

i アプリコンテンツ開発ガイド for 504i

～ 詳細編 ～

第 2.2 版

平成 15 年 7 月 9 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。サンのロゴマークは、米国 Sun Microsystems, Inc. の登録商標です。

「i モード」、「i アプリ／アイアプリ」、「i-appli」ロゴ、「DoJa」、「i メロディ／アイメロディ」は NTT ドコモの商標または登録商標です。

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。なお、本書では、コピーライト及び商標・登録商標表記はしておりません。

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 2000-2003 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、ムーバ 504i シリーズおよび 2002 年冬以降発売の FOMA 携帯電話に搭載される Java アプリケーション実行環境（本書中では i アプリ実行環境と呼びます）をターゲットとして記述されています。本書内ではデジタル・ムーバ 503i シリーズの i アプリ実行環境を DoJa-1.0 プロファイル、ムーバ 504i シリーズなどの i アプリ実行環境を DoJa-2.0 プロファイルと呼びます。DoJa-2.1 プロファイルと記載されている場合、特に 2002 年冬以降発売の FOMA 携帯電話に搭載されている i アプリ実行環境を指します。また DoJa-2.2 プロファイルと記載されている場合、特に 2003 年春以降発売の FOMA 携帯電話に搭載されている i アプリ実行環境を指します。

本書の内容は、以前のプロファイルから存在している機能であっても、以前のプロファイルにのみ対応した携帯電話には適用されませんのでご注意ください。本書で解説するプロファイルには、既存機能に関する未定義の仕様の明確化も含まれています。

目 次

本書の目的.....	7
対象読者	7
本書の構成.....	7
表記上の規則.....	8
関連ドキュメント	8
第1章 はじめに	9
1.1 i アプリ実行環境の利点	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 i アプリ対応携帯電話の 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.....	17
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.....	18
第2章 i アプリ実行環境.....	19
2.1 i アプリ実行環境	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 API	20
2.1.5 i アプリ API	20
2.2 i アプリ実行環境における API の概要.....	21
2.2.1 ネットワークアーキテクチャ	21
2.2.2 CLDC API	22
2.2.3 i アプリ API	28
第3章 設計上の考慮事項.....	39
3.1 i アプリ対応携帯電話の特性	39
3.2 メモリの問題	40
3.2.1 メモリ使用量の最小化	41
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	43
3.3.1 ターゲットユーザー向けの設計.....	43
3.3.2 アプリケーション操作方法の設計指針	43
3.3.3 スクロール	44
3.3.4 キーおよびソフトキー	44
3.3.5 データ入力	44

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	46
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	47
3.7	ハードウェアのアクセス	49
3.8	エラー処理	49
3.9	待ち受けアプリケーション	49
3.10	携帯電話の機種判別	50
第4章	ユーザーインタフェースの設計	51
4.1	高レベル UI コンポーネントによるプログラミング	54
4.1.1	Panel の使用	54
4.1.2	Dialog の使用	55
4.1.3	コンポーネントの使用	55
4.1.4	コンポーネントの配置	63
4.1.5	フォーカスとスクロール	66
4.1.6	高レベルイベントの処理	67
4.2	低レベル API によるプログラミング	67
4.2.1	Canvas の使用	67
4.2.2	グラフィックスの描画	68
4.2.3	Image オブジェクトへの描画	70
4.2.4	カラーサポート	71
4.2.5	低レベルイベントの処理	72
4.3	イベントリスナー	73
4.3.1	ComponentListener	74
4.3.2	SoftKeyListener	75
4.3.3	KeyListener	76
4.3.4	MediaListener	76
4.3.5	TimerListener	77
4.4	マルチメディアデータの使用	77
4.4.1	プレゼンタの利用	77
4.4.2	複数のサウンドデータの同時再生	83
4.4.3	サウンド再生の同期イベント	83
第5章	通信の制御	85
5.1	クライアント/サーバプログラミング	85
5.1.1	接続の終了	89
5.2	セッション管理	90
5.3	HTTPS による安全な通信	92
第6章	テキスト処理	93
6.1	テキスト処理	93

6.2	i アプリ対応携帯電話の絵文字	94
第7章 ScratchPad とリソースの使用.....		98
7.1	ScratchPad に対する読み取りと書き込み	98
7.2	リソースの読み取り	100
7.3	エラー処理.....	102
第8章 ハードウェアのアクセス		103
第9章 待ち受けアプリケーション.....		105
9.1	待ち受けアプリケーションの概要.....	105
9.1.1	待ち受けアプリケーションの特徴	105
9.1.2	待ち受けアプリケーションのユーザー設定.....	107
9.2	待ち受けアプリケーションの作成.....	108
9.2.1	待ち受けアプリケーションクラス	108
9.2.2	待ち受けアプリケーションのライフサイクルと状態遷移	108
9.2.3	システムイベント	112
9.2.4	ADF による待ち受けアプリケーションの宣言.....	113
9.3	待ち受けアプリケーション実行中の他機能の起動.....	113
9.4	i アプリ API 利用上の注意点	114
9.4.1	ユーザーインターフェース.....	114
9.4.2	入出力	116
9.4.3	ハードウェア制御	116
9.4.4	ネイティブアプリケーション連携	116
第10章 OBEX 外部接続		117
10.1	OBEX によるデータ送受信	117
10.2	OBEX 外部接続 API.....	118
10.2.1	OBEX クライアント API.....	119
10.2.2	OBEX サーバーAPI.....	122
10.3	外部機器接続のヒント	125
第11章 ネイティブアプリケーション連携.....		127
11.1	ブラウザ連携.....	127
11.1.1	ブラウザからのアプリケーションの起動.....	127
11.1.2	アプリケーションからのブラウザ起動	129
11.2	メール連携.....	130
11.2.1	メールからのアプリケーションの起動	130
11.3	音声通話連携.....	132
11.3.1	アプリケーションからの音声発信機能の起動	133
11.4	外部機器連携.....	133
11.4.1	外部機器からのアプリケーションの起動.....	133
第12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.....		137
12.1	i アプリ開発環境のインストール.....	137

12.2	i アプリ開発環境のコンポーネント	138
12.3	開発サイクルの概要	139
12.4	通信先アプリケーションの作成	140
12.5	アプリケーションのパッケージ化	140
12.5.1	ADF の作成	140
12.5.2	JAR ファイルの作成	144
12.6	アプリケーションのテスト	144
第 13 章	アプリケーションの配布	146
13.1	アプリケーション配布のための Web サーバーの構成	146
13.2	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	147
13.3	特定メーカーの i アプリ対応携帯電話へのダウンロード	148
付録 A	FOMA 携帯電話の i アプリ実行環境 – FOMA02 仕様モデル、 FOMA03 前期仕様モデル	149
A.1	DoJa-2.1 プロファイル	149
A.1.1	携帯電話の個体識別情報の取得	149
A.1.2	ADF へのプロファイルバージョン (ProfileVer キー) の設定	150
A.1.3	携帯電話からのテレビ電話の発信	150
A.2	DoJa-2.2 プロファイル	150
A.2.1	ADF へのプロファイルバージョン (ProfileVer キー) の設定	150
A.3	FOMA に向けた i アプリ実行環境最適化	151
A.3.1	ネットワークアーキテクチャ	151
A.3.2	i アプリ実行環境	151
用語集		153

本書の目的

『i アプリコンテンツ開発ガイド』は、i モード携帯電話向けに 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（i アプリ）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 アプリコンテンツ開発ガイド』には、i アプリを利用したサービス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 アプリ実行環境の紹介と概要	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 11 章
開発プロセス	第 12 章、第 13 章

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 アプリコンテンツ開発ガイド for 504i API リファレンス編』
- 『i アプリコンテンツ開発ガイド for 504i i アプリオプション・i アプリ拡張編』
- 『i アプリコンテンツ開発ガイド for 504i オプション・拡張 API リファレンス編』
- 『DoJa2.0 API 向け i α ppli Development Kit 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



i アプリ対応携帯電話では、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（i アプリ）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、i アプリ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。

i アプリ対応携帯電話では、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 i アプリ対応携帯電話のことを指します）のようにメモリや CPU、消費電力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 ファミリーにおける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。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・どの機種種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 モード携帯電話への Java の搭載により、コンテンツプロバイダはより新しく、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、i アプリ対応携帯電話におけるアプリケーション実行環境（i アプリ実行環境）の全般的な特徴と機能について説明します。

1.1 i アプリ実行環境の利点

Java コンテンツプロバイダは、i アプリを Java プログラミング言語を使って作成します。i アプリ実行環境では、携帯電話のユーザーインタフェース、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 (i アプリ API) が提供されます。i アプリ API は以下の 3 つのカテゴリの API 群から構成されます。

- i アプリ基本 API

共通仕様として API および動作が規定されており、全ての機種に共通に搭載される API です。

- i アプリオプション API

共通仕様として API および動作が規定されていますが、搭載有無の判断はメーカーに委ねられる API です。このカテゴリの API の使用にあたっては、API が搭載されていない機種があることを前提にしなければなりません。

- i アプリ拡張 API

同一の機能を実現するものであっても、API および動作がメーカー毎に異なる可能性のある API です。このカテゴリの API の使用にあたっては、API が搭載されていない機種があることと、機種により API の使用方法などが異なる（同機能を持つ機種間であってもソースコードレベルでの互換性が保持されない）ことを前提にしなければなりません。

【DoJa-2.0】

i アプリ実行環境の API カテゴリ分類は、DoJa-1.0 プロファイルでのカテゴリ分類から変更されています。また、DoJa-2.0 プロファイルにおいては、上記の 3 カテゴリ全ての API について i アプリコンテンツ開発ガイドにて公開されます。

なお、本書「i アプリコンテンツ開発ガイド 詳細編」では主に J2ME/CLDC API および i アプリ基本 API を解説の対象とします。i アプリオプション API および i アプリ拡張 API の詳細については、「i アプリコンテンツ開発ガイド i アプリオプション・i アプリ拡張編」および「i アプリコンテンツ開発ガイド オプション・拡張 API リファレンス編」を参照してください。

i アプリ実行環境には、この他に次の機能があります。

- アプリケーションの動的な配布
- 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マに提供されます。

i アプリ実行環境は、次の 4 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 アプリ API (i アプリ基本 API)
- i アプリ API (i アプリオプション API)
- i アプリ API (i アプリ拡張 API)

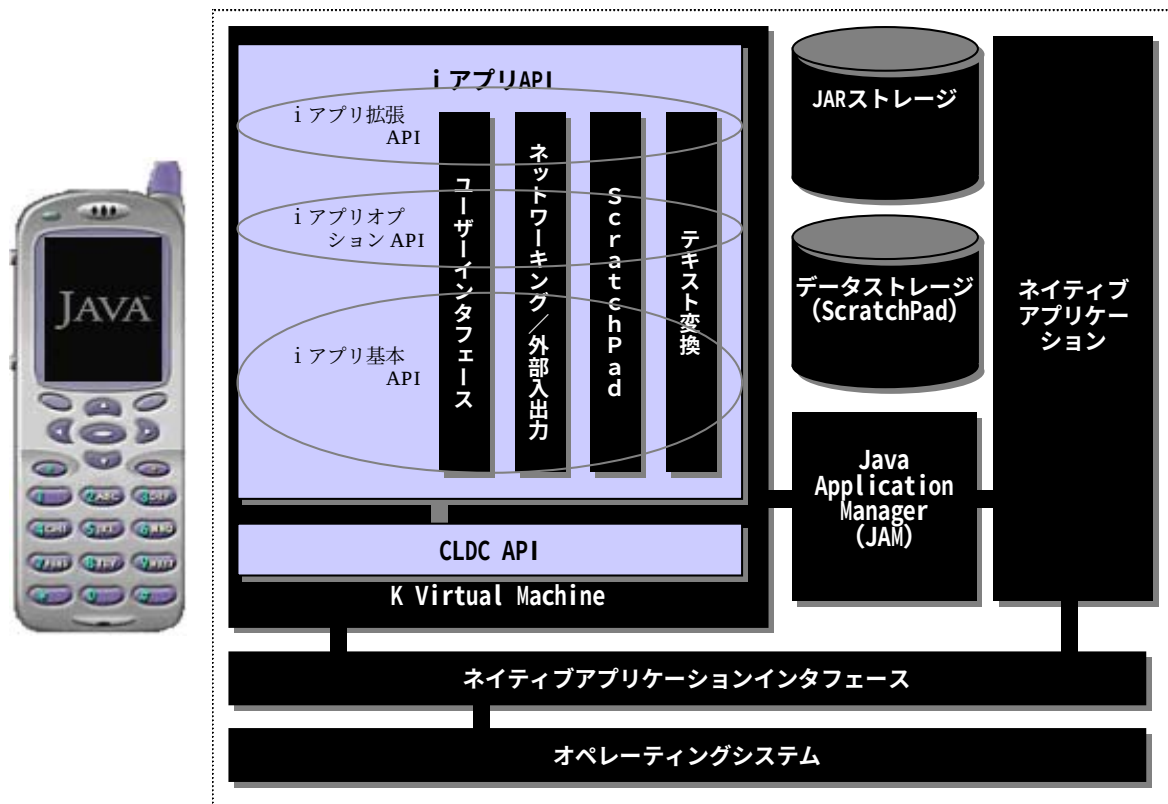


図 1. i アプリ対応携帯電話における i アプリ実行環境

注意事項：

- 多くのアプリケーションでは、J2ME/CLDC API と i アプリ基本 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、アプリケーションの表現力を増すために i アプリオプション API や i アプリ拡張 API を使用するものがあります。これらの API を使用するアプリケーションは、メーカーにより動作しない可能性がある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。
- デバイスの提供するサービスに最適化されたユーザーインター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、PDA などの携帯情報端末、家電機器、POS 端末などがあります。

J2ME/CLDC は、i アプリ実行環境に導入されている安定したアプリケーション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
- 多くの開発者および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 アプリ実行環境には、次のような機能を持つ 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
- 待ち受け状態の携帯電話にアプリケーションを常駐させる待ち受けアプリケーション制御
- ネイティブアプリケーション（ブラウザ、メーラなど）との連携動作
- 赤外線ポートを利用した外部接続（IrOBEX）

【DoJa-2.0】

待ち受けアプリケーション、IrOBEX 外部接続、ネイティブアプリケーション連携は DoJa-2.0 プロファイルにおいて新規に導入された機能です。待ち受けアプリケーションは第 9 章、IrOBEX 外部接続は第 10 章、ネイティブアプリケーション連携は第 11 章でそれぞれ詳細に解説しています。

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 アプリサービスでは、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サポートしています。

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 13 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、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 アプリ実行環境には、インターネットを介して接続されるクライアント/サーバー型アプリケーション

ン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。また、赤外線ポートを介して近距離の携帯電話間などでクライアント/サーバー型アプリケーションを動作させるための IrOBEX API も含まれています。

HTTP(S)接続

HTTP(S)接続を利用するクライアント/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、インターネット上にある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。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のフロントエンド・プロセッサ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、i アプリ対応携帯電話のユーザーは、快適に情報にアクセスしたり、電子商取引のようなリモ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。

HTTP(S)接続を利用したクライアント/サーバー型アプリケーションの作成については、第 5 章を参照してください。

IrOBEX 外部接続

IrOBEX（以降、単に OBEX と呼びます）外部接続は、携帯電話に装備された赤外線ポートを使用して、近距離の外部機器との間でデータ送受信を行うためのシンプルなクライアント/サーバーモデルを実現します。i アプリ実行環境は OBEX 外部接続のクライアント API とサーバーAPI の両方を持っており、携帯電話間でのデータの送受信もサポートしています。この機能を利用することで、名刺交換アプリケーションやスケジュール交換アプリケーションのような、携帯電話間で小さなデータをやりとりするアプリケーションを実現することができます。

OBEX 外部接続では、接続先の機器のハードウェア条件や周囲の環境などによっては通信を行えない場合があります。

なお、以降は特に注記のない場合クライアント、サーバーはそれぞれ HTTP(S)通信を行う場合のクライアント、サーバーを指すものとします。OBEX においてはそれぞれ OBEX クライアント、OBEX サーバーと記載します。

1.1.2 永続データのローカル記憶装置サポート

i アプリ実行環境では、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。

i アプリ対応携帯電話のセキュリティ機能の一つに、Java Application Manager (JAM：次項参照) 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 アプリ実行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アプリケーションを管理するコンポーネントが含まれています。このコンポーネントは Java Application Manager (JAM) と呼ばれ、次のような機能を持っています。

- 携帯電話に格納されているアプリケーションをリスト表示する。
- アプリケーションの実行管理（起動や強制終了、i アプリ実行環境とネイティブアプリケーションの間の仲介など）を行う。
- アプリケーションをインストールまたは更新する。
- 携帯電話に格納されている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、携帯電話の終話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。

【DoJa-2.0】

DoJa-2.0 プロファイルでは、アプリケーションの強制終了は終話キーに統一されます。なお、DoJa-1.0 プロファイルでは強制終了のキーサインはメーカーにより異なります。

1.3 i アプリ対応携帯電話の 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、i アプリ実行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 KVM はマルチスレッドをサポートしており、1 つのアプリケーションの中で通信とユーザーインタフェース処理を並行して行うといったマルチスレッドプログラミングは可能です。

- JAM は KVM から独立したネイティブコンポーネントであり、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。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。
- 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、他のアプリケーションのために ScratchPad 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の不正な行為を行うことはできません。
- アプリケーションがメモリダイヤルなどの個人情報を含んだネイティブデータにアクセスすることはできません。また、ブラウザやダイヤラーなどのネイティブアプリケーションにアクセスする機能では、ユーザーの意思に反して Web ブラウジングや音声通話発信を行うことのないよう、API 呼び出し時に必ずユーザーに動作への同意を求めます。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。

【DoJa-2.0】

DoJa-1.0 プロファイルでは、セキュリティ上の問題からアプリケーションと他のネイティブアプリケーションの連携動作をサポートしませんでした。DoJa-2.0 プロファイルでは、アプリケーションとネイティブアプリケーション間の連携動作が行われる際に必ずユーザーの同意を得る機構を取り入れ、ユーザーにとって安全なネイティブアプリケーション連携をサポートします。

- i アプリ実行環境の通信機能では、i アプリ対応携帯電話でサポートされる HTTP 通信および HTTPS 通信に対する簡単で使いやすいインタフェースが提供されます。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ィ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、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~数 10K バイトの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、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 12.5.1 項を、アプリケーションを携帯電話に配布するための Web サーバーの準備については第 13 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。

