより一般的なインターネット上のドキュメントであれば、難なく i アプリからアクセスできるようになり、実際これがなければ RSS Viewer は実現していなかったでしょう。FOMA の通信はもともと高速ですし、加えて前述したように 2004 年 6 月 1 日からパケット定額制、通称「パケ・ホーダイ」が開始されました。FOMA とその i アプリにとって、高速・大容量通信が今後の鍵となる可能性が高いと言えるでしょう。

基礎体温表&バイオリズム kiso.jar 実用

対応機種：P900i ／ F900i ／ N900i ／ SH900i ／ D900i　



「基礎体温表です。基礎体温計ではないので、腋に挟んでも体温は測れません。」

このアプリは以前の書籍で、このような紹介文のもとに公開されました。

最初は編集者の提案を受けてグラフ表示のサンプルとして作成したものでしたが、その後、予想もしなかった反響があり、さまざまな要望や叱咤激励をいただきました。反響に応えるかたちで今回は、体重とバイオリズム表示機能をあらたに追加し、高解像度表示に対応しています。さらに実用的なアプリになったのではないでしょうか。

■ 使い方

[ソフトキー2]を押すと以下のような操作方法が表示されます。

[↑][↓]キーによる体温／体重の記録は前日のデータを基準にしていますので、操作がしやすくなっています。生年月日の入力はバイオリズムの表示を行うためのものです。５種類の多目的フラグもついていますので記録帳的な使い方もできます。

キー／ボタン	動作
[セレクト]キー	体温／体重モード切替
[↑] [↓]	値変更
[←] [→]	日付変更
[1]～[5]キー	各フラグの ON ／ OFF
[0]キー	バイオリズム表示 ON ／ OFF
[ソフトキー1]	生年月日の入力
[ソフトキー2]	操作ヘルプ表示

■ プログラムの解説

初回起動時間と誕生日を、それぞれ long 値でスクラッチパッドの先頭部分に保存しています。

その後に、体温、体重、フラグ情報の順で１バイトずつ、合計３バイト保存されており、値の変更があるたびに、スクラッチパッドの該当部分を書き換えています。

生年月日の入力には、Panel 画面を使うことも考えましたが、今回は Canvas.imeOn() で IME を起動させ、IME の入力完了イベントで入力された文字列を解析して、誕生日データを取得しています。

待ち受けブックマーク MBookmark.jar 実用

対応機種：P900i ／ F900i ／ N900i ／ SH900i ／ D900i ／ SO505iS　© M.Tokashiki

待ち受けアプリとして動作する、iモードのブックマークとiアプリのランチャー機能を持つ実用アプリです。iモードとiアプリを統合したリスト形式のインターフェイスにより、シンプルな操作で素早いアクセスを実現します。リスト表示の文字色やアイコンの変更、QRコードからのブックマーク登録、待ち受け画像の設定も可能です。使用しない間は休眠状態になっているため、待ち受けアプリとして利用しても消費電力は低く抑えられています。

なお、本アプリを活用して、インターネットでブックマーク情報を簡単に取得できるサービスとして「i ちょい」というサイト（http://ichoi.jp）を立ち上げています。サーバーを介すことで、ユーザー間でブックマーク情報をやり取りしたり、デザインを変更するなどの機能を追加しています。より実用的に利用したいという方はぜひお試しください。

■ 使い方

待ち受けブックマークは2つのiアプリから構成されているため、Redirectorと MBookmarkをそれぞれダウンロードする必要があります（2つ必要な理由は後述）。MBookmarkはダウンロード後、待ち受け起動するように設定します。

MBookmarkを活性化状態にするキーを押すと、ブックマーク・アプリランチャーリストが表示されます。[↑][↓]キーでフォーカスを移動させて、目的のiモードページもしくはiアプリを選択して[セレクト]キーで、iモード接続やiアプリ起動を実行します。上から10番目までの項目に関しては[0]～[9]の数字キーを押すことでダイレクトに選択することもできます。また、iモード接続の場合はRedirectorのアプリ連携起動時とブラウザの起動時に確認ダイアログが表示されますが、［はい］（YES）を選択して処理を進めてください。同様にiアプリランチャーの場合はアプリ連携起動に確認ダイアログが表示されますので、［はい］（YES）を選択します。

リスト表示

設定メニュー

Bookmark追加の入力画面

リストの編集や待ち受け画像の設定を行う場合は、[ソフトキー1]（設定）を押します。設定メニューは下記に示す項目から構成されます。

- Bookmark 追加：タイトル、URL、文字色、アイコンを設定しブックマークを追加する。また、QR コードからのブックマーク登録も可能
- アプリ追加：タイトル、文字色、アイコンと起動するアプリを設定しランチャー項目を追加する
- 順番替え：順番を替えたい項目を選んで[↑][↓]キーで好きな位置に移動する
- 修正：修正したい項目を選んで、タイトル、URL、文字色、アイコンの編集を行う
- 削除：削除したい項目を選んで削除する
- 待ち受け画像：待ち受け画面をマイピクチャにある画像にすることができる。画像サイズは待ち受け画面サイズ以下に制限している

特別なケースとして、登録してある項目が 1 つの場合はリスト選択の意味がないため、活性化状態にするとすぐに Redirector を起動するようになっています。特定の 1 サイトに頻繁にアクセスするという場合は、手数が 1 つ減るためより便利になります。設定変更のためにリスト画面を表示したい場合は Redirector を起動する際の確認ダイアログで［いいえ］（NO）を選択してください。

■ プログラムの解説

待ち受けブックマークは MBookmark と Redirector の 2 つのアプリから構成されます。これは待ち受けアプリの実行中の制限として、活性化状態になっていてもブラウザ起動が不可になっているためです。一方、i アプリ連携起動に関しては活性化状態であれば実行可能なため、リストを管理する MBookmark から、Redirector をパラメータとして URL を渡す形で起動し、Redirector では取得した URL を指定してブラウザ起動を行っています。

MBookmark は、省電力のため待ち受けアプリとして起動すると、すぐに非活性化状態から休眠状態に遷移させています。リストを表示している状態でユーザーからのキー入力を監視し、節電タイマーとして ShortTimer を用いて時間をカウントして 30 秒に達すると休眠状態に移行させています。また、携帯電話を閉じた場合も FOLD_CHANGED_EVENT イベントを拾うことで休眠状態に遷移させています。

MBookmark は実用性を考慮すると待ち受けアプリとして利用すべきですが、通常起動でも動作するようになっています。getLaunchType() で確認できる起動形態によって、下記のように動作を切り替えています。

	起動直後	節電タイマー	ブラウザ起動	待ち受け画像
待ち受けアプリ起動	待ち受け状態	作動する	Redirector 経由	待ち受け状態時に表示
通常起動	リスト表示	作動しない	直接ブラウザ起動	利用しない

