

i モード対応 Java コンテンツ開発ガイド

～ 詳細編 ～

第 1.1 版

平成 13 年 5 月 14 日
株式会社 N T T ドコモ



本製品または文書は著作権法により保護されており、その使用、複製、再頒布および逆コンパイルを制限するライセンスのもとにおいて頒布されます。NTTドコモ（その他に許諾者がある場合は当該許諾者も含めて）の書面による事前の許可なく、本製品および関連する文書のいかなる部分も、いかなる方法によっても複製することが禁じられます。フォントを含む第三者のソフトウェアは、著作権法により保護されており、その提供者からライセンスを受けているものです。

Sun、Sun Microsystems、Java、J2ME および J2SE は、米国およびその他の国における米国 Sun Microsystems, Inc.の商標または登録商標です。サンロゴマークは、米国 Sun Microsystems, Inc.の登録商標です。

「i モード／アイモード」は NTT ドコモの登録商標、「i アプリ／i appli／アイアプリ」は NTT ドコモの商標です。

その他、掲載されている会社名、製品名、サービス名は各社の商標または登録商標です。

文書は現状有姿で提供されており、その商品性、特定目的への適合性、第三者の権利の非侵害を含み、明示的または黙示のいかなる条件、表明および保証も行われず、また提供されません。但し、かかる保証の否定が、適用される法律の下で無効である場合は、この限りではありません。

Copyright 2000 NTTDoCoMo, Inc. 2-11-1 Nagata-cho, Chiyoda-ku, Tokyo, 100-6150, Japan and Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California, 94303, U.S.A. All rights reserved.

本書は、デジタル・ムーバ 503i シリーズに搭載される Java アプリケーション実行環境（本書中では i モード Java アプリケーション環境と呼びます）をターゲットとして記述されています。i モード Java とは、i モード対応 Java の便宜的な名称です。

本書では、i モード対応 Java API 向け Java 2, Micro Edition, Wireless SDK のサンプルアプリケーションをベースとしたコード例を記載しております。ただし SDK のバージョンアップなどの理由により、必ずしもサンプルアプリケーションとコード例は一致しませんのでご了承ください。

目 次

本書の目的.....	6
対象読者	6
本書の構成.....	6
表記上の規則.....	7
関連ドキュメント.....	7
第 1 章 はじめに	9
1.1 i モード Java アプリケーション環境の利点.....	10
1.1.1 スタンドアロン型およびクライアント/サーバー型アプリケーションの最適化されたサポート.....	12
1.1.2 永続データのローカル記憶装置サポート	13
1.2 Java Application Manager	14
1.3 Java 対応 i モードの Java セキュリティモデル.....	14
1.4 アプリケーションのインストールとライフサイクル	15
1.4.1 アプリケーションのダウンロード	15
1.4.2 アプリケーションの起動	16
1.4.3 アプリケーションの更新	17
1.4.4 アプリケーションの削除	17
第 2 章 i モード Java アプリケーション環境.....	19
2.1 i モード Java アプリケーション環境.....	19
2.1.1 ネイティブアプリケーションとオペレーティングシステム.....	20
2.1.2 Java Application Manager.....	20
2.1.3 K Virtual Machine.....	20
2.1.4 CLDC ライブラリ	21
2.1.5 i モード Java の拡張.....	21
2.1.6 メーカーによる拡張.....	21
2.2 i モード Java アプリケーション環境における API の概要.....	21
2.2.1 ドコモシステムアーキテクチャ	22
2.2.2 CLDC	23
CLDC でサポートされる J2SE クラス	23
システムクラス	24
データ型クラス	24
Collection クラス	24
入出力クラス	24
Micro Edition クラス	25
カレンダーと時間のクラス	25
追加のユーティリティクラス	25
エラーと例外クラス	25
CLDC プラットフォームではサポートされない JVM 機能.....	26
浮動小数点サポート	26
JNI	26
スレッドグループ	26
ファイナライズ.....	26
Generic Connection フレームワーク (javax.microedition.io パッケージ)	27
2.2.3 i モード Java 拡張.....	29
com.nttdocomo.io パッケージ	30
com.nttdocomo.net パッケージ	31

com.nttdocomo.util パッケージ	31
com.nttdocomo.lang パッケージ	32
com.nttdocomo.ui パッケージ	33
IApplication.....	37
テキスト変換.....	37
ScratchPad.....	38
2.2.4 メーカーによる拡張	38
第3章 設計上の考慮事項.....	39
3.1 Java 対応 i モードの特性.....	39
3.2 メモリの問題.....	40
3.2.1 メモリ使用量の最小化	41
アプリケーション実行メモリ	41
アプリケーション記憶メモリ	42
3.2.2 アプリケーションサイズ	42
3.3 使い易さと GUI の設計.....	42
3.3.1 ターゲットユーザー向けの設計	43
3.3.2 操作カテゴリの理解.....	43
3.3.3 スクロール	43
3.3.4 ソフトキー	44
3.3.5 データ入力.....	45
3.3.6 パスワード入力	45
3.3.7 スレッドの使用	45
3.4 セキュリティ	45
3.5 無線ネットワークでの操作.....	46
3.6 処理の中断と再開.....	46
3.7 ハードウェアのアクセス.....	47
3.8 エラー処理.....	47
第4章 ユーザーインタフェースの設計.....	48
4.1 高レベル UI コンポーネントによるプログラミング	51
4.1.1 Panel の使用	51
4.1.2 Dialog の使用	51
4.1.3 コンポーネントの使用	52
Label	52
ImageLabel.....	53
Button	54
ListBox.....	54
TextBox.....	57
Ticker	57
VisualPresenter	58
4.1.4 コンポーネントの配置	59
4.1.5 フォーカスとスクロール	61
4.1.6 高レベルイベントの処理	61
4.2 低レベル API によるプログラミング	61
4.2.1 Canvas の使用	61
4.2.2 グラフィックスの描画	62
4.2.3 カラーサポート	64
4.2.4 低レベルイベントの処理	64
4.3 イベントリスナー	66
4.3.1 ComponentListener	66
4.3.2 SoftKeyListener.....	68

4.3.3 KeyListener	68
4.3.4 MediaListener.....	68
4.3.5 TimerListener	69
4.4 マルチメディアデータの使用.....	69
第5章 通信の制御	76
5.1 クライアント/サーバプログラミング	76
5.1.1 接続の終了	80
5.2 セッション管理.....	80
5.3 HTTPS による安全な通信.....	82
第6章 テキスト処理	85
6.1 テキスト処理.....	85
6.2 Java 対応 i モードのグラフィックス文字	86
第7章 ScratchPad とリソースの使用	89
7.1 ScratchPad に対する読み取りと書き込み	89
7.2 リソースの読み取り	91
7.3 エラー処理.....	92
第8章 ハードウェアのアクセス.....	94
第9章 アプリケーションのビルド、テスト、およびパッケージ化.....	95
9.1 アプリケーション開発環境のインストール	95
9.2 アプリケーション開発環境のコンポーネント	95
9.3 開発サイクルの概要.....	96
9.4 サーバー側アプリケーションの作成.....	98
9.5 アプリケーションのパッケージ化.....	98
9.5.1 ADF の作成.....	98
9.5.2 JAR ファイルの作成.....	100
9.6 アプリケーションのテスト	100
第10章 アプリケーションの配布.....	102
10.1 アプリケーション配布のための Web サーバーの構成.....	102
10.2 サーバーアプリケーション実行のための Web サーバーの構成.....	103
10.3 特定メーカーの Java 対応 i モードへのダウンロード	103
付録 A サンプルコード: バンキングアプリケーション.....	104
クライアントアプリケーション	104
サーバーアプリケーション	104
付録 B サンプルコード: 株価情報/株価監視アプリケーション.....	105
付録 C サンプルコード: スペースインベーダゲーム.....	107
付録 D サンプルコード: ユーザーインタフェースアプリケーション.....	108
用語集	109

本書の目的

『i モード対応 Java コンテンツ開発ガイド』は、i モード対応の携帯電話向けに Java ベースの新しいサービスを提供するアプリケーションの設計や開発に必要な情報を、i モード向けコンテンツプロバイダに提供するものです。このガイドの目的は、アプリケーションの作成や開発に役立つシンプルなモデルを提供することにあります。さらに、このガイドには、この技術に関する説明と、使い易い Java アプリケーションを作成する上でアプリケーション開発者が準拠すべきプロセスの詳細が述べられています。この巻頭部分では、さらに次の項目について説明しています。

- 対象読者
- 本書の構成
- 表記上の規則
- 関連ドキュメント

対象読者

『i モード対応 Java コンテンツ開発ガイド』には、i モード対応の携帯電話で使用するアプリケーションの開発者向けの情報が記載されています。したがって、アプリケーション設計者や、サーバーおよびクライアントアプリケーションのプログラマ、アプリケーションの導入担当者は、このガイドを読む必要があります。このガイドでは、次の知識を前提としています。

- Java プログラミング言語を使ったネットワークアプリケーションの基礎知識
- 開発したアプリケーションが実行される i モード対応携帯電話の製品基本仕様

本書の構成

次の表に、各主題について記載している章を示します。

内容	参照先
i モード Java アプリケーション環境の紹介と概要	第 1 章、第 2 章
設計上の考慮事項	第 3 章
仕様の解説	第 4 章～第 8 章
開発プロセス	第 9 章、第 10 章
サンプルコードの解説	付録 A～付録 D

表記上の規則

このガイドで使用されている表記上の規則を次に示します。

表記	意味
タイプ文字	タイプ文字は、そのテキストがコード例、クラス、インタフェース、メソッド、またはデータ型であることを示します。
ボールドテキスト	ボールドは、そのテキストがメニューやソフトキーのオプションであることを示します。

関連ドキュメント

このガイドに関連するドキュメントを次に示します。

- 『i モード対応 Java コンテンツ開発ガイド API リファレンス編』
- 『i モード対応 Java API 向け Java 2, Micro Edition Wireless SDK ユーザーズガイド』
- 『Java 2 Platform, Micro Edition (J2ME) Connected Limited Device Configuration (CLDC) specification』 (<http://java.sun.com/products/cldc/>)

なお、上記に含まれる URL は予告なく変更される場合があります。

第 1 章

はじめに



Java 対応 i モードでは、電話やインターネットアクセスの他に、i モードコンテンツプロバイダの Web サイトから Java アプリケーションをダウンロード、実行する機能がサポートされます。このガイドは、Java 対応 i モード向けに Java アプリケーションを開発するソフトウェアエンジニア向けのプログラミングガイドです。

Java 対応 i モードでは、Java 2 Platform Micro Edition (J2ME) の Connected, Limited Device Configuration (CLDC) がサポートされます。J2ME CLDC では、小型のネットワーク接続デバイス向けの Java 仮想マシン仕様およびクラスライブラリに関して必要最小限の機能を定義します。CLDC は、K Virtual Machine (KVM) をベースに作成されています。KVM は、携帯電話（本書では特段の注記がない限り、「携帯電話」は Java 対応 i モード携帯電話のことを指します）のようにメモリや CPU、電池消費に制約があるデバイス向けに設計されたコンパクトで移植性の高い Java Virtual Machine です。KVM は、Java 2 ファミリの中でより上位に位置付けられるエディションの Java 実行環境に含まれる機能の多くを共有していますが、小型デバイスの要件を満たすために、小型デバイスの制約に合わせて最適化されています。J2ME/CLDC プラットフォームはコンパクトであるとはいえ、必要な Java Virtual Machine やクラスライブラリを備えていますので、i モード向けコンテンツプロバイダは、i モードユーザーが必要に応じてインストールできる内容豊富でエキサイティングなアプリケーションを開発できます。

Java には write-once, run-anywhere という特性があり、コンテンツプロバイダの作成したアプリケーションは、CPU やオペレーティングシステムに関係なく、どのメーカーの携帯電話でも容易に動作させることができます。Java にはアプリケーションをインターネット経由で安全に流通させる手段が備わっているため、i モードユーザーは、i モードコンテンツプロバイダの Web サイトに接続するだけでアプリケーションをいつでもダウンロードし、インストールすることができます。Java を使えば、i モードコンテンツの開発者は、ゲームや強固な電子商取引サービスといった広範囲のアプリケーションを開発することができます。i モードへの Java の搭載により、より新しく、よりダイナミックなサービスを展開することが可能になります。

この章では、Java 対応 i モードにおけるアプリケーション実行環境（i モード Java アプリケーション環境）の全般的な特徴と機能について説明します。

1.1 i モード Java アプリケーション環境の利点

Java コンテンツプロバイダは、Java 対応 i モード向けアプリケーションを Java プログラミング言語を使って作成します。i モード Java アプリケーション環境では、携帯電話のユーザーインターフェース、通信、テキスト変換、グラフィックス、マルチメディア、携帯電話上へのデータの保管といった処理を制御する API が提供されます。さらに、i モード Java アプリケーション環境では、メーカーから特定の機種向けに拡張 API が提供されることがあります。

i モード Java アプリケーション環境でこのアーキテクチャを選択した理由は 2 つあります。1 つは、Java 対応 i モード向けアプリケーションの開発と導入をできるだけ容易にするためであり、もう 1 つは、携帯電話のユーザーがこれらのアプリケーションをできるだけ快適に使用できるようにするためです。

i モード Java アプリケーション環境には、この他に次の機能があります。

- Java アプリケーションやコンテンツの動的な配布
- Java Application Manager (JAM)
- シンプルなサンドボックスモデルなどのセキュリティ機能。このモデルでは、`java.lang.*`パッケージや `java.util.*`パッケージのクラスなど、安全な Java API セットだけがアプリケーションプログラムに提供されます。

i モード Java アプリケーション環境は、次の 3 つの API レイヤから構成されています。

- J2ME/CLDC で規定された API
- i モード Java の拡張 API
- メーカーによる拡張 API

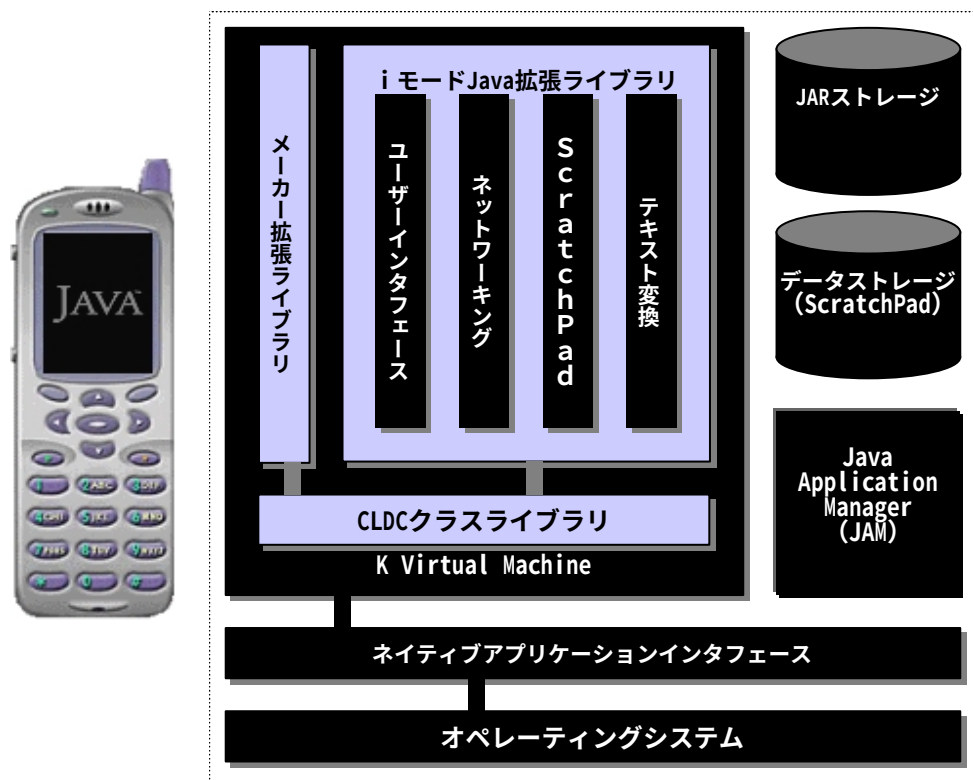


図 1. Java 対応 i モードにおける i モード Java アプリケーション環境

注意事項：

- 多くのアプリケーションでは、J2ME/CLDC API と i モード Java 拡張 API だけを使用します。しかし、特殊なアプリケーションの中には、特定の携帯電話固有の機能を実行するためにメーカーによる拡張 API を使用するものがあります。メーカーによる拡張 API を使用するアプリケーションは、そのメーカーの携帯電話でしか動作しないため注意が必要です。

J2ME/CLDC は、資源に制約がある小型のネットワーク接続デバイス向けの標準的な Java プラットフォームを定義したものです。このようなデバイスには、次の特徴があります。

- ・ メモリの全体量が少ない。
- ・ 消費電力が限られている。一般には電池で動作する。
- ・ 何らかのネットワークに接続される。途切れやすい無線ネットワークに接続されていることも多く、帯域幅も限られている。
- ・ サービスのために最適化されたユーザーインタフェースが必要となる。

CLDC 仕様でサポートされる一般的なデバイスには、携帯電話、双方向ポケットベル、携帯情報端末、オーガナイザ、家電機器、POS 端末などがあります。

J2ME/CLDC は、Java 対応 i モード向けアプリケーションの導入に使用される安定したプラットフォームです。上位エディションの Java 実行環境と同じように、J2ME でも、Java 技術の利点として知られている次のような特性があります。

- 時、場所、デバイスを選ばない、製品間での実装の一貫性
- 多くの開発者と多くの開発ツールに支えられた高レベルなオブジェクト指向プログラミング言語環境
- コードの移植性
- ネットワークによる安全なアプリケーションの配布
- J2SE や J2EE への上位拡張性

CLDC 上に構築される i モード Java 拡張には、次の機能を行う API が含まれています。

- HTTP や HTTPS によるネットワーク接続
- ユーザーインターフェースを定義するコンポーネント
- 低レベルのグラフィックス制御
- アプリケーションデータの保存に使用する ScratchPad

さらにメーカーによっては、自社の携帯電話の商品性を高めるため、メーカー特有の機能を制御するために独自の拡張を提供する場合があります。メーカーによる拡張についての詳細は別途公開されます。

1.1.1 スタンドアロン型およびクライアント/サーバー型アプリケーションの最適化されたサポート

i モード Java アプリケーション環境では、スタンドアロン型アプリケーションとクライアント/サーバー型アプリケーションの両方を作成することができます。

スタンドアロン型アプリケーションでは、アプリケーションとデータの両方が携帯電話上に存在し、サーバーと対話動作することはありません。ゲームや計算機、ユーティリティの多くはスタンドアロン型アプリケーションとして作成することができます。

スタンドアロン型アプリケーションでは、そのアプリケーション本来の目的を達するためにサーバーに接続することはありませんが、アプリケーションを携帯電話にダウンロードする際には、サーバーとの接続が必要です。アプリケーションを、インターネットを介して携帯電話にダウンロードできるようにするための方法などについては、第 10 章を参照してください。

スタンドアロン型アプリケーションにも価値があるとはいえ、Java 対応 i モードの魅力あるアプリケーションの多くはクライアント/サーバー型アプリケーションの形態を取るでしょう。クライアント/サーバー型アプリケーションでは、携帯電話の制約された処理能力が、サーバーのはるかに大きな処理能力や強力なデータ検索能力によって拡張されます。i モード Java アプリケーション環境には、クライアント/サーバー型アプリケーションの開発をサポートするために、HTTP および HTTPS に基づくネットワーク接続を行う API が含まれています。

クライアント/サーバー型アプリケーションでは、クライアントコンピュータ(この場合は携帯電話)の処理能力に、リモートサーバーの処理能力やデータアクセス機能を結合することができます。クライアント/サーバー型アプリケーションは、Java 対応 i モードのような Web 対応の小型デバイスを強力で魅力的なものにします。携帯電話は処理能力や記憶装置容量が限られていますが、強力な記憶装置とプロセッサを持つサーバーへの入り口として機能することができます。携帯電話にはデータ入力や表示サイズに制限があるものの、適切な設計やプログラミングに基づいてアプリケーションを作成することにより、Java 対応 i モードのユーザーは、快適に情報にアクセスしたり、電子商取引のようなりもートサービスを受けることができるようになります。

クライアント/サーバー型アプリケーションにおけるサーバーの代表的な役割は、要求されたデータをデータベースから検索し、それを要求元のクライアントに返すというものです。しかし場合によっては、クライアントの限られた処理能力とメモリ容量では不適切または不可能な計算処理をサーバーで代行することもあります。

クライアント/サーバー型アプリケーションの作成については、5.1 項を参照してください。またクライアント/サーバープログラミングの例については、付録 A と C のバンキングデモとスペースインベーダーのサンプルコードを参照してください。

1.1.2 永続データのローカル記憶装置サポート

i モード Java アプリケーション環境では、ScratchPad と呼ぶローカル記憶装置のメカニズムがサポートされます。このメカニズムでは、永続性を持つデータ（アプリケーションの終了後も保持され続けるデータ 永続データとも言います）の保管領域が携帯電話内の記憶装置に確保されます。永続データを格納する必要があるスタンドアロン型アプリケーションは ScratchPad にデータを格納しますが、クライアント/サーバーアプリケーションは、通常は一部の情報を ScratchPad に格納するだけで、残りのデータはサーバー側に格納します。

サーバー側の永続データは、ほとんどの場合、データベースに格納されます。クライアントデータは HTTP リクエストでサーバーに送信され、HTTP リクエストがサーブレットや CGI スクリプトと対話します。そして、サーブレットや CGI スクリプトが HTTP リクエストとデータベースアクセスプロトコルとの仲介として働きます。このため、サーバー側には常に HTTP サーバーが必要です。

永続データの格納場所を決定するには、データの性質を考慮する必要があります。永続データを失っても影響が小さい場合は、ScratchPad に格納し、そうでない場合は、サーバー側に格納します。ScratchPad に保存されたデータは、携帯電話のハードウェア的な故障などの原因によって失われてしまう可能性があるためです。

Java 対応 i モードのセキュリティ機能の一つに、Java Application Manager(JAM)によって個々のアプリケーション用に個別に割り当てられた ScratchPad メモリの領域は、他のアプリケーションからはアクセスできないという点があります。また、携帯電話上で同時に動作可能な Java アプリケーションは 1 つだけであり、複数のアプリケーションが RAM 上でデータを共有することもできません。複数のアプリケーション間でデータを共有する場合は、サーバーを各アプリケーション共通のデータ格納領域として使用する必要があります。

1.2 Java Application Manager

i モード Java アプリケーション環境が搭載されている携帯電話には、携帯電話にインストールされている Java アプリケーションを管理する機能が含まれています。Java Application Manager (JAM)は次のことを行います。

- 携帯電話に格納されている Java アプリケーションをリスト表示する。
- Java アプリケーションの実行管理（起動や強制終了など）を行う。
- Java アプリケーションをインストールまたは更新する。
- 既存の Java アプリケーションを削除する。アプリケーションの削除はユーザーの明示的な指示に基づいてのみ行われます。

JAM は KVM から独立したネイティブコンポーネントであり、これをアプリケーションから制御することはできません。

アプリケーションを終了させようとしても終了しない場合は、携帯電話のシステム割り込みコマンドに対応するメーカーの指定するキーシーケンスに従ってキーを押すことによって、JAM にアプリケーションの強制終了を指示することができます。強制終了の方法は携帯電話各機種により異なります。

1.3 Java 対応 i モードの Java セキュリティモデル

CLDC に用意されたセキュリティ機能の他に、Java Application Manager (JAM)では、Java 対応 i モードの Java 環境特有の機能として、次のセキュリティ機能が提供されます。

- 携帯電話上では、同時に 1 つの Java アプリケーションしか動作できません。そのため、あるアプリケーションが別のアプリケーションに干渉したり、別のアプリケーションのデータにアクセスしたりすることはできません。ただしこれは、アプリケーションが、情報をダウンロードしながらユーザー入力を受け取るといったいくつかの操作を同時に実行できないことを意味するわけではありません。
- Java Application Manager (JAM)は KVM から独立したネイティブコンポーネントであり、Java アプリケーションからこれを制御することはできません。Java アプリケーションのダウンロードや管理は JAM が行います。そのためユーザーにとっては、ダウンロードしたアプリケーションの不正な行為やバグによる誤動作などから携帯電話が保護されることになります。
- Java アプリケーションは、携帯電話の記憶装置 (ScratchPad) に、制限されたアクセスしかできません。あるアプリケーションに割り当てられた ScratchPad の領域に、他のアプリケーションからアクセスすることは許可されません。したがって、ダウンロードされたアプリケーションが携帯電話の限られた記憶メモリで不正な行為（他のアプリケーションのために保存された個人情報などをユーザーの知らないうちに読み込んで別のサーバーに送信するなど）を行うことはできません。
- Java アプリケーションがメモリダイアル機能などのネイティブアプリケーションにアクセスすることはできません。Java アプリケーションが携帯電話の基本機能に干渉する心配はないため、アプリケーションの不正行為によって携帯電話の基本的なサービスが使用不可能になるようなことはありません。そのためユーザーは、Java アプリケーションを安心してダウンロードおよび実行することができます。
- i モード Java アプリケーション環境の通信機能では、Java 対応 i モードでサポートされる HTTP 通信および HTTPS 通信に対する簡単で使いやすいインターフェースが提供されます。
- Java アプリケーションは、そのアプリケーション自身のダウンロード元であるサーバーとしか通信できません。このネットワークセキュリティ機能により、アプリケーションが、ユーザーの知らない別のサーバーに対して情報を送信するようなことはありません。

1.4 アプリケーションのインストールとライフサイクル

携帯電話上でアプリケーションを実行するためには、コンテンツプロバイダの i モード対応 Web サイトからアプリケーションをダウンロード（インストール）しなければなりません。アプリケーションがダウンロードされると、アプリケーションの保存のために必要な各メモリを JAM が割り当てます。アプリケーション JAR ファイルとアプリケーションの ScratchPad は、どちらも携帯電話の不揮発性メモリに格納されます。

次の各項では、アプリケーションのダウンロード、実行、削除といったライフサイクルについて説明します。

1.4.1 アプリケーションのダウンロード

アプリケーションのダウンロードは一般的に以下のような流れで行われます。なお、携帯電話の具体的な動作の詳細はメーカーにより一部異なる場合があります。

1. i モードブラウザを使って、必要なアプリケーションのダウンロードリンク先を決めます。

アプリケーションのダウンロードは、i モードブラウザの操作を契機として開始されます。ダウンロードを行うための Web ページ（HTML コンテンツ）には、通常、Java アプリケーションの説明文や、ダウンロードを開始するためにクリックするリンクが表示されています。リンクには、アプリケーションデスクリプタファイル（「.jam」拡張子をもつファイル ADF、JAM ファイルとも呼ばれます。以降本書では ADF とします）への参照が含まれています。ADF は SJIS コードのテキストファイルであり、アプリケーション情報を表すキーと値のペアが 1 行ずつ記述されています。