なお以下の場合には、アプリケーションはすでに携帯電話にインストールされているもの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）。

- ・ 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 URL の ADF 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
- ・ 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）および 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

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、インストール済みアプリケーションのリストに前項でインストールされた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。

ADF でアプリケーションが HTTP 通信を行うように設定されている場合は、ここでアプリケーションに HTTP 通信を許可するかをユーザーに問い合わせます。また ADF でアプリケーションが待ち受けアプリケーションとして動作可能となるよう設定されている場合は、そのアプリケーションを待ち受けアプリケーションとして登録するか否かをユーザーに問い合わせます。

1.~5.の過程を経てアプリケーションは携帯電話上にインストールされ、利用可能（実行可能）な状態になります。

1.4.2 アプリケーションの起動

アプリケーションの通常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アプリケーションの画面が表示されます。

また、上記のユーザー操作によるアプリケーションの起動だけではなく、i アプリ実行環境は以下の起動形態のバリエーションを持っています。

- ・ タイマー起動
- ・ ネイティブアプリケーションとの連携による、ブラウザ、メーラ、外部機器（赤外線ポート）からのアプリケーション起動
- ・ 待ち受けアプリケーションの登録による、待ち受け状態復帰時の自動起動

タイマー起動については、12.5.1 項の LaunchAt キーの項を参照してください。またネイティブアプリケーション連携については第 11 章を、待ち受けアプリケーションについては第 9 章を参照してください。

【DoJa-2.0】

DoJa-1.0 プロファイルでは、ユーザー操作による起動とタイマー起動のみをサポートしていました。DoJa-2.0 プロファイルでは、ネイティブアプリケーションからの起動および待ち受けアプリケーションの待ち受け時起動の 2 つが追加されています。

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。

バグの修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時を設定します。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 URL は以前のものと同じにします。ADF の URL を変更すると、すでに携帯電話にアプリケーションをダウンロードしているユーザーがアプリケーションの更新を行えなくなるため注意が必要です。

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 アプリ実行環境

この章では、i アプリ対応携帯電話に搭載される Java アプリケーション実行環境（i アプリ実行環境）の機能構成、API 構成の概要を解説します。

2.1 i アプリ実行環境

i アプリ実行環境は、携帯電話のネイティブアプリケーション、Java Application Manager、i アプリ API、K Virtual Machine、および CLDC API から構成されています。次に i アプリ実行環境の構成を示します。

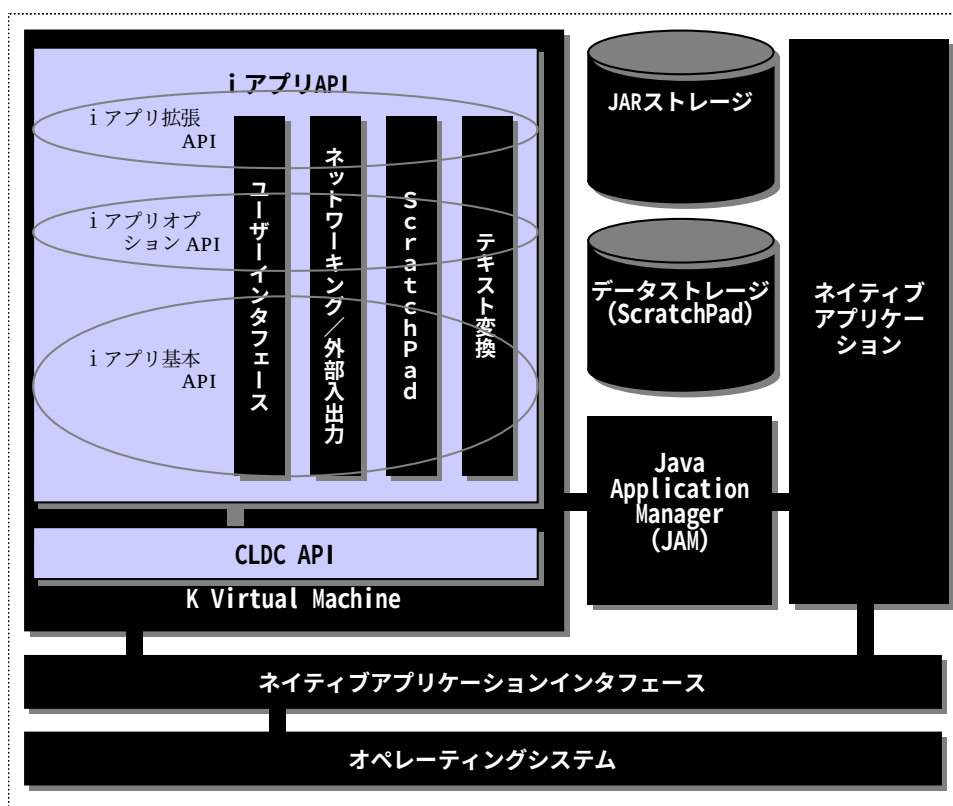


図 2: i アプリ実行環境

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アプリケーション（ブラウザ、ネットワーキングコンポーネント、ダイアラーなど）を使って行われます。i アプリ実行環境には、JAM を通じてこれらのネイティブアプリケーションと連携動作するための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、消費電力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 API

CLDC API 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 API 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 API については、2.2.2 項を参照してください。

2.1.5 i アプリ API

i アプリ API は、i アプリ対応携帯電話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 アプリ API では CLDC を土台として、i アプリ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 アプリ API は、次のような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
- OBEX による外部接続

i アプリ API に含まれる 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。

なお i アプリ API は、以下の 3 つのカテゴリに分類されます。

- 共通仕様として API および動作が規定されており、全ての機種に共通的に搭載される i アプリ基本 API
- 共通仕様として API および動作が規定されているが、搭載有無の判断がメーカーに委ねられる i アプリオプション API
- 同一の機能を実現するものであっても、API および動作がメーカー毎に異なる可能性のある i アプリ拡張 API

全機種で動作することが求められるアプリケーションでは、i アプリオプション API および i アプリ拡張 API は使用しないようにしてください。

2.2 i アプリ実行環境における API の概要

次の各項では、i アプリ実行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 API
- 2.2.3 i アプリ API

CLDC API には標準 J2SE ライブラリのサブセットクラス、および機能的には J2SE にマッピング可能ですが小型デバイス向けに再設計された CLDC 独自のクラスが含まれています。これらは Java Community Process (JCP) を経て策定されました。

一方、i アプリ API は i モードサービスに特化した API 群であり、NTT ドコモにより規定されています。

なお、本項では i アプリ API のうち i アプリ基本 API についてのみ記載します。i アプリオプション API および i アプリ拡張 API の詳細については、「i アプリコンテンツ開発ガイド i アプリオプション・i アプリ拡張編」および「i アプリコンテンツ開発ガイド オプション・拡張 API リファレンス編」を参照してください。

2.2.1 ネットワークアーキテクチャ

i アプリサービス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。

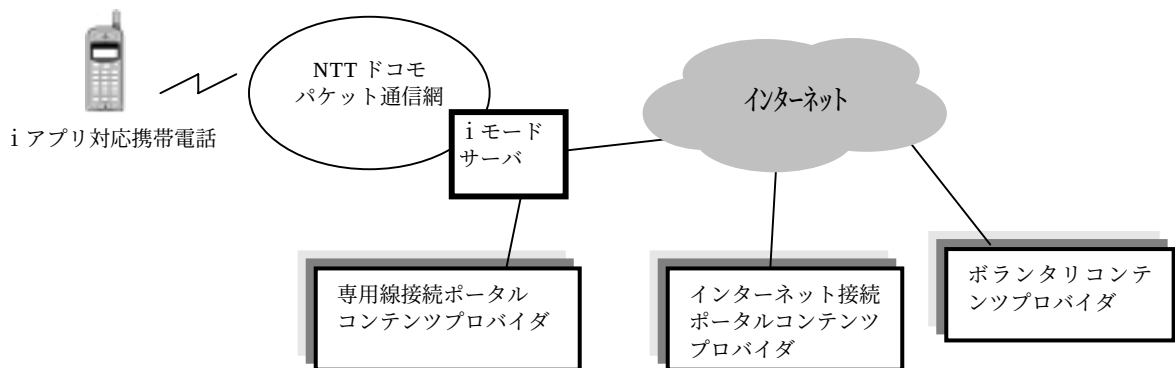


図 3: ネットワークアーキテクチャ概念図

i アプリサービスに関する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ーアクセス

i アプリサービス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 モードサーバー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 モードサーバー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 i アプリ対応携帯電話では、携帯電話側にて SSL (Secure Sockets Layer) によるサーバー認証をサポートしています。i アプリ対応携帯電話の通信機能と 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 サーバーまでの間となります。

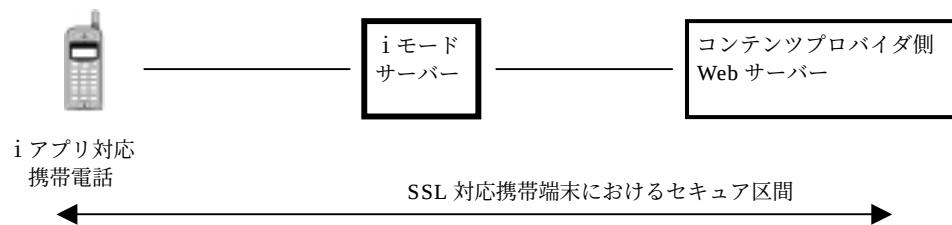


図 4: i モードサービスの SSL サポート

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おりません。

2.2.2 CLDC API

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、ユーザーインターフェース、ネットワーキング、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、特にネットワーキング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。これらのクラスの機能や、継承されたメソッドの使用方法に変更はありません。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 コンフィギュ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

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 アプリ実行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。アプリケーションは `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 クラスのクラスメソッド `open()` 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 J2SE や他のアプリケーション実行環境における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。次に、Generic Connection クラス階層の主なクラス間の関係を示します。

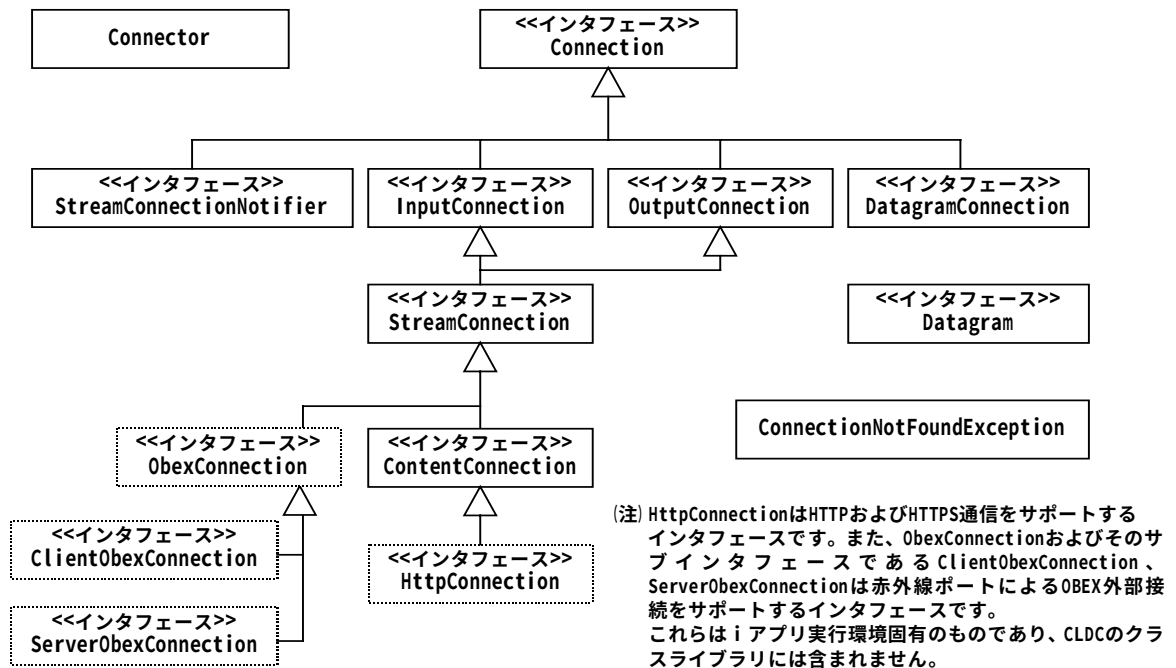


図 5: Generic Connection クラス階層

i アプリ実行環境では、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 アプリ API に含まれる、`ContentConnection` のサブインタフェース)
- ・ ScratchPad : `StreamConnection`
- ・ リソース : `InputConnection`
- ・ OBEX 外部接続 : `ObexConnection`、`ClientObexConnection`、`ServerObexConnection`
(i アプリ API に含まれる、`StreamConnection` のサブインタフェース)

【DoJa-2.0】

DoJa-2.0 プロファイルでは赤外線ポートを介した OBEX 外部接続をサポートしますが、このための API も Generic Connection フレームワークの枠組みの中で規定されています。

2.2.3 i アプリ API

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、i アプリ基本 API に属するクラスについて説明します（一部のクラスには、i アプリオプション API に属するメソッドを含んでいるものがあります）。

com.nttdocomo.io パッケージ

CLDC の Generic Connection フレームワークは、i アプリ実行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/HTTPS プロトコルによるネットワーク接続、および赤外線ポートを介した OBEX 外部接続をサポートします。

Generic Connection フレームワークについては 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インタフェース：

名前	説明
ClientObexConnection	OBEX クライアントとしての赤外線ポートへのアクセスを定義するインタフェースです。 javax.microedition.io.Connector.open() メソッドに URL "obex:/irclient" 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。 ClientObexConnection は、i アプリ実行環境において OBEX サーバー、OBEX クライアントの共通インタフェースを定義する ObexConnection のサブインタフェースです。 【DoJa-2.0】本インタフェースは DoJa-2.0 プロファイルで新設されました。
HttpConnection	HTTP 接続へのアクセスを定義するインタフェースです。 javax.microedition.io.Connector.open() 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。 HttpConnection は Generic Connection フレームワークにおける javax.microedition.io.ContentConnection インタフェースのサブインタフェースです。
ObexConnection	OBEX サーバー、OBEX クライアントの共通インタフェースを定義します。このインタフェースを直接実装したインスタンスをアプリケーションで使用することはありません。 このインタフェースには、レスポンスステータスやオペレーションコード定数、OBEX サーバーと OBEX クライアントで共通のアクセッサメソッドが定義されています。 ObexConnection は Generic Connection フレームワークにおける javax.microedition.io.StreamConnection インタフェースのサブインタフェースです。 【DoJa-2.0】本インタフェースは DoJa-2.0 プロファイルで新設されました。
ServerObexConnection	OBEX サーバーとしての赤外線ポートへのアクセスを定義するインタフェースです。 javax.microedition.io.Connector.open() メソッドに URL "obex:/irserver" 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。 ServerObexConnection は、i アプリ実行環境において OBEX サーバー、OBEX クライアントの共通インタフェースを定義する ObexConnection のサブインタフェースです。 【DoJa-2.0】本インタフェースは DoJa-2.0 プロファイルで新

設されました。

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、getStatus()メソッドで取得します。

HttpConnection を使用したクライアント/サーバプログラミングについては第 5 章で解説しています。また OBEX による外部接続プログラミングについては第 10 章で解説しています。

com.nttdocomo.net パッケージ

このパッケージに含まれるクラス：

名前	説明
URLEncoder	このクラスには、String を URL エンコード形式（MIME タイプ x-www-form-urlencoded で使用されるエンコード形式）にエンコード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、URL エンコード形式を String にデコード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。

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。

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 アプリ 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。
Phone	Phone クラスはダイヤラーを表し、携帯電話のネイティブの音声通話機能にアクセスする手段を提供します。 アプリケーションから音声通話機能呼び出すと、JAM はユーザーに音声通話発信を行ってもよいか確認を求めます。この確認を省略することはできません。 【DoJa-2.0】本クラスは DoJa-2.0 プロファイルで新設されました。
TimeKeeper	i アプリ 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。ただし、アプリケーションが中断された場合や、待ち受けアプリケーションが休眠状態に入るとき、タイマーは停止します。

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
IllegalStateException	操作やメソッド呼び出しが行われてはならない状態の時にそのような操作やメソッド呼び出しが行われると、実行時例外として IllegalStateException がスローされます。 【DoJa-2.0】本クラスは DoJa-2.0 プロファイルで新設されました。
UnsupportedOperationException	サポートされない操作やメソッドを実行すると、実行時例外として UnsupportedOperationException 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 <code>Panel.setComponentListener()</code> メソッドを使用して <code>Panel</code> 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 <code>componentAction()</code> 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とは、 <code>Panel</code> 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に関する指定が特に行われていない場合、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。現在までのプロファイルでは、デフォルトフォーカスマネージャ以外のフォーカスマネージャを規定していません。
Interactable	ユーザーが操作できるコンポーネント (<code>Button</code> 、 <code>ListBox</code> 、 <code>TextBox</code> 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 <code>Panel</code> 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 <code>keyPressed()</code> と <code>keyReleased()</code> 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とは、 <code>Panel</code> 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。i アプリ実行環境では、レイアウトマネージャに関する指定が特に行われていない場合に使用されるデフォルトレイアウトマネージャ、および HTML 風のレイアウトを行う HTML レイアウトマネージャが定義されています。 【DoJa-2.0】HTML レイアウトマネージャは DoJa-2.0 プロファイルで新設されました。
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 <code>mediaAction()</code> 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	<code>MediaData</code> の実装オブジェクトは <code>MediaManager.getData()</code> 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。 <code>MediaData</code> 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	<code>MediaImage</code> 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) および JPEG フォーマット (JFIF または EXIF 形式、Baseline エンコーディング) です。 【DoJa-2.0】DoJa-1.0 プロファイルでは、使用可能なデータは GIF フォーマットのみです。
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。実際の端末では i メロディ形式フォーマットのサウンドデータを使用することができます。
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
AnchorButton	AnchorButton コンポーネントは、HTML のアンカー風のボタン (ボタンとしての操作が可能な下線付きテキスト) を表現します。テキストの前に見出し用のイメージ画像を指定することもできます。また、テキストは複数行に渡ることができます。アンカーボタン押下により処理を行う場合は、ComponentListener を実装したオブジェクトを、Panel の setComponentListener() メソッドによりリスナー登録します。アンカーボタンが操作されると、リスナーは BUTTON_PRESSED イベントを受け取ります。 【DoJa-2.0】本クラスは DoJa-2.0 プロファイルで新設されました。
AudioPresenter	このクラスは、サウンドメディアデータ (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、 ComponentListener を実装したオブジェクトを、 Panel の setComponentListener() メソッドによりリスナー登録します。ボタンが操作されると、リスナーは BUTTON_PRESSED 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 paint() 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定します。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、 Display.setCurrent() メソッドを、切り替えたい画面に対応する Frame オブジェクトを引数として呼び出すことで行います (ただし Dialog のみ、 Dialog.show() メソッドにより表示します)。
Graphics	Canvas および Image への描画とそれに関連する操作をカプセル化します。なお Image への描画は、その Image が Image.createImage() メソッドにより新規に作成された Image インスタンスである場合に限り可能です。 【DoJa-2.0】 Image インスタンスからの Graphics インスタンス取得と描画の機能は、DoJa-2.0 プロファイルで追加されました。