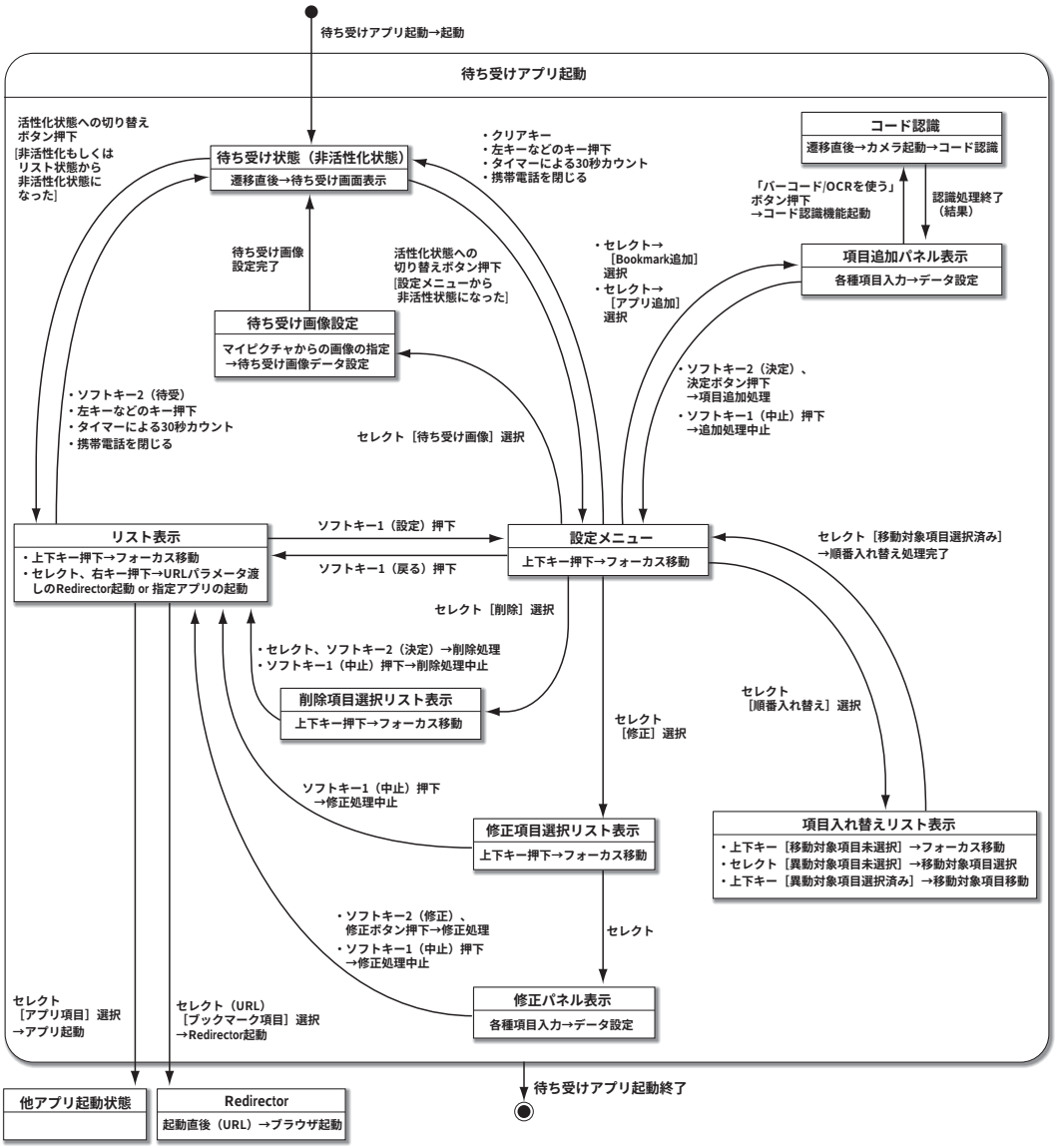
QR コード認識からのブックマーク登録に関しては、ブックマーク登録機能用のフォーマットを前提とした簡易的な解析処理を行っています。

■ フォーマットの例

```
MEBKM:TITLE:xxx;URL:yyy;;
```

StringParse クラスでは、先頭の「:」までの識別子と最後の「;」を除く文字列に対し、「:」までをプロパティ、「;」までをパラメータとして解析した結果を Hash テーブルで保持しています。この時、「¥」「:」「;」「,」はエスケープ文字として記述されているため「¥」の次の文字はスキップしています。MBookmark では識別子（MEBKM）を確認した場合に、TITLE と URL を取り出しブックマーク登録に使っています。

最後に、参考までに本アプリの状態遷移図を載せておきます。



ProxyVoice proxyvoice.jar

実用

対応機種：P900i ／ F900i ／ N900i ／ SH900i ／ D900i　© 志村 拓

入力した平仮名を音声で読み上げるiアプリです。編集長の強い要望によって作成しましたが、風邪で声が出ない時、直接面と向かって話したくない内容を話す時（痴情のもつれの末の別れ話等）、身元が知られたくない匿名の電話をする時（身代金の要求等）など、利用範囲は広大と思われます。

残念ながら、「あ」から「ん」までの51音しか用意していませんので、濁音、半濁音等は発音できません。そうでなくともイントネーションが皆無のため、ちょっと怪しげな発音であることに加え、濁音、半濁音を清音として発音してしまうため、外国人の話すたどたどしい日本語に勝るとも劣りません。しかし、逆にこの辺りが、このソフトのチャームポイントでもあります。

対象機種は音声データをjarファイル内に持っているため、DoJa3.5以降でFOMA 900iのみ利用可能です。また、ADPCMのiメロディが機種ごとに異なるため、jarファイルも機種ごとに用意しています。

■ 使い方

使い方は、いたって簡単です。画面のテキストボックスに、発音させたい言葉を「全部ひらがな」で入力し、ソフトキー1を押すと、携帯電話が話し始めます。携帯電話のテンキーで入力するのが面倒という人（作者もその一人）のために、コードリーダーから読み込んだ文字列をテキストボックスに追加する機能もついています。画面上の「Camera（CodeReader）」ボタンを押すと、コードリーダーが呼び出されますので、「QRファクトリー」等で作ったQRコードを読み込ませます。読み込んだデータは、テキストボックスの文字列の末尾に追記されます。あらかじめしゃべらせたい内容をQRコードで複数用意しておくことで、素早くしゃべらせる内容を切り替えることができます。

なお、テキストボックスに入力した文字列は、スクラッチパッドに保存しますので、終了しても次に起動したときは、終了した時の状態が復元されます。

■ プログラムの解説

51音の音声は、mldファイルとしてjarにリソースとして格納しています。プログラム起動時にこれらのmldファイルを読み込み、MediaSoundクラスの51要素の1次元配列に読み込んで保持しています。

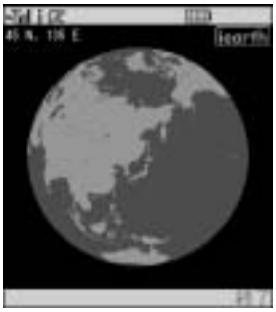
発音を自分の声にしたいという人は、リソースディレクトリにあるmldファイルを作成しなおして、リビルドしてください。なお、このソフトに添付されている音声データは、wav2mldという市販アプリの試用版を使って作成しました。このソフトは、なかなかよくできていて、各機種用のmldファイルを作成できるようになっています。wav2mldの詳しい内容については、同ソフトのWebページ（http://www.hundredsoft.jp/wav2mld/index.html）を参照してください。

さて、本ソフトは、ソフトキー1を押すというイベントをきっかけに、発音を開始するわけですが、発音は1文字発音しては、その発音が終わると次の文字の発音といった具合に、キーイベントとはまったく別に処理を続行する必要があります。本ソフトでは、AudioPresenterのイベントであるAUDIO_COMPLETE（音声再生が完了した際に発生するイベント）を利用して、1音ずつの再生を連続的に文字列の最後まで実行するようにしています。

iEarth ieearth.jar

癒し系

対応機種：P900i ／ F900i ／ N900i ／ SH900i ／ D900i　ⓒ Hidetoshi Ohtomo



端末の画面に地球を表示します。カーソルキーで緯度、経度ともに 45 度単位で回転させることが可能です。地球や宇宙が大好きな方にお勧めです。忙しい方や疲れている方も、これを見て「ほっ」と一息入れていただければと思います。

[編集者注] なお、このプログラムは自然が大好きな作者が編集長を拝み倒して、もう 1 つの担当プログラムを没にしてまで作った作品です。

■ 使い方

使い方は、以下のとおりです。大量の画像ファイルを読み込むため、起動から最初の描画まで少々時間を要しますのでご注意ください。なお、SH900i では約 8 秒かかりました。