このファイルの目的の 1 つは、ダウンロードの開始前に、選択された Java アプリケーションを携帯電話に正しくダウンロードできるかどうかを JAM が判定できるようにすることです。事前に正しくダウンロードできないと判定できれば、Java アプリケーションを実際に携帯電話にダウンロードするコストが節約されます。数 K バイトの Java アプリケーションに対して ADF は 100～300 バイト程度と小さく、Java アプリケーション全体をダウンロードする前に ADF をダウンロードして保存に必要な記憶容量やその機種に対する互換性をチェックすることは、通信コストの点から有利となります。

なお ADF の内容に関する詳細は 9.5.1 項を、アプリケーションを携帯電話に配布するための Web サーバの準備については第 10 章を参照してください。

2. リンクをクリックして、インストールプロセスを開始します。

このユーザー操作により、ブラウザは、JAM に ADF の位置を示す URL を渡します。以降、アプリケーションのダウンロードとインストールは JAM により行われます。

3. ADF の内容を検査し、アプリケーションのインストール可否を確認します。

JAM は、指定された ADF にすべての必須キーが指定されているかチェックします。そして ADF の記述内容（アプリケーションサイズや ScratchPad サイズ、アプリケーションのターゲットとする機種情報など）から、そのアプリケーションをその携帯電話にインストール可能かどうかチェックし、可能であれば次のステップに進みます。

なお、インストールしようとしているアプリケーションと同じアプリケーション名（ADF の AppName キー）、JAR ファイル URL（ADF の PackageURL キー）を持つアプリケーションがすでに携帯電話にインストールされている場合、そのアプリケーションは同一と見なされインストールすることはできません（1.4.3 項に示すアプリケーションの更新は可能です）。

4. アプリケーションのダウンロードが行われます。

アプリケーションが携帯電話上にインストール可能であると判断されたら、JAM は ADF の PackageURL キーを使って JAR ファイルの URL を入手し(PackageURL キーが相対 URL で記述されている場合は、ADF の位置をベース URL として使用します)、HTTP ダウンロードプロセスを開始します。もしダウンロードプロセス中にユーザーの中断指示や通信エラーが発生したら、JAM はダウンロード中のアプリケーションを破棄して携帯電話をダウンロードが行われる前の状態に戻します。

5. インストール済みアプリケーションのリストを更新します。

JAM は、インストール済み Java アプリケーションのリストにアプリケーションを追加します。また ADF の情報とアプリケーションの JAR ファイルを不揮発メモリ上に保存します。

インストールされたアプリケーションは、ユーザーの好きなときに起動することができます。

1.4.2 アプリケーションの起動

アプリケーションの起動手順やユーザーインターフェースはメーカーによって異なりますが、一般には次のようにして行われます。

1. ユーザーの指示によりインストール済み Java アプリケーションのリストを表示します。
2. ユーザーがリストから起動する Java アプリケーションを選択します。
3. JAM が ADF の AppClass キーで示されるアプリケーションのメインクラス名を指定して Java Virtual Machine を起動します。
4. 携帯電話のディスプレイに Java アプリケーションの画面が表示されます。

1.4.3 アプリケーションの更新

アプリケーションの更新は、ユーザーがインストール済み Java アプリケーションのリストから、メニューなどによりアプリケーション更新を指示することにより行われます（詳細な操作方法はメーカーにより異なります）。携帯電話が自動的にアプリケーションを更新することはありません。ユーザーにアプリケーションの更新を促すには、Web サイトやメールなどを使用して告知を行うようにしてください。

アプリケーションをインストールした際に、JAM は ADF の URL を保存します。ユーザーがアプリケーションの更新を指示すると、JAM は再度この URL にアクセスし、新たな ADF を取得します。そしてその内容に従ってアプリケーションを更新する必要があるかどうかを判定し、必要と判断された場合のみ実際の更新処理を行います。

バグの訂正や新しい機能の追加を行うなどの理由でアプリケーションを更新する場合には、コンテンツプロバイダは次のことを行う必要があります。

- ADF の LastModified キーに前のバージョンより後の日付を設定します。その他のキーは必要に応じて適切な値に設定します。ただし ScratchPad サイズ（SPsize キー）の値は、前のバージョンより小さい値は指定しないようにしてください。更新しようとしているアプリケーションの ScratchPad サイズが以前のものと同じか大きい場合に限り、以前の ScratchPad の内容が更新後のアプリケーションに引き継がれます。
- コンテンツプロバイダの Web サイトに更新済み JAR ファイルと ADF ファイルを登録します。新しい ADF の名前は以前のものと同じにします。

アプリケーションの更新を指示されても、ADF の LastModified キーが更新されていなければ、JAM はアプリケーションの更新は行いません。

1.4.4 アプリケーションの削除

ユーザーの要求によりアプリケーションを削除する際の手順は、メーカーによって異なります。JAM はアプリケーションの削除をユーザーに指示されると、携帯電話上の JAR ファイルやアプリケーションに割り当てられた ScratchPad を削除します。

第 2 章

i モード Java アプリケーション環境

この章では、Java 対応 i モードに搭載される Java アプリケーション実行環境（i モード Java アプリケーション環境）の機能構成、API 構成の概要を解説します。

2.1 i モード Java アプリケーション環境

i モード Java アプリケーション環境は、携帯電話のネイティブアプリケーション、Java Application Manager、i モード Java 拡張 API、K Virtual Machine、および CLDC クラスライブラリから構成されています。i モード Java アプリケーション環境の構成を図 2 に示します。

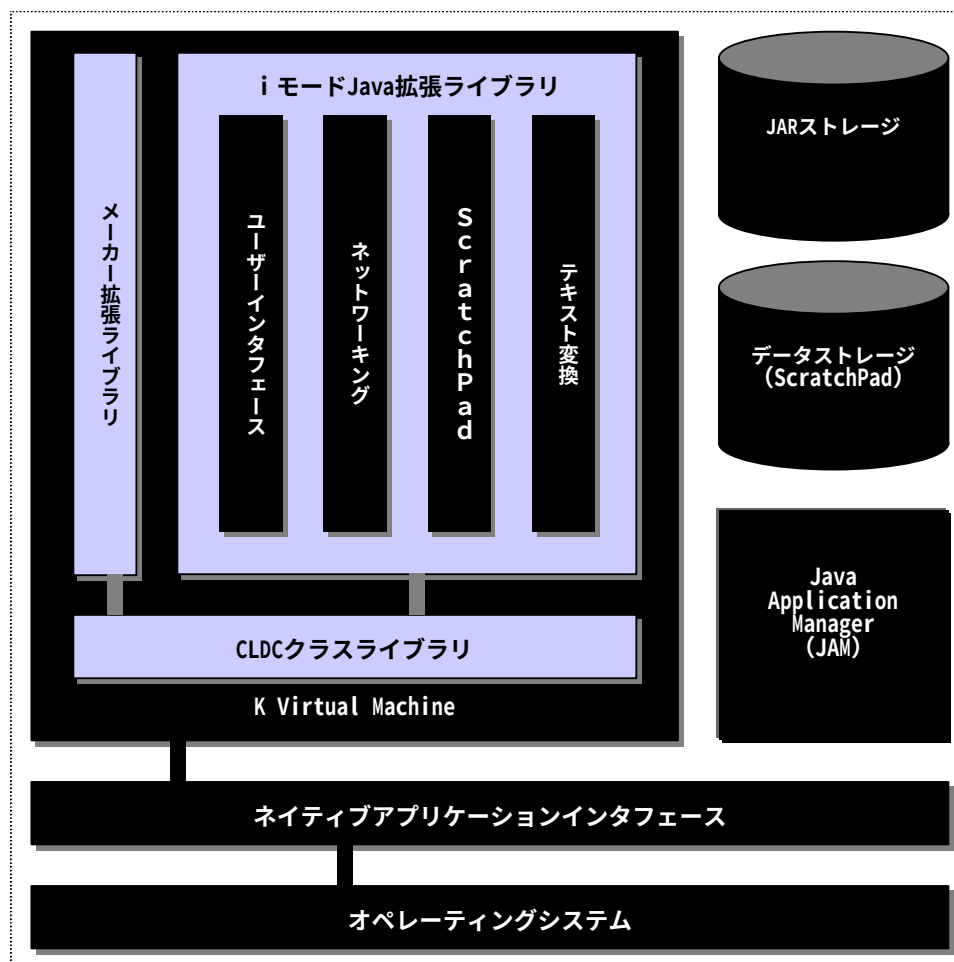


図 2: i モード Java アプリケーション環境

2.1.1 ネイティブアプリケーションとオペレーティングシステム

通常の電話や従来のブラウザベースの i モードサービスは、携帯電話上のオペレーティングシステムおよびネイティブソフトウェア(ブラウザ、ネットワーキングコンポーネント、Phone Dialer など)を使って行われます。これらの中には、ネットワーキングコンポーネントなど Java アプリケーションから使用される機能も含まれています。

2.1.2 Java Application Manager

Java Application Manager (JAM)により、JAR ファイルとして格納されている個々の Java アプリケーションの管理や K Virtual Machine との通信が行われます。

Java Application Manager の主な機能については 1.2 項を参照してください。

2.1.3 K Virtual Machine

K Virtual Machine は Java Virtual Machine を小型組み込みデバイス向けに再設計したもので、メモリや CPU 能力、電池消費に制約があるデバイスで使用するよう設計されています。

2.1.4 CLDC ライブラリ

CLDC ライブラリには、Java 2 Standard Edition (J2SE)から継承されたサブセットクラスが含まれています。CLDC ライブラリは上位エディションの Java 実行環境に含まれる機能の多くを共有していますが、小型デバイスの要件を満たすために小型デバイスの制約に合わせて最適化されています。

CLDC ライブラリの提供する API については、2.2.2 項を参照してください。

2.1.5 i モード Java の拡張

i モード Java の拡張は、Java 対応 i モード向けに設計されたものです。CLDC が小型組み込みデバイスで共通的に使用可能なアプリケーション基盤を提供しているのに対して、i モード Java の拡張では CLDC を土台として、Java 対応 i モードサービスに特化したアプリケーション基盤を提供しています。i モード Java の拡張は、次の各要素から構成されています。

- ユーザーインタフェース API
- ネットワーキング API
- SJIS テキスト処理
- ScratchPad ローカル記憶装置

i モード Java の拡張の提供する API については、2.2.3 項を参照してください。また各機能の具体的な利用方法は 4 章以降で詳細に解説されています。

2.1.6 メーカーによる拡張

メーカーは、ユーザーインタフェース、Look&Feel、ハードウェア特性などに関する特別な機能(カラーまたはワイドディスプレイ、特殊なサウンド機能など)をサポートするために、そのメーカー独自の API を個々に提供する場合があります。メーカー拡張 API を使用する場合は、アプリケーションが正しく動作するかどうかをその対象の携帯電話でテストする必要があります。

メーカー拡張 API を使用するのは、必要な場合だけにすべきです。メーカー拡張 API を使用した場合、そのアプリケーションは他のメーカーの機種で動作させることはできません。メーカー拡張 API に関する個々の情報については別途公開されます。

2.2 i モード Java アプリケーション環境における API の概要

次の各項では、i モード Java アプリケーション環境でサポートされる API について説明します。この節は、次の各項に分割されています。

- 2.2.1 ドコモシステムアーキテクチャ
- 2.2.2 CLDC
- 2.2.3 i モード Java 拡張
- 2.2.4 メーカーによる拡張

i モード Java アプリケーション環境で提供されるライブラリは、次の4つのカテゴリに分類されます。

- CLDC に含まれる、標準 J2SE ライブラリのサブセットクラス
- CLDC 独自のクラス（機能的には J2SE にマッピング可能）
- i モード Java 拡張固有のクラス
- メーカー拡張固有のクラス

最初の2つのカテゴリは、Java Community Process (JCP) を経て策定されました。また後の2つはそれぞれ NTT ドコモとメーカーにより規定されています。メーカー拡張固有のクラスに関する詳細は別途公開されます。

2.2.1 ドコモシステムアーキテクチャ

Java 対応 i モードのネットワークアーキテクチャは、従来の i モードサービスにおけるネットワークアーキテクチャとほぼ同様です。携帯電話とコンテンツプロバイダサイトの間のアプリケーションレベルの通信は、HTTP を使用して行われます。以下にネットワークアーキテクチャの概念図を示します。

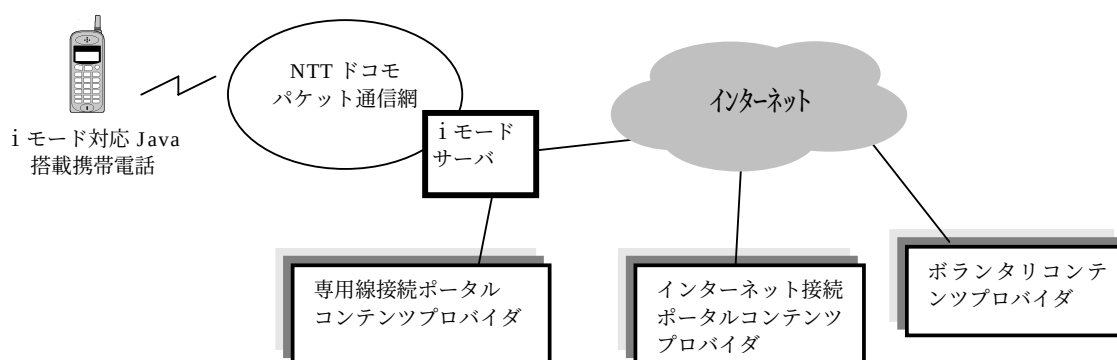


図 3: ネットワークアーキテクチャ概念図

Java 対応 i モードのネットワークオペレーションは以下の2種類に分類されます。

- アプリケーションのダウンロードとインストール
- 実行中のアプリケーションからのサーバアクセス

Java 対応 i モードのネットワークオペレーションは、従来の i モードサービスと同様に i モードサーバを介して行われます。携帯電話上でもアプリケーションレベルの通信プロトコルでは HTTP を使用しており、i モードサーバではプロトコル変換に類する処理は行いません。ダウンロードされるアプリケーションも、コンテンツプロバイダ側 Web サイトに搭載されているファイルがそのまま携帯電話まで送信されます。

また Java 対応 i モードでは、携帯電話側にて SSL (Secure Socket Layer) によるサーバ認証をサポートしています。Java 対応 i モードの通信機能と SSL サポートを利用することで、セキュア HTTP (HTTPS) 通信を行うこともできます。この場合のセキュア区間は、携帯電話からコンテンツプロバイダ側 Web サーバまでの間となります。

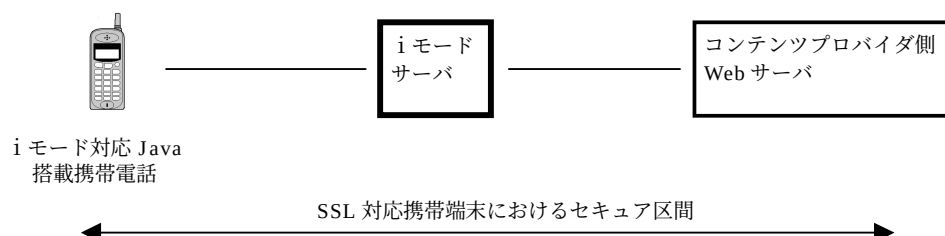


図 4: i モードサービスの SSL サポート

なお、証明書を携帯電話（クライアント）側に持たせるクライアント認証には対応していません。

2.2.2 CLDC

CLDC に含まれているクラスライブラリの大半は、アプリケーションの上位互換性と移植性を確保するため、上位エディションの Java 実行環境(J2SE)のサブセットになっています。上位互換性は非常に望ましい目標ですが、J2SE ライブラリには、セキュリティ、入出力、ユーザーインタフェース、ネットワーク、記憶装置といった重要な分野でサブセットを作成することを非常に困難にする内部的な強い依存関係があります。これらの依存関係は、Java クラスライブラリの発展とともに必要になった設計の革新と再利用の必然的な結果です。残念ながら、この依存関係があるために、ライブラリの一部だけを使用し、他の部分を使用しないということは非常に困難です。そのため、ライブラリの一部、特にネットワークと入出力の部分は再設計されています。

CLDC でサポートされる J2SE クラス

CLDC には、Java 2 Standard Edition (J2SE)から継承されたクラスが多数あります。これらのクラスのセマンティクスとメソッドには変更はありません。これらのクラスには、対応する J2SE クラスにない public や protected のメソッドやフィールドは追加されていません。これは、J2ME コンフィグレーションの規則により、J2SE と同じパッケージ名とクラス名をもつクラスは、対応する J2SE クラスの厳密なサブセットでなければならず、独自のメソッドやフィールドを追加することは許されていないためです。

システムクラス

標準の Java クラスライブラリには、Java Virtual Machine と密接に結びついたクラスが含まれています。またいくつかの標準的な Java ツールも、特定のクラスがシステムに存在するものと仮定しています。たとえば標準的な Java コンパイラは、クラス String と StringBuffer の特定のメソッドを使用するコードを生成します。

```
java.lang.Object
java.lang.Class
java.lang.Runtime
java.lang.System
java.lang.Thread
java.lang.Runnable (インタフェース)
java.lang.String
java.lang.StringBuffer
java.lang.Throwable
```

データ型クラス

CLDC では、java.lang.*パッケージの次の基本データ型クラスがサポートされます。これらのクラスはそれぞれ、Java 2 Standard Edition の対応するクラスの厳密なサブセットです。CLDC では浮動小数点数をサポートしていないため、浮動小数点型データを表す基本データ型クラスはサポートされません。

```
java.lang.Boolean
java.lang.Byte
java.lang.Short
java.lang.Integer
java.lang.Long
java.lang.Character
```

Collection クラス

CLDC では、java.util.*パッケージの次の Collection クラスがサポートされます。

```
java.util.Vector
java.util.Stack
java.util.Hashtable
java.util.Enumeration (インタフェース)
```

入出力クラス

CLDC では、java.io.*パッケージの次のクラスがサポートされます。国際化のサポートには、Reader、Writer、InputStreamReader、OutputStreamWriter の各クラスが必要です。

```
java.io.InputStream
java.io.OutputStream
java.io.PrintStream
java.io.ByteArrayInputStream
java.io.ByteArrayOutputStream
java.io.DataInput (インタフェース)
java.io.DataOutput (インタフェース)
java.io.DataInputStream
java.io.DataOutputStream
java.io.Reader
java.io.Writer
java.io.InputStreamReader
java.io.OutputStreamWriter
```

Micro Edition クラス

CLDC では、javax.microedition.io.*パッケージの次のクラスがサポートされます。

```
javax.microedition.io.Connector
javax.microedition.io.ConnectionNotFoundException
javax.microedition.io.Connection (インタフェース)
javax.microedition.io.ContentConnection (インタフェース)
javax.microedition.io.Datagram (インタフェース)
javax.microedition.io.DatagramConnection (インタフェース)
javax.microedition.io.InputConnection (インタフェース)
javax.microedition.io.OutputConnection (インタフェース)
javax.microedition.io.StreamConnection (インタフェース)
javax.microedition.io.StreamConnectionNotifier (インタフェース)
```

カレンダーと時間のクラス

CLDC には、J2SE の標準クラスである java.util.Calendar、java.util.Date、java.util.TimeZone の小さなサブセットが含まれています。CLDC では、時間帯として GMT と UTC がサポートされます。他の時間帯については実際の実装で提供できます。i モード Java アプリケーション環境では追加の時間帯として JST をサポートしています。

```
java.util.Calendar
java.util.Date
java.util.TimeZone
```

追加のユーティリティクラス

CLDC には、追加のユーティリティクラスが2つあります。java.util.Random クラスは、ゲームなどのアプリケーションを実装するときに使用する簡単な擬似乱数ジェネレータです。java.lang.Math クラスには、他の Java ライブラリクラスによって頻繁に使用される(データ型 int と long 用の)min()、max()、abs()メソッドがあります。

```
java.lang.Math
java.util.Random
```

エラーと例外クラス

CLDC のクラスライブラリは一般には J2SE のクラスライブラリと高度な互換性をもつことが定められているため、CLDC に含まれているクラスは、対応する J2SE クラスと全く同じ例外をスローします。このため CLDC のクラスライブラリには、数多くの例外クラスが組み込まれています。

エラークラス

```
java.lang.Error
java.lang.VirtualMachineError
java.lang.OutOfMemoryError
```

例外クラス

```
java.lang.Exception
java.lang.ClassNotFoundException
java.lang.IllegalAccessException
java.lang.InstantiationException
java.lang.InterruptedException
java.lang.RuntimeException
java.lang.ArithmeticException
java.lang.ArrayStoreException
java.lang.ClassCastException
```

```
java.lang.IllegalArgumentException  
java.lang.IllegalThreadStateException  
java.lang.NumberFormatException  
java.lang.IllegalMonitorStateException  
java.lang.IndexOutOfBoundsException  
java.lang.ArrayIndexOutOfBoundsException  
java.lang.StringIndexOutOfBoundsException  
java.lang.NegativeArraySizeException  
java.lang.NullPointerException  
java.lang.SecurityException  
java.util.EmptyStackException  
java.util.NoSuchElementException  
java.io.EOFException  
java.io.IOException  
java.io.InterruptedIOException  
java.io.UnsupportedEncodingException  
java.io.UTFDataFormatException
```

CLDC プラットフォームではサポートされない JVM 機能

仮想マシンのサイズを小さくするために、CLDC の仮想マシン（KVM）では次の機能が標準の Java Virtual Machine から削除されています。

浮動小数点サポート

KVM では浮動小数点演算はサポートされません。

JNI

KVM では Java Native Interface (JNI)はサポートされません。

スレッドグループ

KVM にはマルチスレッド機能が実装されていますが、スレッドグループはサポートされません。したがって、スレッドの開始や停止などのスレッド操作は、個々のスレッドオブジェクトにしか適用できません。アプリケーションプログラマがスレッドグループに対してスレッド操作を行いたい場合は、アプリケーションレベルでコレクションオブジェクトを明示的に使用してスレッドオブジェクトをグループ管理する必要があります。

ファイナライズ

CLDC では `Object` クラスに `finalize()` メソッドが含まれておらず、KVM ではオブジェクトのファイナライズはサポートされません。i モード Java アプリケーションでは、`finalize()` メソッドを定義することはできません。

Generic Connection フレームワーク (javax.microedition.io パッケージ)

J2ME の実装サイズを小さくするために、J2SE のネットワーククラスと入出力クラスを汎用化する必要がありました。この新しいフレームワークの目標は、これが J2SE におけるネットワーク機能や入出力機能の厳密なサブセットとして機能するようにすることです。このサブセットは、一般的な低レベルハードウェアや任意の J2SE 実装に容易にマップ可能であると同時に、新しいデバイスやプロトコルのサポートでは、よりよい拡張性、柔軟性、一貫性を備えています。さまざまな種類の通信先に対し個々の概念に基づくアクセス手段を提供するのではなく、次に示すような統合された概念に基づいたアクセス手段がアプリケーションプログラミングレベルで使用されます。

すべての接続は、**Connector** クラスの **open()** という単一の static メソッドを使って行われます。接続が正常に行われると、**open()** メソッドは Generic Connection インタフェース群 (**Connection** インタフェースをルートとするサブインタフェース群) の中の 1 つを実装したオブジェクトを返します。このインタフェース群は、図 5 に示されるように多数のインタフェースから構成されています。

open() メソッドは、URL パラメータを次の一般的な形式で受け取ります。

```
Connector.open("<protocol>:[target][parameters]");
```

この設計の目的は、アクセス対象毎の初期化の違いをできる限り、アクセス対象のタイプを特徴付ける文字列 (この場合 URL 文字列) に集約することです。**Connector.open()** へのパラメータにはこの URL 文字列を指定します。このアプローチの主な利点は、使用される接続の種類が何であれ、アプリケーションコードの大部分は変更の必要がないことです。これは従来の実装とは異なります。従来の実装では、アクセス対象を変更すると (例えばファイル入出力をソケット入出力に変更すると)、アプリケーションに適用する概念が大幅に変更になることがよくあります。

プロトコルの特定の初期化に伴う違いが **Connector.open()** のパラメータで識別でき、その違いをプラットフォーム側で吸収することができれば、プロトコル固有の処理をアプリケーション側で行う必要はありません。そのため、このような機構を採用することによりアプリケーションの移植性を向上させることができます。パラメータ文字列を実行時環境から取得するアプリケーションは、プラットフォームの点でもプロトコルの点でも高度な移植性が備わります。したがってこのようなアプリケーションでは、接続の形式が変わっても、コードの書き直しは必要ないか、最小限に抑えることができます。

プロトコルと J2ME プログラムとのバインドは実行時に行われます。実装レベルでは、**Connector.open()** の URL パラメータのプロトコル部により、サポートされているプロトコル実装のうちどのプロトコル実装を使用するかが決定されます。プログラムが、実行時に異なるプロトコルに動的に適応できるのは、この実行時のバインドメカニズムによるものです。概念的に言えば、これは、PC やワークステーションのアプリケーションプログラムとデバイスドライバの関係に似ています。

タイプが異なるデバイスへの接続では、それぞれのデバイスに特有の操作を行えるようにしなければならない場合があります。例えばファイルはその名前を変更することができますが、TCP/IP にはこれに似た操作はありません。Generic Connection フレームワークにはこのような個別の機能が反映されており、論理的に同じ操作は同じ API を共有します。

この新しいフレームワークは、同じセマンティクスをもつプロトコルクラスをグループ化した **Connection** インタフェースから継承されるサブインタフェース階層として実装されています。Generic Connection フレームワークで扱う基本的なインタフェースタイプには、全てのインタフェースのスーパーインタフェースである **Connection** インタフェースを除き、次の 6 つがあります。

- 基本的なシリアル入力 : **InputConnection**
- 基本的なシリアル出力 : **OutputConnection**
- データグラム指向の通信 : **DatagramConnection**
- ストリーム指向の通信 (**InputConnection** および **OutputConnection** のサブインタフェース)
: **StreamConnection**
- クライアント/サーバー接続をサーバーに知らせる通知メカニズム
: **StreamConnectionNotifier**
- コンテンツ送受信用通信 (**StreamConnection** のサブインタフェース)
: **ContentConnection**

Connection インタフェース群は 1 つの階層を形成し、階層がルート **Connection** インタフェースから継承されるに従って階層の機能が向上します。この編成により、アプリケーション開発者は、作成するアプリケーションに最も適した、プロトコル間の移植性レベルを選択できます。図 5 に、Generic Connection クラス階層の主なクラス間の関係を示します。