HTMLLayout	HTML 風のレイアウトを行うためのレイアウトマネージャです。Panel に追加された順にレイアウトしていく点ではデフォルトレイアウトマネージャと同様ですが、HTML レイアウトマネージャでは、用意されたメソッドにより HTML の改行、段落およびアライメントに相当する指定を行うことができます。 【DoJa-2.0】本クラスは DoJa-2.0 プロファイルで新設されました。
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。IApplication クラスは、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を管理します。
Image	Image クラスは静止画像を表現する abstract クラスです。静止画像 (GIF または JPEG) から MediaImage インスタンスを作成し、getImage() メソッドを呼び出すことで Image の実装インスタンスを取得することができます。また Image.createImage() メソッドにより、新規に Image インスタンスを作成することもできます。 【DoJa-2.0】createImage() メソッドは DoJa-2.0 プロファイルで追加されました。また、DoJa-1.0 プロファイルでは JPEG はサポートしていません。
ImageLabel	ImageLabel コンポーネントは、Panel に静止画像を表示するときに使用します。Image インスタンスを ImageLabel に設定し、それを Panel 上に配置することができます。
ImageButton	ImageButton コンポーネントは、静止画像を貼り付けることのできるボタンを表現します。イメージボタン押下により処理を行う場合は、ComponentListener を実装したオブジェクトを、Panel の setComponentListener() メソッドによりリスナー登録します。イメージボタンが操作されると、リスナーは BUTTON_PRESSED イベントを受け取ります。 【DoJa-2.0】本クラスは DoJa-2.0 プロファイルで新設されました。
Label	Label コンポーネントは、Panel にテキストを表示するときに使用します。Label 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が、ユーザーが直接編集することはできません。
ListBox	ListBox 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 ListBox の外観や機能を選択することができます。
MApplication	MApplication クラスは、待ち受けアプリケーションの雛型クラスです。MApplication クラスは IApplication クラスを継承しており、待ち受けアプリケーションのメインクラス（待ち受け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）はこのクラスのサブクラスでなければなりません。MApplication クラスは、待ち受けアプリケーションの起動・終了・実行状態管理などのライフサイク

ルを管理します。

【DoJa-2.0】本クラスは DoJa-2.0 プロファイルで新設されました。

MediaManager

MediaResource を管理するためのクラスです。
MediaManager はメディアデータ (MediaData、MediaImage、MediaSound) のファクトリクラスです。

Panel

Panel 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。Panel 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。Panel は、同時に 1 つしか画面に表示できません。

PhoneSystem

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。

ShortTimer

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。
ShortTimer は、本クラスの static メソッド getShortTimer() を使用して取得します。
getShortTimer() メソッドの引数には Canvas を指定しますが、タイマーイベントはこの Canvas の processEvent() 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。

TextBox

TextBox 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。TextBox 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。

Ticker

Ticker 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

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実行時例外クラスです。

VisualPresenter

VisualPresenter コンポーネントは、画像メディアデータ (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 と MApplication

`com.nttdocomo.ui.IApplication` クラスは、全てのアプリケーションの雛型となるクラスです。通常、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。
- アプリケーションダウンロード後の初回起動時、およびネイティブアプリケーション（ブラウザ、メール）からアプリケーションの起動が指示された場合には、HTML やメール本文からアプリケーションにパラメータを与えることができます。
- 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。
- `IApplication.launch()`メソッドを使用することで、アプリケーションから他のネイティブアプリケーション（i アプリ実行環境では、起動可能なネイティブアプリケーションはブラウザに限定されます）を起動することができます。ネイティブアプリケーションの起動に成功すると、元のアプリケーションの実行は終了します。

また、`com.nttdocomo.ui.MApplication` クラスは、待ち受けアプリケーション特有の複雑なライフサイクルに対応するために `IApplication` クラスを継承、拡張したものです。待ち受けアプリケーションのメインクラスは `MApplication` クラスを継承していなければなりません。

`MApplication` クラスには、`IApplication` の持つ機能の他に以下のような機能が含まれています。

アプリケーションの状態制御：

待ち受けアプリケーションには、通常のアプリケーションが実行されている状態と同等の状態である「活性化状態」の他に、

- アプリケーションは動作しているが、キーイベントおよびソフトキーイベントは携帯電話のネイティブの待ち受け機能を使用するためフックされ、アプリケーションに通知されない「非活性化状態」
- 電池の消費を抑えるため、画面表示は保持したままアプリケーションの動作が停止する「休眠状態」

という3つの状態を持ちます。`MApplication` クラスは、これらの状態の遷移を管理します。

システムイベントのハンドリング：

待ち受けアプリケーションでは、常駐しているアプリケーションに外部からのイベントがシステムイベントとして通知されます。待ち受けアプリケーションは、`MApplication` クラスの持つシステムイベントハンドラをオーバーライドすることで、上記に示した3つのいずれの状態にあってもこれらのイベントの通知を受け、処理することができます。

待ち受けアプリケーションの詳細については、第9章を参照してください。

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 アプリ実行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メインクラスの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 アプリ実行環境は、メインクラスのインスタンス化時に引数のあるコンストラクタは使用しません。明示的にメインクラスの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.Sysytem.exit()** メソッドが定義されています。しかし i アプリ実行環境では、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

i アプリサービス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。

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 i アプリ対応携帯電話の特性

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。

i アプリを活用したサービスを設計する場合、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

次に、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。

i アプリ対応携帯電話

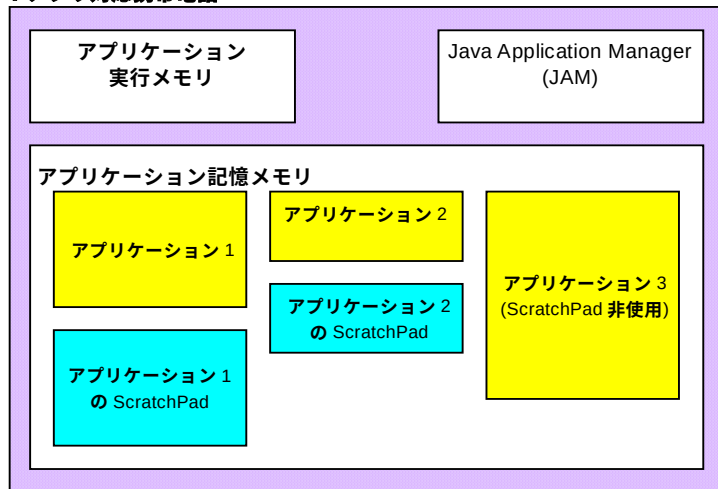


図 6: i アプリ対応携帯電話のメモリ

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 30K バイト（以降、1K バイト=1024 バイト）まで、ScratchPad が 100K 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 100K 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。

【DoJa-2.0】

DoJa-1.0 プロファイルでは JAR ファイルは最大 10K バイト、ScratchPad は最大 5K バイトと規定されていました。DoJa-2.0 プロファイルでは、JAR ファイル、ScratchPad とともに上記サイズまで拡張されています。

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 アプリ実行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か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 アプリ実行環境では、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。
- 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およびサイズを最小限に抑える。
- 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、および個々のクラスのサイズを最小限に抑える。
- 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やイメージは速やかに破棄（`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。
- メモリを消費しやすい処理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、必要最小限のものに絞り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

アプリケーションを保存するために、アプリケーション記憶メモリの中に以下の2つのブロックが確保されます。

- 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。i アプリオプシオン API や i アプリ拡張 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 13 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 12.5.1 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たとともに、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。

なお、NTT ドコモの提供する i アプリ開発環境のビルド機能を利用してアプリケーションをビルドする場合、コンパイル時には `-g:none` オプションが設定されています。i アプリ開発環境の詳細については、第 12 章を参照してください。

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。特にユーザー操作可能な UI コンポーネント（ボタンなど）を1つの画面に多数表示した場合、フォーカス移動操作などの手間により、ユーザーにとってアプリケーション操作が煩わしく感じられることがあります。
- 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、Panel は垂直方向のスクロールをサポートしますが、Canvas を使用する場合は画面のスクロールはアプリケーション側で制御する必要があります。
- 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 アプリケーション操作方法の設計指針

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こと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つ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。

【DoJa-2.0】

DoJa-1.0 プロファイルにおいては、Panel のスクロールサポート有無はメーカーの実装に委ねられていました。DoJa-2.0 プロファイルでは、デフォルトレイアウトマネージャまたは HTML レイアウトマネージャが設定されている Panel は縦方向のスクロールをサポートします。

3.3.4 キーおよびソフトキー

i アプリ対応携帯電話では、アプリケーションは以下の標準的なキーを使用することができます。

- ・ 上下左右キーおよび選択（決定）キー
- ・ 0～9 までの数字キーおよび#キー、*キー
- ・ 待ち受けアプリケーション切り替えキー

メーカーによっては、i アプリオプションのカテゴリに属する機能としてこれら以外の特別なキーをサポートする場合があります。

また、i アプリ対応携帯電話には上記以外に 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。

【DoJa-2.0】

待ち受けアプリケーション切り替えキーは、DoJa-2.0 プロファイルの待ち受けアプリケーション機能のために新設されました。待ち受けアプリケーションが休眠状態や非活性化状態である場合を除き、アプリケーションは直接このキーを使用（キーイベントの取得）することができます。待ち受けアプリケーションが休眠状態や非活性化状態である場合は、待ち受けアプリケーション切り替えキーのイベントはシステムが使用します。

なお、待ち受けアプリケーションの詳細については第 9 章を参照してください。

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。テキスト入力是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。

1つ目の方法を使用する場合、一般的にはパスワードには数字のみを使用します。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な作業となります。パスワードに使用する文字を数字のみとする場合、`TextBox.setInputMode()` メソッドを使用して `TextBox` の初期入力モードを数字入力モードとしておくことで、ユーザーの誤操作を防ぐことができます。

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つ目の方法を使用します。

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なくても、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ためです。

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ー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、サーバーアドレス、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電波が届く区域を時に頻繁に出入りします）。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。このように、アプリケーションを実行中の携帯電話上で、より優先度の高い別の機能が動作する必要がある場合、アプリケーション実行の中断や再開が行われる場合があります。

アプリケーションが中断される条件は、音声通話の着信時以外にもいくつか存在します。以下に、アプリケーションの中断、再開が行われるケースを示します。

- アプリケーション実行中に音声着信があった場合（アプリケーションが通信実行中の場合、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もあります）。および、待ち受けアプリケーションが非活性化状態または休眠状態で実行されている際にメール・メッセージ着信があった場合。なお待ち受けアプリケーションが活性化状態で実行されている場合は、通常のアプリケーションの実行時と同様に、メール・メッセージの着信を受けてもアプリケーションの実行は継続されます。
- アプリケーション実行中に、他のネイティブアプリケーションが起動された場合。これには、アプリケーション実行中に時刻アラームなどのネイティブアプリケーションが自動起動した場合や、ネイティブアプリケーション連携機能によりアプリケーションから他のネイティブアプリケーションを起動した場合を含みます。なお、起動されたネイティブアプリケーションがパケット通信を行うタイプのもの（ブラウザやメール）である場合、呼び出し元のアプリケーションは中断ではなく終了します。
- ユーザーがアプリケーションの強制終了操作を行い、終了確認ダイアログでユーザーが強制終了をキャンセルした場合。
- 携帯電話が上記以外にアプリケーションを中断、再開するための機構（中断・再開キーなど）を持っており、その操作が行われた場合。

【DoJa-2.0】

アプリケーションの中断・再開条件については DoJa-2.0 プロファイルにおいて明確化されました。DoJa-1.0 プロファイルでは必ずしも上記の動作とはならない場合がありますので注意してください。

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、アプリケーションの実行が再開されたら通信結果をチェックし、必要に応じて通信のリトライを行います。これは、アプリケーションが通信処理を行っている際に中断・再開が発生すると、その通信処理は失敗するためです。

- 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）は停止しています。これらのタイマーは、アプリケーション再開時に明示的に再スタートを行う必要があります。
 なお DoJa-1.0 プロファイルでは、タイマーを使用中にアプリケーションの実行が中断された場合、アプリケーションは再開後にタイマーのインスタンスを破棄した上で再作成する必要があります。
- アプリケーション中断時に AudioPresenter がサウンドを再生していた場合、AudioPresenter は動作を停止します。アプリケーション再開後も自動的に再生を開始することはありません。アプリケーションから明示的に再生指示（play()）を行うことにより、サウンドデータの再生はデータの先頭から行われます。なお、VisualPresenter が動画を再生しているときにアプリケーションの実行が中断された場合は、動画再生は一時的に停止しますが、アプリケーションの再開とともに中断時のフレームから動画再生も再開されます。
- DoJa-2.0 プロファイルでは、アプリケーションの中断・再開により Canvas が再表示される際、i アプリ実行環境からその Canvas の paint() メソッドが 1 回呼び出されることが保証されます。しかしその paint() メソッド呼び出しで全画面が完全には復元されないようなケース（アプリケーションがイベントに応じて差分描画のみ行うようになっているなど）では、どの程度システムが画面を復元できるかは機種により異なります。
- アプリケーションの中断・再開が発生した場合、再開したアプリケーションが適切なプロンプトメッセージを表示して一旦停止することが望ましい場合があります。たとえば、多くのアクションゲームのようにユーザーとの即座対話が必要なケースでは、アプリケーションが再開されてからユーザーの操作の準備が整うまでの間に若干の時間的な猶予を持たせることで、再開時のアプリケーションの操作性を向上させることができます。

【DoJa-2.0】

中断・再開がアプリケーションに与える影響については DoJa-2.0 プロファイルにおいて詳細に規定されました。DoJa-1.0 プロファイルでは必ずしも上記の動作とはならない場合がありますので注意してください。

注意事項：

- メーカーによっては、アプリケーションの中断と再開をサポートしない場合があります。そのような携帯電話では、アプリケーションの実行が中断されるべき状況になるとそのアプリケーションは終了します。
- アプリケーションが通信を行っている場合、音声通話の着信でアプリケーションが中断するか否かは、携帯電話の i モード着信設定の内容に従います。i モード通信中に音声通話の着信を許可しない設定になっている場合は、アプリケーションが通信を行っている間は音声通話の着信を受けてもアプリケーションの実行は中断されません。
- アプリケーションの実行が中断された際に未処理のイベントがイベントキューにキューイングされていた場合、アプリケーション再開時にはそれらは全て破棄されます。

3.7 ハードウェアのアクセス

i アプリ実行環境では、携帯電話のハードウェア（ネイティブリソース）にアクセスする手段として `PhoneSystem` クラスを提供しています。このクラスを使用することで、ディスプレイのバックライトやバイブレータの制御などを行うことができます。アプリケーションからハードウェアに頻繁にアクセスすると、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。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、ハードウェアアクセス機能の多くは i アプリオプション API のカテゴリに含まれます。したがって、メーカーによっては一部の機能が搭載されないことがありますので注意してください。

3.8 エラー処理

i アプリ実行環境の提供する 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）やエラー（`java.lang.Error` およびサブクラス）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。

3.9 待ち受けアプリケーション

DoJa-2.0 プロファイルでは、時計アプリケーションをはじめとする常駐タイプのアプリケーションを使い勝手よく実装するための機構として、待ち受けアプリケーションをサポートしています。

待ち受けアプリケーションは、実行中のアプリケーションの画面を、待ち受け画像のように待ち受け画面に表示することのできる常駐型アプリケーションです。もちろん、アプリケーションが画面の内容を変更すると、それは直ちに待ち受け画面に反映されます。

DoJa-1.0 プロファイルの上でも、アプリケーションを常に実行し続ける形でアプリケーションをシステムに常駐させることは可能でした。しかしこの方法では、以下のような点において問題がありました。

- ・ 電話をかけたり Web ブラウジングを行おうとした場合、ユーザーは明示的にアプリケーションを終了させなければならない。また、それらの作業が済んだあとはユーザーが明示的にアプリケーションを再起動しなければならない。
- ・ アプリケーション実行中はメールの着信を受けることができない。このため、ユーザーはメールが来ていることに気が付かないことがある。
- ・ アプリケーションの実行時間がそのまま電力消費量に結びつき、アプリケーションを長時間駆動し続けることができない。

待ち受けアプリケーションでは上記のような問題を解消し、ユーザーにとって使い勝手のよい常駐型アプリケーションを実現するための機構が取り入れられています。常に実

行され続けていることを前提としたアプリケーションを開発する場合、そのアプリケーションを待ち受けアプリケーションとして実装することを検討してください。

なお、待ち受けアプリケーションの詳細は、第 9 章を参照してください。

3.10 携帯電話の機種判別

アプリケーションが現在どのような機種で実行されているかを知るための手段として、以下の 2 つが用意されています。

- ・ i アプリ実行環境におけるシステムプロパティ `microedition.platform`

アプリケーションは、`java.lang.System.getProperty()` メソッドを使用してシステムプロパティ（システムの情報を取得するためのプロパティセット）を参照することができます。

`microedition.platform` プロパティは、CLDC でホストプラットフォームを示すシステムプロパティとして規定されていますが、i アプリ実行環境ではこのプロパティの値に携帯電話の機種名が格納されます。携帯電話上のアプリケーションは、この値を調べることで自身が動作している携帯電話の機種を知ることができます。

このプロパティに格納される機種名は、ユーザーエージェントに含まれている名前と同等のものとなります。

- ・ HTTP 通信時にサーバーに送出される `User-Agent` リクエストヘッダ

ブラウザからの Web アクセスと同様、アプリケーションから HTTP 通信を実行した際にはリクエストヘッダにユーザーエージェント（`User-Agent` ヘッダ）が含まれます。携帯電話上のアプリケーションと通信を行うサーバーアプリケーションは、`User-Agent` ヘッダの内容を調べることで現在対話を行っている携帯電話の機種を知ることができます。