キー	機能
← →	視点経度変更（45 度単位）
↑ ↓	視点緯度変更（45 度単位）
ソフトキー2	終了

■ プログラムの解説

あらかじめ用意した 40 枚の画像ファイルを、視点位置ごとに表示することで地球を描画しています。画像データは、X Window System 用の名作プログラム「xearth」を用いて生成しました。xearth の作者である Kirk Lauritz Johnson 氏にこの場を借りてお礼申し上げます。

当初は緯度・経度とも 1 度単位での視点変更を可能にするため、レンダリングによる描画を行うつもりでした。しかし 1 回の描画に 30 秒以上かかることが判明したので泣く泣く断念し、今回は 40 の視点からの画像を用いることにしました。jar ファイルサイズの制限が 100K バイトと格段に増えたからこそできた、CPU スピードとメモリ使用量のトレードオフと言えます。

少なくとも 5 度単位での描画が可能であれば、xearth にあるような以下の機能も実現したかったところです。

- 描画時刻での太陽からの視点による表示
- 描画時刻での月からの視点による表示

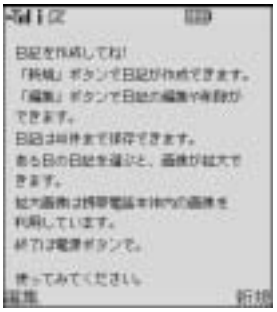
また以下は、秘かにバージョンアップをして、ぜひ実装しようと考えている機能です。

- 太陽光が当たっていない面を暗くする「シェーディング」
- 緯度、経度を示す点線の表示
- 南極大陸を白くする

PhotoLog PhotoLog.jar

実用

対応機種：P900i ／ F900i ／ N900i ／ SH900i ／ D900i　© H.Aonuma



携帯電話で撮影した写真を利用して日記をつけるiアプリです。アプリからもカメラを利用でき、撮影した画像は本体内部にも保存されます。写真にはそれぞれサムネイルが作られ、60文字までのメモがつけられます。日記を選択すると、サムネイル写真のもととなる本体の写真が表示できます。

常に持ち歩くケータイで、手軽に一覧性の高い写真日記が作成できます（最大の登録件数は40件）。また、このプログラムを改造することで、自分のホームページに写真日記をアップすることも可能になります。

■ 使い方

最初に起動すると、簡単な利用説明が表示されます。日記が記入されていると、起動時には日記一覧画面になります。日記は、日付の降順にソートされており、↑ ↓キーでフォーカスを移動します。また、上下端に達するとスクロールを開始します。

日記にフォーカスがあたっている状態で セレクト キーを押すと、サムネイルとして表示されている画像が拡大されます。ただし、携帯電話本体内部の画像を利用していますので、本体内部の画像の名前が作成時と変更されていたり、削除されている場合には拡大表示できません。

ソフトキー1（新規）で新規に日記を追加し、ソフトキー2（編集）で現在フォーカスが当たっている日記の編集をすることができます。

▶▶ 日記の新規作成

日記作成画面の入力フォームは、年月日、時間、画像、文章があります。各入力フォーム間のフォーカス移動は、↑ ↓キーで行います。年月日時は、デフォルトでいまの年月日時が入力されています。必要に応じて修正してください（ただし、年は下2桁で入力）。カメラ撮影を行う場合には、［撮影］ボタンを押してください。カメラ画面が起動して、撮影可能になります。撮影が完了すると、本体内部への保存を行いサムネイルが入力フォームに表示されます。画像を入力する場合には、［画像選択］ボタンで携帯電話内部に保存されている画像の選択画面に移ります。画像を選択すると、縮小されたサムネイルが入力フォームに表示されます。画像を変更したい場合は、［画像選択］ボタンで再度選択をし直してください。画像を削除する場合は、［画像削除］ボタンを使います。

なお、N900i、P900iでは、携帯電話本体に保存されている画像サイズが240 × 320を越える場合には利用することができません。携帯電話本体の機能で、画像を事前に縮小して利用してください。文章を入力する場合には、テキストボックスを選択して入力します。入力できる最大文字数は60文字に設定されています。

すべての入力が完了したら［決定］ボタンで日記一覧の画面に戻ります。なお、画像と文章が両方とも入力されていないと日記を追加することはできません。新規に追加せずに日記一覧画面に戻る場合には、ソフトキー1（戻る）を押します。

▶▶日記の編集・削除

日記を編集する場合には、編集したい日記を選択しソフトキー1（編集）で、編集画面に入ります。それぞれの入力フォームは、↑ ↓キーで選択してください。編集も新規作成の場合と操作は同じです。

変更した日記の保存は［決定］ボタンを押し、完了すると日記一覧の画面に戻ります。また、日記を削除する場合には、［削除］ボタンを押してください。その日記が削除され、一番古い日記にフォーカスが当たった日記一覧の画面に戻ります。

編集を行わずに日記一覧画面に戻る場合には、ソフトキー1（戻る）を押します。

▶▶日記画像の拡大

拡大したい画像を持つ日記を選択して、セレクトキーを押します。携帯電話本体に、対応する画像がある場合には画像を拡大表示します。再度、日記一覧画面に戻るには、セレクトキーかソフトキー1（戻る）を押します。

▶▶アプリの終了

本アプリの終了は、携帯電話の終話キーを利用します。

■ プログラム解説

PhotoLog のソースコードは、以下の構成になっています。

- PhotoLog.java

  アプリ実行クラス。メイン画面（MainCanvas）、新規追加・編集画面（AddPanel）、画像拡大表示画面（SubCanvas）の切り替えを行います。

- MainCanvas.java

  日記一覧画面を構成します。日記ごとに構成される Contets を作成保持します。

- AddPanel.java

  新規追加・編集画面を構成します。携帯電話本体の画像選択や選択された画像の縮小、日記の作成を行います。

- SubCanvas.java

  日記の画像を拡大表示します。携帯電話本体に保持されている画像を取得し、表示します。

- Contents.java

  スクラッチパッドに保存されている日記データをプログラムで利用するために、各日記ベースに保持するための Bean です。

● StorageAccess.java

スクラッチパッドへのアクセスを行うクラスです。スクラッチパッドへの日記の保存、読み出しを行います。また、日記のデータを扱いやすくするために、スクラッチパッド内のデータへアクセスするための Index の作成、日にちによるソート等が含まれています。スクラッチパッドのデータ設計はソースコードをご覧ください。

このプログラムでは、多量に画像を扱うため画像まわりの扱いに注意しています。特に、MediaImage、Image クラスでは dispose を明示的に呼ばないと、ヒープに残り続けメモリが不足する状態になります。そのため各表示用クラスで必ず画像関連のオブジェクトを管理しています。表示用クラスのインスタンスの終了を宣言する destory メソッドを用意し、その中で関連する画像クラスをすべて dispose しています。また、DoJa3.0 より追加された携帯電話内部の画像へのアクセス API である、ImageStore を多く利用したサンプルコードになっています。

スクラッチパッドの管理を行っている StorageAccess では、画像の大きさ、文章の大きさを static int で宣言しています。この部分を調整することで、505 シリーズに対応させることも可能です。

日記をネットワーク上にあるサーバーにアップロードする仕組みを入れるのも面白い課題だといえます。たとえば、編集画面に［UpLoad ボタン］を用意すればよいのですが、それは AddPanel.java において編集画面用のコンストラクタ内で行います。そして、componentAction メソッド内で相当する処理を書いていけばよいでしょう。AddPanel で画像、携帯本体内部の画像への ID、文章、年月日を保持していますので、ネットワークアクセスを行い、それらを OutputStream で流し込むことで作成できます。