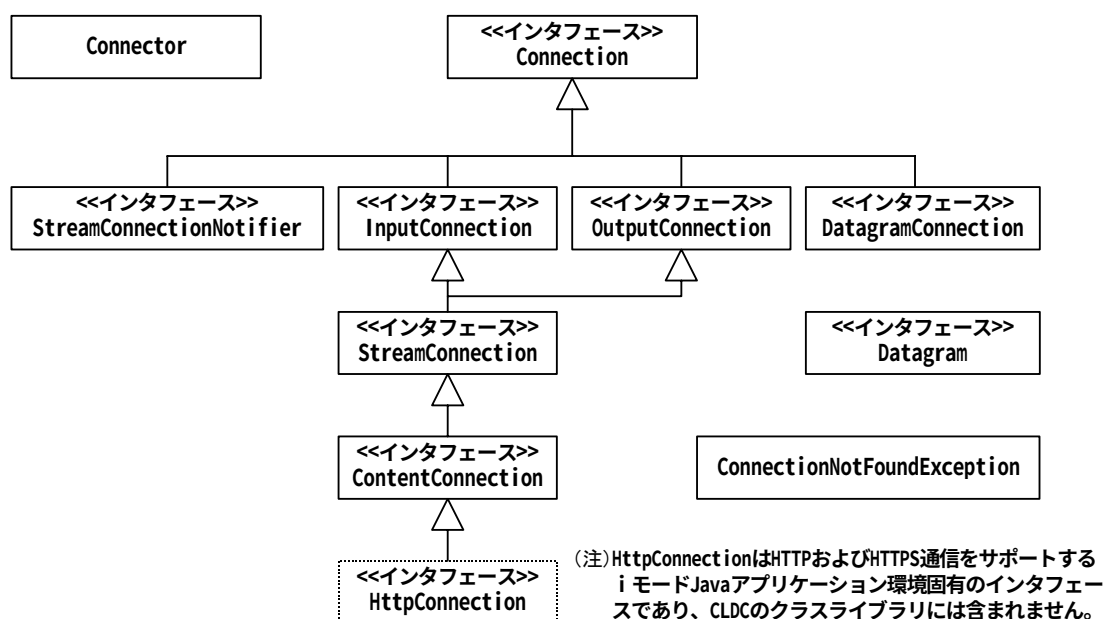


図 5: Generic Connection クラス階層

i モード Java アプリケーション環境では、アプリケーションが外部にアクセスする際に使用する以下のインタフェースをサポートしています。

- ・ HTTP(S)通信 : HttpConnection
(i モード Java 拡張に含まれる、ContentConnection のサブインタフェース)
- ・ ScratchPad : StreamConnection
- ・ リソース : InputConnection

2.2.3 i モード Java 拡張

次の各項では、com.nttdocomo パッケージ中の io、net、util、lang、ui の各サブパッケージに含まれるクラスについて説明します。

com.nttdocomo.io パッケージ

CLDC の Generic Connection フレームワークは、i モード Java アプリケーション環境の入出力パッケージの基礎をなすものです。com.nttdocomo.io パッケージでは、URL の形式で指定されたネットワーク資源に接続して入出力を行うためのインタフェースおよび例外クラスが提供されます。このインタフェースは、HTTP と HTTPS プロトコルによるネットワーク接続をサポートします。

このパッケージに含まれるインタフェース：

名前	説明
HttpConnection	HTTP 接続へのアクセスを定義するインタフェースです。 <code>javax.microedition.io.Connector.open()</code> メソッドに HTTP または HTTPS の URL を指定することで、本インタフェースの実装インスタンスが得られます。 <code>HttpConnection</code> は Generic Connection フレームワークにおける <code>javax.microedition.io.ContentConnection</code> インタフェースのサブインタフェースです。 Generic Connection フレームワークの詳細は、2.2.2 項を参照してください。

このパッケージに含まれるクラス：

名前	説明
ConnectionException	このクラスは、com.nttdocomo.io パッケージにおける入出力処理で異常が検出されたときにスローされます。例外の詳細は、 <code>getStatus()</code> メソッドで取得します。

`HttpConnection` を使用したクライアント/サーバープログラミングについては第 5 章で詳細に解説しています。

com.nttdocomo.net パッケージ

このパッケージには、次のクラスが含まれています。これらのクラスは、それぞれ Java 2 Standard Edition の java.net パッケージに含まれる同名のクラスと同じ機能を持ちます。

名前	説明
URLEncoder	このクラスには、String を x-www-form-urlencoded 形式と呼ばれる MIME 形式に変換するユーティリティメソッドが含まれています。エンコードの際、String は SJIS コードに変換されます。
URLDecoder	このクラスには、MIME 形式 x-www-form-urlencoded を String に変換するユーティリティメソッドが含まれています。入力データは SJIS コードをベースとしているものと見なされます。.

com.nttdocomo.util パッケージ

このパッケージには、次のクラスおよびインタフェースが含まれています。

名前	説明
EventListener	i モード Java 拡張 API で扱うイベントのリスナーインタフェースは、全てこのインタフェースを継承しています。
TimeKeeper	i モード Java 拡張 API では、一定時間が経過したことをイベントとしてアプリケーションに通知するタイマー機能をサポートしています。タイマークラス (com.nttdocomo.util.Timer および com.nttdocomo.ui.ShortTimer) はこのインタフェースを実装しています。
TimerListener	このインタフェースは com.nttdocomo.util.Timer の通知するタイマーイベントを取得するリスナーを実装するためのインタフェースです。なお com.nttdocomo.ui.ShortTimer のイベント処理は Canvas の processEvent() メソッドを使用して行うため、本インタフェースは使用しません。
Timer	タイマークラスです。このクラスは、ワンショットタイマーとインタバルタイマーをサポートします。TimerListener オブジェクトを登録することでタイマーイベントを受け取ることができます。タイマーイベントが発生すると、リスナーオブジェクトの timerExpired() メソッドが呼び出されます。タイマーの有効期間はアプリケーション全体に渡り、画面が切り替わってもタイマー動作は継続されます。

com.nttdocomo.lang パッケージ

このパッケージには、次の例外クラスが含まれています。

名前	説明
UnsupportedOperationException	サポートされない操作やメソッドを実行すると、例外として <code>UnsupportedOperationException</code> がスローされます。

com.nttdocomo.ui パッケージ

このパッケージに含まれているインタフェースは以下の通りです。

名前	説明
ComponentListener	コンポーネントイベントを受け取るリスナーインタフェースです。コンポーネントイベントの処理を行うクラスはこのインタフェースを実装する必要があります。リスナーオブジェクトは Panel.setComponentListener() メソッドを使用して Panel に登録します。コンポーネントイベントが発生するとリスナーオブジェクトの componentAction() メソッドが呼び出され、イベントのタイプ、イベントが発生したコンポーネント、イベントパラメータが引数として渡されます。
FocusManager	フォーカスマネージャクラスは、 Panel 上に配置されたコンポーネント間のフォーカス移動を管理するオブジェクトです。このインタフェースはフォーカスマネージャクラスが実装するインタフェースです。フォーカスマネージャクラスは、メーカー拡張クラスとしてメーカーから提供されます。拡張が提供されていない場合は、デフォルトのフォーカスマネージャクラスが使用されます。
Interactable	ユーザーが操作できるコンポーネント (Button 、 ListBox 、 TextBox など) が実装するインタフェースです。このインタフェースでは、コンポーネントへのフォーカス設定や操作可否設定を行うためのメソッドが定義されています。
KeyListener	高レベル API において、個別のキー操作を待ち受けるためのリスナーインタフェースです。本インタフェースの実装オブジェクトは Panel に登録して使用します。このインタフェースには、キープレスイベントとキーリリースイベントに対応する keyPressed() と keyReleased() の2つのメソッドが定義されています。
LayoutManager	レイアウトマネージャクラスは、 Panel 上へのコンポーネントの配置を管理するオブジェクトです。このインタフェースはレイアウトマネージャクラスが実装するインタフェースです。レイアウトマネージャクラスは、メーカー拡張クラスとしてメーカーから提供されます。拡張が提供されていない場合は、デフォルトのレイアウトマネージャクラスが使用されます。
MediaListener	このインタフェースは、マルチメディアデータを再生するメディアプレゼンターからイベントを受け取るためのメディアリスナーが実装するインタフェースです。このインタフェースで定義されている mediaAction() メソッドでは、メディアプレゼンターの再生開始・終了・停止などのイベントを受け取ることができます。
MediaData	MediaData の実装オブジェクトは MediaManager.getData() メソッドを使って取

得します。実際に返されるオブジェクトは実装によって異なります。**MediaData** はメーカーが独自のメディアデータを新たに実装する際のために用意されたインタフェースです。

MediaImage

MediaImage は静止画像または動画像を表現します。**MediaImage** の実装オブジェクトは **MediaManager.getImage()** メソッドを使って取得します。**MediaImage** は **VisualPresenter** オブジェクトを使って再生します。使用可能なデータは GIF フォーマット (GIF87/87a/89a) です。

MediaPresenter

メディアデータの再生オブジェクトに実装されるインタフェースを定義します。プレゼンタークラスは、メディアデータを実際の内容に従って再生します。サポートされるメディアデータのタイプは端末に依存します。端末がサポートしないメディアデータが設定されると、**UIException** がスローされます。

MediaResource

メディアデータインタフェース(**MediaData**、**MediaImage**、**MediaSound**)が継承しなければならないインタフェースを定義します。**MediaManager** を使用して取得した **MediaResource** インスタンスを実際に使用する前には **use()** メソッドを、使用した後は **unuse()** メソッドをそれぞれ呼び出す必要があります。**MediaResource** インスタンスをそれ以上使用しない場合は、**dispose()** メソッドを呼び出してインスタンスを破棄すべきです。

MediaSound

MediaSound はサウンドを表現します。**MediaSound** の実装オブジェクトは **MediaManager.getSound()** メソッドを使って取得します。**MediaSound** は **AudioPresenter** オブジェクトを使って再生します。使用可能なデータは、PC 上のアプリケーション開発環境では MIDI フォーマット、実際の端末では i メロディ形式フォーマットです。

SoftKeyListener

高レベル API において、携帯電話に 2 つ用意されているソフトキー操作を待ち受けるためのリスナーインタフェースです。本インタフェースの実装オブジェクトは **Panel** に登録して使用します。このインタフェースには、ソフトキープレスイベントとソフトキーリリースイベントに対応する **softKeyPressed()** と **softKeyReleased()** の 2 つのメソッドが定義されています。

このパッケージに含まれているクラスは以下の通りです。

名前	説明
AudioPresenter	このクラスは、音声メディアデータ (MediaSound) を高レベル API と低レベル API で再生するときに使用します。

Button	Button コンポーネントはラベル付きボタンを表現します。ボタン押下により処理を行う場合はアプリケーションに <code>ComponentListener</code> を実装し、Panel の <code>setComponentListener()</code> メソッドを呼び出してこの新しいリスナーを登録し、このボタンが通知する <code>BUTTON_PRESSED</code> イベントを受け取ります。
Canvas	Canvas は画面上の空白の矩形域です。アプリケーションでは、この領域に描画したり、この領域でユーザーの入力イベントをトラップしたりすることができます。アプリケーションは、通常 Canvas を継承するサブクラスを定義して使用します。アプリケーションが Canvas に描画を行うには、Canvas クラスの <code>paint()</code> メソッドをオーバーライドします。
Component	Component はグラフィックスで表現されたユーザーインタフェース部品であり、画面に表示したり、これを使ってユーザーと対話したりすることができます。Component のサブクラスには Button、ListBox、TextBox などがあります。Component は高レベル API に属し、Panel と組み合わせて使用します。
Dialog	Dialog は、タイトル、テキストメッセージ、1つ以上のボタンを備えたトップレベルウィンドウです。Panel および Canvas のいずれの上にも Dialog を表示することができます。
Display	Display は画面領域を管理するクラスです。Display のすべてのメソッドは static であり、そのインスタンスが作成されることはありません。Display クラスは、画面 (Panel または Canvas) を切り替えたり、画面上に表示されている表示可能オブジェクトに入力イベントを渡したりします。このクラスは、低レベルイベントを受け取り、それを適切な表示可能オブジェクトに渡します。そして、このオブジェクトがイベントの処理方法を決めます。さらに、Display には、表示領域の幅と高さや、キーやイベント処理で使用するさまざまな定数といったデバイス情報が含まれています。
Font	低レベル API において、Canvas に文字を表示する際のフォントを表します。どのようなフォントがサポートされるかは、メーカーにより異なります。
Frame	Frame は、画面を表現する abstract クラスです。携帯電話のディスプレイに一度に表示できる内容は、1つの Frame インスタンスに対応します。Frame のサブクラスには Canvas、Dialog、Panel があります。画面の切り替えは、 <code>Display.setCurrent()</code> メソッドを切り替えた画面を引数として呼び出すことで行います。
Graphics	低レベル API における Canvas への描画とそれに関連する操作をカプセル化します。
IApplication	IApplication クラスは、アプリケーションの雛型クラスです。アプリケーションのメインクラス

(アプリケーションの起動時に最初に呼び出されるクラス)は `IApplication` のサブクラスでなければなりません。

<code>Image</code>	<code>Image</code> クラスは静止画像を表現する abstract クラスです。静止画像 (GIF) から <code>MediaImage</code> インスタンスを作成し、 <code>getImage()</code> メソッドを呼び出すことで <code>Image</code> の実装インスタンスを取得することができます。
<code>ImageLabel</code>	<code>ImageLabel</code> コンポーネントは、 <code>Panel</code> に静止画像を表示するときに使用します。 <code>Image</code> インスタンスを <code>ImageLabel</code> に設定し、それを <code>Panel</code> 上に配置することができます。
<code>Label</code>	<code>Label</code> コンポーネントは、 <code>Panel</code> にテキストを表示するときに使用します。 <code>Label</code> は、1 行の読み取り専用テキストを表現します。テキストはアプリケーションからは変更することはできません、ユーザーが直接編集することはできません。
<code>ListBox</code>	<code>ListBox</code> コンポーネントは、複数のテキスト項目からなるスクロールリストを表示するときに使用します。インスタンス化時の指定によって、開発者は <code>ListBox</code> の外観や機能を選択することができます。
<code>MediaManager</code>	<code>MediaResource</code> を管理するためのクラスです。 <code>MediaManager</code> はメディアデータ (<code>MediaData</code> 、 <code>MediaImage</code> 、 <code>MediaSound</code>) のファクトリクラスです。
<code>Panel</code>	<code>Panel</code> は、高レベル API 使用時において、コンポーネントを画面に表示するためのコンテナクラスです。 <code>Panel</code> が表すスペースには、アプリケーションで任意のコンポーネントを付加することができます。 <code>Panel</code> は、同時に 1 つしか画面に表示できません。
<code>PhoneSystem</code>	このクラスは、プラットフォームのネイティブ資源にアクセスするときに使用します。
<code>ShortTimer</code>	低レベル API で使用することを想定した、比較的オーバーヘッドの低いタイマーです。このクラスは、ワンショットタイマーとインタバルタイマーをサポートします。 <code>ShortTimer</code> は、本クラスの static メソッド <code>getShortTimer()</code> を使用して取得します。 <code>getShortTimer()</code> メソッドの引数には <code>Canvas</code> を指定しますが、タイマーイベントはこの <code>Canvas</code> の <code>processEvent()</code> メソッドに通知されます。アプリケーションが別の画面に切り替わると、タイマーは自動的に停止します。
<code>TextBox</code>	<code>TextBox</code> コンポーネントは、テキストの入力や表示を行うときに使用します。 <code>TextBox</code> は、ユーザーが任意の文字列を入力するための唯一のコンポーネントです。
<code>Ticker</code>	<code>Ticker</code> コンポーネントは、左方向に自動スクロールするテキストを表示するために使用するコンポーネントです。このコンポーネントは、テキストを

UIException	ティックカーテープ形式（i モード HTML の MARQUEE タグ相当）で表示します。 UIException は、UI パッケージや UI パッケージの使用するネイティブ関数などが動作しているときに異常を検出した際にスローされる例外クラスです。
VisualPresenter	VisualPresenter コンポーネントは、画像メディアアダプタ（MediaImage）を高レベル API で再生するときに使用するプレゼンタークラスです。

IApplication

com.nttdocomo.ui.IApplication クラスは、全ての i モード Java アプリケーションの雛型となるクラスです。アプリケーションのメインクラス（ADF の AppClass キーで指定される、アプリケーション起動時に最初に呼び出されるクラス）はこのクラスを継承していなければなりません。これは、PC のブラウザ上で動作する Java アプレットが java.applet.Applet クラスを継承しなければならないことと似ています。

IApplication クラスには、主にアプリケーションの起動から終了までのライフサイクルに関わるメソッドが定義されています。

- アプリケーションの実行は、IApplication.start()メソッドから開始されます。アプリケーションはこのメソッドをオーバーライドして、アプリケーションの処理をクラスに実装します。この際、IApplication.getArgs()メソッドを呼び出してアプリケーションの起動パラメータ（ADF の AppParam キーで指定された値）を取得することができます。
- アプリケーションの実行を終了するには、IApplication.terminate()メソッドを呼び出します。
- アプリケーションの実行中断と再開をサポートする機種では、電話着信などにより中断されたアプリケーションの再開時に IApplication.resume()メソッドが呼び出されます。アプリケーションはこのメソッドをオーバーライドすることで、アプリケーション実行再開時に必要な処理を実装することができます。

注意事項：

- アプリケーションは、IApplication およびそのサブクラスを直接インスタンス化してはいけません。これらは i モード Java アプリケーション環境がインスタンス化します。
- アプリケーションが明示的にコンストラクタを定義する場合、必ずアクセス指定子 public を指定してください。また、i モード Java アプリケーション環境は引数のあるコンストラクタは使用しません。明示的にコンストラクタを定義する際は、必ず引数なしのコンストラクタを設けるようにしてください。
- CLDC には、java.lang.Runtime.exit()メソッドおよび java.lang.System.exit()メソッドが定義されています。しかし i モード Java アプリケーション環境では、アプリケーションの実行を終了するためにこれらのメソッドを使用することはできません（これらを呼び出した場合には、SecurityException がスローされます）。アプリケーションの実行を終了するためには、IApplication.terminate()メソッドを呼び出すようにしてください。

テキスト変換

Java 対応 i モード向けに開発されるアプリケーションでは、テキスト処理に日本語文字セットと ASCII 文字セットを使用します。java.io パッケージには、Unicode 文字ストリームと、非 Unicode テキストのバイトストリームとの間で変換を行うクラスが含まれています。byte ストリームを character ストリームに変換するには InputStreamReader クラスを、character ストリームを byte ストリームに変換するには

`OutputStreamWriter` クラスをそれぞれ使用します。テキスト変換に関する詳細は、第 6 章を参照してください。

ScratchPad

携帯電話のアプリケーション向け記憶装置モデルを ScratchPad と呼びます。ScratchPad に接続するには、`Connector` クラスの `open()` メソッドを使用します。このメソッドに ScratchPad の URL を渡すと ScratchPad への接続がオープンされ、CLDC で定義されている `javax.microedition.io.StreamConnection` インタフェースの実装インスタンスが返されます。ScratchPad 領域の読み取りや書き込みには、`StreamConnection` インタフェースを使用します。ScratchPad の利用に関する詳細は、第 7 章を参照してください。

2.2.4 メーカーによる拡張

メーカーは自社製品の商品性を高めるために、その機種機能をフル活用するための独自の拡張 API を提供する場合があります。このような API により、バイブレータモード、音声、表示輝度といった携帯電話の資源にアクセスできるようになる可能性があります。メーカー拡張 API の詳細は別途公開されます。

なお、メーカー拡張 API を使用した場合、そのアプリケーションは他のメーカーの機種では動作させることができなくなるため注意してください。

第 3 章

設計上の考慮事項

この章では、利用可能なシステム資源を効率的に使用し、無線通信環境に適合した使いやすいアプリケーションを設計するためのガイドラインを提供します。

3.1 Java 対応 i モードの特性

携帯電話の表示画面やアプリケーション実行メモリ、アプリケーション記憶メモリは、携帯電話の物理的なサイズをできるだけ小さくし、バッテリー寿命をできるだけ長くするため、パーソナルコンピュータに比べ非常に小さくなっています。さらに、プロセッサもパーソナルコンピュータより遅い速度で動作します。液晶(LCD)画面の電力消費量は非常に少ないのですが、リフレッシュ速度は他のタイプのディスプレイに劣ります。テキスト入力には携帯電話のキーパッドを使って行い、しかも各キーに複数の文字が割り当てられているため、テキストの入力はユーザーにとって煩雑な操作です。したがって、テキスト入力は最小限に留めるべきです。

Java 対応 i モードでは、このようなプラットフォーム固有の特性を考慮してアプリケーションの設計を行う必要があります。次の各節では、これらの問題を設計の観点から検討し、使い易いアプリケーションを設計するためのガイドラインを提供します。

3.2 メモリの問題

図 6 に、3 つのアプリケーションに対して JAM が記憶領域（アプリケーションファイルと ScratchPad の保存領域）を割り当てる例を示します。

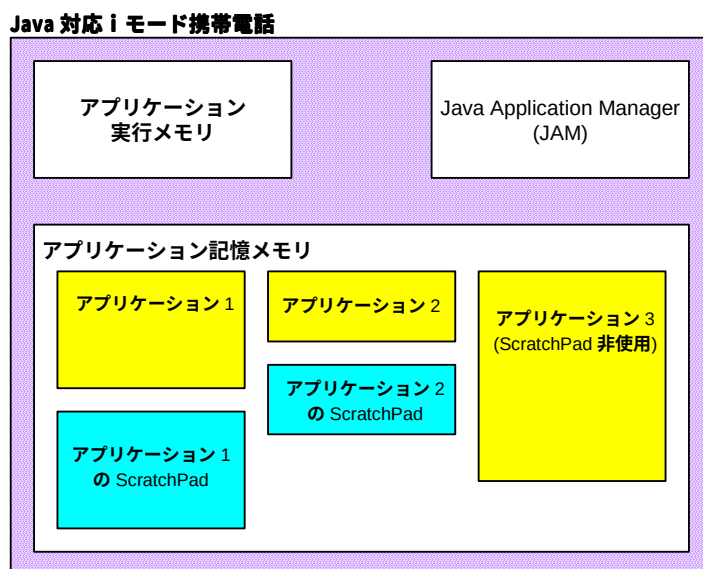


図 6: Java 対応 i モードのメモリ

アプリケーション開発者にとって、メモリには次の 2 つの制約があります。

- 限られたアプリケーション実行メモリ。アプリケーションはこの領域で実行されます。実行時に生成されるオブジェクトもここに格納されます。
- 限られたアプリケーション記憶メモリ。ここにはアプリケーション（JAR ファイル）と、個々のアプリケーションに対応する ScratchPad が格納されます。

アプリケーション実行時サイズに対する絶対的な制約は、アプリケーションが実行される携帯電話に搭載されているアプリケーション実行メモリの量によります。

またアプリケーション記憶サイズは、JAR ファイルが 10K バイト（以降、1K バイト=1024 バイト）まで、ScratchPad が 5K バイトまでに制限されます。このサイズの制限を越えるアプリケーションは携帯電話に正しくダウンロードできないのでご注意ください。

注意事項：

- ScratchPad サイズはメーカーにより 5K バイトより大きいサイズをサポートする場合があります。またメーカーの採用する領域管理方式によっては、新たにインストールしようとしているアプリケーションの ScratchPad サイズが、インストール済みアプリケーションが使用している ScratchPad サイズの合計サイズにより制限を受ける場合もあります。いずれの場合にも、アプリケーションが ADF の SPSIZE キーで宣言した容量の ScratchPad を確保できない場合、そのアプリケーションは携帯電話にダウンロードすることはできません。

3.2.1 メモリ使用量の最小化

ここでは、大きなカテゴリとして、アプリケーション実行メモリとアプリケーション記憶メモリに分けて説明します。

アプリケーション実行メモリ

通常、アプリケーション実行メモリを占める最大のコンポーネントは実行可能プログラム（i モード Java アプリケーション環境およびアプリケーションを構成するクラス群）そのものです。アプリケーションとそれに必要なすべてのオブジェクトが使用可能なアプリケーション実行メモリに入りきらない場合は、実行可能プログラムのサイズが生成されるオブジェクトセットのサイズ、またはその両方を削減する以外に方法はありません。

実行可能プログラムのサイズを削減する方法については、3.2.2 項を参照してください。

生成されるオブジェクトのサイズを削減する場合は、次に示す内容を検討します。

- ガベージコレクションを頻繁に呼び出す。特に搭載メモリ量の少ない i モード Java アプリケーション環境では、アプリケーション実行メモリ上に発生したフラグメンテーションが原因で、一度にまとまったサイズのメモリを確保できなくなるケースが発生します（OutOfMemoryError、UIException、ConnectionException などの例外が発生したり、アプリケーションの動作が不安定になるなどの問題として現れます）。この場合、できるだけ早め早めにガベージコレクションを呼び出すようにすることで問題が解決することがあります。
- 個々のオブジェクトサイズをできるだけ小さくする。
- メモリにはできるだけ少ないオブジェクトを保持する。または生成するオブジェクトの数を最小限に抑える。
- アプリケーションの分割を検討する（例えば複数のレベルをもつゲームの設計では、レベル毎にアプリケーションを分割するなど）。
- 可能な限り static を使用する。
- 過度のメソッド呼び出しのネストや再帰を避けてスタックの深さを最小限に抑える。
- メソッドに渡すパラメータの数を最小限に抑える。
- クラスの数を最小限に抑える。
- スレッドの多用を避ける。
- サイズがあらかじめ決められたクラスを作成する。HashTable、ByteArrayInputStream、ByteArrayOutputStream、Vector など、サイズが動的に大きくなるクラスを使用すると、メモリのフラグメンテーションを引き起こすことがあります。
- 必要がなくなったメディアデータは速やかに破棄（dispose()）する。

生成されるオブジェクトサイズを可能な限り小さくし、RAM を効率的に使用しているにもかかわらず、まだメモリが不足する場合は、次の内容を検討してください。

- 変数に保存しているデータを ScratchPad に移す。
- 処理やデータ操作の負荷をクライアント側からサーバー側に移す。
- サーバー側からデータをダウンロードする場合、一括してダウンロードするのではなく、必要に応じて部分的にダウンロードする。

アプリケーション記憶メモリ

アプリケーションには、アプリケーション記憶メモリの次の2つのブロックが使用されます。

- JAR ファイルの格納に割り当てられるブロック
- アプリケーションの ScratchPad に割り当てられるブロック

JAR ファイル

JAR ファイルには、できるだけコンパクトにした必須のファイルだけを入れる必要があります。メーカー拡張 API を使用するなどの理由で携帯電話の機種ごとに異なるモジュールをアプリケーションに入れる必要がある場合は、機種ごとに別のアプリケーション(別の JAR ファイル)を作成することを検討してください(機種別にダウンロードを制御する方法については第 10 章を参照)。このようにすれば、携帯電話にダウンロードされる JAR ファイルに、不要なエントリが含まれることはありません。