【DoJa-2.0】

DoJa-1.0 プロファイルでは、`microedition.platform` にどのような値を設定するかはメーカーの判断に委ねられているため、メーカーによっては機種名が取得できないことがあります。

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、サウンドファイルの再生機能

i アプリ実行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、少ないコード量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。

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。ディスプレイには、Panel または Canvas のいずれかを 1 面だけ表示することが可能です。Canvas によって構成される画面でコンポーネントを使用することはできません。ただし Panel によって構成される画面には、イメージを取り扱うためのコンポーネントを使用することで低レベル描画を取り入れることができます。

1 つのアプリケーションは複数の Panel や Canvas を持つことができます。アプリケーションは `Display.setCurrent()` メソッドを使用して、ディスプレイに表示する Panel や Canvas を切り替えることができます。

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、キーの押下や解放などによって発生した低レベルイベントは **Canvas** オブジェクトに通知されます。低レベル API では、アプリケーション開発者は **Canvas** クラスを継承したサブクラスを作成して使用しますが、このとき **Canvas** クラスの `processEvent()` メソッドをオーバーライドすることで低レベルイベントを受け取ることができるようになります。イベントが発生す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。`processEvent()` メソッドには、発生したイベントのタイプと、イベントに特有の引数が渡されます。

高レベル API の場合は、ボタン押下などの高レベルイベントは **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高レベルイベント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、イベントが発生すると **Panel** オブジェクト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、イベントの発生を通知します。高レベルイベントのリスナーインタフェースは、イベント発生源ごとに異なるものが用意されています。イベント発生時にリスナーメソッドに通知される情報も、リスナーの種類により異なります。

イベントリスナーの詳細は、4.3 項を参照してください。

次に、ユーザーインタフェースのためのクラス構成を示します。

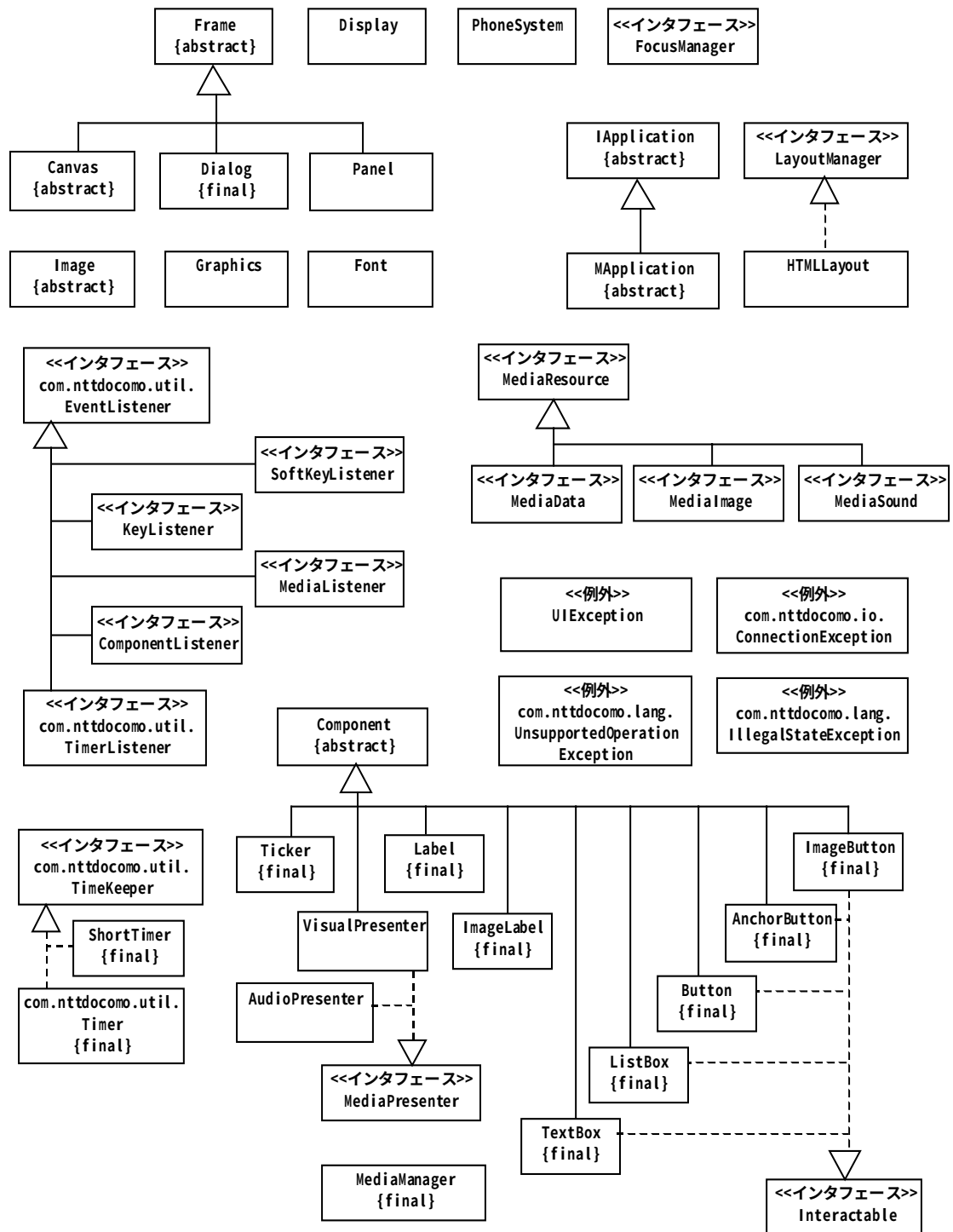


図 7: UI クラス階層

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 アプリ実行環境では、9つの基本的な UI コンポーネントが提供されます。

- Label
- ImageLabel
- Button
- AnchorButton
- ImageButton
- 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。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。

【DoJa-2.0】

AnchorButton、ImageButton は DoJa-2.0 プロファイルで新設されたコンポーネントです。

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。

Panel には、その機能や目的を表示するためのタイトルを設定することができます。

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。
- **Dialog** は、**Panel** や **Canvas** と同様に **Frame** クラスのサブクラスですが、**Dialog** を表示するために **Display.setCurrent()** メソッド（画面に表示されているフレームを切り替えるメソッド）を使用することはできません。**Dialog** を画面に表示するには、**Dialog.show()** メソッドを使用します。

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**、**AnchorButton**、**ImageButton**、**TextBox**、**ListBox**、**Ticker**、**VisualPresenter** があります。**AnchorButton** を除き、コンポーネントのサイズが画面サイズ（**Panel** にタイトルが設定されている場合はタイトル領域を除いた画面サイズ）を超えることはできません。

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。

【DoJa-2.0】

DoJa-1.0 プロファイルでは、コンポーネントに画面サイズを超えるサイズを指定した場合の動作はメーカーにより異なります。

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** コンポーネント内の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/不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、そのコンポーネントの背景色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 アプリ実行環境では別個のコンポーネントが定義されています。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 {
    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);
        ...
    }
    ...
}

```

AnchorButton

AnchorButton クラスは HTML のアンカー風ボタン（下線付きテキストとして表示されるボタン）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。AnchorButton コンポーネントには、テキストおよびその見出しとなるイメージを設定することができます。コンポーネントの幅より長いテキストが指定された場合、テキストは自動的に折り返され複数行に渡って表示されます。

AnchorButton の縦方向のサイズは、画面の縦方向のサイズに制限されることはありません。ただし横方向のサイズは画面の横方向のサイズに制限されます。

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。無効化された AnchorButton のテキストは下線付きテキストではなく通常のテキストのような

Look&Feel で表示されます。このため無効化された `AnchorButton` は、`Panel` においてユーザーに長い文章などを提示するためのコンポーネントとしても使用することができます。

次のコードは、`AnchorButton` コンポーネントの例です。

例: `AnchorButton` コンポーネント

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

public class AnchorButtonDemo extends UIDemoPanel implements ComponentListener {
    AnchorButton anchorButton1, anchorButton2;
    AnchorButtonDemo() {
        // 操作可能なアンカーボタンを作成する。
        anchorButton1 = new AnchorButton("次画面へジャンプ");
        this.add(anchorButton1);
        // 操作不能なアンカーボタンを作成し、ユーザーに文章を提示するために使用する。
        anchorButton2 = new AnchorButton(
            "操作不能に設定されたアンカーボタンは、ユーザーに" +
            "長いテキストを提示するために使用することもできます。");
        anchorButton2.setEnable(false);
        this.add(anchorButton2);
        ...
    }
    ...
}
```

ImageButton

`ImageButton` クラスはイメージを貼り付けることのできるボタンコンポーネントで、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。有効なイメージが設定されていない `ImageButton` は、そのコンポーネントの背景色で塗りつぶされています。

このコンポーネントは有効または無効な状態に設定できます。

次のコードは、`ImageButton` コンポーネントの例です。

例: `ImageButton` コンポーネント

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

public class ImageButtonDemo extends UIDemoPanel implements ComponentListener {
    MediaImage mImage;
    Image image;
    ImageButton imageButton;
    ImageButtonDemo() {
        // イメージボタンを作成し、それをパネルに追加する。
        try {
            mImage = MediaManager.getImage("resource:///label.gif");
            mImage.use();
            image = mImage.getImage();
        } catch (ConnectionException ce) {
            ...
        }
        imageButton = new ImageButton(image);
        this.add(imageButton);
        ...
    }
    ...
}
```

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ラジオボタン、番号付きリスト、チェックボックス、
 * オプションメニュー、単一選択リスト、複数選択リストを示す。
 */
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。
- カレントフレームに追加済みの `ListBox` へのアプリケーションからの項目設定処理（`setItems()` や `append()` など）と、その `ListBox` に対するユーザー操作が同時に行われた場合の振る舞いはメーカーにより異なります。カレントフレーム上にある `ListBox` に項目の設定を行う際は、`setEnabled()` メソッドによりユーザー操作を無効化するようにしてください。

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を入力したり変更したり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);
            ...
        }
        ...
    }
}

```

注意事項：

- ユーザーが **TextBox** でテキストの編集を行っている状態（フロントエンドプロセッサが起動している状態）では、バックグラウンドのスレッドがフロントエンドプロセッサ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。例えばフロントエンドプロセッサが起動されているときにバックグラウンドスレッドが背後の **Panel** を他のフレームに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。

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。
// ティッカーは可視または不可視に設定可能

public class TickerDemo extends UIDemoPanel implements ComponentListener {
    Ticker ticker;          // ティッカーウィジェット
    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、コンポーネントの配置をレイアウトマネージャに任せるか、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ごとに配置位置を指定するかを選択することができます。

レイアウトマネージャは **Panel** に設定して使用します。**Panel** にレイアウトマネージャを設定するには、**Panel.setLayoutManager()** メソッドを使用します。

i アプリ実行環境で用意されているレイアウトマネージャには、**Panel** を生成した初期状態で設定されているデフォルトレイアウトマネージャと、HTML 風の指定方法によりレイアウトを行うための HTML レイアウトマネージャの 2 つがあります。

デフォルトレイアウトマネージャの基本レイアウト方針は、シンプルなフローレイアウトです。パネルに追加されたコンポーネントは画面の左から右に向けて順次横並びに配

置られていき、画面の横幅で収まりきらなくなった時点で次のコンポーネント配置が可能な位置（1 段下の列）へと配置されます。

また HTML レイアウトマネージャでは、コンポーネントを **Panel** に追加していく過程で改行（BR タグに相当）、段落（P タグに相当）、アライメント（CENTER や DIV タグなどに相当）を指定することで、アプリケーション開発者がより明示的にレイアウトを指示することができます。

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** インタフェースはメーカーがレイアウトマネージャを実装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。**Panel** オブジェクトの **setLayoutManager()** メソッドを呼び出すことで、その **Panel** が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に制限が発生します。
- 1 つの HTML レイアウトマネージャオブジェクトは、2 つ以上の **Panel** に同時に設定することはできません。アプリケーション内に HTML レイアウトを適用したい **Panel** が複数ある場合には、それぞれの **Panel** に対し別々の HTML レイアウトマネージャオブジェクトを割り当てるようにしてください。
- HTML レイアウトマネージャは、**setLayoutManager()** メソッドで **Panel** に設定される度に初期状態（アライメントが左寄せで改行、段落指定のない状態）に戻ります。

次のコードは、HTML レイアウトマネージャを使用してコンポーネントの配置を行う例を示したものです。

例: HTML レイアウトマネージャによるコンポーネントの配置

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

public class HTMLLayoutDemo extends UIDemoPanel {
    Label          label1, label2, label3, label4;
    HTMLLayout      lm;

    HTMLLayoutDemo() {
        label1 = new Label("Label 1");
        label2 = new Label("Label 2");
        label3 = new Label("CENTER");
        label4 = new Label("RIGHT");

        // HTML レイアウトマネージャを生成し、Panel に設定する。
        lm = new HTMLLayout();
        setLayoutManager(lm);

        // レイアウトを指定しつつ Panel に Label を追加する。
        add(label1);
        //以降のコンポーネントは次行に配置する（BR タグ相当）。
        lm.br();
    }
}
```

```

        add(label2);
        // 以降のコンポーネントは次行に配置する（BR タグ相当）。
        lm.br();

        // begin()から対応する end()までの間に追加されたコンポーネントを
        // 中央寄せで表示する。
        lm.begin(HTMLLayout.CENTER);
        add(label3);
        lm.end();

        // begin()から対応する end()までの間に追加されたコンポーネントを
        // 右寄せで表示する。
        lm.begin(HTMLLayout.RIGHT);
        add(label4);
        lm.end();
    }
}

```

また次のコードは、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);
    }
}

```

【DoJa-2.0】

HTML レイアウトマネージャ（HTMLLayout クラス）は DoJa-2.0 プロファイルにて新設されました。

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の詳細な振る舞いは、メーカーにより異なる場合があります。i アプリ対応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。典型的な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。ただし、フォーカスが設定されているコンポーネントがディスプレイの領域外に配置されている状態では、そのコンポーネントを操作することはできません。

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：

- **FocusManager** インタフェースはメーカーがフォーカスマネージャを実装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。`Panel` オブジェクトの `setFocusManager()` メソッドを呼び出すことで、その `Panel` が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。ただし現在までのプロファイルでは、デフォルトフォーカスマネージャ以外のフォーカスマネージャは規定されていません。

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。

4.1.6 高レベルイベントの処理

Panel により構成された画面では、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ー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ことを保証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。`paint()` メソッドに渡される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。また、アプリケーションが `Canvas` オブジェクトや `Image` オブジェクトから `getGraphics()` メソッドを使用してグラフィックスオブジェクトを取得した場合、そのグラフィックスオブジェクトが不要になった時点で `dispose()` メソッドを呼び出してください。
- `i` アプリ実行環境では、`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の再描画を行うことができます。DoJa-1.0 プロファイルでは、描画デバイスの持つクリッピング描画能力によりこの機能をサポートしない携帯電話があることを想定していますが、DoJa-2.0 プロファイルに対応した携帯電話は全てこの機能をサポートします。
- DoJa-2.0 プロファイルでは、アプリケーションの中断・再開により `Canvas` が再表示される際、`i` アプリ実行環境からその `Canvas` の `paint()` メソッドが 1 回呼び出されることが保証されます。しかしその `paint()` メソッド呼び出しで全画面が完全には復元されないようなケース (アプリケーションがイベントに応じて差分描画のみ行うようになっているなど) では、どの程度システムが画面を復元できるかは機種により異なります。
- DoJa-2.0 プロファイルでは、低レベル文字列描画に使用されるデフォルトフォントは 12 ドットフォントです。これに対し DoJa-1.0 プロファイルでは、デフォルトフォントサイズはメーカーにより異なります。なお、いずれのプロファイルにおいても、デフォルトフォント以外にどのようなフォントをサポートするかについてはメーカーにより異なります。一般には、携帯電話の限られたメモリにフォント情報を保持する必要があるため、実際に搭載されるフォントの数は 1~2 種類となります。
- DoJa-2.0 プロファイルでは、グラフィックス描画に関して、`i` アプリオプションおよび `i` アプリ拡張のカテゴリに属する機能 (スプライトや 3D グラフィックスなど) が追加されています。これらの詳細については、「`i` アプリコンテンツ開発ガイド `i` アプリオプション・`i` アプリ拡張編」および「`i` アプリコンテンツ開発ガイド オプション・拡張 API リファレンス編」を参照してください。

4.2.3 Image オブジェクトへの描画

DoJa-2.0 プロファイルでは、Image オブジェクトへの描画をサポートします。

Image オブジェクトからの Graphics オブジェクトの取得は、Image.getGraphics() メソッドを呼び出すことにより行います。Image オブジェクトから取得した Graphics オブジェクトへの描画指示は、元となる Image オブジェクトの表示内容に反映されます。このようにして描画された Image オブジェクトを ImageLabel や ImageButton に設定することで、アプリケーションが行った描画処理の結果を Panel 使用時にも表示することができます。

Image オブジェクトから Image.getGraphics() メソッドにより Graphics オブジェクトを取得するには、その Image オブジェクトが Image.createImage() メソッドにより生成された新規の Image オブジェクトでなければなりません。

MediaImage.getImage() メソッドで取得した Image オブジェクトに対して getGraphics() メソッドを呼び出した場合、UnsupportedOperationException がスローされます。

なお、Image.createImage() で生成されたイメージは、初期状態ではその機種における Canvas のデフォルト背景色と同じ色で塗りつぶされています。ただし Image 自身には背景色の概念はありません。Image から取得した Graphics に対して描画内容のクリアを指示するには、Graphics.clearRect() メソッドではなく、適切な色の設定を行った上で Graphics.fillRect() メソッドを呼び出します。

次のコードは、アプリケーションからの指示により描画された Image オブジェクトを ImageLabel で使用する例です。

例: Image オブジェクトへの描画

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

public class ImageDrawDemo extends UIDemoPanel {
    ImageLabel    iLabel;
    Image          img;
    Graphics       g;

    ImageDrawDemo() {
        // Image オブジェクトを新規に生成し、Graphics オブジェクトを取得する。
        int width = 100;
        int height = 50;
        img = Image.createImage(width, height);
        g = img.getGraphics();