■ ネットワーク対応版

上記で説明したサーバーにアップロードするネットワーク対応版の簡単なサンプルを作成しました。

アーカイブに含まれる CGI を設置し日記のアップロードを行うと、選択した日記の画像、文章がファイルとしてサーバーに保存されます。

▶▶準備

アーカイブを解凍し、bin ディレクトリ内にある PhotoLog.jar、PhotoLog.jam をダウンロード用のディレクトリに置きます。cgi ディレクトリ内の up.cgi を PhotoLog.jam を置いたディレクトリ下に cgi という名前のディレクトリを作成し、そこに置きます。cgi を置いたディレクトリ下に data という名前のディレクトリを作成します。この data 内にアップロードした日記がファイルとして保存されます。

▶▶日記のアップロード

アップロードしたい日記を選択し、ソフトキー2（編集）ボタンで編集画面に移り［アップ］ボタンを押します。通信が始まり日記がアップロードされます。画像は 240 × 240 程度に補正されたものがアップされ、文章は文字コード SJIS でアップされます。アップロードされた画像、文章は年月日時を名前としたファイルとして保存されています。同一名のファイルが存在した場合には、「n_ファイル名」（n は数字）として保存されます。

▶▶プログラムの解説

アーカイブを解凍したsrcディレクトリ内にソースコードが保存されています。ネットワーク非対応版との違いは、AddPanel.javaのみです。AddPanelクラス内のcomponentActionメソッド内に、アップボタンのイベントのハンドリングを追加しています。全体の流れは、日記で利用している画像の元画像を240×240の枠に収まるように縦横等率で変更します。その画像をJPEGでエンコードします。

HTTP Postリクエストメソッドを利用して、データをアップロードします。文章のアップデートも同様にHTTP Postリクエストメソッドを利用してアップロードしています。画像、文章ともバイナリとして扱い、メッセージボディはバイナリデータだけで構成されます。サーバー側のCGIでもバイナリとして扱っています。そのため画像のフォーマットも文章の文字エンコードも、クライアント側で設定したものとなっています。up.cgiを置くディレクトリを変更したい場合は、AddPanelクラスのpathフィールドの値を変更してください。このpathはjamファイルを置いたディレクトリからの相対ディレクトリになっています。

また、アップロードした画像、文章を保存するファイルのディレクトリを変更したい場合には、up.cgi内の$pathを変更してください。up.cgiは非常に簡単なサンプルとなっています。このサンプルを拡張することで、自分のBlogへの追加を行うのも楽しい試みといえます。

詳しい使い方は、アーカイブ中のreadme.txtも参照してください。

com.nttdocomo.ui
AnchorButton

クラス

このクラスは、Panel に配置する HTML のアンカー風のボタンを定義します。このコンポーネントにはテキストやその見出しとなるイメージを設定することができます。

このコンポーネントがユーザー操作可能の場合は、テキストなら下線付き、画像なら周囲に枠線の付いた形で表示されます。setEnabled() を用いてユーザー操作不可にした場合、テキストなら下線なし、画像なら周囲の枠線を取り除いた形で表示されます。

画像はコンポーネントの表示領域の左上から表示されます。同時にテキストも設定されている場合は、画像の下端にテキストの下端を合わせた高さで画像の右側に表示されます。さらに、同じ状況でテキストが折り返される場合、折り返されたテキストの表示位置は画像の左端からになります。

DoJa3.0 から XString に対応しました。表示文字列として従来の文字列もしくは XString を指定することができます。これらは排他的に動作します。

折り返されたテキストの表示位置

AnchorButton の縦方向のサイズは、画面の縦方向のサイズには制限されません。しかし、横方向のサイズは画面の横方向のサイズに制限されます。コンポーネントの幅より長いテキストが設定された場合、テキストが自動的に折り返されて複数行に渡って表示されます。そのため、ユーザー操作不可の AnchorButton オブジェクトは、複数行に渡る文字列を表示するのに使用されます。

Panel.setLayoutManager() の引数に null を指定して呼び出した場合は、位置設定後にコンポーネントが画面の横にはみ出した部分は、画面端で折り返されることはなく表示されません。コンポーネントのサイズが変更された場合は、変更後のコンポーネントの幅でテキストを折り返して表示し直します。

Component から継承するメソッド：

`getHeight()`, `getWidth()`, `getX()`, `getY()`, `setBackground()`, `setForeground()`, `setLocation()`, `setSize()`, `setVisible()`

Interactable の実装：

`requestFocus()`、`setEnabled()`

例外：

- Image オブジェクトを引数に取るコンストラクタやメソッドで、引数で指定された Image オブジェクトがすでに dispose() を呼ばれ破棄されていた場合、UIException(ILLEGAL_STATE) が発生

関連項目：

親クラス ☞ `Component`

実装しているインターフェイス ☞ `Interactable`

イベントの処理 ☞ `ComponentListener`.componentAction(Component *c*, int *type*, int *param*)

Panel への配置 ☞ `Panel`.add(Component *c*)

```
public final class AnchorButton
extends Component
implements Interactable {

  //コンストラクタ
  public AnchorButton();
  public AnchorButton(Image image);
  public AnchorButton(Image image, java.lang.String text);
  public AnchorButton(java.lang.String text);
  public AnchorButton(Image image, XString xText);
  public AnchorButton(XString xText);

  //インスタンスメソッド
  public void requestFocus();                  //Overrides Interactable.requestFocus
  public void setEnabled(boolean b);           //Overrides Interactable.setEnabled
  public void setImage(Image image);
  public void setText(java.lang.String text); //Overrides Interactable.setText
  public void setText(XString xText)
}
```

コンストラクタの概要

■ public **AnchorButton**()

何も表示するものがないアンカーボタンを生成する。テキスト文字列とラベル画像の両方に null が指定されたものとみなす。

■ public **AnchorButton**(java.lang.String ***text***)

テキスト文字列を指定してアンカーボタンを生成する。ラベル画像には null が指定されたものとみなす。

引数： *text*　表示するテキスト文字列

■ public **AnchorButton**(Image ***image***)

ラベル画像を指定してアンカーボタンを生成する。テキスト文字列には null が指定されたものとみなす。

引数： *image*　表示するラベル画像

■ public **AnchorButton**(Image ***image***, java.lang.String ***text***)

ラベル画像とテキスト文字列を指定してアンカーボタンを生成する。

引数： *image*　表示するラベル画像
text　表示するテキスト文字列

■ public **AnchorButton**(XString *xText*)

テキスト文字列に XString の文字列を指定してアンカーボタンを生成する。ラベル画像には null が指定されたものとみなす。

引数：*xText*　表示する XString のテキスト文字列

■ public **AnchorButton**(Image *image*, XString *xText*)

ラベル画像と、テキスト文字列に XString の文字列を指定してアンカーボタンを生成する。

引数：*image*　表示するラベル画像
xText　表示する XString のテキスト文字列

メソッドの概要

■ public void **requestFocus**()

Panel に追加された AnchorButton にフォーカスをセットするように要求する。Panel に追加されていない場合、要求は無視される。

■ public void **setEnabled**(boolean *b*)

AnchorButton へのユーザー操作の可否を設定する。デフォルトはユーザー操作可能（true）。

引数：*b*　ユーザー操作を受け付ける（true）か、受け付けない（false）か

■ public void **setText**(java.lang.String *text*)

AnchorButton オブジェクトに表示するテキスト文字列を設定する。

引数：*text*　表示するテキスト文字列

■ public void **setImage**(Image *image*)

AnchorButton オブジェクトに表示するラベル画像を設定する。

引数：*image*　表示するラベル画像

■ public void **setText**(XString *xText*)

AnchorButton オブジェクトに表示する XString の文字列を設定する。

引数：*xText*　表示する XString のテキスト文字列

com.nttdocomo.ui

AudioPresenter

クラス

このクラスでは、i メロディ形式などの音声メディアデータ（MediaData、MediaSound）を再生するための機能が提供されています。ディスプレイのカレントに設定されている Frame が、Canvas もしくは Panel のどちらであってもこのクラスを利用することができます。

setMediaListener() で MediaListener を登録すると、再生状態をリスナーオブジェクトに通知します。setMediaListener() を複数回呼び出した場合は、最後に登録したリスナーオブジェクトだけが有効となります。引数に null を指定すると、リスナーの登録を削除します。