また JAR ファイルには画像やサウンドなどのリソース(第 7 章参照)を含めることができますが、これは JAR ファイルサイズに大きな影響を与えます。使用頻度の低いリソースは、アプリケーション実行時に通信機能を使用して入手することを検討してください。

なお、i モードサービスでは通信料金にパケット課金システムが採用されています。JAR ファイルを小さくすることは、アプリケーションダウンロード時のユーザーの通信コストを削減することにもつながります。

ScratchPad

必要以上に大きな ScratchPad は指定しないでください。さらに、どの位の ScratchPad 容量を割り当てるにしろ、できる限り効率的に使用してください。アプリケーションの ScratchPad サイズは、アプリケーションの ADF に指定します(詳細は 9.5.1 項を参照)。なお、アプリケーションが使用可能な ScratchPad サイズには、3.2 項で解説した制限があります。

3.2.2 アプリケーションサイズ

コンパイルした Java クラスファイルにはデバッグのための情報が含まれることがあり、この情報によりクラスファイルのサイズ(アプリケーションサイズ)が増大することがあります。定義するクラスの数をもっと抑えようとするとともに、コードを `-g:none` オプションでコンパイルすることでデバッグ情報を取り除き、アプリケーションサイズを小さくするようにしてください。

3.3 使い易さと GUI の設計

この節では、簡単に直感的に使用できるアプリケーションを作成するためのガイドラインを示します。携帯電話のようなプラットフォームでは、一般的にはたくさんのキー操作を必要とする複雑なアプリケーションは敬遠され、よりキー操作が少ないアプリケーションが求められることが想定されます。注意すべき点を次に列挙します。

- アプリケーションを通して一貫した Look&Feel を維持します。
- 物理的な画面領域が小さいため、1つの画面に表示する UI コンポーネントの数を最小限に抑えます。

- UI レイアウト管理はメーカーによって異なりますが、ほとんどの場合、デフォルトのレイアウト管理は簡単なフローレイアウトです。アプリケーションの UI レイアウトをシンプルにし、アプリケーションの外観が、メーカー間で大きく変わらないようにします。
- スクロールを最小限に抑えます。なおメーカーによっては、アプリケーションの画面スクロールをサポートしない場合もありますのでご注意ください。
- ソフトキーは、ユーザー操作を簡単にする便利な方法です。複雑なユーザー操作が発生するような画面では、ソフトキーをショートカットキーのように使用することを検討してください。たとえば多くの入力用コンポーネント部品が配置された画面において、ソフトキーはどのコンポーネントにフォーカスがある状態でも確定動作を行わせるためのキーとして使用することができます。
- 携帯電話でのテキスト入力は煩雑なため、入力する文字数を最小限に抑えます。可能な限り選択肢を表示し、その中からユーザーが選択する形の入力方式にします。
- よく行われる操作は、多数のメニュー階層を移動したり、多数のキー操作をしなくても使用できるように画面構成を設計します。

3.3.1 ターゲットユーザー向けの設計

電話はだれでも使いますが、コンピュータはだれもが使うわけではありません。したがって、UI コンポーネント間のフォーカス移動やスクロールという考え方は、全てのユーザーにとって直感的にわかるわけではありません。広範なユーザーに使用してもらうためには、アプリケーションは簡単に使用できなければなりません。アプリケーションは、それを使用すると見込まれているターゲットユーザー層を見極めて設計すべきです。

3.3.2 操作カテゴリーの理解

ユーザー操作は、それがどのくらい頻繁に行われるかや、ユーザーにとってどの程度重要なものであるかによって、いくつかのカテゴリーに分類する必要があります。一般には、頻繁に行われる操作はより簡単に使えるようにすべきです。たとえば、銀行残高の確認は口座間の振り替えよりも頻繁に行われます。したがって理想的には、残高の確認は振り替えよりも少ないキー操作でできるようにすべきです。

一方、個人識別 ID の入力など、いくつかのキー操作が必要な操作は、それが重要な機能である限り(この場合は、ユーザーの銀行口座などへの不正なアクセスを防ぐ)、ユーザーに受け入れられます。さらにアプリケーションを携帯電話にインストールして最初に使い始める際に、多少煩雑ではあってもその後の操作を簡単にする目的で、一定の情報入力をユーザーが行うようにするという点も検討の余地があります。

操作には一貫性がなければなりません。項目の選択方法は、どの画面でも一貫しているべきです。ソフトキーは、誤操作や使いづらさを避けるため、アプリケーションを通して一貫した考え方に基づいて使用できなければなりません。また最も重要かつ頻繁に使用される操作は、最も少ないキー操作でできるようにすべきです。

3.3.3 スクロール

理論的には、物理画面よりも大きな仮想画面を設計することができます。しかしそのような画面では、画面全体を見たり UI コンポーネント間をフォーカス移動するには、多くのスクロールが必要になります。スクロールは必ずしもレスポンスのよい画面操作ではないため、ユーザーはスクロールの多い画面を使いづらいと感じる可能性があります。スクロールは最小限に抑えることを目標にすべきです。なお、Panel は垂直方向のスク

ロールのみサポートします。またメーカーによってはスクロール機能をサポートしない場合がありますのでご注意ください。

3.3.4 ソフトキー

Java 対応 i モードの携帯電話には2つのソフトキーが設けられます。ソフトキーの外観や場所はメーカーによって異なります。前述の通り、ソフトキーを使用するとアプリケーションの使用感が大幅に改善されることがあります。アプリケーションの設計ではソフトキーの使用を考慮すべきですが、特定の外観や場所に依存することは避けるようにしてください。

3.3.5 データ入力

ユーザーがデータを入力する手段は、原則的に携帯電話のキーパッドだけです。したがって、テキストデータを入力するよりも数字データを入力する方がより簡単に行えます。これはテキストデータの場合、1文字を入力するために複数回のキー操作が必要になるためです。テキスト入力は最小限に抑えるべきです。

3.3.6 パスワード入力

パスワードの入力は次のいずれかの方法で行います。

- パスワードを表示しない場合は、`TextBox` のコンストラクタで表示モードパラメータに `TextBox.DISPLAY_PASSWORD` を渡します。すべての入力文字は「*」で表示されます。
- パスワードを表示する場合は、`TextBox` のコンストラクタで表示モードパラメータに `TextBox.DISPLAY_ANY` を渡します。この場合、入力された文字はすべて画面上に表示されます。

パスワードに数字以外の文字（テキスト文字）を許可する場合は、2つめの方法を使用します。携帯電話では PC と異なり、数字以外の文字をパスワード入力することはユーザーにとって非常に困難です。

3.3.7 スレッドの使用

スレッドは、ネットワーク入出力や UI での対話など、長時間を要するブロック操作を行うときには非常に便利です。しかし、どのようなマルチスレッドシステムでも、アクティブスレッド間のコンテキスト切り替えにはオーバーヘッドが必要です。携帯電話の処理能力は限られているため、ユーザー操作をストレスなく可能にするためにスレッドの数は最小限にすべきです。

また、アプリケーションがマルチスレッドプログラミングを行っていないとしても、システムがイベント処理などのためにスレッドを生成することがあります。どの機種でも動作するアプリケーションを作成するには、スレッドセーフなプログラム設計が必要です。

3.4 セキュリティ

クライアントとサーバー間で機密情報を送受信するアプリケーションを設計する場合には、HTTPS プロトコルを使用することを検討します。ただし、クライアント側でこのメカニズムをサポートするには、CPU 負荷が高くなることに注意してください。これは、各 SSL セッションの始めにクライアント側のアプリケーションがマスターキーを生成する必要があるのに加え、データが処理されてサーバーとのデータ送受信が行われるたびに、これらのキーに基づくデータの暗号化や暗号化解除が必要になるからです。

HTTPS プロトコルを利用した通信については、5.3 項を参照してください。

なお、アプリケーションがサーバとの通信を行う場合、その接続先はアプリケーションのダウンロード元とプロトコル、サーバアドレス、ポート番号が同じでなければなりません。したがって、HTTPS プロトコルを使用するアプリケーションは HTTPS プロトコルを使用してダウンロードされている必要があります。

3.5 無線ネットワークでの操作

アプリケーションは、通信接続の中断がごく普通に発生する環境で使用されることに留意してください(ユーザーは携帯電話のサービス区域を出たり入ったりします)。また i モードサービスではパケット課金システムが採用されているため、アプリケーションが通信を行う場合は、必要なデータだけを送受信するようにしてください。

アプリケーションでは、小さいデータパケットトランザクションをいくつも送信する(切断によるデータの喪失を最小限にする)か、1つの大きなデータパケットを送信する(通信オーバーヘッドを最小限にする)かについてバランスをとる必要があります。

接続の確立、テスト、終了に関する問題や、トランザクションやセッションの管理に関する問題の詳しい説明は、第5章を参照してください。さらに、次項「処理の中断と再開」を参照してください。

3.6 処理の中断と再開

JAM はアプリケーション実行中に音声通話の着信があると、動作しているアプリケーションを自動的に中断します。そしてユーザーが音声通話を終了すると、Resume (再開) メッセージをアプリケーションに送信して実行を再開します(アプリケーションの再開をサポートしないメーカーもあります。また通信実行中は、携帯電話の i モード着信設定により処理が中断されない場合があります)。

アプリケーションの実行が再開されると、アプリケーション (`com.nttdocomo.ui.IApplication` を継承したクラス) の `resume()` メソッドが呼び出されます。アプリケーション開発者は `resume()` メソッドをオーバーライドすることによりアプリケーションの再開時処理を実装することができます。

スタンドアロンの電卓アプリケーションなど、ある種のアプリケーションでは、アプリケーション開発者が中断/再開のために特別なステップを行う必要はありません。これは、アプリケーションが中断されてもアプリケーションには何の変化も起らないことと、アプリケーションの処理進行が完全にユーザー生成のイベント(キーの操作)によって駆動されるからです。しかし特定の種類のアプリケーションでは、プログラムの実行を再開する際に特別なステップを行う必要があります。一般に、次のような状況では、プログラムの実行を再開する際に何らかのアクションをとる必要があります。

- クライアントサーバー型のアプリケーションでは、JAM から Resume メッセージを受信したら通信結果をチェックし、必要なら通信のリトライを行います。これは、通信実行中にアプリケーションの動作が中断される可能性があるためです。
- アプリケーションが中断、再開する際には、そのアプリケーションが中断する前に動作していたタイマー (Timer および ShortTimer) は無効化された状態になります。アプリケーション再開時には、それまで使用していたタイマーを破棄し、インスタンスを作成し直した後で再スタートを行ってください。
- 機種によっては、Canvas を表示した状態でアプリケーションの中断が発生すると画面描画情報を解放する場合があります。このような機種では、アプリケーション再開時に Canvas の再描画 (repaint) メソッドの呼び出しを行うことにより中断前の画面を再表示させることができます。
- 多くのアクションゲームのように、アプリケーションが動的なもので、ユーザーとの即時対話が必要な場合は、Resume メッセージを受信したらアプリケーションを再度停止し、適切なプロンプトメッセージを表示します。それによって、ユーザーは都合のよいときにアプリケーションを再開することができます。

3.7 ハードウェアのアクセス

メーカーは、音量、LED オプション、バイブレータモードといったハードウェアの設定値にアクセスするためのメーカー拡張機能を提供する場合があります。アプリケーションでこれらの設定を変更した場合でも、アプリケーション終了時には自動的に元の設定に戻されます。なお、アプリケーションからハードウェアにアクセスした結果、携帯電話のバッテリーを大量に消費する場合がありますので注意してください。

また、メーカー拡張機能を使用した場合、アプリケーションはそのメーカーの機種でしか動作しないことにも注意してください。

3.8 エラー処理

CLDC API および i モード Java 拡張 API では、処理中に異常を検出した場合に、一般的な Java API と同じくエラーや例外をスローします。アプリケーションでは発生した例外をキャッチして、必要に応じて適切な処理（例外メッセージをユーザーに提示するなど）を行うようにしてください。

なお、実行時例外（`java.lang.RuntimeException` およびサブクラス）などがアプリケーションでキャッチされなかった場合、そのアプリケーションは異常終了します。

第 4 章

ユーザーインタフェースの設計

ユーザーインタフェース（本書では UI と記すことがあります）機能には、次のような基本的な要件があります。

- ボタンやラベルといった高レベルなユーザーインタフェース部品（コンポーネント）の作成
- イベント処理
- レイアウト処理
- 画像ファイルの表示、音声ファイルの再生機能

i モード Java アプリケーション環境のユーザーインタフェース API には、高レベル API と低レベル API の 2 種類があります。次の各節では、個々の基本的な機能分野について説明し、そのコード例を示します。

高レベル API では、さまざまなコンポーネントが提供されます。アプリケーション開発者は、これらを組み合わせてアプリケーションのユーザーインタフェースを作成します。高レベル API を使用したアプリケーションはハードウェア特性を考慮する必要性が低く、メーカーの異なる携帯電話でも容易に動作させることができます。高レベル API ではアプリケーション開発者の自由度は限定されますが、少ないコード行で高機能なアプリケーションを作成することができます。高レベル API のユーザーインタフェースを構成する部品をコンポーネントと呼びます。アプリケーション開発者は、これらのコンポーネントを組み合わせてアプリケーションを作成します。高レベル API を使用した画面は、おおむね i モード HTML と同程度の表現を行うことができます（文字への色指定など一部の機能を除きます）。

低レベル API では、携帯電話の画面サイズやキーパッドなどハードウェアの特性を考慮しながらアプリケーションを作成します。低レベル API を使用すれば、携帯電話の各機種の特徴を引き出したアプリケーションを作成することができます。この場合、アプリケーションプログラムの自由度は増しますが、グラフィックスの描画や画面サイズと論理座標値といった細かい点を考慮する必要があります。

1 枚の画面は、**Panel** または **Canvas** のインスタンスとして表現されます。**Panel** は高レベル API を使用する画面（コンポーネントベースで構築される画面）で、**Canvas** は低レベル API を使用する画面（描画ベースで構築される画面）で利用されます。同じ画面の中で高レベル API と低レベル API を混在して使用することはできません。ただし、複数の画面から構成される 1 つのアプリケーションが、高レベル API を使用する画面（**Panel**）と低レベル API を使用する画面（**Canvas**）を切り替えて表示することは可能です。

ボタンを押したり、タイマーが切れるといったイベントの処理は、高レベル API と低レベル API では異なります。低レベル API では、キーの押下や解放を含め、すべてのイベントをアプリケーションで受け取ることができます。一方高レベル API では、コンポーネントによっては、いくつかの低レベルのイベントを内部的に処理したり、より高レベルのイベントに変換してアプリケーションに通知したりします。低レベル API の場合は、ボタンの押下や解放によって生成されたイベントは **Canvas** オブジェクトに報告されます。このイベントを受け取るには、**Canvas** オブジェクトに対して、**processEvent()** と呼ばれるイベントハンドラメソッドをオーバーライドする必要があります。イベントが生成されるとこのメソッドが呼び出されるので、その中でイベントに対する処理を行います。高レベル API の場合は、ボタン押下などのアクションは **Panel** オブジェクトに通知されます。アプリケーションの中でこのようなアクションに対する処理を行う場合は、**Panel** オブジェクトにリスナーオブジェクトを登録する必要があります。**Panel** オブジェクトは、コンポーネントでアクションが発生すると、**Panel** オブジェクト自体に登録されているリスナーオブジェクトメソッドを呼び出し、それにイベントを送信します。その際には、アクションが発生したコンポーネント、発生したアクションのタイプ、アクション特有の引数がリスナーメソッドの引数に渡されます。イベントリスナーの詳細は、4.3 項を参照してください。

図 7 に、ユーザーインターフェースのためのクラス構成を示します。

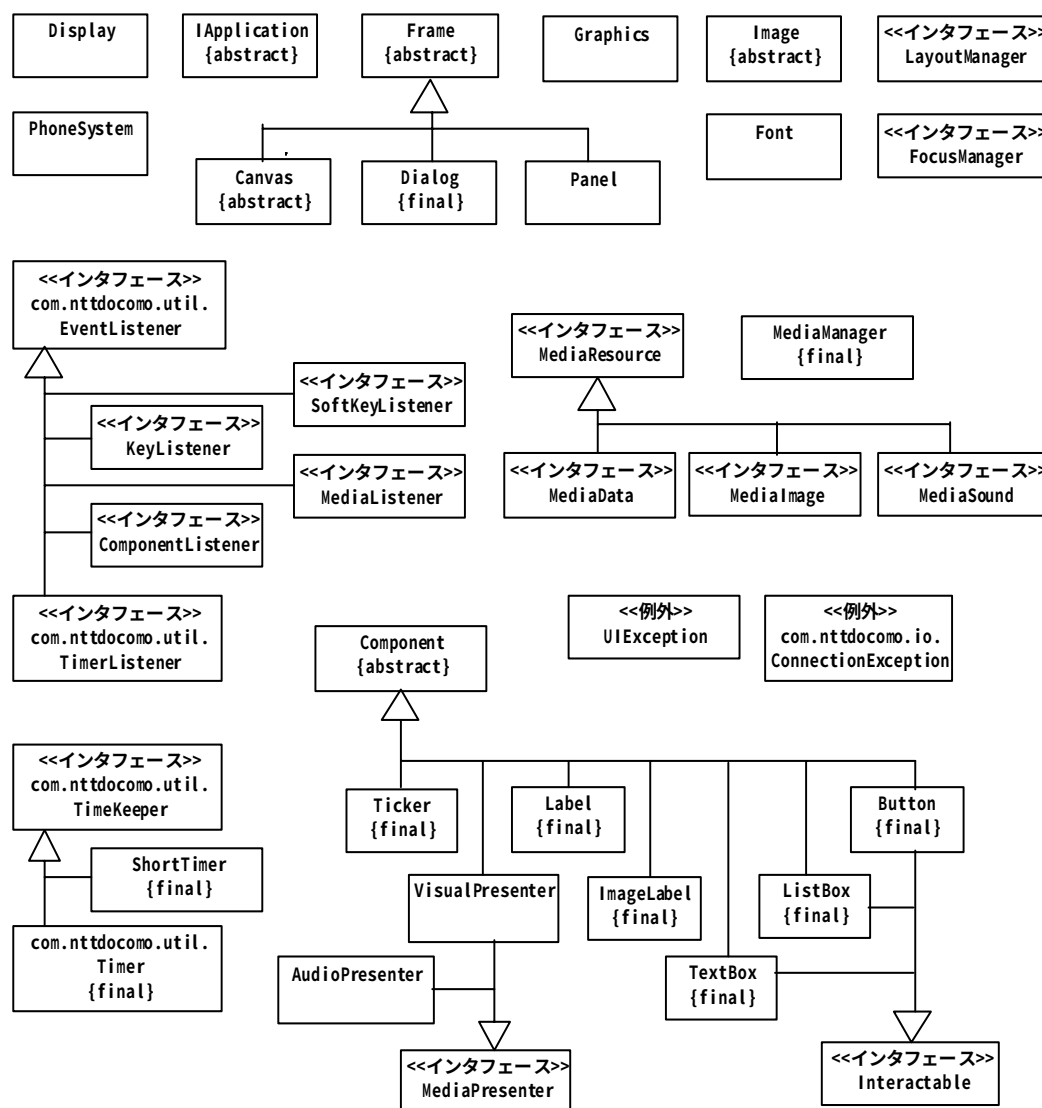


図 7: UI クラス階層

4.1 高レベル UI コンポーネントによるプログラミング

i モード Java アプリケーション環境では、7つの基本的な UI コンポーネントが提供されます。

- Label
- ImageLabel
- Button
- ListBox
- TextBox
- Ticker
- VisualPresenter

この他の基本 UI 部品として **Dialog** があります。上の各要素の動作は携帯電話の各メーカー間でほぼ同様ですが、Look&Feel は異なる可能性があります。これは、Windows、Macintosh、UNIX といったコンピュータプラットフォーム向けに開発された標準的な GUI アプリケーションの場合と同じです。これらのアプリケーションはどのプラットフォームでも同じように動作しますが、Look&Feel は異なっています。たとえば、ボタンの形状は、プラットフォームによって丸みがかっていたり四角だったり、枠が太かったり細かったりとそれぞれ異なります。これは携帯電話の場合も同様です。低レベルのグラフィックルーチンは、メーカーによって異なる外観を生成することがあります。デフォルトのレイアウト、スクロール、フォーカスの管理に関する方針は、どのメーカーでもほぼ同様です。ただしメーカーは個別に、自社の携帯電話向けにレイアウト、スクロール、フォーカスの管理を拡張している場合があります（メーカーによってはスクロールはサポートされません）。次の各項の要素やレイアウトマネージャの説明は一般的なものであり、携帯電話各機種の詳細な振舞いは各機種毎に異なる場合があります。

4.1.1 Panel の使用

Panel クラスは、UI 部品のシンプルなコンテナとして機能します。画面には、同時に 1 つのパネルしか表示できません。ただし、これは、アプリケーションで 1 つのパネルしか定義できないことを意味するわけではありません。多くのアプリケーションは複数のパネルから構成されています。ユーザーは、通常、これらのパネル間をナビゲーションしながらアプリケーションを操作します。

4.1.2 Dialog の使用

ダイアログは、テキストメッセージを表示して警告や情報をユーザーに示したり、ユーザーに決定を促がすときに使用します。**Dialog** は **Panel** または **Canvas** の上に重なるように表示される画面であり、**Frame** クラスのサブクラスです。ダイアログには、タイトルとテキストメッセージ、それに最低 1 つのボタンがあります。ダイアログは同時に 1 つしか表示できません。

ダイアログには次のような種類があります。

- 情報
- 警告

- エラー
- Yes/No
- Yes/No/Cancel

注意事項：

- 同時に2つ以上のダイアログを表示しようとすると、例外がスローされます。
- ダイアログはモーダルです。
- バックグラウンドのスレッドがダイアログのモーダル操作に干渉しないようにアプリケーション側では注意が必要です。例えばダイアログの表示中にバックグラウンドスレッドが背後の `Panel` や `Canvas` を切り替えようとした際の動作は、機種により異なりますので注意してください。

4.1.3 コンポーネントの使用

UI コンポーネント (`com.nttdocomo.ui.Component` のサブクラス) には、`Label`、`ImageLabel`、`Button`、`TextBox`、`ListBox`、`Ticker`、`VisualPresenter` があります。これらのコンポーネントのサイズが画面サイズ以上になる場合の動作は、メーカーにより異なる場合があります (コンポーネントサイズが画面サイズに制限されるなど)。PC 上で動作する Java 対応 i モードの携帯電話シミュレータ (以降、本書ではエミュレータと表記する場合があります) では、コンポーネントサイズは画面サイズを最大とし、画面領域より大きなサイズをコンポーネントに設定すると画面領域のサイズに制限されます。次の各項では、各 UI コンポーネントについて詳しく説明します。

Label

`Label` クラスは `Panel` 上に1行のテキストを表示するためのコンポーネントです。`Label` コンポーネントでは、枠のない文字列が表示されます。`Label` コンポーネントは可視または非可視に設定することが可能で、テキストの表示位置は、左、中央、または右に位置合わせすることが可能です。次のコードは、`Label` コンポーネントの例です。

例: `Label` コンポーネント

```
package uidemo;

import com.nttdocomo.ui.*;

/**
 * ラベルウィジェットとそれに対して呼び出された setSize メソッドの
 * 機能を示すクラス。
 * このクラスは1つのボタンと2つのラベルウィジェットを作成する。
 * ラベルウィジェットの可視/非可視の機能を示すため、
 * ボタンウィジェットを使用する。
 */
public class LabelDemo extends UIDemoPanel {
    Label lbl1, lbl2;
    Button btn1;
    int count = 0;

    LabelDemo() {
        lbl1 = new Label("Label1");
        lbl2 = new Label("This is a Label2");
    }
}
```

```

        btn1= new Button("Button1");
        lbl1.setSize(25,15);    //ラベルのサイズを設定する。

        this.add(btn1);
        this.add(lbl1);
        this.add(lbl2);
        ListenerClass lclass = new ListenerClass();
        this.setComponentListener(lclass);
    }
}

```

注意事項：

- Label コンポーネントがフォーカスを得ることはありません。

ImageLabel

ImageLabel クラスは Panel に画像を表示するためのコンポーネントです。ImageLabel コンポーネントでは、静的なイメージ（静止画像）のみ表示可能です。イメージが設定されていない ImageLabel は、バックグラウンドカラーで塗りつぶされます。Java GUI の標準 Swing クラスでは Label コンポーネントにイメージを設定してアイコンを作成可能ですが、i モード Java アプリケーション環境では、別個のコンポーネントが定義されています。ImageLabel は、可視または非可視に設定することができます。次のコードは、ImageLabel コンポーネントの例です。

例: ImageLabel コンポーネント

```

import com.nttdocomo.ui.*;
import com.nttdocomo.io.*;

public class ImageLabelDemo extends UIDemoPanel {
    ImageLabelDemo() {
        ImageLabel l1;

        // メディアイメージを取得する。
        MediaImage mi =
            MediaManager.getImage("resource:///UIDemo/img/cup.gif");
        try {
            mi.use();
        } catch (ConnectionException ce) {
            // リソース入出力に関する例外処理を記述
        }
        Image im = mi.getImage();

        // イメージオブジェクトをもつイメージラベルを作成する。
        l1 = new ImageLabel(im);
        // イメージラベルをパネルに追加する。
        add(l1);
    }
    ...
}

```

注意事項：

- ImageLabel コンポーネントがフォーカスを得ることはありません。

Button

Button クラスはボタンを表現するコンポーネントで、ユーザーの操作を受け付けるためのインタフェース `Interactable` を実装しています。Button コンポーネントには、テキストを設定することができます。このコンポーネントは有効または無効な状態に設定できます。次のコードは、Button コンポーネントの例です。

例: Button コンポーネント

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

public class ButtonDemo extends UIDemoPanel implements ComponentListener {
    Button b1, b2, b3, b4;
    ButtonDemo() {
        // 4つのボタンを作成し、それをパネルに追加する。
        b1 = new Button("B4 enabled");
        b2 = new Button("B4 disabled");
        b3 = new Button("ChangeB4");
        b4 = new Button("B4");

        this.add(b1);
        this.add(b2);
        this.add(b3);
        this.add(b4);
        ...
    }
    ...
}
```

ListBox

ListBox クラスは各種の選択リストを表現するためのコンポーネントで、ユーザーの操作を受け付けるためのインタフェース `Interactable` を実装しています。ListBox コンポーネントは矩形の枠をもっており、その中で項目をスクロール表示することが可能です。ListBox に表示する項目全体が画面に入りきらないと、上下のスクロールを示すインジケータが表示されます。ListBox には次のような種類があります。

- 単一選択リスト
- 複数選択リスト
- ラジオボタン
- チェックボックス
- 番号付きリスト
- オプションメニュー(ポップアップ)

枠の内部に含まれる項目の表示と選択方法は、使用する ListBox のタイプによって異なります。

タイプ	選択	
	単一選択	複数選択
リスト	単一選択リスト (SINGLE_SELECT)	複数選択リスト (MULTIPLE_SELECT)
番号付きリスト	番号付きリスト (NUMBERED_LIST)	なし
チェックボックス	なし	チェックボックスリスト (CHECK_BOX)
ラジオボタン	ラジオボタンリスト (RADIO_BUTTON)	なし
ポップアップ	オプションメニュー (CHOICE)	なし

このコンポーネントは有効または無効な状態に設定できます。次のコードは、ListBox コンポーネントの例です。

例: ListBox コンポーネント