        // 取得した Graphics オブジェクトに描画する。
        g.lock();
        g.drawRect(0, 0, width-1, height-1);
        g.drawString("Graphics による", 5, 16);
        g.drawString("Image への", 5, 32);
        g.drawString("描画のサンプル", 5, 48);
        g.unlock(true);

        // Image オブジェクト ImageLabel に設定し、Panel 上に表示する。
        iLabel = new ImageLabel(img);
        add(iLabel);
    }
}
```

注意事項：

- `Image.createImage()`メソッドで生成可能な `Image` のサイズは、縦横とも携帯電話の描画エリアのサイズまでです。携帯電話の描画エリアより大きなサイズの `Image` を生成可能かどうかはメーカーにより異なりますので注意してください。

4.2.4 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。

色の管理方法はメーカーに依存するため、i アプリ実行環境内部で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。アプリケーションは上記のメソッドを使用して、機種間で共通の色の定義 (名前または RGB 値) を各機種固有の色の定義に変換する必要があります。

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。
        }
        ...
    }
}
```

【DoJa-2.0】

アプリケーションはカラー対応絵文字 (i モードサービス向けに NTT ドコモが定義した外字セット) を使用することができます。カラー対応絵文字では、1つ1つの絵文字に対しデフォルトの表示色が割り当てられています。

DoJa-2.0 プロファイルの `Graphics` クラスには、`Graphics` オブジェクトに描画色が明示的に指定されている場合に、絵文字をデフォルトの表示色で描画するか `Graphics` オブジェクトに指定された描画色で描画するかを選択するためのメソッド `setPictoColorEnabled()` が追加されています。

DoJa-1.0 プロファイルでは、`Graphics` オブジェクトに描画色が明示的に指定された状態で絵文字を描画した際に、その絵文字をデフォルトの表示色で描画するか `Graphics` オブジェクトに指定された描画色で描画するかはメーカーにより異なります。

注意事項：

- アプリケーションからの色指定において、透明色を指定することはできません。i アプリ実行環境で透明色を取り扱うことのできるケースは、透過 GIF イメージを使用する場合に限られます。

4.2.5 低レベルイベントの処理

Canvas 使用時に適用される低レベルイベント処理の方法は、Panel 使用時における高レベルイベント処理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 クラスを継承する際に、`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 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 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);
    }
}
}

```

【DoJa-2.0】

DoJa-2.0 プロファイルでは、アプリケーション再開時の低レベルイベント発生の流れを以下のように規定しています。

- ・ アプリケーション再開時は、アプリケーションが再開したことを示す RESUME_VM_EVENT の通知を行った後、アプリケーションの resume() メソッドが呼び出されます。
- ・ アプリケーション再開時に、描画データの破壊を意味する UPDATE_VM_EVENT が発生する場合は、RESUME_VM_EVENT が発生した後で UPDATE_VM_EVENT が発生します。

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 MediaPlayer 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、そのイベントを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。

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、ディスプレイに表示されている Panel 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 インタフェースおよび `componentAction()` メソッドを実装します。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 の選択内容が確定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 の文字入力が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/不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不可視に、
             * 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;
                }
            }
        }
    }
}

```

注意事項：

- TextBox および ListBox は、それらのクラスが備えるメソッドで TextBox に設定されているテキストや ListBox の選択状態を変更することができます。このように、ユーザーの操作ではなくメソッドを使用して状態の変更を行った場合もコンポーネントイベントが通知されます。

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、以下のようになります。

- リスナーとして使用するクラスに、SoftKeyListener インタフェースおよび softKeyPressed()、softKeyReleased()の2つのメソッドを実装します。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、以下のようになります。

- リスナーとして使用するクラスに、**KeyListener** インタフェースおよび `keyPressed()`、`keyReleased()` の 2 つのメソッドを実装します。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。

【DoJa-2.0】

Panel では、画面のスクロールやコンポーネントの操作などのために方向キーおよび選択キーを使用します。DoJa-2.0 プロファイルでは、高レベル API 利用時におけるこれらのキーのキーイベントについて以下のように規定されています。

- 方向（上下左右）キーはキーイベントを発生しません。これらのキーは、スクロールおよびフォーカス制御のために常に **Panel** で処理されます。
- 選択（決定）キーは、フォーカスを得ているコンポーネントがディスプレイ上に表示されている場合に限りキーイベントを発生しません。フォーカスを得ているコンポーネントがディスプレイ上に表示されている状態では、選択キーはそのコンポーネントの操作のために **Panel** で処理されます。

なお、DoJa-1.0 プロファイルでは、どのような状況でどのようなキーイベントが **Panel** で処理されるかについてはメーカーにより異なります。

4.3.4 MediaListener

MediaListener インタフェースは、メディアデータ再生においてメディアイベント（**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、以下のようになります。

- リスナーとして使用するクラスに、**MediaListener** インタフェースおよび `mediaAction()` メソッドを実装します。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** : サウンドメディアデータの再生完了
- **AUDIO_PLAYING** : サウンドメディアデータの再生開始
- **AUDIO_STOPPED** : サウンド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、メディアデータの詳細については 4.4 項を参照してください。

【DoJa-2.0】

メディアデータ再生開始・停止・完了イベントのうち、DoJa-1.0 プロファイルで発生が保証されていたものは AUDIO_COMPLETE のみでしたが、DoJa-2.0 プロファイルではメディアデータ再生開始・停止・完了イベントは全ての機種でサポートされます。また、DoJa-2.0 プロファイルでは、サウンドメディアデータの再生中断・再開・再生同期といった i アプリオプションのイベントタイプ、およびそれらに関連する API が追加されています。これらの詳細は、「i アプリコンテンツ開発ガイド i アプリオプション・i アプリ拡張編」および「i アプリコンテンツ開発ガイド オプション・拡張 API リファレンス編」を参照してください。

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 インタフェースおよび timerExpired() メソッドを実装します。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

4.4.1 プレゼンタの利用

i アプリ実行環境では、アニメーションを含む画像の再生やサウンドの再生を行う機能が提供されます。画像を再生するクラスとサウンド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。VisualPresenter では、標準的に再生可能な画像として GIF アニメーションをサポート
しています。
- 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。AudioPresenter では、標準的に再生可能なサウ
ンドとして i メロディ形式データをサポートしています。
- NTT ドコモの提供する DoJa-2.0 プロファイル対応 i アプリ開発環境では、AudioPresenter は i メ
ロディ形式データと MIDI 形式データをサポートしています。i アプリ開発環境を使用した実際の開
発手順などについては第 12 章を参照してください。

【DoJa-2.0】

NTT ドコモの提供する DoJa-1.0 プロファイル対応 i アプリ開発環境では、サウンドデータとして
i メロディ形式データを使用することはできません。DoJa-1.0 対応 i アプリ開発環境で使用可能
なサウンドデータは MIDI 形式データのみとなります。

次に、VisualPresenter クラスを使って画像を再生する例と、AudioPresenter ク
ラスを使ってサウンドを再生する例を示します。

この例では、画像ファイルやサウンドファイルとして、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 サーバー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

    List<String>  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.isIndexSelected(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: サウンドファイルの再生

```

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;   // サウンド再生ボタン
    Button          stopButton;   // サウンド停止ボタン
    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サウンド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 {

```

```

    /**
    * サウンドデータの再生/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) {
                // サウンド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.isIndexSelected(0))
                        ((AudioPresenter)source).play();
                    ;
                    break;

                // サウンド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;

                // サウンド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、サウンド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();
}
}

```

【DoJa-2.0】

- DoJa-1.0 プロファイルでは、メディアデータがセットされていないメディアプレゼンタに対し stop() メソッドを呼び出した際の振る舞いはメーカーに依存していました。これに対し DoJa-2.0 プロファイ

ルでは、メディアデータがセットされていないメディアプレゼンタに対し `stop()` メソッドを呼び出すと例外 (`UIException`) が発生します。メディアプレゼンタにメディアデータがセットされているかどうかは `MediaPresenter.getMediaResource()` メソッドを使用して判定してください。

- DoJa-2.0 プロファイルでは、`VisualPresenter` によるアニメーション GIF 再生について以下の規定が追加されています。
 - ・ 再生可能なフレーム数は、最低限 8 フレームが保証されます。
 - ・ ループ型アニメーション GIF を再生した場合、1 回のループが終了する毎に再生完了イベントが発生します。
 - ・ ループ型アニメーション GIF の最大ループ回数は 16 回です。これを超えたループ回数が指定されている場合でも、16 回ループした時点で再生は停止します。

4.4.2 複数のサウンドデータの同時再生

DoJa-2.0 プロファイルにおいては、アプリケーションは複数の `AudioPresenter` を取得し、それらに対して同時に再生指示を与えることができます。ただし、実際にいくつのサウンドを同時に再生することができるかは端末により異なります（各機種で共通に保証される数は 1 つです）。

アプリケーションから携帯電話の能力を超えた数のサウンドデータの再生を指示した場合、どのサウンドが有効なものとして再生されるかは、`AudioPresenter` に設定された優先順位属性 (`AudioPresenter.PRIORITY`) の値により決定されます。同時に 1 つのサウンドだけ再生可能な携帯電話において、優先順位の低いサウンドを再生中に優先順位の高いサウンドの再生が指示された場合、優先順位の高い方のサウンドが低い方のサウンドに割り込んで再生されます。優先順位の高い方のサウンドの再生が終了すると、低い方のサウンドの再生が再開されます。

なお、優先順位属性の設定には `AudioPresenter.setAttribute()` メソッドを使用します。設定可能な優先順位の値としては、高 (`AudioPresenter.MAX_PRIORITY`)、通常 (`AudioPresenter.NORM_PRIORITY`)、低 (`AudioPresenter.MIN_PRIORITY`) の 3 つが用意されています。なお、優先順位属性をサポートしていない端末では、後に再生したサウンドの方が再生中のサウンドより優先度が高いものとして再生中のサウンドに割り込んで再生されます。

4.4.3 サウンド再生の同期イベント

DoJa-2.0 プロファイルの `AudioPresenter` では、i アプリオプション API に属する機能としてサウンド再生における同期イベント発生機能を規定しています。同期イベントとは、サウンドの再生進行に同期してアプリケーション（メディアリスナー）に通知されるメディアイベントです。`AudioPresenter.setAttribute()` メソッドを使用して、同期イベント発生可否属性 (`AudioPresenter.SYNC_MODE`) に対しオン

（`AudioPresenter.ATTR_SYNC_ON`）またはオフ

（`AudioPresenter.ATTR_SYNC_OFF`）を指定することでイベントの発生有無を制御することができます。

同期イベント機能を利用する場合、サウンドデータを構成するうちの任意の 1 チャンネルをイベント発生源として指定します。指定されたチャンネルの指定されたノートメッセージに連動して、メディアリスナーに同期イベントが通知されます。

なお、イベント発生源として指定されたチャンネルも発音の対象となります。発音が不要な場合、サウンドデータ作成時においてそのチャンネルに対し消音指定を行うようにしてください。

第 5 章

通信の制御

i アプリ実行環境では、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）。i アプリ実行環境では、アプリケーションは HTTP (S) プロトコル（HEAD、GET、POST メソッド）を使用して Web サーバーと通信を行うことができます。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 アプリ実行環境では、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 アプリ実行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り込んでいます。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。クラスの数を見最小限に抑えるため、汎用的な `URLConnection` API は `HttpURLConnection` 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 アプリ実行環境では、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 アプリ API の `HttpURLConnection` 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。i アプリサービスでは、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 アプリ実行環境では、J2SE における `HttpURLConnection` 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.getOutputStream();  
  
        // 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.getInputStream();  
        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.getOutputStream() 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.getInputStream() 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必須ではなく必要に応じて行うものですが、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.getOutputStream() 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

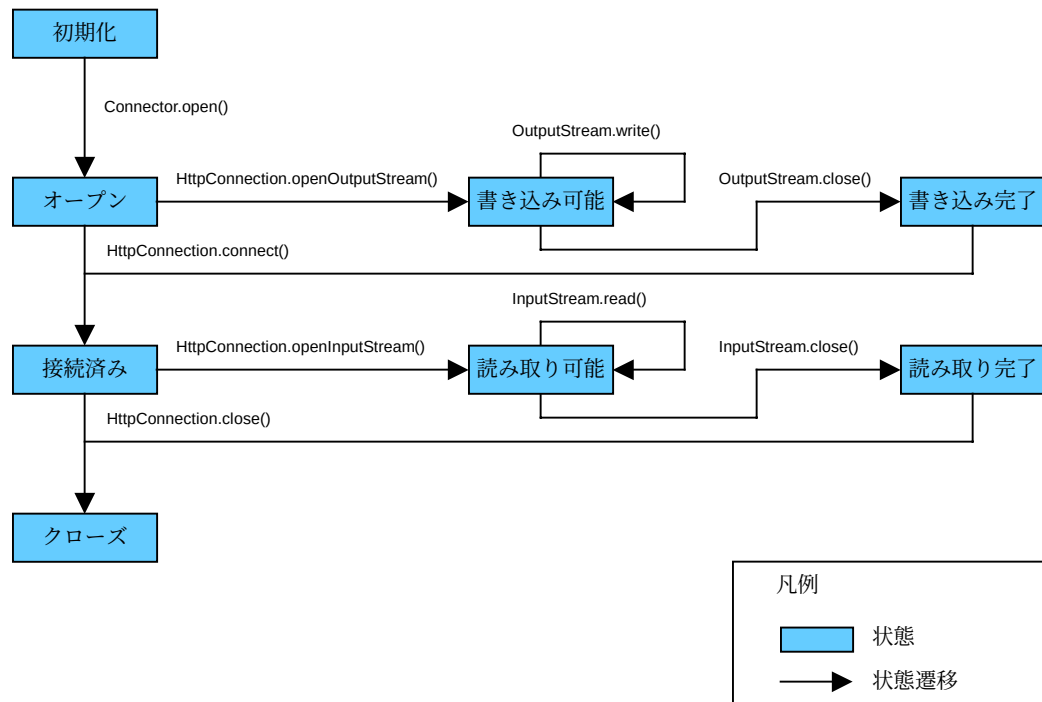


図 8: HttpURLConnection の状態遷移

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 キーでの指定）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接続可能な HttpURLConnection の数は 1 つだけです。
- i アプリサービスにおけるクライアント/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 モードサーバーのインターネット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 アプリ実行環境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

この制限は、アプリケーションが直接 `HttpConnection` を操作する場合だけでなく、メディアデータを HTTP 通信を介して取得する際にも適用されます。

- 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。アプリケーションからの HTTP(S)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。
- アプリケーションが通信を行っている状態でアプリケーションの中断（3.6 項参照）が発生すると、通信は中断されます。この場合、アプリケーションが再開された時点で例外が発生します。

【DoJa-2.0】

- DoJa-2.0 プロファイル対応の携帯電話では、外部との通信機能を一切停止して、通信を使用しない機能のみ利用可能とするセルフモード機能が搭載されます。ユーザーが携帯電話をセルフモードにしている状態では、アプリケーションの起動を行うことはできますが起動されたアプリケーションが HTTP(S)通信を行うことはできません。
- DoJa-2.0 プロファイルでは、HTTP(S)通信で指定する URL の長さとして、スキーム/パス部に 255 バイト、クエリー部に 512 バイトまで指定することができます。
- DoJa-1.0 プロファイルでは、HTTP レスポンスのステータスコードについて以下のように定義されています。

- `HttpConnection.HTTP_SERVER_ERROR` (500)
- `HttpConnection.HTTP_INTERNAL_ERROR` (501)

これに対し、DoJa-2.0 プロファイルでは

- `HttpConnection.HTTP_INTERNAL_ERROR` (500)
- `HttpConnection.HTTP_NOT_IMPLEMENTED` (501)

に改められます。なおコンパイル上の互換性を保持するため、DoJa-2.0 プロファイルでも `HttpConnection.HTTP_SERVER_ERROR` (500) は引き続き定義されます。

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。

なお DoJa-2.0 プロファイルでは、`connect()` メソッドが通信を行っている状態で、他のスレッドからその `HttpConnection` インスタンスに対し `close()` メソッドを呼び出すことで通信を切断させることができます。

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。これは、`Connector.open()`メソッドで取得した `Connection` 実装オブジェクトを使用した結果として、通信エラーなどにより例外が発生した場合でも同様です。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、クライアントが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。一方、特定のユーザーにのみ使用を許可するサーバーではユーザーの認証が必要になりますが、接続が行われるたびにアプリケーションからサーバーへログインすることは現実的ではありません。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を保つことが必要となります。

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 アプリ実行環境では 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 アプリ実行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 = (HttpURLConnection)Connector.open(url,
        Connector.READ_WRITE, true);
    conn.setRequestMethod(HttpURLConnection.POST);
    conn.setRequestProperty("Content-Type", "text/plain");
    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);
        conn.setRequestProperty("Content-Type", "text/plain");
        OutputStream out = conn.openOutputStream();
        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時か
        // 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、サーバー（backflip.sfbay）のポート 8080 番において HTTP プロトコルを使用したサービスが提供されており、またポート 7070 番で HTTPS プロトコルを使用した同様のサービスが提供されている場合、URL”http://backflip.sfbay:8080/”を”https://backflip.sfbay:7070/”で置き換えます。

注意事項：

- HttpURLConnection 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、i アプリ対応携帯電話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。i アプリ対応携帯電話では、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 セキュアサーバー・ルート証明書
 - 2) Baltimore 社
 - GTE CyberTrust Root CA
 - GTE CyberTrust Global Root

【DoJa-2.0】

DoJa-2.0 プロファイルでは、SSL 通信実行時に認証エラーなどの SSL エラーが発生した場合、`ConnectionException`（`SSL_ERROR`）がスローされます。例外ステータス `SSL_ERROR` は DoJa-2.0 プロファイルで追加されました。

第 6 章

テキスト処理

一般にアプリケーションのテキスト処理では、ASCII 文字セット以外に各国固有の文字セットも多く使用されます。i モードサービスでは日本語テキストの符号化方式として Shift-JIS を採用していますが、i アプリ実行環境では CLDC の国際化されたストリーム入出力機構を利用して、Shift-JIS の日本語テキストを容易に取り扱うことができます。

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。CLDC には J2SE と同様に国際化されたストリーム入出力機構が備えられており、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。次に、この変換プロセスを示します。