AudioPresenter インスタンスの取得には、コンストラクタを用いるのではなく、クラスメソッドの getAudioPresenter() を用います。

AudioPresenter に再生させる音声メディアデータは setData() か setSound() を用いて設定します。これらのメソッドを複数回呼び出した場合は、最後に設定したデータが有効になります。

stop() で停止した場合、play() で再開してもデータの先頭からの再生となる点に注意してください。DoJa3.0 では、状態管理として、再生と一時停止は同じ状態であると判断されます。そのため、一時停止中も同時再生中として扱われたり、一時停止中に stop() を呼び出されると再生中に呼び出した場合と同様に停止します。

メディアデータに同期イベントが埋め込まれている場合、メディア再生時に同期イベントのタイミングに合わせて MediaListener に通知します。同期イベントを発生させるかどうかを play() を呼び出す前に設定しておく必要があります。デフォルトでは同期イベントは発生しません。

DoJa2.0 では新たに優先順位の属性が追加され、サウンドに再生優先順位を持たせることができます。優先順位は setAttribute() により設定します。優先順位は、端末が持つ音源チップの能力を超えてサウンドを再生しようとした時に、再生するサウンドを決定する要因になります。複数のサウンドが同じ優先順位を持つ場合、どのサウンドが優先されるかは機種に依存します。

DoJa3.0 では、複数の AudioPresenter の再生がサポートされています。2 つ以上再生しようとした場合、指定した優先順位により再生する AudioPresenter が決定されます。

DoJa3.0 では、同時再生に関してポートの概念が導入されました。ポートは、再生可能数分だけ用意された仮想再生器で、AudioPresenter.getAudioPresenter() の引数に指定してポートを割り当てます。複数再生した場合、同時再生可能な範囲内ではすべての AudioPresenter の音が再生されますが、発音数やチャンネル数等の制限により、すべての音が再生されるとは限りません。同時再生可能数を超えて再生しようとした場合、ポート指定なしの時は優先順位の高いほうが優先して再生されます。ポート指定がある場合は、優先順位は、同じポートを指定した AudioPresenter の中でのみ有効となります。

DoJa3.0 では、AUDIO_COMPLETE、AUDIO_PLAYING、AUDIO_STOPPED、AUDIO_PAUSED、AUDIO_RESTARTED、AUDIO_SYNC の各イベントの発生が保証されます。

MediaPresenter の実装：

```
getMediaResource(), play(), setAttribute(), setData(), setMediaListener(), stop()
```

例外：

- setData()、setSound() は、端末で再生できない音声メディアデータを設定した場合には UIException(UNSUPPORTED_FORMAT)、データ再生中に呼び出した場合には UIException(ILLEGAL_STATE) が発生
- ミニマムスペックでは setData() を呼ぶと UnsupportedOperationException が発生
- play()、play(int)、stop()、pause() を呼び出したとき、メディアデータがセットされていなかったり、メディアデータが再生可能でない場合、UIException(ILLEGAL_STATE) が発生
- play()、play(int)、restart() で再生しようとしたとき、オーディオ再生のためのリソースが確保できない場合、UIException(NO_RESOURCES または BUSY_RESOURCE) が発生
- setSound()、setData の引数に null が指定された場合、NullPointerException が発生
- setData() を端末がサポートしていない場合、UnsupportedOperationException が発生
- setAttribute() の第一引数で指定された有効な属性に対し、第二引数に不正な値を指定した場合、IllegalArgumentException が発生
- getAudioPresenter() の引数に指定可能でない値を指定した場合、IllegalArgumentException が発生
- play()、play(int) を通話中に呼び出すと UIException(BUSY_RESOURCE) が発生
- play(int) の引数に 0 未満の値が指定されると、IllegalArgumentException が発生

- setData()、setSound() の引数で指定されたメディアデータ、サウンドが use() されていない場合、UIException(ILLEGAL_STATE) が発生
- getCurrentTime() で、メディアデータがセットされていなかったり、use() されていない場合に UIException(ILLEGAL_STATE) が発生
- play()、play(int)、restart() ネイティブ側で再生に失敗した場合に UIException(ILLEGAL_STATE) が発生

ミニマムスペック：

- setData() を呼び出すと例外 UnsupportedOperationException が発生
- setAttribute() は何もしない
- 再生方法に関する属性値は定義されていない
- AudioPresenter の同時再生数は 2

機種依存：

- pause()、restart()、setData() がサポートされているかどうか
- 同じ優先度を持つ複数のサウンドデータの再生順序
- 再生位置の指定が可能かどうか
- SMF と MFi をサポートしている FOMA 端末で、それらの 2 つが同時再生可能かどうか
- 同時可能な発音数やチャンネル数を超えて複数再生した場合に再生される音
- 最低優先順位の AudioPresenter を同時再生可能数を超えて再生した場合に停止する AudioPresenter

関連項目：

実装しているインターフェイス ☞ `MediaPresenter`
リスナーの登録 ☞ `setMediaListener(MediaListener listener)`
設定するメディアリソース ☞ `setData(MediaData data)`, `setSound(MediaSound sound)`

```
public class AudioPresenter
extends java.lang.Object
implements MediaPresenter {

  // コンストラクタ
  protected AudioPresenter();

  // ベンダー定義の属性項目の境界

  // ベンダー定義属性の最小値
  protected static final int MIN_VENDOR_ATTR        =  64;
  // ベンダー定義属性の最大値
  protected static final int MAX_VENDOR_ATTR        = 127;

  // MediaListener に通知するイベント
  public static final int AUDIO_PLAYING             =   1; //再生開始
  public static final int AUDIO_STOPPED             =   2; //再生中断
  public static final int AUDIO_COMPLETE            =   3; //再生完了
  public static final int AUDIO_SYNC                =   4; //同期イベント
  public static final int AUDIO_PAUSED              =   5; //再生一時停止
  public static final int AUDIO_RESTARTED           =   6; //再生再開
```

```
  // ベンダー定義のイベント種類の境界

  // ベンダー定義のイベント種類の最小値
  protected static final int MIN_VENDOR_AUDIO_EVENT =  64;
  // ベンダー定義のイベント種類の最大値
  protected static final int MAX_VENDOR_AUDIO_EVENT = 127;

  // 属性種類の ID
  public static final int PRIORITY                  =   1; //再生優先順位属性
  public static final int SYNC_MODE                 =   2; //再生同期イベントの発生属性
  public static final int TRANSPOSE_KEY             =   3; //音程の変化
  public static final int SET_VOLUME                =   4; //音量
  public static final int CHANGE_TEMPO              =   5; //テンポ

  // 優先順位属性の値
  public static final int MIN_PRIORITY              =   1; //最も低い優先度
  public static final int NORM_PRIORITY             =   5; //普通の優先度
  public static final int MAX_PRIORITY              =  10; //最も高い優先度

  // 再生イベント発生の属性値
  public static final int ATTR_SYNC_OFF             =   0; //同期イベントを発生させない
  public static final int ATTR_SYNC_ON              =   1; //同期イベントを発生させる

  // オプションのオーディオ属性項目の境界

  // オプションのオーディオ属性の最小値
  public static final int MIN_OPTION_ATTR           = 128;
  // オプションのオーディオ属性の最大値
  public static final int MAX_OPTION_ATTR           = 255;

  // クラスメソッド
  public static AudioPresenter getAudioPresenter();
  public static AudioPresenter getAudioPresenter(int port);

  // インスタンスメソッド
  public MediaResource getMediaResource();              // Overrides MediaPresenter.getMediaResource()
  public void play();                                   // Overrides MediaPresenter.play()
  public void play(int time);
  public void pause();
  public void restart();
  public void setAttribute(int attr, int value);        // Overrides MediaPresenter.setAttribute()
  public void setData(MediaData data);
  public void setMediaListener(MediaListener listener); // Overrides MediaPresenter.setMediaListener()
  public void setSound(MediaSound sound);
  public void stop();                                   // Overrides MediaPresenter.stop()
  public int  getCurrentTime();
  public void setSyncEvent(int channel, int key);
}
```