```
package uidemo;

import com.nttdocomo.ui.*;

/**
 * リストボックスウィジェットの機能を示すクラス。
 * ここでは、リストタイプとして radio_button、numbered_list、
 * checkbox、choice、single_select、multiple select を示す。
 */
public class ListBoxDemo extends UIDemoPanel implements SoftKeyListener {
    ListBox l1, l2, l3, l4, l5, l6;
    int count = 0;
    int newItemCnt=0;
    ListBoxDemo() {
        Label radioLabel, checkLabel, numberLabel;
        Label choiceLabel, singleLabel, multipleLabel;

        radioLabel = new Label("Radio Button List");
        // RADIO_BUTTON タイプのリストボックスを作成し、いくつかの項目を追加する。
        l1 = new ListBox(ListBox.RADIO_BUTTON);
        l1.append("radio1");
        l1.append("long radio label");
        l1.append("radio3");
        l1.append("radio4");

        checkLabel = new Label("Checkbox List");
        // CHECK_BOX タイプのリストボックスを作成し、いくつかの項目を追加する。
        l2 = new ListBox(ListBox.CHECK_BOX);
        l2.append("checkbox1");
        l2.append("checkbox2");
        l2.append("checkbox3");
        l2.append("checkbox4");
        l2.append("checkbox5");

        numberLabel = new Label("Number List");
        // NUMBERED_LIST タイプのリストボックスを作成し、いくつかの項目を追加する。
        l3 = new ListBox(ListBox.NUMBERED_LIST);
        l3.append("number1");
        l3.append("number2");
        l3.append("number3");
        l3.append("number4");
        l3.append("number5");
    }
}
```

```

singleLabel = new Label("Single List");
// SINGLE_SELECT タイプのリストボックスを作成し、いくつかの項目を追加する。
l4 = new ListBox(ListBox.SINGLE_SELECT);
l4.append("single1");
l4.append("single2");
l4.append("single3");
l4.append("single4");
l4.append("single5");

multipleLabel = new Label("Multiple List");
// MULTIPLE_SELECT タイプのリストボックスを作成し、いくつかの項目を追加する
l5 = new ListBox(ListBox.MULTIPLE_SELECT);
l5.append("multiple1");
l5.append("multiple2");
l5.append("multiple3");
l5.append("multiple4");
l5.append("multiple5");

choiceLabel = new Label("Choice List");
// CHOICE タイプのリストボックスを作成し、いくつかの項目を追加する。
l6 = new ListBox(ListBox.CHOICE);
l6.append("choice1");
l6.append("choice2");
l6.append("choice3");
l6.append("choice4");
l6.append("choice5");

this.add(radioLabel);
this.add(l1);
this.add(checkLabel);
this.add(l2);
this.add(numberLabel);
this.add(l3);
this.add(singleLabel);
this.add(l4);
this.add(multipleLabel);
this.add(l5);
this.add(choiceLabel);
this.add(l6);

//このパネルに softkeylistener を設定する。
this.setSoftLabel(Frame.SOFT_KEY_2, "List");
setSoftKeyListener(this);
    }
    ...
}

```

注意事項：

- 番号付きリストは単一選択リストの特別な形式です。ユーザーは、番号付きリストの先頭から 9 個目までの項目を、携帯電話の数字キーを使用して選択することができます。最初の項目を選択するには数字 1 のキー、2 つめの項目を選択するには数字 2 のキーというように使用します。10 個目以後の項目は単一選択リストと同じ操作により選択します。
- オプションメニュー(ポップアップ)も単一選択リストの特別な形式です。オプションメニューにフォーカスがある状態で携帯電話の「選択」ボタンを押すと、ポップアップが表示されます。ユーザーはこのポップアップから目的の項目を選択します。なお、他の種類の `ListBox` と異なり、オプションメニューは初期状態では先頭の項目が選択された状態になっています。

TextBox

TextBox クラスは入力フィールドを表現するコンポーネントで、ユーザーの操作を受け付けるためのインタフェース **Interactable** を実装しています。TextBox コンポーネントでは、単一行または複数行のテキストが表示されます。テキスト入力可能なコンポーネントは TextBox だけです。アプリケーションで TextBox の編集モードを編集可能に設定すると、ユーザーがテキスト文字列を入力したり変更することが可能になります。編集モードを編集不可に設定すると、TextBox は読み取り専用になります。

TextBox のコンストラクタには、表示モードパラメータとして **DISPLAY_PASSWORD** または **DISPLAY_ANY** を指定します。入力テキストを非表示とする必要のあるアプリケーションでは、TextBox コンポーネントを **DISPLAY_PASSWORD** モードで使用すると、すべての文字が"*"で表示されます。表示モードパラメータを **DISPLAY_ANY** に設定すると、TextBox コンポーネントは入力テキストをそのまま表示します。

初期入力モード（かな漢字入力、アルファベット入力などの入力モード）を変更するには、**setInputMode()** メソッドに初期入力モードパラメータとして **KANA**（かな漢字入力モード）、**ALPHA**（アルファベット入力モード）、または **NUMBER**（数字入力モード）を指定します。TextBox は、初期状態では **KANA** になっています。

このコンポーネントは、有効または無効な状態に設定することができます。次のコードは、TextBox コンポーネントの例です。

例: TextBox コンポーネント

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

public class TextBoxDemo extends UIDemoPanel implements ComponentListener {
    TextBox t;                //単一行テキストボックス(テキストフィールド)
    TextBox t2;               //複数行テキストボックス(テキスト領域)
    Button b;
    TextBoxDemo() {
        t2 = new TextBox(
            "This is a test for textbox",20,3,TextBox.DISPLAY_ANY);
        t = new TextBox(
            "This is a test for textbox",20,1,TextBox.DISPLAY_ANY);
        b = new Button("TestText");

        this.add(b);
        this.add(t2);
        this.add(t);
        ...
    }
    ...
}
```

Ticker

Ticker クラスはスクロールする文字列を Panel に表示させるためのコンポーネントです。Ticker コンポーネントでは、テキストがティックカーテープ表示形式で表示されます。文字列は、画面を右から左にスクロールします。

次のコードは、Ticker コンポーネントの例です。

例: Ticker コンポーネント

```

package uidemo;

import com.nttdocomo.ui.*;

//ティックカーウィジェットの機能を示すクラス。
//最初にパネルが表示されたときには、ティックカーのデフォルト文字列が動いている。
//ユーザーは、テキストボタンを押してテキストを変更できる。
//ティックカーは可視または非可視に設定可能

public class TickerDemo extends UIDemoPanel implements ComponentListener {
    Ticker ticker;          //ティックカーウィジェット
    Button textButton;      //テキストの変更

    static int    whichText  = 1;
    static String textString1 = "This is a test text 1 for the ticker panel";
    static String textString2 = "Now you see test text 2";

    TickerDemo() {
        ticker      = new Ticker(textString1);
        textButton = new Button("change text");

        this.add(ticker);
        this.add(textButton);
        this.setComponentListener(this);
    }

    // Component c でイベントが発生したことを示している。
    //
    // @param c Component。このイベントが発生したコンポーネント
    // @param type イベントタイプ
    // @param param イベントのパラメータ
    public void componentAction(Component c, int type, int param) {

        //コンポーネントアクションが発生した。textButton がクリックされると、
        //ティックカーの文字列が他のものに更新される。

        if (c == textButton) {
            if (whichText==1) {
                whichText=2; ticker.setText(textString2);
            } else if (whichText==2) {
                whichText=1; ticker.setText(textString1);
            }
        }
    }
}

```

注意事項：

- コンポーネント内のテキストのスクロール動作をアプリケーションから制御することはできません。
- Ticker コンポーネントがフォーカスを得ることはありません。
- テキストのスクロール速度はメーカーにより異なります。

VisualPresenter

VisualPresenter クラスは動画像を再生するためのコンポーネントで、MediaPresenter を実装しています。VisualPresenter コンポーネントは、画面に表示されるメディアデータ（GIF アニメーションなど）の再生を行う高レベル API のコ

ンポーネントです。このコンポーネントでは、メディアのデータの再生に関するイベント処理のために **MediaListener** を使用することができます。イベント処理の詳細は、4.3 項を参照してください。メディアデータを再生するコード例は、4.4 項で説明します。

注意事項：

- **VisualPresenter** はコンポーネントであり、高レベル API でのみ使用することができます。
- **VisualPresenter** コンポーネントがフォーカスを得ることはありません。

4.1.4 コンポーネントの配置

アプリケーション開発者は、デフォルトのレイアウトモードを使って、コンポーネントの配置をデフォルトのレイアウトマネージャに任せるか、デフォルトレイアウトマネージャを無効にして、**Component** クラスの **setLocation()** メソッドでコンポーネントを特定のピクセル位置に配置できます。典型的なデフォルトレイアウト管理では、パネルに追加されたコンポーネントが画面の左上より順次横並びに配置され、画面の横幅で収まりきらない場合には、次のコンポーネント配置が可能な位置（1 段下の列）へ配置します。どのメーカーの携帯電話にも適用できる汎用的なアプリケーションを作成するためには、コンポーネントのレイアウト構成をシンプルに保つことを推奨します。なお、配置可能なコンポーネントの数は、メモリ使用状況や各機種固有の制限などの影響を受けます。

注意事項：

- **LayoutManager** インタフェースにはメソッドは定義されていません。このインタフェースは、メーカーが独自のレイアウトマネージャを提供するために用意されているものであり、アプリケーション開発者が本インタフェースを利用して独自のレイアウトマネージャを作成することはできません。メーカーが独自のレイアウトマネージャを提供している場合、**Panel** オブジェクトの **setLayoutManager()** メソッドを呼び出すことで、使用するレイアウトマネージャを変更することができます。
- レイアウトマネージャは無効にすることが可能です。レイアウトマネージャを無効にするには、**Panel** オブジェクトに対して、パラメータに **null** を指定して **setLayoutManager()** メソッドを呼び出します。レイアウトマネージャを無効にした場合、アプリケーションは各コンポーネントの **setLocation()** メソッドを使用して、コンポーネント毎にそれを配置する位置を指定します。なお、レイアウトマネージャを無効にした場合、機種によってはスクロール機能も無効になります。

次のコードは、レイアウトマネージャを無効化して、アプリケーションが独自にコンポーネントの配置を行う例を示したものです。

例: **setLocation()** によるコンポーネントの配置

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

public class NullLayoutDemo extends UIDemoPanel {
    Button button1, button2;
    Button offScreen;
    TextBox text;
    Label title, button1Label, button2Label;

    NullLayoutDemo() {
```

```
//ここですべてのコンポーネントを使用する。
title           = new Label("Null Layout ");
button1         = new Button("Button1");
button2         = new Button("Button2");
button1Label    = new Label("ONE:");
button2Label    = new Label("TWO:");
text            = new TextBox("A simple sentence in a
                           textbox", 40,
                           2, TextBox.DISPLAY_ANY);

/*
 *画面外に配置するコンポーネントの例として
 *このボタンを作成する。
 */
offScreen = new Button("This button is never seen");

/*
 *ここでレイアウトマネージャに null を設定する。
 *したがって、個々のコンポーネントの場所を設定する
 *必要がある。注：物理的な画面の座標外に配置される
 *コンポーネントは全く表示されない。null レイアウト
 *により、スクロールは無効にされる場合がある。
 */
setLayoutManager(null);

/*注： この場合でも、コンポーネントを追加する順序には
 *意味がある。コンポーネントは、この順序に従って
 *フォーカスを与えられる。
 */
add(title);
add(button1Label);
add(button1);
add(button2Label);
add(button2);
add(text);
//このコンポーネントも追加する。
add(offScreen);

/*
 *ここで、各コンポーネントの場所を設定する必要がある。
 *つまり、パネルのレイアウトを独自に指定する。
 *コンポーネントのサイズは自動的に設定することも、
 * setSize を直接呼び出してサイズを指定することもできる。
 */

title.setLocation(12, 0);
button1Label.setLocation(10, 18);
button1.setLocation(45, 18);
button2Label.setLocation(10, 40);
button2.setLocation(45, 40);
text.setLocation(2, 60);

/*
 *注： このボタンは画面外に配置される。したがって、
 *表示もされないし、フォーカスを得ることもない。
 */
offScreen.setLocation(1000, 1000);

}
}
```

4.1.5 フォーカスとスクロール

フォーカス管理とスクロール管理は各機種仕様の仕様に依存します。一般には、Java 対応 i モードの携帯電話にはシンプルなデフォルトフォーカスマネージャが搭載されています。デフォルトフォーカスマネージャでは、コンポーネントが Panel に追加された順序がフォーカスを得る順序になります。アプリケーションが起動されて画面が表示されると、有効な状態に設定されている最初の対話可能なコンポーネントにフォーカスが設定されます。デフォルトのスクロール動作は、フォーカス管理と緊密な関係があります。これは、画面のスクロールがフォーカスの移動に伴って行われるためです。

注意事項：

- フォーカスマネージャの実装する FocusManager インタフェースにはメソッドは定義されていません。このインタフェースは、メーカーが独自のフォーカスマネージャを提供するために用意されているものであり、アプリケーション開発者が本インタフェースを利用して独自のフォーカスマネージャを作成することはできません。メーカーが独自のフォーカスマネージャを提供している場合、Panel オブジェクトの setFocusManager() メソッドを呼び出すことで、使用するフォーカスマネージャを変更することができます。なおレイアウトマネージャと異なり、フォーカスマネージャを無効化することはできません。
- メーカーによっては、Panel のスクロール機能をサポートしない場合があります。

4.1.6 高レベルイベントの処理

高レベル API の場合、コンポーネントによって生成されたイベントは、ComponentListener を実装したオブジェクトの componentAction() メソッドに渡されます。またソフトキーイベントは、SoftKeyListener を実装したオブジェクトの softKeyPressed() または softKeyReleased() メソッドに渡されます。このように、高レベルイベント処理にはリスナーモデルが採用されています。

ComponentListener も SoftKeyListener も Panel に登録して使用します。高レベルイベント処理におけるイベントリスナーとコード例の詳細は、4.3 項を参照してください。

4.2 低レベル API によるプログラミング

ゲームアプリケーションなどの分野では、コンポーネントを使用したユーザインタフェースよりもグラフィックス描画機能を多用する場合があります。この節では、主にグラフィックス描画を行うための低レベル API の使い方とイベントの処理方法について説明します。

4.2.1 Canvas の使用

Canvas は、グラフィックスを描画するための画面領域です。アプリケーション開発者は、Canvas クラスのサブクラスを作成し、paint() メソッドをオーバーライドすることによって、Canvas に描画することができます。paint() メソッドは、システムが Canvas

の描画が必要と判断したときやアプリケーションが明示的に再描画を要求したとき（repaint()メソッドを呼び出したとき）にシステムからコールバックされます。

Panel と同様に、1 つの Canvas オブジェクトは 1 つの画面に相当します。したがって、1 つの画面に複数の Canvas を表示したり、Canvas と Panel を同時に 1 つの画面に表示することはできません。ただし複数の Canvas や Panel を、1 つの画面上に順次切り替えて表示することは可能です。

次に Canvas に簡単な描画を行う例を示します。

例: Canvas への描画

```
import com.nttdocomo.ui.*;
public class CanvasDemo extends Canvas {
    int x = 20;
    int y = 10;

    CanvasDemo() {
        // ソフトラベル「back」を作成する。
        setSoftLabel(Frame.SOFT_KEY_1, "Back");
    }

    /**
     * キャンバス領域に文字列と矩形を描画する。
     */
    public void paint(Graphics g) {
        g.lock();
        g.clearRect(0, 0, 1000, 1000);
        g.drawString("Hello", x, y);
        g.drawString("Canvas", x, y+15);
        g.drawRect(x+10, y+30, 20, 5);
        g.unlock(true);
    }
    ...
}
```

注意事項：

- repaint()メソッドはシステムに対し描画の必要があることを指示するためのものであり、1 回の repaint()メソッド呼び出しが 1 回の paint()メソッド呼び出しに対応するものではありません。このためアプリケーションから見た場合に、複数の連続した repaint()メソッド呼び出しが 1 回の paint()メソッド呼び出しにまとめられることがあります。アプリケーションは、paint()メソッド呼び出しが 1 回にまとめられたような場合にも画面の描画が正しく行われるよう paint()メソッドを記述する必要があります。

4.2.2 グラフィックスの描画

線や図形、テキストなどの描画には、グラフィックスオブジェクト（Graphics クラスのオブジェクト）を使用します。このオブジェクトには描画可能な領域、フォント、色などの情報が含まれているため、グラフィックスコンテキストとも呼ばれます。前項の例ではグラフィックスコンテキスト g を使い、clearRect()メソッドによる画面のクリアの後に drawString()メソッドを呼び出してテキストを描画し、続いて drawRect()メソッドを呼び出して矩形を描画しています。

上の例で使用している `lock()` メソッドと `unlock()` メソッドは、グラフィックス描画のダブルバッファリングを管理するために `Graphics` クラスに用意されているメソッドです。ダブルバッファリング中は、指示された描画はオフスクリーンバッファに対して行われ、ダブルバッファリング終了とともにオンスクリーンにコピーされます。このため、ダブルバッファリングを使用するとフリッカーが発生しません。

ダブルバッファリングは、メーカーによりサポートされない場合があります。ダブルバッファリングをサポートしない機種では、この2つのメソッドは何も行いません。ダブルバッファリングをサポートするデバイスでは `lock()` メソッドが呼び出されると(1回または複数回)、`unlock()` メソッドが引数に `true` が指定されて呼び出されるか、引数に `false` が指定されて `lock()` メソッド呼び出しと同じ回数だけ呼び出されるまで、画面への描画は一時的に保留されます。

注意事項：

- ダブルバッファリングの実装は携帯電話によって異なります。アプリケーションの正しい動作を保証するために、ダブルバッファリングを使用する際は次のガイドラインに従うようにしてください。
 - 異なる `Graphics` オブジェクトの `lock()` メソッドと `unlock()` メソッドの呼び出しを混在して行わない。
 - `paint()` メソッドを終了する前には必ず `Graphics` オブジェクトのロックを解除する。

次のコードに、ロックおよびロック解除の正しい使い方を示します。

例：正しい実装（上記ガイドラインに従っている）

```
import com.nttdocomo.ui.*;
public void paint(Graphics g) {
    Graphics g2 = g.copy();
    g.lock();
    //何かを描画する .....
    g.unlock(false);

    g2.lock();
    // さらに描画する .....
    g2.unlock(false);
}
```

次のコードに、ロックおよびロック解除の正しくない使い方を示します。

例：正しくない実装

```
import com.nttdocomo.ui.*;
public void paint(Graphics g) {

    Graphics g2 = g.copy();
    g.lock();
    //何かを描画する .....
    g2.lock();                      // g と g2 のロック区間が重複している
    //さらに描画する .....
    g.unlock(false);                // g2 のロック解除が行われていない
}
```

- `paint()` メソッドの引数で渡されるグラフィックスオブジェクトに対し、`Graphics.dispose()` メソッドは発行しないでください。グラフィックスオブジェクトの破棄は、必要に応じて Java 実行環境により行われます。

- i モード対応 Java では、repaint()メソッドの引数に矩形領域を設定することで、指定された矩形領域のみ再描画を行うことができます。ただしこの機能は、端末の描画デバイスの持つクリッピング描画能力に依存します。クリッピング描画機能を持たない端末で repaint()メソッドに矩形領域を指定した場合、その指定は無視され画面全体が再描画されます。

4.2.3 カラーサポート

Graphics オブジェクトには、オブジェクトの色情報が含まれています。Graphics クラスには2つのメソッドがあり、それぞれが色を表す整数値を返します。

getColorOfName(int name) : 色名 (Graphics 定数) を表す整数値を返します。
 getColorOfRGB(int r, int g, int b) : RGB 値を表す整数値を返します。

色はメーカーに依存するため、色に割り当てられる整数値（これらのメソッドの返り値）はメーカーによって異なる可能性があります。アプリケーション開発者は、この整数値にいかなる仮定も置くことはできません。たとえば、あるメーカーの白黒画面では黒の値が1であったとしても、別のメーカーのカラー画面では黒の値は255 かもしれません。

以下に、正しくないコードの例を示します。

例: グラフィックスキャンバス

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

public class MyCanvas extends Canvas {
    ...
    public void paint(Graphics g) {
        ...
        if(g.getColorOfName(Graphics.BLACK) == 1) { //何かをする。
            //あるメーカーの機種では結果が真になり、このコードが実行されたとしても、
            //他のメーカーの機種では偽になり、コードはスキップされるかもしれない。
        }
        ...
    }
}
```

4.2.4 低レベルイベントの処理

低レベル API でのイベント処理は、高レベル API の場合とは異なります。Canvas オブジェクトは低レベルイベントだけを処理します。アプリケーションは Canvas クラスを拡張する際に、processEvent()メソッドをオーバーライドすることができます。このメソッドには低レベルイベント発生時に、イベント種別とそれに伴うパラメータが渡されます。次の表は、Canvas に通知される有効なイベントタイプを示したものです。これらのイベントタイプは Display クラスに定義されています。

イベントタイプ	イベントタイプ値	パラメータ
KEY_PRESSED_EVENT	0	押されたキーの値
KEY_RELEASED_EVENT	1	離されたキーの値
RESUME_VM_EVENT	4	使用されない
UPDATE_VM_EVENT	6	使用されない
TIMER_EXPIRED_EVENT	7	タイマーイベントの発生した時刻

		ShortTimer のタイマーID
--	--	--------------------

次のコードは、低レベルイベントを処理する `processEvent()` メソッドの実装例です。

例: イベント処理の実装

```
package spaceinv;

import com.nttdocomo.ui.*;
import com.nttdocomo.ui.ShortTimer;
import java.util.Random;

/**
 * このクラスは、特定のハイスコアサーバーに設定されている
 * 歴代の殿堂入りハイスコアを表示するときに使用する。
 * スコアとその名前がアニメーション風に表示される。
 * このクラスは、ユーザー入力を受け取るか、このクラスが
 * 維持するアイドルタイマーが切れると、次の画面に移る。
 */
public class HighScoreScreen extends Canvas implements Runnable {
    private boolean done;
    private ShortTimer timer;

    /** このクラスの新しいインスタンスを作成する。 */
    public HighScoreScreen() {
        ...
    }

    /** このインスタンスを初期状態にリセットする。 */
    public void reset(String[] list) {
        ...
    }

    /** Runnable インタフェースに対して必要。 */
    public void run() {
        ...
    }

    /** この画面を表示する。 */
    public void paint(Graphics g) {
        ...
    }

    /** イベント処理を行うために Canvas の processEvent() をオーバーライドする。 */
    public void processEvent(int type, int param) {
        if (type == Display.KEY_PRESSED_EVENT ||
            type == Display.TIMER_EXPIRED_EVENT) {
            done = true;
            timer.stop();
            timer.dispose();
            SpaceInvaders.WELCOME.reset();
            Display.setCurrent(SpaceInvaders.WELCOME);
        }
    }
}
```

注意事項：

- `processEvent()` メソッドの呼び出しは、ユーザーの操作やタイマーなど非同期なイベント発生を契機として行われます。`processEvent()` メソッドの呼び出し時に、それがどのようにスレッドに割り当てられるかはメーカーにより異なりますので注意してください。例えばあるメーカーの機種では、

同時に発生した複数のイベントに対して1つのスレッドで処理を行うかもしれません。
`processEvent()`メソッド内で行われる処理は極力短時間で完結するようにしてください。

4.3 イベントリスナー

イベントリスナーとは、イベントを通知するオブジェクト（`Panel`や`MediaPresenter`など）に登録され、特定タイプのイベントを待ち受けるオブジェクトです。イベントが発生すると、オブジェクトは適切なイベントタイプを構築し、そのイベントに対応するイベントリスナーに通知します。イベントリスナーには、次の5つのタイプがあります。

- `ComponentListener`
- `SoftKeyListener`
- `KeyListener`
- `MediaListener`
- `TimerListener`

ダイアログで使用されるイベントリスナーはありません。ダイアログを画面に表示するには、`Dialog.show()`メソッドを使用します。`Dialog.show()`メソッドは、ユーザーがダイアログとの対話を終了したときにアプリケーションに制御を戻します。`Dialog.show()`メソッドの戻り値は、ユーザーがダイアログ内のどのボタンを押したかを表しています。

イベントリスナーの実装方法とイベントハンドラのコード例については、サンプルアプリケーションのUIDemoも参照してください。

注意事項：

- リスナーメソッドの呼び出しは、ユーザーの操作やタイマーなど非同期なイベント発生を契機として行われます。リスナーメソッドの呼び出し時に、それがどのようにスレッドに割り当てられるかはメーカーにより異なりますので注意してください。例えばあるメーカーの機種では、同時に発生した複数のイベントに対して1つのスレッドで処理を行うかもしれません。リスナーメソッド内で行われる処理は極力短時間で完結するようにしてください。

4.3.1 ComponentListener

`ComponentListener` インタフェースは、表示されているフレーム上のUIコンポーネントに対して起動されたイベントを待ち受けるときに使用します。このインタフェースのメソッドは`componentAction()`だけです。コンポーネントリスナーの定義と登録は、以下のように行います。

- リスナーとして使用するクラスに`ComponentListener`インタフェースを実装し、`componentAction()`メソッドを定義します。コンポーネントイベントが発生すると、`componentAction()`メソッドが呼び出され、イベントタイプ、イベントが発生したコンポーネント、およびその他の必要なパラメータがイベントリスナーに渡されます。
- `Panel`オブジェクトに対して`setComponentListener()`メソッドを呼び出し、上記で実装したリスナークラスのインスタンスを登録します。

`ComponentListener` には、次の3つのイベントタイプが定義されています。

- `BUTTON_PRESSED`：

Button が押されたときに発生するイベント。イベントパラメータは使用しません。

- **SELECTION_CHANGED:**
ListBox の選択内容が変更されたときに発生するイベント。イベントパラメータとして、選択または状態変更された項目のインデックスが渡されます(複数選択リストでは選択された項目の中で最も小さいもののインデックスが渡されます)。
- **TEXT_CHANGED:**
TextBox の文字入力終了したときに発生するイベント。イベントパラメータは使用しません。