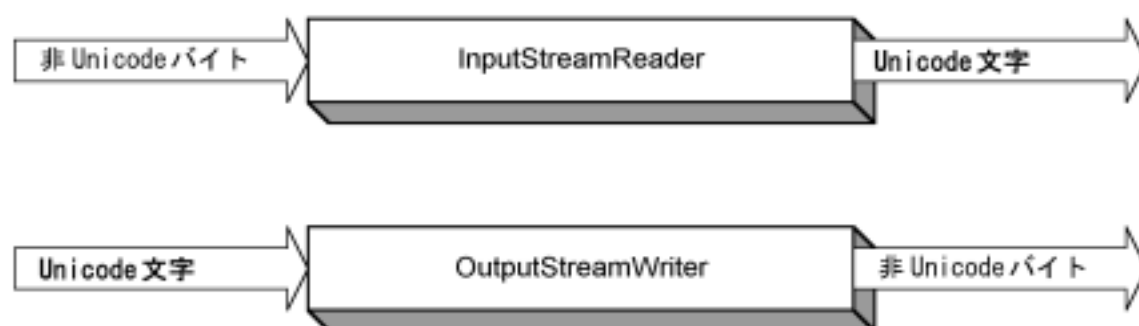


図 9: 入出力ストリームの変換

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。なお、i モードサービスでは日本語テキストの符号化方式として Shift-JIS (SJIS) を採用しており、これらのクラスのデフォルトエンコーディングも SJIS となっています。

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 バイトデータを Unicode 文字に変換

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

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

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

// SJIS バイトデータを Unicode 文字に変換する。
isr = new InputStreamReader (conn.openInputStream()); //デフォルトの符号化は SJIS
```

例: Unicode 文字を SJIS バイトデータに変換

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

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

// Unicode 文字を SJIS バイトデータに変換する。
osw = new OutputStreamWriter (conn.openOutputStream()); //デフォルトの符号化は SJIS
```

6.2 i アプリ対応携帯電話の絵文字

i アプリ対応携帯電話では、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。

<基本絵文字>

文字	SJIS	Unicode	タイトル	文字	SJIS	Unicode	タイトル
•	F89F	E63E	晴れ	•	F8CA	E669	ホテル
•	F8A0	E63F	曇り	•	F8CB	E66A	コンビニ
•	F8A1	E640	雨	•	F8CC	E66B	ガソリンスタンド
•	F8A2	E641	雪	•	F8CD	E66C	駐車場
•	F8A3	E642	雷	•	F8CE	E66D	信号
•	F8A4	E643	台風	•	F8CF	E66E	トイレ
•	F8A5	E644	霧	•	F8D0	E66F	レストラン
•	F8A6	E645	小雨	•	F8D1	E670	喫茶店
•	F8A7	E646	牡羊座	•	F8D2	E671	バー
•	F8A8	E647	牡牛座	•	F8D3	E672	ビール
•	F8A9	E648	双子座	•	F8D4	E673	ファーストフード
•	F8AA	E649	蟹座	•	F8D5	E674	ブティック
•	F8AB	E64A	獅子座	•	F8D6	E675	美容院
•	F8AC	E64B	乙女座	•	F8D7	E676	カラオケ
•	F8AD	E64C	天秤座	•	F8D8	E677	映画
•	F8AE	E64D	蠍座	•	F8D9	E678	右斜め上
•	F8AF	E64E	射手座	•	F8DA	E679	遊園地
•	F8B0	E64F	山羊座	•	F8DB	E67A	音楽
•	F8B1	E650	水瓶座	•	F8DC	E67B	アート
•	F8B2	E651	魚座	•	F8DD	E67C	演劇
•	F8B3	E652	スポーツ	•	F8DE	E67D	イベント
•	F8B4	E653	野球	•	F8DF	E67E	チケット
•	F8B5	E654	ゴルフ	•	F8E0	E67F	喫煙
•	F8B6	E655	テニス	•	F8E1	E680	禁煙
•	F8B7	E656	サッカー	•	F8E2	E681	カメラ
•	F8B8	E657	スキー	•	F8E3	E682	カバン
•	F8B9	E658	バスケットボール	•	F8E4	E683	本
•	F8BA	E659	モータースポーツ	•	F8E5	E684	リボン
•	F8BB	E65A	ポケットベル	•	F8E6	E685	プレゼント
•	F8BC	E65B	電車	•	F8E7	E686	パースデー
•	F8BD	E65C	地下鉄	•	F8E8	E687	電話
•	F8BE	E65D	新幹線	•	F8E9	E688	携帯電話
•	F8BF	E65E	車（セダン）	•	F8EA	E689	メモ
•	F8C0	E65F	車（RV）	•	F8EB	E68A	TV
•	F8C1	E660	バス	•	F8EC	E68B	ゲーム
•	F8C2	E661	船	•	F8ED	E68C	CD
•	F8C3	E662	飛行機	•	F8EE	E68D	ハート
•	F8C4	E663	家	•	F8EF	E68E	スベード
•	F8C5	E664	ビル	•	F8F0	E68F	ダイヤ
•	F8C6	E665	郵便局	•	F8F1	E690	クラブ
•	F8C7	E666	病院	•	F8F2	E691	目
•	F8C8	E667	銀行	•	F8F3	E692	耳
•	F8C9	E668	ATM	•	F8F4	E693	手（グー）

文字	SJIS	Unicode	タイトル	文字	SJIS	Unicode	タイトル
•	F8F5	E694	手 (チョキ)	•	F984	D6DF	フリーダイヤル
•	F8F6	E695	手 (パー)	•	F985	E6E0	シャープダイヤル
•	F8F7	E696	右斜め下	•	F986	E6E1	モバQ
•	F8F8	E697	左斜め上	•	F987	E6E2	1
•	F8F9	E698	足	•	F988	E6E3	2
•	F8FA	E699	くつ	•	F989	E6E4	3
•	F8FB	E69A	眼鏡	•	F98A	E6E5	4
•	F8FC	E69B	車椅子	•	F98B	E6E6	5
•	F940	E69C	新月	•	F98C	E6E7	6
•	F941	E69D	やや欠け月	•	F98D	E6E8	7
•	F942	E69E	半月	•	F98E	E6E9	8
•	F943	E69F	三日月	•	F98F	E6EA	9
•	F944	E6A0	満月	•	F990	E6EB	0
•	F945	E6A1	犬	•	F991	E6EC	黒ハート
•	F946	E6A2	猫	•	F992	E6ED	揺れるハート
•	F947	E6A3	リゾート	•	F993	E6EE	失恋
•	F948	E6A4	クリスマス	•	F994	E6EF	ハートたち (複数ハート)
•	F949	E6A5	左斜め下	•	F995	E6F0	わーい (嬉しい顔)
•	F950	E6AC	カチンコ	•	F996	E6F1	ちっ (怒った顔)
•	F951	E6AD	ふくろ	•	F997	E6F2	がく〜 (落胆した顔)
•	F952	E6AE	ペン	•	F998	E6F3	もうやだ〜 (悲しい顔)
•	F955	E6B1	人影	•	F999	E6F4	ふらふら
•	F956	E6B2	いす	•	F99A	E6F5	グッド (上向き矢印)
•	F957	E6B3	夜	•	F99B	E6F6	るんるん
•	F958	E6B7	soon	•	F99C	E6F7	いい気分 (温泉)
•	F95C	E6B8	on	•	F99D	E6F8	かわいい
•	F95D	E6B9	end	•	F99E	E6F9	キスマーク
•	F95E	E6BA	時計	•	F99F	E6FA	びかびか (新しい)
•	F972	E6CE	phone to	•	F9A0	E6FB	ひらめき
•	F973	E6CF	mail to	•	F9A1	E6FC	むかつ (怒り)
•	F974	E6D0	fax to	•	F9A2	E6FD	パンチ
•	F975	E6D1	i モード	•	F9A3	E6FE	爆弾
•	F976	E6D2	i モード (粹付き)	•	F9A4	E6FF	ムード
•	F977	E6D3	メール	•	F9A5	E700	バッド (下向き矢印)
•	F978	E6D4	ドコモ提供	•	F9A6	E701	眠い (睡眠)
•	F979	E6D5	ドコモポイント	•	F9A7	E702	exclamation
•	F97A	E6D6	有料	•	F9A8	E703	exclamation&question
•	F97B	E6D7	無料	•	F9A9	E704	exclamation×2
•	F97C	E6D8	ID	•	F9AA	E705	どんっ (衝撃)
•	F97D	E6D9	パスワード	•	F9AB	E706	あせあせ (飛び散る汗)
•	F97E	E6DA	次項有	•	F9AC	E707	たらーっ (汗)
•	F980	E6DB	クリア	•	F9AD	E708	ダッシュ (走り出すさま)
•	F981	E6DC	サーチ (調べる)	•	F9AE	E709	ー (長音記号 1)
•	F982	E6DD	NEW	•	F9AF	E70A	ー (長音記号 2)
•	F983	D6DE	位置情報	•	F9B0	E70B	決定

<拡張絵文字>

文字	SJIS	Unicode	タイトル	文字	SJIS	Unicode	タイトル
•	F9B1	E70C	i アプリ	•	F9D7	E732	トレードマーク
•	F9B2	E70D	i アプリ (枠付き)	•	F9D8	E733	走る人
•	F9B3	E70E	T シャツ (ボーダー)	•	F9D9	E734	マル秘
•	F9B4	E70F	がま口財布	•	F9DA	E735	リサイクル
•	F9B5	E710	化粧	•	F9DB	E736	レジスタードトレードマーク
•	F9B6	E711	ジーンズ	•	F9DC	E737	危険・警告
•	F9B7	E712	スノボ	•	F9DD	E738	禁止
•	F9B8	E713	チャペル	•	F9DE	E739	空室・空席・空車
•	F9B9	E714	ドア	•	F9DF	E73A	合格マーク
•	F9BA	E715	ドル袋	•	F9E0	E73B	満室・満席・満車
•	F9BB	E716	パソコン	•	F9E1	E73C	矢印左右
•	F9BC	E717	ラブレター	•	F9E2	E73D	矢印上下
•	F9BD	E718	ランチ	•	F9E3	E73E	学校
•	F9BE	E719	鉛筆	•	F9E4	E73F	波
•	F9BF	E71A	王冠	•	F9E5	E740	富士山
•	F9C0	E71B	指輪	•	F9E6	E741	クローバー
•	F9C1	E71C	砂時計	•	F9E7	E742	さくらんぼ
•	F9C2	E71D	自転車	•	F9E8	E743	チューリップ
•	F9C3	E71E	湯のみ	•	F9E9	E744	バナナ
•	F9C4	E71F	腕時計	•	F9EA	E745	りんご
•	F9C5	E720	考えてる顔	•	F9EB	E746	芽
•	F9C6	E721	ほっとした顔	•	F9EC	E747	もみじ
•	F9C7	E722	冷や汗	•	F9ED	E748	桜
•	F9C8	E723	冷や汗 2	•	F9EE	E749	おにぎり
•	F9C9	E724	ぶっくっく顔	•	F9EF	E74A	ショートケーキ
•	F9CA	E725	ボケーとした顔	•	F9F0	E74B	とっくり (おちょこ付き)
•	F9CB	E726	目がハート	•	F9F1	E74C	どんぶり
•	F9CC	E727	指で OK	•	F9F2	E74D	パン
•	F9CD	E728	あっかんべー	•	F9F3	E74E	かたつむり
•	F9CE	E729	ウィンク	•	F9F4	E74F	ひよこ
•	F9CF	E72A	うれしい顔	•	F9F5	E750	ペンギン
•	F9D0	E72B	がまん顔	•	F9F6	E751	魚
•	F9D1	E72C	猫 2	•	F9F7	E752	うまい!
•	F9D2	E72D	泣き顔	•	F9F8	E753	ウッシッシ
•	F9D3	E72E	涙	•	F9F9	E754	ウマ
•	F9D4	E72F	NG	•	F9FA	E755	ブタ
•	F9D5	E730	クリップ	•	F9FB	E756	ワイングラス
•	F9D6	E731	コピーライト	•	F9FC	E757	げっそり

注意事項：

- 拡張絵文字をサポートしていない携帯電話では、拡張絵文字はスペースなどに置換されて表示されます。どのような文字に置換されるかは機種により異なります。

第 7 章

ScratchPad とリソースの使用

この章では、i アプリ対応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（画像ファイルやサウンドファイルなど）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

アプリケーションは、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 アプリ実行環境では、アプリケーション終了後もデータを保存し続け、次にアプリケーションが実行されたときにそれを再び利用できるようにするための、ScratchPad と呼ばれる特別な記憶領域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、12.5.1 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 (IOException e) {
        // 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`、`Connector.WRITE`、`Connector.READ_WRITE` の各モードが指定できます。
`Connector.READ_WRITE` モードでオープンした ScratchPad からは入力ストリーム、出力ストリームのいずれも取得できます。ただし、1つの `StreamConnection` について、入力ストリームと出力ストリームを同時に有効な状態にすることはできません。一方をオープンする際には、必ず他方がクローズされた状態となるようにしてください。

【DoJa-2.0】

`Connector.READ_WRITE` での ScratchPad のオープンは、DoJa-2.0 プロファイルでのみ可能です。DoJa-1.0 プロファイルでは、ScratchPad を `Connector.READ_WRITE` でオープンした際の動作はメーカーにより異なります。

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サウンド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画像ファイルやサウンドファイルを、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 アプリ実行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。

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。

【DoJa-2.0】

DoJa-1.0 プロファイルでは、ScratchPad の末尾を越えて読み取りを行った場合には `EOFException` が発生することが規定されています。これに対し DoJa-2.0 プロファイルでは、ScratchPad の末尾を越えて読み取りを行った場合の動作も、CLDC で定義されている各ストリームクラスのメソッドの動作規定に従います。例えば、`InputStream.read()` メソッドは、ScratchPad の末尾を越えた位置のバイト読み取りを指示されると -1 を返します。

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、ハードウェア・リソースの属性値を参照する `getAttribute()` メソッド、および特定のハードウェア・リソースをその機種上で制御可能かテストする `isAvailable()` 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 アプリ実行環境で規定されているハードウェア・リソースタイプには以下のようなものがあります。

- ・ 画面のバックライト（オンまたはオフに切り替え可能）
- ・ バイブレータ（オンまたはオフに切り替え可能）
- ・ 機体の折りたたみ状態（属性値の参照のみ）
- ・ メール・メッセージ受信状態（属性値の参照のみ）

これらの属性は全て `PhoneSystem` クラスで定義されています。

また `PhoneSystem` クラスには、携帯電話にあらかじめ組み込まれた効果音を発音させるための `playSound()` メソッドが用意されています。効果音の種類は `PhoneSystem` クラスに定義されていますが、実際に発音される音はメーカーにより異なります。

なお、`PhoneSystem` クラスで定義されている属性や機能の一部は i アプリオプション API のカテゴリに含まれます。i アプリオプション API に含まれる機能は、メーカーにより搭載されない場合がありますので注意してください。

【DoJa-2.0】

DoJa-1.0 プロファイルにおいては、`PhoneSystem` クラスで制御・参照可能なハードウェア・リソースは画面のバックライトのみです。

注意事項：

- アプリケーションから取得できるメール・メッセージ受信状態は携帯電話のディスプレイ上のメール・メッセージ受信アイコンの表示内容に連動します。アプリケーションがメール・メッセージの受信状態を参照するためには、ADF の `GetSysInfo` キーにてそのアプリケーションが携帯電話のアイコン情報を参照することを宣言し、かつユーザーがそのアプリケーションに対しアイコン情報の参照を許可する必要があります。ADF の記述方法については、12.5.1 項を参照してください。なお、メール・メッセージ受信状態は携帯電話のメール・メッセージ受信アイコンに連動するため、電波状況により携帯電話がメール・メッセージ着信の通知を受けられないようなケースでは必ずしも正しい結果が得られない場合がありますのでご注意ください。

- 画面のバックライトやバイブレータなど、電力消費の大きいハードウェアリソースをアプリケーションが長時間オンにし続けた場合、機種によっては省電力のためシステムにより自動的にオフの状態に戻されます。`PhoneSystem.getAttribute()`メソッドの返すハードウェアリソース状態には、このようなシステムによる自動的な状態変更の結果も反映されます。
- 待ち受けアプリケーションが非活性化状態の場合、そのアプリケーションからバックライトの制御を行うことはできません。
- 携帯電話の設定により、携帯電話のキー操作が無効化されている状態では、アプリケーションはバイブレータの制御を行うことはできません。

第 9 章

待ち受けアプリケーション

DoJa-2.0 プロファイルでは携帯電話の待ち受け画面に、カレンダーや画像などを表示するかわりにシステムに常駐したアプリケーションの実行画面を表示する、待ち受けアプリケーション機能をサポートします。DoJa-1.0 プロファイルでは、アプリケーションを常駐させておきたいという要求に対しては、通常のアプリケーションを常に実行し続けるという形で解決する必要がありました。しかしこの方法では、アプリケーションおよび携帯電話機能自身の操作性を低下させるいくつかの問題があります。DoJa-2.0 プロファイルの待ち受けアプリケーション機能では、このような問題点を解決し、より操作性のよい常駐型アプリケーションを実現するための機構が採用されています。

9.1 待ち受けアプリケーションの概要

9.1.1 待ち受けアプリケーションの特徴

待ち受けアプリケーション機能を使用して常駐アプリケーションを作成した場合、DoJa-1.0 プロファイルの機能の範囲で常駐アプリケーションを作成した場合と比較して以下のようなメリットがあります。

操作性の向上

通常のアプリケーションを実行している状態では、携帯電話のキー操作はアプリケーションのイベントとして受信されます。例えば、携帯電話は待ち受け状態で数字キーを押されるとネイティブオペレーティングシステムのユーザーインタフェース機構（以降ネイティブ OS とします）がイベントを捕捉して自動的に音声発信を行うためのモードに移りますが、アプリケーション実行中は数字キーの操作は全てキーイベントとしてアプリケーションに配送されます。従って、従来の手法で作成された常駐アプリケーションのユーザーは、電話を掛けたり Web ブラウザを使用する場合には、手動操作で一旦アプリケーションを終了させる必要がありました。また、それらの作業が終了した後は、手動操作で再度アプリケーションを起動し直さなければなりません。

これに対し待ち受けアプリケーション機能では、アプリケーションがキーイベントを受け取るモード（活性化状態と呼びます）と携帯電話のネイティブ OS がキーイベントを受け取るモード（次項参照）をユーザーが簡単なキー操作で切り替えることができます。ユーザーは、通常はネイティブ OS がキーイベントを受け取るモードでアプリケーションを常駐させておき、必要の都度モードを切り替えてアプリケーションと対話する、というスタイルで待ち受けアプリケーションを利用します。