コンストラクタの概要

■ protected **AudioPresenter**()

アプリケーションはこのコンストラクタを用いて、直接このクラスのインスタンスを生成することはできない。

メソッドの概要

■ `public MediaResource` **`getMediaResource`**`()`

設定されている MediaResource を返す。リソースが設定されていない場合は null を返す。

■ `public static AudioPresenter` **`getAudioPresenter`**`()`

再生する MediaResource が設定されていない、メディアイベントを通知する MediaListener が未登録である AudioPresenter を返す。

■ `public void` **`play`**`()`

設定されている MediaResource を先頭から再生する。リスナーが登録されている場合は、リスナーに AUDIO_PLAYING イベントを通知する。データを最後まで再生すると停止し、リスナーに AUDIO_COMPLETE イベントを通知する。また、データ再生中に play() を再度呼び出すと、再生中のデータを停止し、先頭から再生し直す。

■ `public void` **`setAttribute`**`(int` ***`attr`***`, int` ***`value`***`)`

再生方法に関する属性値を設定する。

引数： *attr* 属性項目を示す整数値
value 属性値

■ `public void` **`setData`**`(MediaData` ***`data`***`)`

再生する MediaResource として、MediaData を設定する。

引数： *data* 再生するメディアデータ

■ `public void` **`setMediaListener`**`(MediaListener` ***`listener`***`)`

メディアイベントを通知する MediaListener を登録する。

引数： *listener* メディアイベントを受け取る MediaListener。null を指定するとリスナーの登録を解除する

■ `public void` **`setSound`**`(MediaSound` ***`sound`***`)`

再生する MediaResource として、MediaSound を設定する。

引数： *sound* 再生するメディアサウンド

■ `public void` **`stop`**`()`

再生を停止する。リスナーが登録されている場合は、リスナーに AUDIO_STOPPED イベントを通知する。データ再生を停止しているときに stop() を再度呼び出すと、AUDIO_STOPPED イベントを通知するが、メディアデータ再生に関わる処理は何もしない。途中で再生を停止した場合でも、play() で再生を再開した場合は、データの先頭からの再生となる。

■ `public int` **`getCurrentTime`**`()`

メソッドが呼ばれた時点で再生しているポイントのメディアサウンドの曲頭からの時間を ms 単位で返す。

■ `public void` **`setSyncEvent`**`(int` ***`channel`***`, int` ***`key`***`)`

引数で指定されたチャンネルとキーに対応するノートメッセージを同期イベントとして設定する。再生中に呼び出しても無視される。また、TRANSPOSE_KEY による音階変更の影響を受けない。

引数： *channel* 同期イベントに利用するチャンネル
key 同期イベントに利用するキー。端末がサポートする MFi の有効キー範囲は 21〜119（MFi のデータとしては－ 24〜74）、MIDI の有効キー範囲は 0〜127

■ public void **play**(int *time*)

メディアデータを引数で指定された位置から再生する。そのほかの内容は play() と同じ。

引数：*time* 再生を開始する位置（単位はメディアデータの先頭からのミリ秒）

■ public void **pause**()

メディアデータの再生を一時停止する。再生の再開には restart() を用いる。一時停止されたときにリスナーが登録されていれば、リスナーに AUDIO_PAUSED イベントを通知する。データの再生が停止しているときにこのメソッドが呼ばれても何もしない。

■ public void **restart**()

pause() で一時停止したメディアデータの再生を再開する。再生が再開されたときにリスナーが登録されていれば、リスナーに AUDIO_RESTARTED イベントを通知する。停止している時にこのメソッドが呼ばれても何もしない。メディアデータの種類によってサポートされない場合がある。

■ public static AudioPresenter **getAudioPresenter**(int *port*)

再生するポート番号を指定して再生オブジェクトを取得。

引数：*port* ポート番号。1 から再生可能数-1 まで指定可能

主要メソッドの詳細

■ **setSyncEvent**(int *channel*, int *key*)

同期イベントを発生させるには、このメソッドを呼び出す前に setAttribute() の第一引数に AudioPresenter.SYNC_MODE を、第二引数に AudioPresenter.ATTR_SYNC_ON を指定して呼び出す必要がある。その後、このメソッドで設定すると、リスナーが登録されている場合、設定されたノートメッセージに対応する音が鳴るのと同時にイベントタイプ AUDIO_SYNC でリスナーが起動されるようになる。

ただし、AudioPresenter オブジェクトには 1 つの同期イベントのみ設定可能であるため、複数回呼び出して設定した場合、最後の設定のみが有効になる。同期イベント（AUDIO_SYNC）の発生の最小間隔は 100ms であるため、100ms 以内に発生した AUDIO_SYNC イベントは破棄される。引数で指定されたチャンネル、キーが端末サポート外の場合、このメソッドを呼び出しても何も行われない。

com.nttdocomo.ui

Button

クラス

このクラスは、Panel に配置するラベル付きのプッシュボタンを定義します。ラベル文字列はボタン中央に配置されます。

インスタンスが生成された時点では、可視状態でユーザー操作可能となっています。表示の可否を設定するには setVisible() を、ユーザー操作の可否を設定するには setEnabled() を用います。

requestFocus() は、ボタンにフォーカスを合わせるように要求しますが、パネルに追加される前に発行された要求は無視されます。ボタンが押されたとき、Button オブジェクトが配置されている Panel オブジェクトに ComponentListener が登録されていると、リスナーオブジェクトの componentAction() が呼び出されます。

DoJa3.0 から XString に対応しました。表示文字列として従来の文字列もしくは XString を指定することができます。これらは排他的に動作します。

Component から継承するメソッド：

getHeight(), getWidth(), getX(), getY(), setBackground(), setForeground(), setLocation(), setSize(), setVisible()

Interactable の実装

requestFocus(), setEnabled()

機種依存：

- ラベル文字列が画面の横幅に収まらない場合の表示

関連項目：

親クラス ☞ **Component**

実装しているインターフェイス ☞ **Interactable**

Panel への配置 ☞ **Panel**.add(Component *c*)

イベントの処理 ☞ **ComponentListener**.componentAction(Component *c*, int *type*, int *param*)

```
public final class Button
extends com.nttdocomo.ui.Component
implements com.nttdocomo.ui.Interactable {

  // コンストラクタ
  public Button();
  public Button(java.lang.String label);
  public Button(XString xLabel);

  // インスタンスメソッド
  public void requestFocus();         // Overrides Interactable.requestFocus()
  public void setEnabled(boolean b); // Overrides Interactable.setEnabled()
  public void setLabel(java.lang.String label);
  public void setLabel(XString xLabel);
}
```

コンストラクタの概要

■ public **Button**()

ラベル文字列が何もない Button を生成する。

■ public **Button**(java.lang.String *label*)

指定された文字列をラベル文字列とする Button を生成する。null の場合は、空文字列となる。

引数：*label*　生成する Button のラベル文字列

■ public **Button**(XString *xLabel*)