次に、ComponentListener を内部クラスとして定義し、使用する例を示します。

例: ComponentListener を実装する内部クラス

```
package uidemo;

import com.nttdocomo.ui.*;

/**
 * ラベルウィジェットとそれに対して呼び出された setSize メソッドの
 * 機能を示すクラス。
 * このクラスは 1 つのボタンと 2 つのラベルウィジェットを作成する。
 * ラベルウィジェットの可視/非可視の機能を示すため、
 * ボタンウィジェットを使用する。
 */
public class LabelDemo extends UIDemoPanel {
    Label lbl1, lbl2;
    Button btn1;
    int count = 0;

    LabelDemo() {
        lbl1 = new Label("Label1");
        lbl2 = new Label("This is a Label2");
        btn1 = new Button("Button1");
        lbl1.setSize(25, 15); // ラベルのサイズを設定する。

        this.add(btn1);
        this.add(lbl1);
        this.add(lbl2);
        ListenerClass lclass = new ListenerClass();
        this.setComponentListener(lclass);
    }

    /**
     * ボタンイベントを処理する内部クラス
     */
    class ListenerClass implements ComponentListener {
        /**
         * イベントが Component c で発生したことを示す。
         *
         * @param c このイベントが発生した Component
         * @param type イベントのタイプ
         * @param param イベントのパラメータ
         */
        public void componentAction(Component c, int type, int param) {
            /**
             * コンポーネントアクションが行われた。
             * コンポーネントがボタンの場合、
             * カウントが 0 なら、このパネルの最初のラベルが非可視に、
             * カウントが 1 なら、同じラベルを可視にそれぞれ設定する。
             */
        }
    }
}
```

```

        */
        if (c == btn1) {
            if (count == 0) {
                lbl1.setVisible(false);
                count++;
            }
            else {
                lbl1.setVisible(true);
                count=0;
            }
        }
    }
}

```

4.3.2 SoftKeyListener

SoftKeyListener インタフェースは、2つのソフトキーが操作されるのを待ち受けるときに使用します。このインタフェースで定義されているメソッドは、`softKeyPressed()`と`softKeyReleased()`の2つです。ソフトキーリスナーの定義と登録は、以下のように行います。

- リスナーとして使用するクラスに SoftKeyListener インタフェースを実装し、`softKeyPressed()`と`softKeyReleased()`という2つのメソッドを定義します。ソフトキーが押されると`softKeyPressed()`メソッドが、放されると`softKeyReleased()`メソッドが呼び出されます。
- Panel オブジェクトに対して `setSoftKeyListener()`メソッドを呼び出し、上記で実装したリスナークラスのインスタンスを登録します。

注意事項：

- ソフトキーは、Canvas から也可以使用することができます。その場合は、低レベルイベントの処理方法に従ってソフトキーリスナーの代わりに `Canvas.processEvent()`を使用します（通常のキー操作と同様に、`KEY_PRESSED_EVENT` または `KEY_RELEASED_EVENT` が通知されます）。

4.3.3 KeyListener

KeyListener インタフェースは、ソフトキー以外のいずれかのキーが操作されるのを待ち受けるときに使用します。このインタフェースで定義されているメソッドは、`keyPressed()`と`keyReleased()`の2つです。キーリスナーの定義と登録は、以下のように行います。

- リスナーとして使用するクラスに KeyListener インタフェースを実装し `keyPressed()`と`keyReleased()`という2つのメソッドを定義します。キーが押されると`keyPressed()`メソッドが、放されると`keyReleased()`メソッドが呼び出されます。
- Panel オブジェクトに対して `setKeyListener()`メソッドを呼び出し、上記で実装したリスナークラスのインスタンスを登録します。

4.3.4 MediaListener

MediaListener インタフェースは、メディアイベント（AudioPresenter または VisualPresenter が通知するイベント）を待ち受けるときに使用します。このインタ

フェースで定義されているメソッドは `mediaAction()` だけです。メディアリスナーの定義と登録は、以下のように行います。

- リスナーとして使用するクラスに `MediaListener` インタフェースを実装し、`mediaAction()` メソッドを定義します。メディアイベントが発生すると、このメソッドが呼び出されます。
- `MediaPresenter` オブジェクトに対して `setMediaListener()` メソッドを呼び出し、上記で実装したリスナークラスのインスタンスを登録します。

`AudioPresenter` には、次の3つのイベントタイプが定義されています。

- `AUDIO_COMPLETE` : 音声メディアデータの再生完了
- `AUDIO_PLAYING` : 音声メディアデータの再生開始
- `AUDIO_STOPPED` : 音声メディアデータの再生停止

`VisualPresenter` には、次の3つのイベントタイプが定義されています。

- `VISUAL_COMPLETE` : 画像メディアデータの再生完了
- `VISUAL_PLAYING` : 画像メディアデータの再生開始
- `VISUAL_STOPPED` : 画像メディアデータの再生停止

なお、メーカーによってはメディアイベントの通知をサポートしない場合があります。

4.3.5 TimerListener

`TimerListener` インタフェースは、タイマー終了イベントを待ち受けるときに使用します。このインタフェースで定義されているメソッドは `timerExpired()` だけです。タイマーリスナーの定義と登録は、以下のように行います。

- リスナーとして使用するクラスに `TimerListener` インタフェースを実装し、`timerExpired()` メソッドを定義します。タイマーが終了すると、このメソッドが呼び出されます。
- `Timer` オブジェクトに対して `setListener()` メソッドを呼び出し、上記で実装したリスナークラスのインスタンスを登録します。

注意事項：

- `ShortTimer` には、これに対応するインタフェースはありません。`ShortTimer` では、`Canvas` の `processEvent()` メソッドを通じて `TIMER_EXPIRED_EVENT` を受け取ります。

4.4 マルチメディアデータの使用

i モード Java アプリケーション環境では、アニメーションを含む画像の表示や音声ファイルの再生を行う機能が提供されます。GIF 形式の画像ファイルを表示するクラスと i

メロディ形式の音声ファイルを再生するクラスとしてそれぞれ、VisualPresenter と AudioPresenter があります。

注意事項：

- VisualPresenter はコンポーネントであり、高レベル API にて Panel オブジェクトに追加して使
用します。
- AudioPresenter はコンポーネントではありません。AudioPresenter は Panel 表示時、Canvas
表示時のいずれの場合も使用することができます。
- 音声ファイルとして、実際の携帯電話では i メロディ形式ファイルが使用されますが、PC 上で動作す
る携帯電話シミュレータでは MIDI 形式ファイルを使用します。携帯電話シミュレータを使用した実
際の開発手順などについては第 9 章を参照してください。

次に、VisualPresenter クラスを使って画像ファイルを表示する例と、AudioPresenter クラスを使って音声ファイルを再生する例を示します。

この例では、画像ファイルや音声ファイルとして、アプリケーションの JAR ファイルに同梱されたリソースを使用しています ("resource:/// "で始まる URL の使用)。リソースについては、第 7 章を参照してください。なお、この URL にはリソースだけではなく、Web サーバ上のファイルを示す HTTP の URL を指定することもできます。

例： 画像ファイルの表示

```
package uidemo;

import java.util.*;
import com.nttdocomo.ui.*;
import com.nttdocomo.io.ConnectionException;

public class VisualPresenterDemo extends UIDemoPanel {

    int          px, py;          // 現在の画像の表示位置 (VisualPresenter 内)
    VisualPresenter theVP;        // プレゼンター

    Button        startButton;    // GIF アニメーション再生ボタン
    Button        stopButton;     // GIF アニメーション停止ボタン

    ListBox       repeatChoice;   // 繰り返し再生指示用チェックボックス
    Ticker        ticker;         // キーマップ説明用ティックカーテープ

    static String instr = "Control ImagePos: Key 2-Down 8-Up 4-Left 6-Right";

    /**
     * VisualPresenterDemo コンストラクタ
     * 必要な画像データを取得して、VisualPresenter に引き渡す。
     */
    VisualPresenterDemo() {

        // 画像データを取得する。
        MediaImage mi =
            MediaManager.getImage("resource:///uidemo/img/singleloop.gif");
```

```

// use()メソッドを呼び出すまで画像データを利用することはできない。
try {
    mi.use();
} catch (ConnectionException ce) {
    // ここで入出力の例外を処理する。
} catch (UIException uie) {
    // ここでUIの例外を処理する。
}

px    = 0;
py    = 0;

theVP    = new VisualPresenter();
theVP.setImage(mi);

ticker    = new Ticker(instr);

startButton    = new Button("start");
stopButton     = new Button("stop ");

repeatChoice    = new ListBox(ListBox.CHECK_BOX);
repeatChoice.append("Repeat");

add(theVP);
add(ticker);
add(startButton);
add(stopButton);
add(repeatChoice);

// メディアリスナーを設定する。
theVP.setMediaListener(new MediaListenerClass());

setTitle("Status Line");

this.setComponentListener(new ListenerClass());
this.setKeyListener(new KeyListenerClass());
}

class KeyListenerClass implements KeyListener {
    /**
     * キーリリースイベントは無視する。
     */
    public void keyReleased(Panel p, int key) {}

    /**
     * キープレスイベントにより画像表示位置の調整を行う
     *
     * @param p イベントの発生した Panel
     * @param key 押されたキーの値
     */
    public void keyPressed(Panel p, int param1) {
        if (VisualPresenterDemo.this == p) {
            if (param1 == Display.KEY_8) { // 画像の位置 (下)
                py--;
            } else if (param1 == Display.KEY_2) { // 画像の位置 (上)
                py++;
            } else if (param1 == Display.KEY_4) { // 画像の位置 (左)
                px++;
            } else if (param1 == Display.KEY_6) { // 画像の位置 (右)

```

```

        px--;
    }

    // VisualPresenter の setAttribute()メソッド呼び出しにより
    // 画像表示位置を設定する。
    theVP.setAttribute(VisualPresenter.IMAGE_XPOS, px);
    theVP.setAttribute(VisualPresenter.IMAGE_YPOS, py);
    }
}

class ListenerClass implements ComponentListener {
    /**
     * コンポーネントイベントにより再生の開始、停止を制御する
     *
     * @param c      イベントの発生したコンポーネント
     * @param type    イベントタイプ
     * @param param   イベントパラメータ
     */
    public void componentAction(Component c, int type, int param) {
        if (c == startButton) {                // 再生ボタンを押すと再生開始
            theVP.play();
        } else if (c == stopButton) {          // 停止ボタンを押すと再生停止
            theVP.stop();
        }
    }
}

class MediaListenerClass implements MediaListener {
    /**
     * VisualPresenter からのイベントを受け取り、必要に応じて再度再生を行うための
     * メディアリスナー。ループ再生機能はここで実現される。
     */
    public void mediaAction(MediaPresenter source, int type, int param) {

        if (source == theVP) {
            switch (type) {

                // 画像の再生完了イベントはここで処理される。Panel タイトルに
                // 再生を完了したことを表示し、必要に応じて再度再生を繰り返す。
                // この処理は非ループ再生型 GIF アニメーションでのみ有用。
                case VisualPresenter.VISUAL_COMPLETE :
                    setTitle("Complete");

                    if ((repeatChoice.isSelected(0))) {
                        ((VisualPresenter) source).play();
                        setTitle("[Comp]Play again");
                    }
                    break;

                // 画像の再生開始イベントはここで処理される。
                // Panel タイトルに再生を開始したことを表示する。
                case VisualPresenter.VISUAL_PLAYING :
                    setTitle("Playing");
                    break;

                // 画像の再生停止イベントはここで処理される。
                // Panel タイトルに再生を停止したことを表示する。
                case VisualPresenter.VISUAL_STOPPED :
                    setTitle("Stopped");
            }
        }
    }
}

```

```

        break;
    }
}
}

/**
 * MediaImage は不要になったら破棄 (unuse()および dispose()) すべきである。
 * 破棄することにより資源の解放が行われる。ここでは MediaImage を再利用する可能性
 * があるため破棄は行わない。
 */
public void leave() {
}
}

```

例: 音声ファイルの再生

```

package uidemo;

import com.nttdocomo.ui.*;
import com.nttdocomo.io.ConnectionException;

public class AudioPresenterDemo extends UIDemoPanel {
    ListBox      loopChoice;    // ループ再生フラグ
    Button       playButton;    // 音声再生ボタン
    Button       stopButton;    // 音声停止ボタン
    Label        statusLabel;   // ステータス表示ラベル
    AudioPresenter theAP;       // AudioPresenter
    MediaSound    theMediaSound; // MediaSound

    /**
     * AudioPresenterDemo コンストラクタ
     * 再生/停止用の 2 つのボタンと、ループ再生を選択するためのチェックボックスを作成する。
     */
    AudioPresenterDemo() {

        loopChoice = new ListBox(ListBox.CHECK_BOX);
        loopChoice.append("loop");

        playButton = new Button("play");
        stopButton = new Button("stop");
        statusLabel = new Label("status line");

        theMediaSound =
            MediaManager.getSound("resource:///uidemo/img/music.mld");

        // use()メソッドを呼び出すまで音声データを利用することはできない。
        try {
            theMediaSound.use();
        } catch (ConnectionException ce) {
            // ここで入出力の例外を処理する。
        } catch (UIException uie) {
            // ここで UI の例外を処理する。
        }

        theAP = AudioPresenter.getAudioPresenter();
        theAP.setSound(theMediaSound);

        this.add(playButton);
    }
}

```

```

        this.add(stopButton);
        this.add(loopChoice);
        this.add(statusLabel);
        this.setComponentListener(new ListenerClass());
        theAP.setMediaListener(new MediaListenerClass());
    }

    class ListenerClass implements ComponentListener {
        /**
         * 音声データの再生/停止を制御するためのボタンのイベントを受け取るリスナー
         * クラス。
         */
        public void componentAction(Component c, int type, int param) {
            if (c == playButton) {
                theAP.play();
            } else if (c == stopButton) {
                theAP.stop();
            }
        }
    }

    class MediaListenerClass implements MediaListener {
        /**
         * AudioPresenter からのイベントを受け取り、必要に応じて再度再生を
         * 行うためのメディアリスナー。ループ再生機能はここで実現される。
         */
        public void mediaAction(MediaPresenter source, int type, int param) {
            if (source == theAP) {
                switch(type) {
                    // 音声の再生完了イベントはここで処理される。ステータス表示
                    // ラベルに再生を完了したことを表示し、必要に応じて再度再生
                    // を繰り返す。
                    case AudioPresenter.AUDIO_COMPLETE:
                        statusLabel.setText("complete");
                        if (loopChoice.isSelected(0))
                            ((AudioPresenter)source).play();
                        break;

                    // 音声の再生開始イベントはここで処理される。
                    // ステータス表示ラベルに再生を開始したことを表示する。
                    case AudioPresenter.AUDIO_PLAYING:
                        statusLabel.setText("playing ");
                        break;

                    // 音声の再生停止イベントはここで処理される。
                    // ステータス表示ラベルに再生を停止したことを表示する。
                    case AudioPresenter.AUDIO_STOPPED:
                        statusLabel.setText("stopped ");
                        break;
                }
            }
        }
    }

    /**
     * この画面が非表示になる際に、音声の再生を停止するため、AudioPresenter.stop()
     * メソッドの呼び出しを行う。MediaSound をこれ以上使用しない場合は、MediaSound.
     * unuse()メソッドおよび MediaSound.dispose()メソッドも呼び出すべきである（各種
     * 資源の解放が行われる）。ここでは MediaSound を再度利用する可能性があるため、これら
     * のメソッドは呼び出さない。
     */

```

```
        public void leave() {  
            theAP.stop();  
        }  
    }
```

第 5 章

通信の制御

i モード Java アプリケーション環境では、ネットワーキング API が提供されます。これは、Web サーバー側コンポーネントと通信するときにアプリケーション開発者が使用する一連のクラスです。これらのクラスの目的は、Java クライアントアプリケーションの開発をできるだけ簡単にすることにあります(第 2 章の図 5「Generic Connection クラス階層」を参照)。使用可能な通信セマンティクスは、HTTP(S)プロトコルの HEAD、GET、POST メソッドです。標準の J2SE ライブラリには、記憶装置との入出力や通信を処理する機能が豊富に揃っています。i モード Java アプリケーション環境では、2.2.2 項で述べた CLDC 仕様の Generic Connection フレームワークに基づいて携帯電話と Web サーバーを接続する基本的な機能が提供されます。以下の各節では、HTTP(S)接続インタフェースを使ってサーバーと通信する手順について説明します。

i モード Java アプリケーション環境のネットワーキング API と J2SE の `java.net.HttpURLConnection` は、次の点で大きく異なります。

- メッセージコンテンツのタイプを認識し、コンテンツを分析してオブジェクト形式に変換するハンドラをプラグインするためのコンテンツハンドラファクトリという概念はありません。
- メモリサイズを削減するため、サポートするコンテンツタイプを絞っています。一般的なアプリケーションでは、使用するコンテンツタイプは限られているため、多様なコンテンツタイプをサポートする総合的なフレームワークは採用していません。
- 汎用的な `URLConnection` (`java.net.URLConnection` クラス)はありません。クラスの数进行最小限に抑えるため、汎用的な `URLConnection` API は `HttpConnection` インタフェースに統合されています。これによってメモリサイズが減少し、開発者にとっても使用すべき API が簡素になるというメリットがあります。
- `URLStreamHandlerFactory` や `URLStreamHandler` API はありません。i モード Java アプリケーション環境では、使用できるネットワークプロトコルは HTTP、HTTPS に限定されており、多様な通信プロトコルに対応するための総合的なフレームワークは採用していません。これによってメモリサイズが減少し、セキュリティの問題も取り除かれます。
- i モード Java 拡張 API の `HttpConnection` を使用する上では、Generic Connection フレームワーク API の厳密な呼び出しシーケンスが定義されています。詳細は、5.1 項を参照してください。

5.1 クライアント/サーバープログラミング

多くのプログラムは、クライアントとサーバーが対になって動作します。Java 対応 i モードでは、サーバーはインターネット経由でアクセスするリモートホストシステムであり、クライアントは携帯電話上で動作する Java アプリケーションです。クライアントとサーバーは、通常、IP アドレス(サーバー固有の識別子)とポート番号(接続を特定のプロ

セスにルーティングするための識別子)を知ることによって相互に通信します。構造をシンプルにするため、ネットワーク接続には CLDC 仕様で定義されている入出力ストリームが使用されます。データの読み書きは、一般的な Java のストリームクラスおよびメソッドを使って行います。そのため、通信対象への接続から入力ストリームを取得するメソッドと出力ストリームを取得するメソッドがそれぞれ用意されています。これによって、クライアントとサーバーは相互に通信することができます。クライアントは、HTTP サーバーから Web ページの内容を入手するなどのインターネットサービスが必要になると、URL を使ってこのサービスにアクセスします。i モード Java アプリケーション環境では、標準 Java の `URLConnection` のサブセットとして `HttpConnection` インタフェースが提供されます。次のコード行は、`HttpConnection` インタフェース（の実装オブジェクト）の設定と使用の例を示したものです。

例: 接続の確立と動作

```
package spaceinv;

import java.io.*;
import javax.microedition.io.*;
import com.nttdocomo.io.*;

private String[] getScoreList(int score) {
    try {
        // 接続を設定する
        HttpConnection conn = (HttpConnection)Connector.open(
            "http://backflip.sfbay:8080/scores",Connector.READ_WRITE,
            true);

        // 要求メソッドとコンテンツタイプを設定する。
        conn.setRequestMethod(HttpConnection.POST);
        conn.setRequestProperty("Content-Type","text/plain");

        // 出力ストリームを取り出す。
        OutputStream out = conn.openOutputStream();

        // データを書き込む(実際の接続がまだ確立
        // されていないため、バッファに入れられる)。
        out.write("Highscore-Game: Space Invaders\n".getBytes());
        out.write(("Highscore-Name: " + SpaceInvaders.USER.getName() +
            "\n").getBytes());
        out.write(("Highscore-Value: " + score + "\n").getBytes());

        // 出力ストリームをクローズする。
        out.close();

        // リモート資源に実際に接続する。
        conn.connect();

        String[] result = new String[HighScoreScreen.MAX_ENTRIES];

        // 入力ストリームを取り出す。
        InputStream in = conn.openInputStream();
        if (in == null)
            throw new IOException("high score servlet unreachable");

        // データを読み取る。
        for (int i=0; i<HighScoreScreen.MAX_ENTRIES; i++) {
            result[i] = readLine(in);
        }
    }
}
```

```

        // 入力ストリームをクローズする。
        in.close();

        // http 接続をクローズする。
        conn.close();

        return result;
    } catch (Exception x) {
        return new String[] {"Scores", "currently", "unavailable."};
    }
}

```

`HttpConnection` は、HTTP や HTTPS プロトコルのネットワーク資源にアクセスするために使用されるインタフェースです。このインタフェースを実装したクラスのインスタンスは、URL で指定する資源の読み取りと書き込みの両方に使用できます。一般に、URL との接続は、次の手順に従って行います。

1. `Connector.open()` メソッドを呼び出して接続オブジェクトを作成する。オープンモードとリクエストメソッドは以下のように対応する。

`Connector.READ` : GET メソッドおよび HEAD メソッドに対応

`Connector.READ_WRITE` : POST メソッドに対応

2. セットアップパラメータを設定する(リクエストメソッドとリクエストヘッダ)。
3. `HttpConnection.openOutputStream()` メソッドを呼び出して、この接続の出力ストリームを取得する。
4. `OutputStream.write()` メソッドを使って出力ストリームに書き込む。
5. `OutputStream.close()` メソッドを使って出力ストリームをクローズする。
6. `HttpConnection.connect()` メソッドを使ってリモートオブジェクトに実際に接続する。この後、必要に応じてレスポンスヘッダを参照することができます。
7. `HttpConnection.openInputStream()` メソッドを使って入力ストリームを取得する。
8. `InputStream.read()` メソッドを使って入力ストリームから読み取る。
9. `InputStream.close()` メソッドを使って入力ストリームをクローズする。
10. `HttpConnection.close()` メソッドを使って接続をクローズする。

上記の 3~5 は、POST メソッド使用時における HTTP リクエストボディの書き込み処理に相当します。また 7~9 は、POST、GET 両メソッドにおける HTTP レスポンスボディの読み込み処理に相当します。

これらの処理はそれぞれオプションの処理ですが、処理を行う場合は全体として上記のシーケンスに合致するようにしてください。例えば `HttpConnection.connect()` を呼び出した後で `HttpConnection.openOutputStream()` を呼び出すと、機種によっては例外が発生することがあります。

次の図に、接続の始めから終わりまでの状態遷移を示します。

URLConnection の状態遷移図

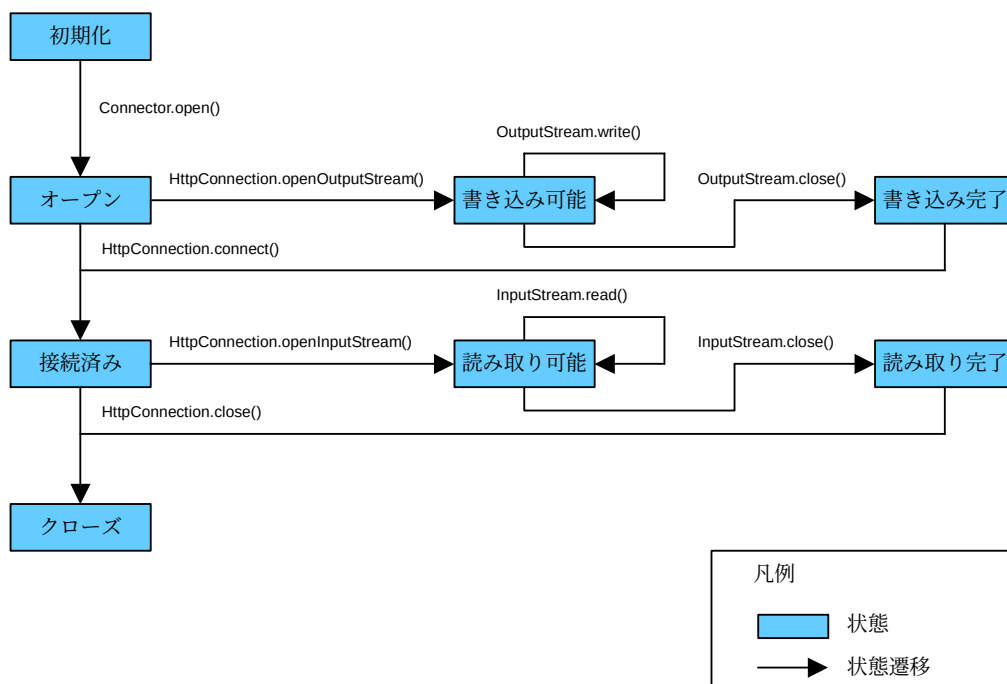


図 8: HttpURLConnection の状態遷移

注意事項：

- HttpURLConnection に渡す URL には、次の制約があります。
 - IP アドレスはシンボリックでなければなりません("www.nttdocomo.co.jp"など)。URLConnection では、数値 IP アドレス("127.0.0.1"など)は使用できません。
 - HttpURLConnection に渡す URL のプロトコル、ホスト名、ポートは、アプリケーションのダウンロード元となった URL (ADF の PackageURL での指定) と同じでなければなりません。
- 1 つのアプリケーションから同時に開設可能な接続の数は 1 つだけです。
- Java 対応 i モードのクライアントサーバーシステムでは、クライアント (携帯電話) とサーバー (Web サーバー) の間に i モードサーバーが介在しています。i モードサーバーでは、以下のような異常検出時にレスポンスステータスを 200 としてメッセージを返すことがあります。このような場合、クライアント側ではレスポンスステータスに基づいた正常性確認を行うことができません。クライアント側で厳密な正常性確認を行うには、アプリケーション独自の HTTP レスポンスヘッダの付加や HTTP レスポンスボディの内容による判断を行うようにしてください。
 - i モードサーバーの対コンテンツプロバイダ向けプロキシサーバー輻輳時
 - 公式メニューサイトのメンテナンス時 (公式メニューサイトのコンテンツプロバイダより申請のあった場合)

なお、異常を示すレスポンスステータスが返却された場合、i モード Java アプリケーションは例外をスローします。
- HTTP リクエスト/レスポンスボディについて x-www-form-urlencoded 形式のエンコード、デコードを行うには、それぞれ以下のユーティリティクラスを使用してください。
 - com.nttdocomo.net.URLEncoder