なお、待ち受けアプリケーションが常駐している状態でユーザーが音声発信や Web ブラウザなどの他のネイティブ機能を起動すると、アプリケーションは自動的に終了または中断します。またユーザーがそれらの作業を終了して再度待ち受け状態に復帰する際、アプリケーションは自動的に再起動または再開されます。

電力消費の抑制

通常のアプリケーションを実行していたり待ち受けアプリケーションが活性化状態である場合には、アプリケーションの実行時間はそのまま電力消費に結びつき、長時間アプリケーションを駆動させ続けることは困難です。待ち受けアプリケーションでは活性化状態の他に、

- ・ キーイベントの発生がアプリケーションに通知されないだけで、アプリケーション処理の続行は可能な非活性化状態
- ・ キーイベントの発生がアプリケーションに通知されず、アプリケーションの処理も停止(スリープ)して余分な電力消費を最低限に抑える休眠状態

の2つを使い分けることで、長時間駆動が可能な常駐アプリケーションを設計することができます。

休眠状態ではアプリケーションロジックの実行は停止しますが、特定のシステムイベント（一定時間が経過した場合など）が発生した際に非活性化状態に復帰し、再びアプリケーションロジックを実行できるようになります。

メール・メッセージ着信のリアルタイム化

通常のアプリケーションを実行している状態では、メールやメッセージの着信通知を受けても、アプリケーションの実行を優先させるためそれらの受信は行いません。このため、従来の手法で作成された常駐アプリケーションのユーザーは、メール・メッセージの着信に気づかないことがありました。

待ち受けアプリケーションでは、活性化状態では通常のアプリケーション実行中と同様メール・メッセージの着信通知を受けてもそれらの受信は行いませんが、非活性化状態または休眠状態では受信処理を行います（その際アプリケーションの実行は中断されます）。

なお DoJa-2.0 プロファイルに対応した携帯電話では、活性化状態および通常のアプリケーションを実行している状態でも、メール・メッセージの着信通知を受けた際は画面上のマークなどによりリアルタイムにユーザーに通知することが規定されています。

待ち受けアプリケーションの例として、次のような株価アラームを考えます。

1. ユーザーは、あらかじめ関心のある銘柄の株価をいくつか選択し、それぞれの銘柄でアラームを発生させるための株価条件を設定しておく。
2. アプリケーションは、通常は1時間に1度、選択された銘柄の最新の株価をサーバーから取得して画面に表示する。株価条件を超えた銘柄が発生した場合、アプリケーションはユーザーに音やバイブレーターでアラームを伝える。
3. アラームの通知を受けたユーザーは、キー操作によりアプリケーションと対話して該当する銘柄の詳細情報を見ることができる。

この例では、2.の処理は非活性化状態および休眠状態で構成することができます。アプリケーションは、非活性化状態の時にサーバーとの通信や画面の更新を行い、その後1時間の間休眠状態に入ることを繰り返します。この処理を行っている間はユーザーのキー操作は全てネイティブ OS の方に伝えられるため、例えばユーザーが数字キーを押すと携帯電話は音声発信のモードに移行します。

また3.の処理は、ユーザー操作によりアプリケーションを活性化状態に移行させることで実現します。活性化状態ではユーザーのキー操作はアプリケーションに伝えられるため、ユーザーとアプリケーションの対話を処理することができます。

注意事項：

- DoJa-2.0 プロファイルではユーザーによるモード切り替えキー（以降、切り替えキーと呼びます）操作は、非活性化状態または休眠状態から活性化状態への遷移のみサポートします。その他の状態遷移については、アプリケーションからメソッドを呼び出すことで行います。
- 切り替えキーのキーアサインはメーカーにより異なります。

9.1.2 待ち受けアプリケーションのユーザー設定

待ち受けアプリケーションは、通常のアプリケーションのようにユーザーが手動操作で起動することも、待ち受けアプリケーションとして携帯電話に登録することによりシステムに常駐させることもできます。システムに常駐したアプリケーションは、ユーザー操作により音声発信や Web ブラウザなどの他のネイティブ機能が起動されると自動的に終了または中断します。またユーザーがそれらの作業を終了して再度待ち受け状態に復帰する際、アプリケーションは自動的に再起動または再開されます。このような起動形態を、通常のアプリケーション起動と区別して「待ち受け起動」と呼びます。

ユーザーが待ち受けアプリケーションをダウンロードしてから待ち受け設定を行い、実際に待ち受け起動されるまでの流れはおおよそ以下の通りです。なお、ここで示す流れは典型的なものであり、携帯電話の操作方法の詳細はメーカーにより異なります。

1. 待ち受けアプリケーションのダウンロード

待ち受けアプリケーションは、通常のアプリケーションと同様の手順でダウンロードします。ブラウザでアプリケーションダウンロード用のリンクをクリックすると、まず ADF の内容がチェックされ、携帯電話上にインストール可能であると判断されれば JAR ファイルがダウンロードされます（詳細は 1.4.1 項を参照してください）。

なお、ここでダウンロードされる ADF には、そのアプリケーションが待ち受けアプリケーションであることの宣言（MyConcierge キー）が含まれます。MyConcierge キーおよび ADF については、9.2.4 項および 12.5.1 項を参照してください。

2. アプリケーション設定

アプリケーションがダウンロードされた後、ADF の内容に応じてアプリケーション設定を行います。アプリケーション設定の項目には、待ち受け設定、ネットワーク許可設定、メール・メッセージ受信状態参照許可設定があります。ADF にて待ち受けアプリケーションであることが宣言されている場合、ユーザーは待ち受け設定を行うことができます。

待ち受けアプリケーションを待ち受け起動可能とするには、待ち受け設定にてそのアプリケーションを携帯電話に登録します。なお、複数のアプリケーションに対し、同時に待ち受け起動可能とすることはできません。また、ユーザーはここで設定した待ち受け設定内容を後で変更することもできます。

3. 待ち受け状態への復帰と待ち受けアプリケーションの起動

2.によりアプリケーションが待ち受け設定された後、ユーザーの終話キー操作などにより携帯電話が待ち受け状態に復帰すると、自動的に待ち受けアプリケーションが待ち受け起動されます。

注意事項：

- 「待ち受け起動」に対する通常の起動とは、待ち受け起動以外の全ての起動形態を指します。具体的には、以下が通常起動に該当します。
 - ・ ユーザーのアプリケーション一覧などのメニュー操作によりアプリケーションが起動される場合
 - ・ ADF の LaunchAt キー（12.5.1 項参照）によりアプリケーションがタイマー起動される場合
 - ・ ネイティブアプリケーション連携（11 章参照）によりアプリケーションが起動される場合

9.2 待ち受けアプリケーションの作成

この項では、待ち受けアプリケーションを作成する上で、通常のアプリケーションを作成する場合と異なる点、留意すべき点について解説します。

9.2.1 待ち受けアプリケーションクラス

待ち受けアプリケーションでは、アプリケーションのメインクラス（ADF の AppClass キーに指定するクラス）は `MApplication` を継承して作成します。`MApplication` クラスは `com.nttdocomo.ui.IApplication` を継承したクラスであり、通常のアプリケーションのライフサイクル制御に加えて待ち受けアプリケーションの複雑なライフサイクルを制御するためのメソッドを備えています。

`MApplication` クラスでは、`IApplication` クラスに加え以下のようなメソッドを使用することができます。

- 状態遷移制御

- ・ `deactivate()` : 活性化状態から非活性化状態に遷移します。
- ・ `sleep()` : 非活性化状態から休眠状態に遷移します。

- システムイベント制御

- ・ `setWakeupTimer()` : ウェイクアップタイマーを開始します。
- ・ `getWakeupTimer()` : ウェイクアップタイマーの残り時間を取得します。
- ・ `resetWakeupTimer()` : ウェイクアップタイマーを解除します。
- ・ `setClockTick()` : 時計イベントの発生有無を設定します。
- ・ `processSystemEvent()` : システムイベントを処理するためのハンドラメソッドです。

`MApplication` を継承したアプリケーションは待ち受け起動だけでなく、通常のアプリケーションと同様にユーザーの手動操作やタイマー起動などによっても起動することが可能です。しかしその場合、アプリケーションはこれらのメソッドを呼び出すことはできません。待ち受け起動されていないアプリケーションがこれらのメソッドを呼び出した場合、例外が発生します。

待ち受けアプリケーションのライフサイクルおよび状態遷移については 9.2.2 項で、システムイベントについては 9.2.3 項で解説します。

9.2.2 待ち受けアプリケーションのライフサイクルと状態遷移

以下に、待ち受けアプリケーションの取る 3 つの状態について解説します。

1. 活性化状態

通常のアプリケーションを実行している時と同様に、アプリケーションがキーイベントおよびソフトキーイベントを受信することのできる状態です。ソフトキーイベントの処理がアプリケーションに委ねられるため、ソフトキーラベルの表示もアプリケーション側で行います。

2. 非活性化状態

アプリケーションは動作を継続しており、画面の表示もアプリケーションの指示通り行われますが、キーイベントおよびソフトキーイベントはアプリケーションには通知されず、ネイティブ OS に通知されます。ソフトキーイベントの処理はネイティブ OS 側に委ねられるため、アプリケーションはソフトキーラベルを設定することはできません。

待ち受けアプリケーションが待ち受け起動された直後、アプリケーションの状態は非活性化状態になっています。

3. 休眠状態

電池の消費を抑えるため、アプリケーションの動作は停止しています。画面には、アプリケーションが休眠状態に入る直前の描画内容が保持されています。休眠状態への遷移は非活性化状態からのみ可能であり、活性化状態から休眠状態に遷移することはできません。

注意事項：

- 非活性化状態ではキーイベントはネイティブ OS 側に通知されますが、画面の表示は待ち受けアプリケーションに委ねられています。従って、非活性化状態のアプリケーションがユーザーにキー操作を促すような画面表示を行っていると、ユーザーが誤った操作を行う可能性もあり得ます。使いやすい待ち受けアプリケーションを作成するためには、非活性化状態における画面表示内容についても留意する必要があります。
- アプリケーション実行中には、アプリケーションではなく、ネイティブコンポーネントである JAM がダイアログを表示することがあります。
 - ・ HTTP 通信時に、BASIC 認証により ID・パスワードが求められた場合
 - ・ HTTPS 通信時に、サーバーの証明書が完全には安全ではないと判断され、ユーザーに継続可否を確認する必要がある場合アプリケーションが非活性化状態にある場合、JAM はダイアログを表示することはできません。その場合は例外が発生しますので注意してください。

アプリケーション実行中は、9.2.1 項で解説した状態遷移制御メソッドの呼び出しや各種システムイベントの発生などによりこれら 3 つの状態の間の遷移が行われます。

以下に、活性化状態、非活性化状態、休眠状態の間の状態遷移の様子を図示します。

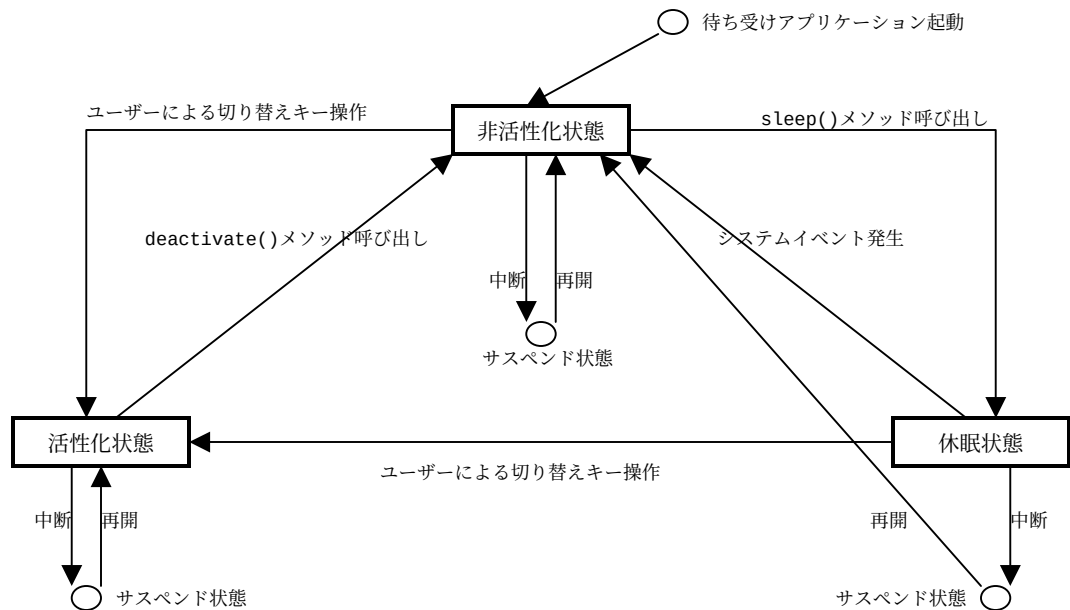


図 10: 待ち受けアプリケーションにおける各状態間の状態遷移

活性化状態、非活性化状態、休眠状態の間の状態遷移について以下に解説します。

1. 非活性化状態から活性化状態への遷移

アプリケーションが非活性化状態のときに、ユーザーにより切り替えキー操作が行われると活性化状態へ遷移します。このとき、待ち受けアプリケーションにはシステムイベント `MODE_CHANGED_EVENT` が通知されます。

2. 活性化状態から非活性化状態への遷移

活性化状態のアプリケーションが、`MApplication.deactivate()` メソッドを呼び出すことで非活性化状態へ遷移します。非活性化状態のアプリケーションがこのメソッドを呼び出しても何もしません。

なお、非活性化状態および休眠状態では切り替えキー操作はキーイベントとしてアプリケーションに通知されませんが、活性化状態では切り替えキー操作はキーイベント (`KEY_IAPP`) としてアプリケーションに通知されます。アプリケーションは、このキーイベントを受けて `MApplication.deactivate()` メソッドを呼び出すようにすることで、ユーザーは切り替えキーを状態切り替えのためのトグルキーのように使用することができます。その際、`MApplication.deactivate()` メソッドの呼び出しは `KEY_IAPP` のリリースイベントを受けて行うようにしてください。`KEY_IAPP` のプレスイベントを受けてこのメソッドを呼び出すようにしている場合、メーカーによってはプレスイベントに続いて発生するリリースイベントをシステムが受信し、再度活性化状態に戻ってしまうことがあります。

3. 非活性化状態から休眠状態への遷移

非活性化状態のアプリケーションが、`MApplication.sleep()` メソッドを呼び出すことで休眠状態へ遷移します。その際、イベントキューに未処理のイベントが蓄えられている場合はそれらは全て破棄されます。休眠状態に入ったアプリケーションは、システムイベントが発生するまで停止します。`sleep()` メソッドを呼び出したスレッドは、システムイベントが発生すると `sleep()` メソッドから復帰し、それに続くロジックの処理を再開します。

特定の時間が経過した際に非活性化状態に復帰させたい場合は、`sleep()` メソッドの前に `MApplication.setWakeupTimer()` メソッドを呼び出します。またネイティブの時計の時刻変化に合わせて非活性化状態に復帰させたい場合は、あらかじめ `MApplication.setClockTick()` メソッドにより時計イベントを有効化しておく必要があります。

なお、休眠状態に入るとアプリケーションのスレッドは全て停止します。イベント処理や描画メソッド (`Canvas.paint()`) など、アプリケーションと非同期で実行される可能性のあるスレッドの実行中に休眠状態に入ると、それらのスレッドも停止しますので注意してください。

4. 休眠状態から活性化状態への遷移

アプリケーションが休眠状態のときに、ユーザーにより切り替えキー操作が行われると活性化状態へ遷移します。このとき、待ち受けアプリケーションにはシステムイベント `MODE_CHANGED_EVENT` が通知されます。`sleep()` メソッドを呼び出したスレッドは、休眠状態から活性化状態に遷移すると `sleep()` メソッドから復帰し、それに続くロジックの処理を再開します。`sleep()` メソッドからの復帰は、`MODE_CHANGED_EVENT` の通知に先立って行われます。

なお、システムイベントの詳細については 9.2.3 項を参照してください。

5. 休眠状態から非活性化状態への遷移

アプリケーションが休眠状態のときに、`MODE_CHANGED_EVENT` 以外のシステムイベントが発生すると非活性化状態へ遷移します。`sleep()` メソッドを呼び出したスレッドは、休眠状態から非活性化状態に遷移すると `sleep()` メソッドから復帰し、それに続くロジックの処理を再開します。`sleep()` メソッドからの復帰は、システムイベントの通知に先立って行われます。

なお、システムイベントの詳細については 9.2.3 項を参照してください。

活性化状態から休眠状態へ遷移することはできません。活性化状態のアプリケーションが `sleep()` メソッドを呼び出すと例外が発生します。活性化状態から休眠状態に遷移させたい場合には、`deactivate()` メソッドにより一旦非活性化状態に遷移した後で `sleep()` メソッドを呼び出すようにしてください。

待ち受けアプリケーションは、自身が現在どのような状態にあるかについて把握する必要があります。アプリケーションに `MODE_CHANGED_EVENT` が通知される契機、および

アプリケーションが `deactivate()` メソッドを呼び出す契機にて、自身の状態を適切に管理するようにしてください。

各状態からのアプリケーションの中断は、以下のような契機で発生します。中断されたアプリケーションが再開される際、通常のアプリケーションの実行が再開される場合と同様に `MApplication.resume()` メソッドが呼び出されます。

状態	アプリケーション中断が発生する契機
活性化状態	携帯電話が音声着信を受けた場合
	そのアプリケーションがネイティブアプリケーション連携機能のメソッドを呼び出した場合(*1)
非活性化状態 ・休眠状態	携帯電話が音声着信またはメール・メッセージ着信を受けた場合
	ユーザー操作により、音声発信が行われた場合
	ユーザー操作により、パケット通信を使用しないネイティブアプリケーションが起動された場合

(*1)`MApplication.launch()`により、ブラウザを起動しようとした場合を除きます。待ち受けアプリケーションはブラウザを起動することはできません。従ってこの場合は、アプリケーションの中断は発生しません。

中断時のアプリケーションの状態が活性化状態または非活性化状態であれば、再開後の状態は中断前の状態と同じとなります。これに対し、中断時のアプリケーションの状態が休眠状態の場合、再開後の状態は非活性化状態に変化しますので注意してください。休眠状態にあるアプリケーションが中断・再開を受けると、`MApplication.sleep()` メソッドでブロックされているスレッドは `sleep()` メソッドから復帰し、後続の処理を再開します。