指定された XString の文字列をラベル文字列とする Button を生成する。null の場合は、空文字列となる。

引数：*xLabel*　生成する Button の XString のラベル文字列

メソッドの概要

- `public void` **`requestFocus`**`()`

  Button にフォーカスをセットするように要求する。

- `public void` **`setEnabled`**`(boolean` ***b***`)`

  Button へのユーザー操作の可否を設定する。

  引数：*b*　ユーザー操作を受け付ける（true）か、受け付けない（false）か

- `public void` **`setLabel`**`(java.lang.String` ***label***`)`

  Button のラベル文字列を設定する。Button のサイズが設定されているときに、文字列がそのサイズに収まらない場合の振る舞いは機種依存。

  引数：*label*　設定する Button のラベル文字列

- `public void` **`setLabel`**`(XString` ***xLabel***`)`

  Button のラベル文字列を XString の文字列で設定する。Button のサイズが設定されているときに、文字列がそのサイズに収まらない場合の振る舞いは機種依存。

  引数：*xLabel*　設定する Button の XString のラベル文字列

コンストラクタの詳細

- **`Button`**`(java.lang.String` ***label***`)`
- **`Button`**`(XString` ***xLabel***`)`

  引数で渡された文字列をラベル文字列とする Button を生成する。サイズが指定されない場合、コンポーネントのサイズは、指定されたラベル文字列が収まる最小のサイズとなる。ただし、幅が画面の横幅より大きくなる場合は、画面に表示できる幅となる。

com.nttdocomo.ui

Canvas　クラス

このクラスは、画面全体を１つのビットマップとして扱い、グラフィックスや文字を描画したり、ユーザーの入力イベントを受け取るための表示画面を定義します。Canvas には paint() が実装されていないので、通常は Canvas を継承するサブクラスで paint() を実装します。

Canvas のサブクラスのインスタンスを Display のクラスメソッド setCurrent() に渡すことでディスプレイに表示させます。

高レベル API の Panel との違いは、以下の点にあります。

- 高レベル API の各種コンポーネントを利用できない
- ComponentListener や KeyListener といったイベントリスナーの登録ができない
- タイトルバーを持たず、画面のスクロール機能もない

しかし、イベントリスナーで処理するイベントの多くは processEvent() でも扱うことができます。

paint() は Canvas への描画を行うメソッドで、Canvas が表示されるタイミングで自動的に呼び出されます。引数には、paint() の呼び出し元が用意し、初期化と必要な設定を行い利用可能にした Graphics オブジェクトを指定します。

repaint() は Canvas の再描画を行うメソッドですが、通常は paint() を呼び出すことと同じです。一定間隔で自動的に呼び出されていますが、明示的に呼び出すことも可能です。再描画する領域を指定することで再描画効率の向上を期待できます。

DoJa3.0 からは、カレントフレームでない Canvas では、repaint() や Graphics オブジェクトに対する描画メソッドの呼び出しは無視されます。また、i アプリ再開時に Canvas の描画内容をすべて描画できた場合、paint() は呼ばれません。

Frame から継承するメソッド：

`getHeight()`, `getWidth()`, `setBackground()`, `setSoftLabel()`

機種依存：

- processEvent() のスレッド割り当て
- paint() の引数の Graphics オブジェクトに対し、アプリケーションプログラマが Graphics.dispose() を呼び出した場合の振る舞い

例外：

- getKeypadState(int) で、引数に 0 未満の値が指定された場合に IllegalArgumentException が発生
- repaint() で、第 3、第 4 引数に 0 未満の値が指定された場合に IllegalArgumentException が発生
- 非活性化状態で imeOn() を呼び出した場合に IllegalStateException が、第 2、第 3 引数に不正な値が指定された場合に IllegalArgumentException が、すでに IME が起動している時に呼び出された場合に UIException(BUSY_RESOURCE) が発生

関連項目：

親クラス ☞ **Frame**

画面への表示 ☞ **Display**.setCurrent(Frame *frame*)

タイマーの設定 ☞ **ShortTimer**.getShortTimer(Canvas *canvas*, int *id*, int *time*, boolean *repeat*)

```
public abstract class Canvas
extends com.nttdocomo.ui.Frame {

  // コンストラクタ
  public Canvas();

  // IME イベントの通知（processIMEEvent() の引数 type に指定）
  public static final int IME_COMMITTED = 0; //入力確定
  public static final int IME_CANCELED  = 1; //入力キャンセル

  // インスタンスメソッド
  public Graphics      getGraphics();
  public int           getKeypadState();
  public abstract void paint();
  public void          processEvent(int type, int param);
  public void          repaint();
  public void          repaint(int x, int y, int width, int height);
  public int           getKeypadState(int group); //i アプリオプション API（option）
  public void          imeOn(String text, int displayMode, int inputMode);
  public void          processIMEEvent(int type, String text);

}
```

コンストラクタの概要

■ `public` **`Canvas`**`()`

Canvas オブジェクトを生成する。

メソッドの概要

■ `public Graphics` **`getGraphics`**`()`

Canvas の表示画面に相当する Graphics オブジェクトを返す。

■ `public int` **`getKeypadState`**`()`

複数のキーパッドの状態をビット列で返す。

■ `public abstract void` **`paint`**`(Graphics` ***`g`***`)`

Canvas への描画を行うタイミングで呼び出される。

引数： *g* paint() の呼び出し元が用意し、初期化、必要な設定を行い利用可能にした Graphics オブジェクト

■ `public void` **`processEvent`**`(int` ***`type`***`, int` ***`param`***`)`

低レベルイベントが通知されると呼び出される。

引数： *type* 通知された低レベルイベントのタイプ。Display クラスで定義される

param イベントのパラメータ。パラメータの意味はイベントにより異なる。パラメータを持たないイベントの場合は 0

■ `public void` **`repaint`**`()`

画面を再描画するタイミングで呼び出される。実際には、paint() を呼び出している。

■ `public void` **`repaint`**`(int` ***`x`***`, int` ***`y`***`, int` ***`width`***`, int` ***`height`***`)`

Canvas 内の一部の矩形領域内を再描画する。

引数： *x, y, width, height* 矩形領域の左上角の座標が（*x*，*y*）、幅 *width*、高さ *height* の矩形領域を再描画領域に指定する（ピクセル単位）

■ `public int` **`getKeypadState`**`(int` ***`group`***`)`

引数で指定されたグループに属するキーパッドの状態を返す。キーの状態を得るためにはあらかじめ Phonesystem.setAttribute(DEV_KEYPAD,group) を呼び出して、そのグループを有効にしておく必要がある。

引数： *group* キーのグループ

■ `public void` **`imeOn`**`(String` ***`text`***`, int` ***`displayMode`***`, int` ***`inputMode`***`)`

Canvas において、IME を起動する。このメソッドはブロックしない。IME での入力が完了すると、processIMEEvent() が呼ばれる。Canvas がカレントフレームでなかったり、ダイアログが表示されている場合、呼び出しは無視される。