- `com.nttdocomo.net.URLDecoder`
- HTTP(S)通信で送受信可能なメッセージのサイズには、以下の制限があります。
 - リクエストにおける送信サイズ： 5K バイト
 - レスポンスにおける受信サイズ： 10K バイト
- アプリケーションがアクセスする URL には、サーバー側で基本認証（BASIC 認証）の設定が行われている場合があります。このような URL にアクセスした場合、携帯電話は自動的にユーザーID とパスワードの入力を促すダイアログを表示します。アプリケーションが独自の処理で基本認証の情報（Authorization リクエストヘッダなど）をサーバーに送信することはできません。
- アプリケーションが通信を行っている最中に通話着信があった場合、その動作は携帯電話の i モード着信設定に従います。
- Web サーバーに POST メソッドで HTTP リクエストを送信する場合、HTTP リクエストボディの設定内容に適したコンテンツタイプを指定するようにしてください。コンテンツタイプ（Content-Type ヘッダ）の設定には `HttpConnection.setRequestProperty()` メソッドを使用します。また、リクエストボディに書き込むデータが存在しない場合、POST メソッドではなく GET メソッドを使用します。GET メソッド使用時はコンテンツタイプの設定は行わないようにしてください。
- i モードサービスでは、Web サーバーからのレスポンスにはコンテンツ長（Content-Length ヘッダ）の設定を必須としています。i モード Java アプリケーションからの HTTP 通信においても、Web サーバーからのレスポンスにはコンテンツ長の設定を行うようにしてください。

5.1.1 接続の終了

サーバーとクライアント間の接続は、`HttpConnection` インタフェースの `close()` メソッドを使用してクローズします。ただし、このメソッドが実際の物理的な資源を解放するのは、`HttpConnection` から取得されたすべての入出力ストリームがクローズされた後になります。

注意事項：

- `Connector.open()` を呼び出して取り出すすべての接続（`Connection` インタフェースの実装オブジェクト）と、そこから取得されたすべての入出力ストリームは明示的にクローズする必要があります。クローズしない場合、資源のリークが発生して新しい接続をオープンすることができなくなる場合がありますので注意してください。

5.2 セッション管理

HTTP プロトコルでは、永続的な接続は維持されません。接続は、サーバーとデータを交換する必要があるときだけ確立されます。許可されたユーザーのみに使用を許可するサーバーではユーザーの検証が必要であり、クライアントが繰り返しサーバーとデータを交換するためには、ユーザーが継続してシステムにログインした状態であることが必要になります。アプリケーション開発者は、通常、接続が行われるたびにアプリケーションがサーバーへログインすることは望みません。この問題を解決するための手段がセッション管理と呼ばれている手法です。セッション管理の手法では、ユーザーがシステムにログインした際にセッションを一意的に識別するための文字列（セッション ID）を作成します。そしてそれ以降は、サーバーとクライアントの間でセッション ID をやり取りして、個々の HTTP 接続の持ち主（ログインユーザー）を識別します。セキュリティを保

つために、一定時間アクセスのなかったセッション ID は無効化されます。無効化されたセッション ID の持ち主は、再度システムを利用する際にログインする必要があります。

一般的には、セッション管理の具体的な実装例としてクッキーが広く利用されています。クッキーは HTTP リクエスト/レスポンスヘッダの特定のヘッダにユーザー情報（この場合セッション ID）を設定してクライアントとサーバーの間でセッションを保ちます。しかし i モード Java アプリケーション環境では HTTP リクエストヘッダに設定できる情報は content-type と if-modified-since の 2 つに制限されており、クッキーを使用することはできません。i モード Java アプリケーション環境ではクッキーを使用する代わりに、リクエストの対象 URL か HTTP のボディ部分にセッション ID を設定するようにしてください。

次の例では、クッキーを使用する代わりに HTTP リクエスト/レスポンスボディを使用してセッション ID のやり取りを行っています。

例: セッション管理の実装例

```
package banking;

import java.io.*;
import javax.microedition.io.*;
import com.nttdocomo.io.HttpConnection;

/**
 * このクラスにはバンクサーバーとの通信処理が実装されている。
 */
public class BankingClient {
    private static final String UNAUTHORIZED =
        "verification of account data failed";

    private String url;
    private String cookie;
    private HttpConnection conn;
    private InputStream in;

    /**
     * このクラスをインスタンス化する。その際、入力された ID とパスワードをチェックする
     * ためにサーバーに初期接続を行う。
     * @param url バンクサーバーへの URL
     * @param acct ユーザーのアカウント名
     * @param pin ユーザーのアカウントパスワード
     * @exception サーバーに接続できないか入力された ID とパスワードが拒否された場合
     * に IOException がスローされる。
     */
    public BankingClient(String url, String acct, String pin)
        throws IOException {
        this.url = url;
        conn = (HttpConnection)Connector.open(url,
            Connector.READ_WRITE, true);
        conn.setRequestMethod(HttpConnection.POST);
        OutputStream out = conn.openOutputStream();

        // HTTP リクエストボディにアカウント情報を設定する。
        out.write((VERIFY + " " + acct + " " + pin).getBytes());
        out.close();

        conn.connect();
    }
}
```

```

        in = conn.openInputStream();
        if (getCookie(readLine(in)) == null) {
            throw new IllegalArgumentException(UNAUTHORIZED);
        }
    }

    private String getCookie(String line) {
        String header = "Cookie: ";
        // HTTP レスポンスボディから読み込んだ行よりセッション ID を取得する。
        if (!line.startsWith(header) || line.length() <= header.length())
            return null;
        return line.substring(header.length());
    }

    private String contactServer(String cmd) {
        try {
            // 古い接続が残っていたらそれをクローズする。
            if (in != null) in.close();
            if (conn != null) conn.close();

            // サーバーとの接続を再度オープンする。その際に、この
            // インスタンスのコンストラクタによる初期接続で得られた
            // セッション ID を HTTP リクエストボディに設定する。
            conn = (HttpURLConnection)Connector.open(url,
                Connector.READ_WRITE, true);
            conn.setRequestMethod(HttpURLConnection.POST);
            OutputStream out = conn.getOutputStream();
            out.write((cmd + "¥n").getBytes());
            out.write(("Cookie: " + cookie + "¥n").getBytes());
            out.close();

            conn.connect();

            in = conn.openInputStream();
            if (in == null) {
                throw new IOException(UNAUTHORIZED);
            }

            return readLine(in);

            // 以降の処理でこの接続からさらにデータの読み込みが行われるので、
            // ここでは入力ストリームと接続のクローズは行わない。
            // 接続のクローズは、このメソッドが次に呼び出されるときか
            // このクラスの close() メソッドを呼び出すことによって行われる。
        } catch (IOException ioe) {
            return null;
        }
    }
}

```

5.3 HTTPS による安全な通信

安全な接続が必要なクライアント/サーバーアプリケーションでは、HTTPS プロトコルを使用します。HTTPS サーバーに接続するクライアントプログラムのコードは、URL 文字列のプロトコルセクションに”http”の代りに”https”と指定する以外、HTTP プロトコルの場合と全く同じです。たとえば、クライアントとサーバーの間でスペースインベータのハイスコアと名前を安全に送受信する場合、必要な変更は URL 文字列だけです。つ

まり、URL”http://backflip.sfbay:8080/scores”
を”https://backflip.sfbay:7070/scores”で置き換えます。

注意事項：

- `HttpConnection` に渡される URL はアプリケーションが最初にダウンロードされたときの URL と同じプロトコル、ホスト、ポート番号が指定されていなければなりません。したがって、HTTPS を介してダウンロードされたアプリケーションでなければ HTTPS プロトコルを使用することはできません。これは、HTTPS を介してダウンロードされたアプリケーションは、HTTP プロトコルは使用できないということでもあります。
- HTTPS 通信で利用する Web サーバーには、Java 対応 i モードが対応している認証局によって署名されたサーバー証明書をインストールしておく必要があります。Java 対応 i モードでは、以下のルート認証局の証明書に対応しています。
 - 1) Verisign 社
 - Verisign クラス 3 Primary CA ルート証明書
 - Verisign クラス 3 Primary CA ルート証明書 G2
 - RSA セキュアサーバ・ルート証明書
 - 2) Baltimore 社
 - GTE CyberTrust Root CA
 - GTE CyberTrust Global Root

第 6 章

テキスト処理

Java 対応 i モード向けに開発するアプリケーションでは、テキスト処理に必要な応じて日本語文字セットと ASCII 文字セットを使用します。i モード Java アプリケーション環境は、日本語テキスト（SJIS）の処理に適した設計になっています。

6.1 テキスト処理

Java Virtual Machine では、すべてのテキスト処理は Unicode で行われます。このため、Unicode 以外の形式で符号化されたテキストデータを Java の文字ストリームとして使用するには、これを Unicode に変換する必要があります。i モードサービスでは、ネットワーク上のテキストデータは Shift-JIS (SJIS) で符号化されることが規定されています。そのため、Unicode 文字ストリームとそれ以外のテキストのバイトストリームとの間の文字コード変換を行うクラスが java.io パッケージで提供されています。

InputStreamReader クラスはバイトストリームを文字ストリームに、OutputStreamWriter クラスは文字ストリームをバイトストリームにそれぞれ変換します。図 9 に、この変換プロセスを示します。

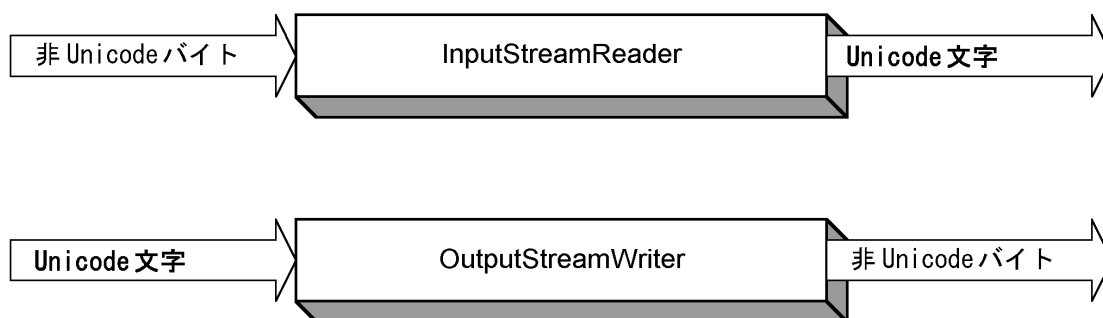


図 9: 入出力ストリームの変換

アプリケーションでは、InputStreamReader や OutputStreamWriter クラスを使ってネットワークから入出力するテキストを適切に変換する必要があります。次に、これらの変換方法を使用するコード例を示します。

例: SJIS を Unicode に変換

```
java.io.*;
import com.nttdocomo.io.*;
import javax.microedition.io.*;
```

```
conn = (HttpURLConnection)Connector.open(url, Connector.READ_WRITE,
true);
```

// SJIS 文字コードを Unicode に変換する。

```
isr = new InputStreamReader(conn.openInputStream()); //デフォルトの符号化は SJIS
```

例: Unicode を SJIS に変換

```
java.io.*;
import com.nttdocomo.io.*;
import javax.microedition.io.*;
```

```
conn = (HttpURLConnection)Connector.open(url, Connector.READ_WRITE,
true);
```

// Unicode 文字コードを SJIS に変換する。

```
osw = new OutputStreamWriter(conn.openOutputStream()); //デフォルトの符号化は SJIS
```

6.2 Java 対応 i モードのグラフィックス文字

Java 対応 i モードでは、従来の i モードコンテンツでもサポートされている絵文字（i モード用外字セット）を利用することができます。以下に NTT ドコモが定義した、各携帯電話に搭載されている絵文字のコード表（SJIS および Unicode の対応表）を示します。各コードの値は 16 進数の数値です。

なお、ここに記載したフォントは、実際の携帯電話に搭載されたフォントとは若干異なりますのでご注意ください。一般的に、フォントを構成するドット数の制約により、実際の携帯電話に搭載されるフォントはここに記載されたものより粗く表示されます。

文字	SJIS	Unicode	タイトル	文字	SJIS	Unicode	タイトル
	F89F	E63E	晴れ		F8C7	E666	病院
	F8A0	E63F	曇り		F8C8	E667	銀行
	F8A1	E640	雨		F8C9	E668	ATM
	F8A2	E641	雪		F8CA	E669	ホテル
	F8A3	E642	雷		F8CB	E66A	コンビニ
	F8A4	E643	台風		F8CC	E66B	ガソリンスタンド
	F8A5	E644	霧		F8CD	E66C	駐車場
	F8A6	E645	小雨		F8CE	E66D	信号
	F8A7	E646	牡羊座		F8CF	E66E	トイレ
	F8A8	E647	牡牛座		F8D0	E66F	レストラン
	F8A9	E648	双子座		F8D1	E670	喫茶店
	F8AA	E649	蟹座		F8D2	E671	バー
	F8AB	E64A	獅子座		F8D3	E672	ビール
	F8AC	E64B	乙女座		F8D4	E673	ファーストフード
	F8AD	E64C	天秤座		F8D5	E674	ブティック
	F8AE	E64D	蠍座		F8D6	E675	美容院
	F8AF	E64E	射手座		F8D7	E676	カラオケ
	F8B0	E64F	山羊座		F8D8	E677	映画
	F8B1	E650	水瓶座		F8D9	E678	右斜め上
	F8B2	E651	魚座		F8DA	E679	遊園地
	F8B3	E652	スポーツ		F8DB	E67A	音楽
	F8B4	E653	野球		F8DC	E67B	アート
	F8B5	E654	ゴルフ		F8DD	E67C	演劇
	F8B6	E655	テニス		F8DE	E67D	イベント
	F8B7	E656	サッカー		F8DF	E67E	チケット
	F8B8	E657	スキー		F8E0	E67F	喫煙
	F8B9	E658	バスケットボール		F8E1	E680	禁煙
	F8BA	E659	モータースポーツ		F8E2	E681	カメラ
	F8BB	E65A	ポケットベル		F8E3	E682	カバン
	F8BC	E65B	電車		F8E4	E683	本
	F8BD	E65C	地下鉄		F8E5	E684	リボン
	F8BE	E65D	新幹線		F8E6	E685	プレゼント
	F8BF	E65E	車（セダン）		F8E7	E686	バースデー
	F8C0	E65F	車（RV）		F8E8	E687	電話
	F8C1	E660	バス		F8E9	E688	携帯電話
	F8C2	E661	船		F8EA	E689	メモ
	F8C3	E662	飛行機		F8EB	E68A	TV
	F8C4	E663	家		F8EC	E68B	ゲーム
	F8C5	E664	ビル		F8ED	E68C	CD
	F8C6	E665	郵便局		F8EE	E68D	ハート

文字	SJIS	Unicode	タイトル	文字	SJIS	Unicode	タイトル
	F8EF	E68E	スペード		F986	E6E1	モバQ
	F8F0	E68F	ダイヤ		F987	E6E2	1
	F8F1	E690	クラブ		F988	E6E3	2
	F8F2	E691	目		F989	E6E4	3
	F8F3	E692	耳		F98A	E6E5	4
	F8F4	E693	手 (グー)		F98B	E6E6	5
	F8F5	E694	手 (チョキ)		F98C	E6E7	6
	F8F6	E695	手 (パー)		F98D	E6E8	7
	F8F7	E696	右斜め下		F98E	E6E9	8
	F8F8	E697	左斜め上		F98F	E6EA	9
	F8F9	E698	足		F990	E6EB	0
	F8FA	E699	くっ		F991	E6EC	黒ハート
	F8FB	E69A	眼鏡		F992	E6ED	揺れるハート
	F8FC	E69B	車椅子		F993	E6EE	失恋
	F940	E69C	新月		F994	E6EF	ハートたち (複数ハート)
	F941	E69D	やや欠け月		F995	E6F0	わーい (嬉しい顔)
	F942	E69E	半月		F996	E6F1	ちっ (怒った顔)
	F943	E69F	三日月		F997	E6F2	がく～ (落胆した顔)
	F944	E6A0	満月		F998	E6F3	もうやだ～ (悲しい顔)
	F945	E6A1	犬		F999	E6F4	ふらふら
	F946	E6A2	猫		F99A	E6F5	グッド (上向き矢印)
	F947	E6A3	リゾート		F99B	E6F6	るんるん
	F948	E6A4	クリスマス		F99C	E6F7	いい気分 (温泉)
	F949	E6A5	左斜め下		F99D	E6F8	かわいい
	F972	E6CE	phone to		F99E	E6F9	キスマーク
	F973	E6CF	mail to		F99F	E6FA	ぴかぴか (新しい)
	F974	E6D0	fax to		F9A0	E6FB	ひらめき
	F975	E6D1	i モード		F9A1	E6FC	むかつ (怒り)
	F976	E6D2	i モード (枠付き)		F9A2	E6FD	パンチ
	F977	E6D3	メール		F9A3	E6FE	爆弾
	F978	E6D4	ドコモ提供		F9A4	E6FF	ムード
	F979	E6D5	ドコモポイント		F9A5	E700	バッド (下向き矢印)
	F97A	E6D6	有料		F9A6	E701	眠い (睡眠)
	F97B	E6D7	無料		F9A7	E702	exclamation
	F97C	E6D8	ID		F9A8	E703	exclamation&question
	F97D	E6D9	パスワード		F9A9	E704	exclamation×2
	F97E	E6DA	次項有		F9AA	E705	どんっ (衝撃)
	F980	E6DB	クリア		F9AB	E706	あせあせ (飛び散る汗)
	F981	E6DC	サーチ (調べる)		F9AC	E707	たらーっ (汗)
	F982	E6DD	NEW		F9AD	E708	ダッシュ (走り出すさま)
	F983	D6DE	位置情報		F9AE	E709	ー (長音記号 1)
	F984	D6DF	フリーダイヤル		F9AF	E70A	ー (長音記号 2)
	F985	E6E0	シャープダイヤル		F9B0	E70B	決定

第 7 章

ScratchPad とリソースの使用

この章では、Java 対応 i モードの携帯電話上でアクセス可能な ScratchPad およびリソースの使用方法について解説します。

ScratchPad とは、携帯電話上にアプリケーションで使用する永続データを保存するための記憶領域です。またリソースとは、アプリケーションの JAR ファイルに同梱されたデータファイル（画像ファイルや音声ファイルなど）のことです。ScratchPad もリソースも、第 5 章で解説した HTTP 通信と同様に、Generic Connection フレームワークを使用してアクセスを行います。

7.1 ScratchPad に対する読み取りと書き込み

Java 対応 i モードの Java アプリケーションでは、永続データ（機密データや長期間の保持が必要なデータなど）の主要な格納場所としてサーバーもしくは ScratchPad を使用します。i モード Java アプリケーション環境では、オフライン時（電波の届かない状況など）のデータ格納やその他の便利な用途のために、ScratchPad と呼ばれる特別な記憶モデルが個々のアプリケーションに与えられます。アプリケーションが使用可能な ScratchPad サイズには、3.2 項で解説した制限があります。さらに、あるアプリケーションに割り当てられた ScratchPad にアクセスできるのは、それを作成したアプリケーションだけです。

アプリケーションで ScratchPad を使用する場合は、ADF の SPsize キーを使って、アプリケーションの使用する ScratchPad サイズを指定する必要があります。JAM は、指定された ScratchPad サイズとプログラムサイズの両面から、携帯電話にアプリケーションをダウンロードできるスペースがあるかどうかを判定します。ADF の記述方法については、9.5.1 項を参照してください。

ScratchPad の記憶モデルには、トランザクションの概念の一部が導入されています。ScratchPad に対する変更は、必ず全部が書き込まれるか、全部が拒否されます。

JAM は、ADF に ScratchPad サイズが指定されているアプリケーションごとに記憶領域を作成し、それを管理します。JAM が記憶装置にデータを実際に書き込むタイミングは、メーカーのメモリやシステムモデルの最適化方法によって異なります。

ScratchPad へのアクセスは Generic Connection フレームワークを使用して行います。ScratchPad の URL は"scratchpad:///0"固定で、これに対して pos オプションを付加することができます。pos オプションに指定する値は、アプリケーションに割り当てられた ScratchPad 領域の先頭からのバイト数を表します。pos オプション指定時は、ここに指定されたオフセットの位置からデータの読み取りまたは書き込みが行えます。

`Connector.open()`に対して ScratchPad の URL を指定すると、このメソッドは `StreamConnection` の実装インスタンスを返します。アプリケーションは、この `StreamConnection` インスタンスから入出力ストリームを取り出して ScratchPad へのアクセスを行います。

次の 2 つの例は、ScratchPad への書き込みと ScratchPad からの読み取りをどのように行うかを示したものです。

例: ScratchPad 領域への書き込み

```
import java.io.*;
import javax.microedition.io.*;

public void save() {
    try {
        OutputStream out = Connector.openOutputStream("scratchpad:///0")
        out.write((hi_score >> 24) & 0xff);
        out.write((hi_score >> 16) & 0xff);
        out.write((hi_score >> 8) & 0xff);
        out.write(hi_score & 0xff);
        out.close();
    } catch (IOException e) {
        // ScratchPad 入出力の例外処理
    }
}
```

例: ScratchPad 領域の読み取り

```
import java.io.*;
import javax.microedition.io.*;

public void load() {
    try {
        InputStream in= Connector.openInputStream("scratchpad:///0")
        hi_score = in.read() << 24;
        hi_score |= (in.read() << 16);
        hi_score |= (in.read() << 8);
        hi_score |= in.read();
        in.close();
    } catch (IOException e) {
        // ScratchPad 入出力の例外処理
    }
}
```

例: ScratchPad 領域の読み取り

```
// ScratchPad の 4 番目のバイトから始まる UTF(つまり UniCode)文字列を
// 取り出す。最初の 4 バイトは他の用途（前の例では
// 4 バイトの数値）で使用されている。
package spaceinv;

import java.io.*;
import javax.microedition.io.*;

public class UserNameScreen extends Panel implements TimerListener {
    private String name;
```

```

/** このクラスの値インスタンスを作成する。 */
public UserNameScreen() {
    try {
        DataInputStream in = Connector.openDataInputStream(
            "scratchpad:///0;pos=4");
        name = in.readUTF();
        in.close();
    } catch (Exception x) {
        // ScratchPad 入出力の例外処理
    }
    ...
}
...
}

```

注意事項：

- ScratchPad への `StreamConnection` をオープンする場合、オープンモードとして `Connector.READ` または `Connector.WRITE` モードが指定できます。`Connector.READ_WRITE` モードでのオープンの可否はメーカーにより異なりますので注意してください。

7.2 リソースの読み取り

JAR ファイルに同梱される画像ファイルや音声ファイルなどのデータファイルをリソースと呼びます。リソースに対しても Generic Connection フレームワークの枠組みでアクセスすることができます。リソースは、読み取り専用でのみオープン可能です。リソースにアクセスするには、`Connector.open()` メソッドのパラメータで `READ` モードであることを明示的に指定するか、`Connection` オブジェクトを得ることなく入力ストリームを得るための簡易 API である `Connector.openInputStream()` を使用しないと例外がスローされます。static メソッド `Connector.open()` には、3 つのパラメータがあります。

- 1) 接続先の URL
- 2) オープンモード
- 3) タイムアウトが予期されるか否かを示すブール値 (true または false)

リソースにアクセスする場合、最初のパラメータは `"resource:/// <jar-relative-path>"` という形式の URL 文字列です。<jar-relative-path>の部分には、JAR ファイルに格納されたリソースの相対パスを指定します。2 つめのパラメータには、`Connector.READ` を指定します (リソースは読み取り専用であるため)。3 つめのパラメータは、HTTP(S)接続をオープンする場合を除き、通常 `false` を指定します。短縮形式の `open()` メソッド (パラメータが URL だけのもの) はオープンモードが `Connector.READ_WRITE` になりますので、リソースアクセス時には使用しないでください。

リソースの URL は、メディアデータにアクセスする際の URL 文字列にも指定することができます。これにより、JAR ファイルに同梱された GIF ファイルや音声ファイルを、アプリケーションから容易に再生することができます。

次に、メディアデータとしてリソースを使用する例を示します。この例では、JAR ファイルに同梱された bank.gif というファイルを取り出して、VisualPresenter に設定する処理を行っています。

例: リソースの利用

```
package banking;

import com.nttdocomo.io.ConnectionException;
import com.nttdocomo.ui.*;

...
/** welcome 画面を表示する。 */
class WelcomePanel extends Panel {

    /** このクラスの新しいインスタンスを作成する。 */
    public WelcomePanel() {
        final Panel instance = this;
        MediaImage bank;

        VisualPresenter vp1 = new VisualPresenter();
        try {

            bank = MediaManager.getImage(
                "resource:///BankingDemo/img/bank.gif");
            bank.use();
        } catch (ConnectionException ce) {
            // リソースアクセス時の例外処理
        }
        vp1.setImage(bank);
        ...
    }
}
```

7.3 エラー処理

ScratchPad またはリソースの取得先を指定した URL を使用するアプリケーションでは、次の例外を処理する必要があります。

ConnectionNotFoundException

URL が無効であったときにスローされます。無効とは、指定された URL の形式を i モード Java アプリケーション環境が解釈できない、または指定された URL の種別とオープンモードが矛盾していることを指します。

ConnectionException

ステータス: **SCRATCHPAD_OVERSIZE**

ScratchPad のサイズを越えて書き込みを行おうとした場合にスローされます。

ステータス: **ILLEGAL_STATE**

接続オブジェクトの状態に反したメソッド呼び出しが行われた場合にスローされます。接続オブジェクトの状態に反したメソッド呼び出しとは、例えば **READ** モードで作成した接続オブジェクトに対して `openOutputStream()` メソッドを呼び出すようなことを指します。

ステータス： **RESOURCE_BUSY**

許された数以上のスレッドが同時に ScratchPad やリソースをオープンしようとしたときにスローされます。

EOFException

ScratchPad やリソースの終りを越えて読み取りを行おうとした場合にスローされます。

その他の例外条件が発生すると、**IOException** または **IOException** のサブクラスがスローされます。

第 8 章

ハードウェアのアクセス

Java アプリケーションから携帯電話のネイティブ資源を制御するために、`com.nttdocomo.ui` パッケージに `PhoneSystem` クラスが定義されています。`PhoneSystem` クラスには、ハードウェア・リソースの属性値を設定する `setAttribute()` メソッドがあります。ハードウェア・リソースを制御するには、ハードウェア・リソースタイプとそれに対応する属性値を指定します。i モード Java アプリケーション環境で規定されているハードウェア・リソースタイプは画面のバックライト（オンまたはオフに切り替え可能）だけです。

バイブレータなどのハードウェア機能をアプリケーションからオンやオフにできるようにするため、ハードウェアにアクセスする拡張 API をメーカーが提供する場合があります。メーカーの拡張 API については別途公開されます。