引数： *text* IME に渡す初期文字列。null の場合は空文字列

displayMode 文字列の表示モード（TextBox.DISPLAY_ANY、TextBox.DISPLAY_PASSWORD）

inputMode 初期入力モード（TextBox.NUMBER、TextBox.ALPHA、TextBox.KANA）

ADF ファイルのエントリ

※太字のエントリ名は、ADF ファイルでの必須項目を示す。

エントリ名	値	目的
AppName	アプリケーション名。最大 16 バイトの日本語を含む任意の文字列を指定	携帯電話で表示されるアプリケーション名に使用される
AppVer	バージョン番号	アプリケーションのバージョン管理を行う
PackageURL	URL（最大 255 バイト）	ADF ファイルに対応する jar ファイルへの URL を指定する。相対パスを指定した場合、ベースディレクトリは ADF ファイルの位置となる
AppSize	ファイルサイズ	PackageURL エントリに指定した jar ファイルのサイズをバイト単位で指定する。最大 102400 バイト
AppClass	クラス名。大文字小文字の区別あり	ここに指定したクラスがメインクラスとして使用される。IApplication クラスのサブクラスを必要ならばパッケージ名を含めて指定する
AppParam	メインクラスのパラメータ。スペースで区切って複数のパラメータを指定可能。最大 255 バイト	IApplication.getArgs() で取得可能なパラメータを指定する。省略時はパラメータなしとして扱われる。ASCII 文字のみ指定可
ConfigurationVer（KvmVer）	J2ME コンフィグレーションバージョン （書式：CLDC-x.x）	i アプリが対応する Configuration と Kvm のバージョン番号を指定する。省略時はすべてのバージョンで動作するものとして扱われる。ConfigurationVer と KvmVer は同義だが、１つの ADF ファイルに同時に指定することはできない。KvmVer は DoJa1.0 プロファイルとの互換性のために用意されている。現在は CLDC-1.0 までサポートされている
ProfileVer	i アプリ実行環境のプロファイル （書式：Doja-x.x）	省略時は i アプリ実行環境のすべてのバージョンで動作するものとして扱われる。現在は Doja-3.5 までサポートされている
SPsize	スクラッチパッドのサイズ	スクラッチパッドのサイズをバイト単位で指定する。省略時はスクラッチパッドを使用しないものとして扱われる。DoJa-3.0 以降では、スクラッチパッドを内部的に最大 16 分割可能。分割する場合は、値に個々のバイト数をカンマで区切って列挙する。この際、カンマの前後に空白を入れてはいけない。分割されたスクラッチパッドへのアクセスは、 scratchpad:///<number> のように URL にスクラッチパッドの番号を指定する。最大 409600 バイト
LastModified	日時（Dow, DD Mon YYYY HH:MM:SS TimeZone） Dow	i アプリの最終更新日時。携帯電話から「アプリケーションのバージョンアップ」を実行したときの判断に使われる。DoJa1.0 プロファイルではタイムゾーンの指定はサポートされていないため、DoJa1.0/2.0 両者で ADF ファイルを共有する場合、タイムゾーンは省略する 曜日（Sun, Mon, Tue, Wed, Thu, Fri, Sat）

機種実装依存一覧

主要機種のシステム情報

スペック	P900i	F900i	N900i	SH900i	D900i
プラットフォーム名	P900i	F900i	N900i	SH900i	D900i
画面の横幅（ピクセル）	240	240	240	240	240
画面の縦幅（ピクセル）	240	240	240	252	270
表示できる色数	65536	65536	65536	65536	262144
デフォルトのフォントサイズ	6 × 12	6 × 12	6 × 12	6 × 12	6 × 12
利用可能な全メモリ（バイト）	2867200	2097152	3072000	8192000	1536000
空きメモリ容量（バイト）	2431796	2076360	2636196	7911356	1514244
サブディスプレイへの描画	none	none	available	available	none
サブディスプレイの画面サイズ			120 × 30	120 × 120	
サブディスプレイの色数			4096	65536	

■ **`System.getProperty("microedition.encoding")`** の値

文字のエンコーディングは、P900i ／ F900i ／ N900i ／ SH900i ／ D900i とも「SJIS」が返る。

■ **`System.getProperty("microedition.configuration")`** の値

コンフィギュレーションは、P900i ／ F900i ／ N900i ／ SH900i ／ D900i とも「CLDC-1.0」が返る。

■ **`System.getProperty("microedition.profiles")`** の値

プロファイルのバージョンは、P900i ／ F900i ／ N900i ／ SH900i ／ D900i とも「DoCoMoProfile-3.5」が返る。

i モード対応 HTML5.0 ユーザーエージェント一覧

機種	ユーザーエージェント	キャッシュの大きさ
F900i	DoCoMo/2.0 F900i(c100;TB;W22H12)	（ブラウザからの通信時、ADF 取得時）*1
	DoCoMo/2.0 F900i(c100;TB;W18H10)	（ブラウザからの通信時、ADF 取得時）*1*3
	DoCoMo/2.0 F900i(c100;TB;W28H15)	（ブラウザからの通信時、ADF 取得時）*1*3
	DoCoMo/2.0 F900i(c100;TD)	（JAR 取得時）
	DoCoMo/2.0 F900i(c100;TJ)	（i アプリからの通信時）
N900i	DoCoMo/2.0 N900i(c100;TB;W24H12)	（ブラウザからの通信時、ADF 取得時）*1
	DoCoMo/2.0 N900i(c100;TB;W20H10)	（ブラウザからの通信時、ADF 取得時）*1*3
	DoCoMo/2.0 N900i(c100;TB;W30H15)	（ブラウザからの通信時、ADF 取得時）*1*3
	DoCoMo/2.0 N900i(c100;TD)	（JAR 取得時）
	DoCoMo/2.0 N900i(c100;TJ)	（i アプリからの通信時）
P900i	DoCoMo/2.0 P900i(c100;TB;W24H11)	（ブラウザからの通信時、ADF 取得時）*2
	DoCoMo/2.0 P900i(c100;TB;W20H09)	（ブラウザからの通信時、ADF 取得時）*2*3
	DoCoMo/2.0 P900i(c100;TB;W30H14)	（ブラウザからの通信時、ADF 取得時）*2*3
	DoCoMo/2.0 P900i(c100;TD)	（JAR 取得時）
	DoCoMo/2.0 P900i(c100;TJ)	（i アプリからの通信時）
SH900i	DoCoMo/2.0 SH900i(c100;TB;W24H12)	（ブラウザからの通信時、ADF 取得時）*1
	DoCoMo/2.0 SH900i(c100;TD)	（JAR 取得時）
	DoCoMo/2.0 SH900i(c100;TJ)	（i アプリからの通信時）
F900iT	DoCoMo/2.0 F900iT(c100;TB;W22H12)	（ブラウザからの通信時、ADF 取得時）*1
	DoCoMo/2.0 F900iT(c100;TB;W18H10)	（ブラウザからの通信時、ADF 取得時）*1*3
	DoCoMo/2.0 F900iT(c100;TB;W28H15)	（ブラウザからの通信時、ADF 取得時）*1*3
	DoCoMo/2.0 F900iT(c100;TD)	（JAR 取得時）
	DoCoMo/2.0 F900iT(c100;TJ)	（i アプリからの通信時）
P900iV	DoCoMo/2.0 P900iV(c100;TB;W24H11)	（ブラウザからの通信時、ADF 取得時）*2
	DoCoMo/2.0 P900iV(c100;TB;W20H09)	（ブラウザからの通信時、ADF 取得時）*2*3
	DoCoMo/2.0 P900iV(c100;TB;W30H14)	（ブラウザからの通信時、ADF 取得時）*2*3
	DoCoMo/2.0 P900iV(c100;TD)	（JAR 取得時）
	DoCoMo/2.0 P900iV(c100;TJ)	（i アプリからの通信時）
N900iS	DoCoMo/2.0 N900iS(c100;TB;W24H12)	（ブラウザからの通信時、ADF 取得時）*1
	DoCoMo/2.0 N900iS(c100;TB;W20H10)	（ブラウザからの通信時、ADF 取得時）*1*3
	DoCoMo/2.0 N900iS(c100;TB;W30H15)	（ブラウザからの通信時、ADF 取得時）*1*3
	DoCoMo/2.0 N900iS(c100;TD)	（JAR 取得時）
	DoCoMo/2.0 N900iS(c100;TJ)	（i アプリからの通信時）
D900i	DoCoMo/2.0 D900i(c100;TB;W20H10)	（ブラウザからの通信時、ADF 取得時）*2
	DoCoMo/2.0 D900i(c100;TD)	（JAR 取得時）
	DoCoMo/2.0 D900i(c100;TJ)	（i アプリからの通信時）

*1 ブラウザからの通信時かつ画像 OFF 設定時は"TB"は"TC"に置換される（ADF 取得時も"TB"は"TC"に置換される）
*2 ブラウザからの通信時かつ画像 OFF 設定時は"TB"は"TC"に置換される（ADF 取得時は常に"TB"）
*3 ユーザー設定フォント時