注意事項：

- メーカーの拡張 API を使用するアプリケーションは、そのメーカーの携帯電話でしか動作しないため注意が必要です。

第 9 章

アプリケーションの ビルド、テスト、およびパッケージ化

この章では、Java 対応 i モード向け Java アプリケーションのアプリケーション開発サイクルについて概要を説明します。NTT ドコモでは、PC 上で Java アプリケーションを開発するためのアプリケーション開発環境（i モード対応 Java API 向け Java 2, Micro Edition, Wireless SDK）を提供していますが、ここでは、開発環境および付属のツールの個々の詳細な情報には触れません。開発環境の詳細については、i モード対応 Java API 向け Java 2, Micro Edition, Wireless SDK ユーザーズガイドを参照してください。

アプリケーション開発者は、Java 2 Standard Edition ソフトウェア開発キット (J2SE SDK バージョン 1.3) を利用したことがあり、SDK に含まれるツールやドキュメントの知識があるものとしします。

なお、開発したアプリケーションを、テストのために実際の携帯電話にインストールするには、アプリケーションを配布するための Web サイト（インターネットに接続されている必要があります）を設定する必要があります。アプリケーションの配布については、第 10 章を参照してください。

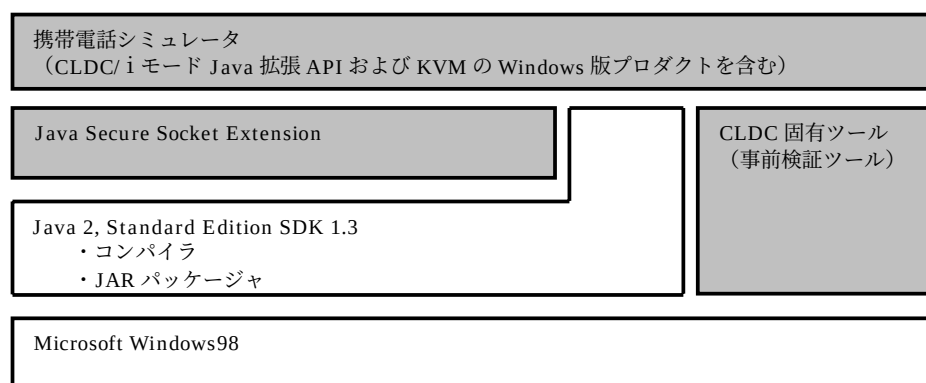
9.1 アプリケーション開発環境のインストール

アプリケーション開発環境は、自己解凍型の J2MEWSDK4DOJA 実行可能ファイルを使って設定します。このファイルを実行するとインストーラが起動され、インストールディレクトリパスの指定など、いくつかのステップが開発者に提示されます。この後、必要なすべてのファイルがインストーラによって展開され、インストールディレクトリに配置されます。PATH 変数など、必要な環境変数はすべて自動的に設定されます。さらに、JDK1.3 のパスがすべてのサンプルバッチファイルに自動的に設定されます。

9.2 アプリケーション開発環境のコンポーネント

アプリケーション開発環境は、PC 上で i モード Java アプリケーションを開発するための簡易な開発環境です。これを使用することで、PC 上で i モード Java アプリケーションをビルド、シミュレート実行することができます。アプリケーション開発環境は Java 対応 i モード開発者キットに含まれています。

アプリケーション開発環境は以下のようなコンポーネント構成を取っています。



(*1)ハッチングされたコンポーネントがアプリケーション開発環境に含まれています。

図 10: アプリケーション開発環境の構成

アプリケーション開発環境をインストールするには、Java 対応 i モード開発者キットのほかに Java 2, Standard Edition (J2SE) SDK 1.3 を用意する必要があります。J2SE SDK 1.3 をインストールした上でアプリケーション開発環境をインストールしてください。

以下にアプリケーション開発環境に含まれる各コンポーネントの概要を示します。

- 携帯電話シミュレータ
アプリケーション開発環境のメインコンポーネントです。開発者は携帯電話シミュレータを使用してアプリケーションのビルド（コンパイル、事前検証、JAR パッケージング）とテスト実行を行うことができます。なおアプリケーションのビルドは、各ビルド用コマンドをコマンドラインから呼び出して行うこともできます。
本書では、携帯電話シミュレータをエミュレータと表記することがあります。
- CLDC 固有ツール
CLDC 固有ツールには事前検証ツール（preverify）があります。事前検証とは、標準的な Java 仮想マシンで行われるバイトコード検証作業の一部を事前（アプリケーションビルド時）に済ませておくことを言います。CLDC では事前検証方式を採用することにより、アプリケーション実行時の検証作業実行コストを削減しています。i モード Java アプリケーション環境では、事前検証を行っていないクラスを使用することはできません。
事前検証ツールは携帯電話シミュレータのアプリケーションビルド機能からだけでなく、コマンドラインから単独のツールとして使用することもできます。
- Java Secure Socket Extension
i モード Java アプリケーション環境では HTTPS による通信をサポートしています。携帯電話シミュレータでは、HTTPS の SSL 処理に Java Secure Socket Extension（JSSE）を使用します。
JSSE はアプリケーション開発環境とともに自動的にインストールされます。

9.3 開発サイクルの概要

図 11 に、一般的な開発サイクルを示します。

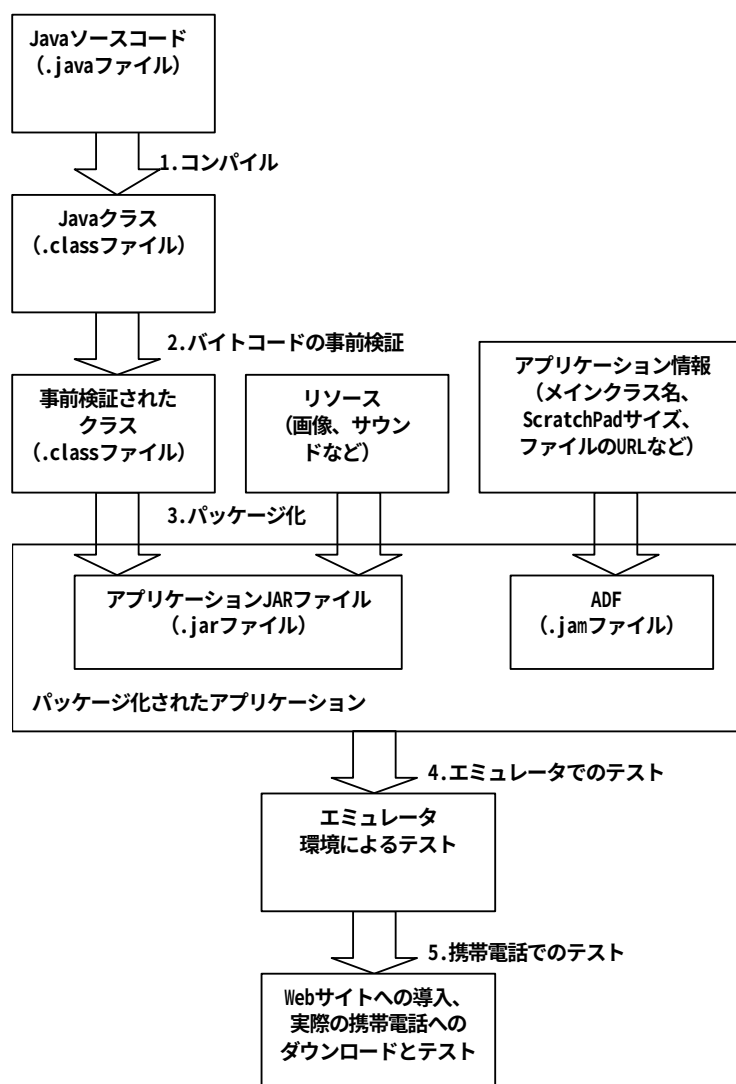


図 11. 開発サイクル

コンパイル – ソースファイルをクラスファイルにコンパイルするには、Java 2 Standard Edition の Java コンパイラ (javac) を使用します。このときコンパイラは、i モード Java アプリケーション環境に属していないパッケージやクラス、メソッドがアプリケーションで使用されていないか検査します。コマンドラインから i モード Java アプリケーションをコンパイルする方法については、i モード対応 Java API 向け Java 2 Micro Edition Wireless SDK ユーザーズガイドを参照してください。

バイトコードの事前検証 – エミュレータのバイトコード事前検証ツール (preverify) では、Java クラスに対する事前検証プロセスを行います。事前検証ツールを実行すると追加の属性情報がクラスファイルに書き込まれるため、CLDC 仮想マシンは、実行時に Java 対応 i モードでバイトコードの検査を効率的に行うことができます。さらに、事前検証ツールでは、Java アプリケーションで浮動小数点演算やファイナライズメソッドが使用されていないか検査します。コマンドラインからの事前検証ツールの実行方法については、i モード対応 Java API 向け Java 2 Micro Edition Wireless SDK ユーザーズガイドを参照してください。

パッケージ化 – 事前検証済みのすべてのクラスファイルとリソースファイルをJARファイルとしてパッケージ化するには、Java 2 Standard Edition のアーカイブツール (jar) を使用します。jar コマンドの使用方法は、J2SE における jar コマンドの標準的な使用方法と同じです。携帯電話に保存されるアプリケーションサイズを最小限に保つため、非圧縮オプションは指定しないようにしてください。

エミュレータでのテスト – アプリケーションの機能が正しく動作するかどうかをエミュレータ環境で確認します。

実際の Java 対応 i モードでのテスト – アプリケーション (JAR ファイル、ADF) およびダウンロードナビゲート用 HTML コンテンツを Web サイトに搭載し、実際の携帯電話にダウンロードします。そして、携帯電話でアプリケーションを実行して機能が正しく動作するか確認します。使用可能なメモリ容量など、携帯電話によってハードウェア・リソースの容量が異なる場合があるので、ターゲットとする全てのメーカーの携帯電話でアプリケーションの動作確認を行うようにしてください。

コンパイル、事前検証、パッケージ化の一連の流れ (ビルド作業) は個々のツールを独立して使用する以外にも、エミュレータの持つアプリケーションビルド機能を使用して一括で行うこともできます。エミュレータのアプリケーションビルド機能については、i モード対応 Java API 向け Java 2 Micro Edition Wireless SDK ユーザーズガイドを参照してください。

9.4 サーバー側アプリケーションの作成

クライアントサーバー型アプリケーションでは、Java 対応 i モードの通信相手となるサーバー側アプリケーションを作成する必要があります。サーバー側アプリケーションは、クライアントコンポーネント (Java 対応 i モード上のアプリケーション) と HTTP プロトコルで通信するものであればどのような技術を使用したものでも構いません。

エミュレータのサンプルアプリケーションでは、サーバー側アプリケーションの例として Java サーブレットを使用しているものがあります。Java サーブレットとは、モジュール形式のサーバー側 Java アプリケーションを開発するための技術です (詳細は、Java サーブレットの Web サイト <http://www.java.sun.com/products/servlet/index.html> を参照)。しかしアプリケーション開発者は、サーバーコンポーネントに必ず Java サーブレットを使用しなければならないわけではありません。

9.5 アプリケーションのパッケージ化

図 11 に示すように、i モード Java アプリケーションは、ADF と JAR ファイル (Java アプリケーションのクラスおよびリソースファイルを含んでいる) の 2 つのファイルから構成されます。

9.5.1 ADF の作成

ADF は、SJIS コードで記述されたテキストファイルです。このファイルは、"="で分離されたキーと値のペアからなる行で構成されます。ファイルの編集には、任意のテキストエディターを使用することができます。このファイルには、次のキーのエントリ (行) を指定します。

AppName: 必須キー

アプリケーション名(最大 16 バイト)。携帯電話でアプリケーション一覧を表示させた際に、ここに指定された名前が表示されます。SJIS で日本語名を指定することができます。

AppVer: オptionalキー

アプリケーションのバージョン。アプリケーション開発者はこのキーを利用してアプリケーションのバージョン管理を行うことができます。ASCII 文字のみ指定可能です。

PackageURL: 必須キー

アプリケーション (JAR ファイル) の URL(最大 255 バイト)。相対パスを指定した場合、そのベースディレクトリは ADF の位置となります。ASCII 文字のみ指定可能です。

AppSize: 必須キー

JAR ファイルのサイズ。バイト単位の整数値を指定します。アプリケーションサイズの制限のため、この値は 10240 以下となります。

AppClass: 必須キー

アプリケーションの起動に使われるメインクラス名 (フルパッケージ名付き)。ASCII 文字のみ指定可能です。

AppParam: Optionalキー

メインクラスの起動パラメータ。パラメータ間はスペースで区切ります(最長 255 バイト)。これらのパラメータを Java アプリケーションから取得するには、`com.nttdocomo.ui.IApplication.getArgs()` メソッドを使用します。ASCII 文字のみ指定可能です。省略時は起動パラメータがないものとします。

KvmVer: オptionalキー

KVM のバージョン。このキーが指定されていない場合は、アプリケーションが CLDC 仮想マシンのすべてのバージョンで動作することを示します。指定する場合は以下のフォーマットに従います。

CLDC-x.x (x は 0~9 の数字。本バージョンでは 1.0 をサポート)

SPsize: オptionalキー

ScratchPad のサイズ。バイト単位の整数値を指定します。アプリケーションが使用可能な ScratchPad サイズには、3.2 項で解説した制限があります。省略時は ScratchPad を使用しないものとします。

LastModified: 必須キー

アプリケーションの最終変更日時(JST)。アプリケーションのバージョンアップは、このキーの内容に基づいて行われます。フォーマットは以下の通りです。

Dow, DD Mon YYYY HH:MM:SS

(Dow: 曜日 DD: 日 Mon: 月 YYYY: 年 HH: 時間 MM: 分 SS: 秒)

指定例: Fri, 25 Feb 2000 12:04:25

UseNetwork: オptionalキー

ネットワーク機能を使用する場合に、通信方式を指定します。i モード Java アプリケーション環境では "http" が唯一の有効な値です (HTTPS 使用時も)。この指定がないアプリケーションは、ネットワーク接続を要求することはできません。

TargetDevice: オptionalキー

ターゲットとする携帯電話の機種名をコンマ区切りで指定します。すべての機種をターゲットとする場合はエントリ自体を指定しません。有効な機種名は、随時 NTT ドコモより公開されます。

LaunchAt: オptionalキー

アプリケーションに自動起動を行わせる場合、その起動タイミングを指定します。先頭の 1 文字は形式指示子です。i モード Java アプリケーション環境では、形式指示子としてインターバルを表す "I" をサポートします。

◇ 時間間隔指定形式（形式指示子：I）

フォーマット： I h

（h： 時間 有効桁数は 2 桁）

指定例： I 6 （毎日 6 時間毎に起動する）

時間間隔の起点は、アプリケーションが携帯電話にダウンロードされた時刻となります。

アプリケーションが自動起動するタイミングで他のアプリケーションが実行されていたり、通話中であつたりした場合、そのアプリケーションは自動起動できません。その場合、携帯電話は各機種固有の方法で、自動起動できなかったアプリケーションがあることを利用者に通知します。

以下に ADF の記述例を示します。ADF は下記に示すとおり、1 つのキーエントリを 1 行で記述しなければなりません。

```
AppName = バンキングデモ
KvmVer = CLDC-1.0
AppClass = banking.BankingDemo
AppVer = 1.0
PackageURL = http://aserver.com:8080/BankingDemo.jar
AppSize = 17556
LastModified = Fri, 25 Feb 2000 12:04:25
```

9.5.2 JAR ファイルの作成

JAR は、アプリケーションを構成するクラスファイルとリソースファイルを圧縮形式で格納するための、標準の Java アーカイブ機能です。JAR ファイルでは、アプリケーションファイルのサイズがおよそ 40% から 50% 削減されます（JAR ファイルに含まれる内容により圧縮率は変動します）。これによって携帯電話に必要なアプリケーション保管容量が減るだけでなく、アプリケーションのダウンロード時間も少なくなります。JAR ファイルにパッケージングされた個々のファイルは、必要に応じて JAM によりアンパックされます。

9.6 アプリケーションのテスト

i モード Java アプリケーションのテストは、エミュレータおよび実際の Java 対応 i モード携帯電話を使って行います。

開発の初期段階では、アプリケーションのテストやデバッグをエミュレータを使って行う方が効率的であり、コスト的にも有利です。エミュレータでは、ユーザーインタフェースや各種入出力、アプリケーションロジックなどのデバッグ作業において、アプリケーションを修正した結果を PC 上ですぐに確認することができます。

しかしエミュレータ上では、プログラムの実行速度やネットワークの帯域幅などに関して、ハードウェアの動作を完全にエミュレートできるわけではありません。また実際のハードウェアには、メモリ容量などの機種固有の制限事項もあります。したがって、実際の Java 対応 i モードで最終的なテストをして、アプリケーションの動作が適切かどうかを確認することは不可欠となります。

なおエミュレータの使用方法の詳細については、i モード対応 Java API 向け Java 2 Micro Edition Wireless SDK ユーザーズガイドを参照してください。

エミュレータ環境には、アプリケーションのテストに関して、さらにいくつかの制約があります。エミュレータでは、Java アプリケーションの実行環境は提供されますが、実際の携帯電話に備わっているアプリケーション管理機能（JAM の持つ機能）はエミュレートされません。たとえば、エミュレータで実行できない 1 つの動作にアプリケーションの更新があります。これは、JAM のアプリケーション管理機能がエミュレータに実装されていないためです。テストできない動作はその他に次のものがあります。

- LastModified、TargetDevice、LaunchAt に関して、実際の動作をエミュレートする機能はありません。
- エミュレータでは、KvmVer キーと CLDC 仮想マシンのバージョン整合性チェックは行われません。
- エミュレータは JAR ファイルを開発コンピュータのハードディスクからロードするため、PackageURL に指定された値は使用されません。実際に Java 対応 i モードにアプリケーションを配布する際には、PackageURL には JAR ファイルをダウンロードするための正しい URL が指定されている必要があります。

実際の携帯電話上でのテストのためにアプリケーションを携帯電話にインストールするには、インターネットに接続された Web サーバーにアプリケーション、ADF、アプリケーションダウンロード用 HTML を搭載する必要があります。これらの詳細については第 10 章を参照してください。

注意事項：

- アプリケーションの **MediaManager** クラスで音声メディアデータを使用している場合は、アプリケーションを実際の携帯電話に配布するフェーズになる前に、エミュレータによる開発とテストで使用している MIDI ファイルを、実際の携帯電話で利用できる i メロディ形式ファイルに置き換える必要があります。場合によっては、Java アプリケーションのソースコード中に指定されたリソースの URL（拡張子）も変更する必要がありますので注意してください。
- アプリケーションがクライアントサーバー構成を取っている場合、テストに利用する Web サーバーには、サーバー側アプリケーションも適切に導入されている必要があります。

第 10 章

アプリケーションの配布

この章では、アプリケーションを配布するために Web サーバーをどのように設定する必要があるかについて説明します。

10.1 アプリケーション配布のための Web サーバーの構成

前の章で述べたように、i モード対応 Java アプリケーションは、ADF(.jam ファイル)と JAR ファイル(.jar ファイル)から構成されています。Web サーバーに置かれた ADF をダウンロードするには、i モード HTML の OBJECT タグでこのファイルが参照されていなければなりません。次に、アプリケーションをダウンロードするための HTML ファイルにおけるタグの定義例を示します。

```
<OBJECT declare id="application.declaration"
data="http://www.nttdocomo.co.jp/java/abc.jam" type="application/x-jam">
サンプルアプリケーション
</OBJECT>
ダウンロードするには
<A ijam="#application.declaration" href="xxx.html">ここ</A>
をクリック。
```

以下に、アプリケーションのダウンロードのために使用されるタグを解説します。

- A タグ

A タグは、ダウンロード対象のアプリケーションに対応する OBJECT タグを参照するために使用されます。ユーザーがこの A タグを選択することで、アプリケーションのダウンロード処理が開始されます。

A タグの ijam 属性には、OBJECT タグの id 属性に指定された名前を指定します。ijam 属性は Java 対応 i モード用に新設された属性であり、従来の i モード携帯電話では解釈することができません。A タグに href 属性を併記することで、従来の i モード機種は href 属性を、Java 対応 i モード機種では ijam 属性を解釈するようになります。

なお、ijam 属性を持つ A タグでは、必ずペアで href 属性を指定するようにしてください。

- OBJECT タグ

OBJECT タグは、アプリケーションに対応する ADF を参照するために使用されます。OBJECT タグの id 属性には、A タグから参照される名前（HTML 内で一意）を指定します。また data 属性には ADF の位置を示す URL を指定します。type 属性には data 属性で示されたデータ（この場合 ADF）のコンテンツタイプを指定します。ADF のコンテンツタイプは"application/x-jam"です。

なお、実際のアプリケーション（JAR ファイル）は、ADF 中に指定された PackageURL キーから参照されます。

Java 対応 i モードでは、OBJECT タグの type 属性で ADF ファイルのコンテンツタイプを確認します。したがって Web サーバー側では、ADF の拡張子に対応するコンテンツタイプのコンフィグレーションを行う必要はありません。ただし ADF の拡張子は".jam"とするようにしてください（ADF の URL の実体が CGI などの場合は、その CGI の拡張子".jam"とします）。

10.2 サーバーアプリケーション実行のための Web サーバーの構成

クライアントサーバー型のアプリケーションを動作させる場合、サーバー側アプリケーションに適用する技術によっては Web サーバーに特定のコンフィグレーションを行う必要がある場合があります。例えばサーバー側アプリケーションに Java サーブレットを使用する場合、Web サーバー側では Java サーブレットエンジンのインストールとコンフィグレーションを行う必要があります。

使用している Web サーバーのドキュメントおよびサーバー側アプリケーションに適用している技術に関するドキュメントを参照し、Web サーバーに適切なコンフィグレーションを行うようにしてください。

10.3 特定メーカーの Java 対応 i モードへのダウンロード

メーカー拡張機能を使用したアプリケーションや厳密な画面サイズを前提としたアプリケーションは、特定メーカーの携帯電話でのみアプリケーション開発者の意図通りに動作します。このようなアプリケーションは、そのターゲットとする機種だけに適切にインストールされることが望まれます。

アプリケーションのダウンロードを各機種毎に適切にコントロールするために、Java 対応 i モードでは次のような機構が利用可能になっています。

1. 携帯電話からの HTTP リクエストには、携帯電話の機種を判別するための User-Agent リクエストヘッダが含まれています。Web サーバー側ではアプリケーションのダウンロード用 HTML を表示する際に、User-Agent の内容に基づいてユーザーに適切なアプリケーションを提示するようナビゲートすることができます。
2. 携帯電話側では、ADF の TargetDevice キーを参照して、ダウンロードするアプリケーションがその機種に対応しているかどうかを判定します。特定機種向けのアプリケーションでは、ADF の TargetDevice キーを適切に設定することで、アプリケーション開発者の意図しない機種へのダウンロードを防止することができます。

付録 A

サンプルコード: バンキングアプリケーション

このバンキングアプリケーションは、一般的なクライアントサーバーアプリケーションの実装例を表したものです。ユーザーは、バンクサーバーにログオンし、残高照会などの銀行取引や、最近行われた預金や引出しの表示、口座間での資金の振り替えを行います。このアプリケーションのクライアント側アプリケーションは、`<emulator directory>/apps/banking` にあるデモプログラムに含まれています。サーバー側アプリケーションは、`<emulator directory>/tools/bankingserver` にあるデモプログラムに含まれています。

クライアントアプリケーション

クライアント側アプリケーションは、ログオン画面を表示し、入力されたログオン情報を検査のためにサーバーに送信します。ユーザーのログオンが完了すると、クライアントアプリケーションは、ユーザーの口座を表示します。この中からユーザーが1つを選択すると、アプリケーションは使用可能な取引の種類を表示します。ユーザーが1つを選択し、操作を行うと、クライアントが取引情報をサーバーに送信し、取引が完了します。クライアントは、サーバーから返されるメッセージを処理し、取引に誤りがある場合は、それを表示します。

サーバーアプリケーション

サーバー側アプリケーションは、クライアントの GET や POST コマンドに応答します。クライアントがトランザクションコマンドやログオン情報をサーバーに POST すると、サーバーはトランザクションコマンドやログオン情報を解析します。次にサーバーは、適切なトランザクションメソッドを呼び出し、トランザクションを完了して、確認メッセージやエラーメッセージをクライアントに返します。

付録 B

サンプルコード: 株価情報/株価監視アプリケーション

スタンドアロンの株価情報/株価監視アプリケーションは、一連の画面を表示する一般的なトレーディングプログラムです。このアプリケーションは、
<emulator directory>/apps/stockdemo にあります。

このアプリケーションは、次のことを行います。

- UI コンポーネントの使用
- イベントリスナーの実装
- 株式を購入するためのデータ入力
- ScratchPad を使った一時データの格納や取り出し
- ソフトウェアによる浮動小数点演算

付録 C

サンプルコード： スペースインベーダゲーム

クライアントサーバー型のゲームアプリケーションであるこのデモプログラムは、かつて流行したスペースインベーダです。このデモプログラムは、
<emulator directory>/apps/spaceinv にあります。

アプリケーションは、次のことを行います。

- インベーダー、砲台、ミサイル、UFO に関する低レベル描画操作とスプライト表示
- ScratchPad を使ったゲームのハイスコアの格納や取り出し
- 低レベルのキーイベント処理

付録 D

サンプルコード: ユーザーインタフェースアプリケーション

スタンドアロンのユーザーインタフェースデモプログラムは、
<emulator directory>/apps/uidemo にあります。

このアプリケーションは、次のことを行います。

- 基本 UI コンポーネントの利用例
- イベントリスナーの実装
- デフォルトのレイアウトマネージャがスクロールやフォーカスの管理に関しどのように動作するか
- デフォルトレイアウトマネージャを無効にする例
- Canvas クラスの使用とグラフィックスオブジェクトへの描画

API	アプリケーションプログラミングインタフェース
Applet	アプレット。Web ページとともにユーザーに送信される小さなプログラム。i モード Java アプリケーションはアプレットではありません。
bps	bits per second。秒当たりのビット数。
CGI	Common Gateway Interface
CHTML	コンパクト HTML
CLDC	Connected Limited Device Configuration
HTTP	Hypertext Transfer Protocol
HTTPS	Secure Hypertext Transfer Protocol
IDE	Integrated Development Environment。統合開発環境。
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JAM	Java Application Manager
JAR	Java アーカイブ(Java Archive)の略
JDK	Java Development Kit
JNI	Java Native Interface
JVMS	Java Virtual Machine Specification
MIME	Multi-Purpose Internet Mail Extensions。インターネット e メールプロトコルを拡張し、このプロトコルを使って多様なデータファイルをインターネットで送信できるようにしたもの。
Servlet	サーブレット。Web コンテンツを動的に生成するモジュール形式のサーバー側 Java アプリケーションを開発するためのメカニズム。
SSL	Secure Socket Layer
UI	ユーザーインタフェース
Unicode	文字を表すための 16 ビットの符号化
URL	Uniform Resource Locator。アクセス可能なインターネット上のファイル(資源)のアドレス。