また、各状態にある待ち受けアプリケーションは、以下のような契機で終了します。終了した待ち受けアプリケーションは、再度待ち受け状態に復帰した際、自動的に再起動されます。

状態	アプリケーションが終了する契機
活性化状態	そのアプリケーションから <code>MApplication.terminate()</code> が呼び出された場合
	強制終了キーが押された場合
	接続ケーブルにより外部機器が接続された場合
	そのアプリケーションの内部で、キャッチされない例外・エラーが発生した場合(*1)
非活性化状態	そのアプリケーションから <code>MApplication.terminate()</code> が呼び出された場合
	ユーザーのメニュー操作により実行中待ち受けアプリケーションの強制終了が指示された場合
	ユーザー操作により、パケット通信を使用するネイティブアプリケーション（ブラウザ、メール）または他の i アプリが起動された場合
	接続ケーブルにより外部機器が接続された場合
	そのアプリケーションの内部で、キャッチされない例外・エラーが発生した場合(*1)
休眠状態	ユーザーのメニュー操作により実行中待ち受けアプリケーションの強制終了が指示された場合
	ユーザー操作により、パケット通信を使用するネイティブアプリケーション（ブラウザ・メール）または他の i アプリが起動された場合
	接続ケーブルにより外部機器が接続された場合

(*1)アプリケーションの内部で発生した、キャッチされない例外・エラーにより待ち受けアプリケーションが終了する場合、その携帯電話の待ち受けアプリケーション設定は解除されます。つまり、キャッチされない例外・エラーの発生により待ち受けアプリケーションが終了した場合、そのアプリケーションが待ち受けアプリケーション

として自動的に再起動されることはありません。待ち受けアプリケーションは、自身のライフサイクルで発生する例外を適切にハンドリングする必要があります。

なお、待ち受けアプリケーションがキャッチされない例外・エラーにより終了した場合、いつアプリケーションが終了したかなどを示す情報が携帯電話に記録されます。ユーザーは、携帯電話のメニュー操作などによりこの記録を参照することができます。

表のように、実行中の待ち受けアプリケーションはさまざまな外部要因により終了する可能性があります。保存する必要のあるデータが生じた際は、その都度 ScratchPad またはサーバーに保存するようにしてください。

9.2.3 システムイベント

待ち受けアプリケーションでは、ライフサイクルを制御する特殊なイベント（システムイベント）をハンドリングすることができます。システムイベントは、アプリケーションのメインクラス（`MApplication` のサブクラス）のインスタンスに対し、`processSystemEvent()` メソッドで通知されます。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システムイベント処理を実装することができます。このメソッドの第 1 引数で発生したイベントの種別が、第 2 引数で発生したイベントに付随するイベントパラメータが通知されます。

システムイベントのイベント種別は以下の通りです。`MApplication` クラスでは、これらのイベント種別を示す定数が定義されています。

● `MODE_CHANGED_EVENT`

ユーザーにより切り替えキー操作が行われ、非活性化状態または休眠状態から、活性化状態への遷移が行われたことを示すイベントです。他の状態遷移ではこのイベントは発生しません。

このイベントでは、イベントパラメータは使用しません。

● `CLOCK_TICK_EVENT`

ネイティブの時計の時刻が変化したことを示すイベントです。DoJa-2.0 プロファイルでは、分単位のイベント発生をサポートします。つまり、ネイティブの時計の時刻が毎分正秒（00 秒）を指した時にこのイベントが発生します。このイベントを発生させるためには、`MApplication.setClockTick()` メソッドを呼び出してこのイベントを有効化する必要があります。

このイベントは、待ち受けアプリケーションがどの状態にある場合でも発生します。

イベントパラメータには、その日の 0 時 0 分 0 秒からの経過秒数が渡されます。

● `WAKEUP_TIMER_EVENT`

ウェイクアップタイマー（アプリケーションが設定した時間が経過したことをシステムイベントとして通知するタイマー）がエクスパイアしたことを示すイベントです。ウェイクアップタイマーは待ち受けアプリケーションが中断・再開されても停止しません。ただし、中断されている期間でタイマーがエクスパイアされた場合、イベントの通知はアプリケーションの実行が再開されるまで遅延します。

ウェイクアップタイマーは、`MApplication.setWakeupTimer()` メソッドを呼び出すことで動作を開始します。また、`MApplication.resetWakeupTimer()` メソッドを呼び出すと、動作中のウェイクアップタイマーを停止させることができます。ウェイクアップタイマーが動作している状態で `MApplication.getWakeupTimer()` メソッドを呼び出すと、その時点からウェイクアップタイマーがエクスパイアするまでの時間を取得することができます。

このイベントは、待ち受けアプリケーションがどの状態にある場合でも発生します。

このイベントが発生した際、時間通りにイベントが発生した場合には値 0 が、アプリケーションの実行中断により遅延してイベントが発生した場合には値 1 が、それぞれイベントパラメータとして渡されます。

● `FOLD_CHANGED_EVENT`

折りたたみ型またはフリップ型の携帯電話において、折りたたみ状態またはフリップの開閉状態が変化したことを示すイベントです。このイベントは折りたたみ型またはフリップ型の携帯電話でのみ発生します。

このイベントは、待ち受けアプリケーションがどの状態にある場合でも発生します。

このイベントが発生した際、閉じた状態に変化した場合には値 0 が、開いた状態に変化した場合には値 1 が、それぞれイベントパラメータとして渡されます。

9.2.4 ADF による待ち受けアプリケーションの宣言

待ち受けアプリケーションは、ADF の MyConcierge キーを使用して、自身が待ち受けアプリケーションであることを宣言（値 "Yes" を設定）する必要があります。このキーが設定されていないアプリケーションは、待ち受けアプリケーションとして携帯電話に登録することはできません。

なお、通常のアプリケーションとして携帯電話にダウンロードされているアプリケーションを、新たに MyConcierge キーが設定された ADF を使用してバージョンアップすることはできません。またその逆に、待ち受けアプリケーションとして携帯電話にダウンロードされているアプリケーションを、新たに MyConcierge キーが削除された ADF を使用してバージョンアップすることはできません。このような場合は、携帯電話にインストールされたアプリケーションを一旦削除し、再度新たなアプリケーションとしてダウンロードし直す必要があります。

9.3 待ち受けアプリケーション実行中の他機能の起動

待ち受けアプリケーションの実行中に携帯電話の他機能を起動させようとする場合、待ち受けアプリケーションが活性化状態でなければ原則的に他機能の方が優先されます。以下に、待ち受けアプリケーション実行中に他機能を起動しようとした場合の、待ち受けアプリケーションの振る舞いを示します。

起動させる機能	待ち受けアプリケーションの振る舞い	
	活性化状態の場合	非活性化状態または休眠状態の場合
ユーザー操作による他機能の起動		
ブラウザ起動	不可 (*1)	待ち受けアプリケーションを終了し、ブラウザを起動します。ユーザーが作業を終えて待ち受け状態に戻ると、待ち受けアプリケーションは再起動されます。
メール起動	不可 (*1)	待ち受けアプリケーションを終了し、メールを起動します。ユーザーが作業を終えて待ち受け状態に戻ると、待ち受けアプリケーションは再起動されます。
音声発信	不可 (*1)	待ち受けアプリケーションを中断し、音声発信動作に移ります。通話が終了するとアプリケーションは再開され、非活性化状態になります。
i アプリ起動	不可 (*1)	待ち受けアプリケーションを終了し、指定された i アプリを起動します。ユーザーが作業を終えて待ち受け状態に戻ると、待ち受けアプリケーションは再起動されます。
ネイティブアプリケーション連携による、待ち受けアプリケーション内部からの他機能の起動		
ブラウザ起動	不可 (*2)	不可 (*2)
音声発信	待ち受けアプリケーションを中断し、音声発信動作に移ります。通話が終了するとアプリケーションは再開され、活性化状態になります。	不可 (*3)
上記以外の方法による他機能の起動		
音声着信	待ち受けアプリケーションを中断し、音声着信動作に移ります。通話が終了するとアプリケーションは再開され、活性化状態になります。	待ち受けアプリケーションを中断し、音声着信動作に移ります。通話が終了するとアプリケーションは再開され、非活性化状態になります。
メール・メッセージ着信	メール・メッセージの受信は行わず、待ち受けアプリケーションの実行を継続します。携帯電話は画面にマークを表示し、i モードサーバーに未受信のメールがあることをユーザーに知らせます。	待ち受けアプリケーションを中断し、メール・メッセージ受信動作に移ります。受信が終了するとアプリケーションは再開され、非活性化状態になります。 ただし、非活性化状態のアプリケーションが HTTP 通信処理を行っている状態でメール・メッセージが着信した場合は、活性化状態での動作に準じます。
i アプリのタイマー起動	i アプリのタイマー起動は行わず、待ち受けアプリケーションの実行を継続します。ユーザーはメニュー操作などにより、タイマー起動できなかった i アプリがあることを知ることができます。	待ち受けアプリケーションを終了し、i アプリをタイマー起動します。タイマー起動された i アプリが終了して待ち受け状態に戻ると、待ち受けアプリケーションは再起動されます。

(*1) 待ち受けアプリケーションが活性化状態である場合、キーイベントは待ち受けアプリケーションに通知されるため、ユーザーはこれらの操作を行うことはできません。

(*2) 待ち受けアプリケーションからは、ネイティブアプリケーション連携機能 (UIApplication.launch()メソッド) を使用してアプリケーションからブラウザを起動することはできません。

(*3) 非活性化状態のアプリケーションは、ネイティブアプリケーション機能 (Phone.call()メソッド) を使用してアプリケーションから音声通話機能を起動することはできません。

9.4 i アプリ API 利用上の注意点

待ち受けアプリケーションが待ち受け起動されている場合、その特性から、i アプリ API を利用する上で通常のアプリケーションとは異なる注意点があります。本項では、これらの注意点について解説します。

9.4.1 ユーザーインタフェース

- フレームに設定されたソフトキーラベルは、アプリケーションが非活性化状態または休眠状態にある間は表示されません。これらの状態では、ソフトキーラベル表示領域はネイティブ OS が使用します。アプリケーションが活性化状態に移行した際に、カレントフレームに設定されているソフトキーラベルがソフトキーラベル表示領域に表示されます。

- Panel 使用時、非活性化状態ではアプリケーションからフォーカスに関する操作（フォーカスマネージャの設定やコンポーネント間のフォーカス移動）を行うことはできません。非活性化状態でアプリケーションがフォーカス操作のメソッドを呼び出しても、そのメソッドは何もせずアプリケーションに復帰します。
- Canvas 使用時、非活性化状態ではキーパッドの状態を取得(`Canvas.getKeyPadState()`メソッド)することはできません。非活性化状態でアプリケーションがこのメソッドを呼び出すと、何もキーが押されていない状態を示す値 0 が返されます。
- 高レベルイベント、低レベルイベントそれぞれについて、非活性化状態の待ち受けアプリケーションへの通知の有無は以下の通りです。原則的にキー操作を伴うイベントは、非活性化状態の待ち受けアプリケーションには通知されません。

高レベルイベント

イベントの種類	通知の有無	備考
コンポーネントイベント		
BUTTON_PRESSED	×	
SELECTION_CHANGED	○	ListBox クラスの <code>select()</code> 、 <code>deselect()</code> メソッドにより選択状態が変化した場合に通知されます。
TEXT_CHANGED	○	TextBox クラスの <code>setText()</code> メソッドによりテキストが再設定された場合に通知されます。
キーイベント	×	
ソフトキーイベント	×	
メディアイベント	○	
タイマーイベント	○	

低レベルイベント

イベントの種類	通知の有無	備考
キープレスイベント	×	ソフトキー操作を含みます。
キーリリースイベント	×	ソフトキー操作を含みます。
レジュームイベント	○	
アップデートイベント	○	
タイマーイベント	○	

- Ticker は、活性化状態および非活性化状態ではスクロール動作を行います。休眠状態に入るとスクロール動作は停止します。休眠状態から活性化状態または非活性化状態に遷移することで、再度スクロール動作が行われるようになります。
- プレゼンタ (VisualPresenter、AudioPresenter) オブジェクトは、活性化状態および非活性化状態では再生処理を行います。再生中のプレゼンタオブジェクトはアプリケーションが休眠状態に入ると停止します。その際、再生停止イベントは発生しません。また、アプリケーションが休眠状態から活性化状態または非活性化状態に遷移しても、自動的に再生を再開することはありません。
- タイマー (Timer および ShortTimer) は、アプリケーションが活性化状態および非活性化状態にある場合に動作します。タイマーが動作しているアプリケーションが休眠状態に遷移すると、使用中のタイマーは停止します (`stop()`メソッドが呼び出された状態)。その後、アプリケーションが休眠状態から活性化状態または非活性化状態に遷移しても、自動的にタイマーの動作を再開することはありません。

9.4.2 入出力

- 非活性化状態のアプリケーションが HTTP(S)通信を行っている場合、音声着信を受けた場合だけではなく、ユーザー操作により音声発信が行われた場合にも通信が中断されます。この場合、音声着信により通信が中断された場合と同様に、通信処理のリトライを行う必要があります。アプリケーション中断、再開が与える影響については、3.6 項を併せて参照してください。
- アプリケーションは、活性化状態および非活性化状態の場合のみ HTTP(S)通信を行うことができます。あるスレッドが HTTP(S)通信を行っているときに他のスレッドが `MApplication.sleep()` メソッドを呼び出して休眠状態に入ると、その通信は中断され例外が発生します。その後、アプリケーションが休眠状態から他の状態に復帰すると、通信を行っていたスレッドは通信の例外処理から動作を再開します。
- ScratchPad が不揮発メモリにデータを書き込んでいるときに他のスレッドが `MApplication.sleep()` メソッドを呼び出して休眠状態に入ろうとすると、実際の休眠状態への遷移は ScratchPad への書き込みが完了するまで遅延します。なお、ScratchPad をオープンしたままアプリケーションが休眠状態に移行した場合は、活性化状態または非活性化状態に復帰した後も ScratchPad はオープンされたままの状態を保っており、継続してデータの書き込みまたは読み込みを行うことができます。
- OBEX 外部接続による外部機器との通信は、アプリケーションが活性化状態にある場合でのみ可能です。OBEX 外部接続により通信を行っているアプリケーションが非活性化状態に遷移すると、通信は中断され例外が発生します。これは、アプリケーションが OBEX クライアントとして動作している場合、OBEX サーバーとして動作している場合の双方に適用されます。

9.4.3 ハードウェア制御

- 非活性化状態にあるアプリケーションは、バックライト制御を行うことはできません。非活性化状態にあるアプリケーションが `PhoneSystem.setAttribute()` メソッドでバックライト状態の変更を指示しても、このメソッドは何も行わず復帰します。
- 活性化状態にあるアプリケーションはバックライト制御を行うことができますが、アプリケーションにより点灯されたバックライトは、一定時間経過後システムにより自動的に消灯されます。
- バイブレータ制御は、アプリケーションが活性化状態または非活性化状態にある場合に行うことができます。バイブレータがオンになっている状態でアプリケーションが休眠状態に遷移すると、バイブレータは停止します。アプリケーションが活性化状態や非活性化状態に復帰しても、自動的にバイブレータ動作を再開することはありません。
- その他、ユーザーがメニュー操作などで状態を設定できるハードウェアリソースは、原則的にアプリケーションから状態を変更しようとしてもユーザーの設定が優先されます。

9.4.4 ネイティブアプリケーション連携

- 待ち受け起動されたアプリケーションは、どのような状態であっても `IApplication.launch()` メソッドを使用してブラウザを起動することはできません。待ち受け起動されたアプリケーションがブラウザを起動しようとした場合は例外が発生します。
- 非活性化状態のアプリケーションは、`Phone.call()` メソッドを使用して音声発信機能を起動することはできません。非活性化状態のアプリケーションが音声発信機能を起動しようとした場合は例外が発生します。

第 10 章

OBEX 外部接続

DoJa-2.0 プロファイルでは、赤外線ポートを介して i アプリと外部機器を接続し、通信を行うための OBEX 外部接続機能が標準でサポートされます。外部機器には DoJa-2.0 プロファイルに対応した携帯電話自身も含まれ、また DoJa-2.0 プロファイルでは OBEX 外部接続におけるクライアント API およびサーバー API の両方をサポートするため、携帯電話間で名刺やスケジュールなどの小さなデータを交換しあうアプリケーションを容易に開発することができます。

本章では、i アプリ実行環境における OBEX クライアント API、OBEX サーバー API、および携帯電話以外の外部機器をターゲットとしたシステムを開発する際のヒントについて解説します。

なお、OBEX 外部接続機能は IrDA 規格に基づいて実装されますが、本書では IrDA 規格そのものに関する解説は行いません。これらの規格に関する詳細は、Infrared Data Association の制定している各種規格書を参照してください。なお、DoJa-2.0 プロファイルにおける OBEX 外部接続機能は、以下のバージョンの各規格に準拠しています。

- Infrared Data Association "Link Manager Protocol version 1.1"
- Infrared Data Association "Serial Infrared Physical Layer Specification version 1.3"
- Infrared Data Association "Tiny TP: A Flow-Control Mechanism for use with IrLMP version 1.1"
- Infrared Data Association "Object Exchange Protocol IrOBEX version 1.2"
- Infrared Data Association "Specification for Ir Mobile Communications(IrMC) version 1.1"

10.1 OBEX によるデータ送受信

OBEX (IrOBEX) は、赤外線ポートを介したデータ送受信の手順を「オブジェクトの交換」という形式で規格化したものです。ここでいうオブジェクトとは、一塊となることで意味を持つ、例えばファイルなどの一般的なデータ実体のことを指します。

OBEX 自体はデータ送受信の手順（プロトコル）を規格化したものであり API を規格化したものではありません。DoJa-2.0 プロファイルでは、Generic Connection フレームワーク上に、OBEX によるデータ送受信を取り扱うための API を規定しています。

OBEX の仕様は HTTP を参考として策定されており、OBEX クライアントが送信するリクエストに対して OBEX サーバーがレスポンスを返すクライアント/サーバーモデルに従っています。OBEX クライアントが OBEX サーバーにオペレーション（HTTP における

メソッドに相当) の処理を要求し、OBEX サーバーはその結果としてレスポンスステータスや、処理結果としてのオブジェクトの内容を OBEX クライアントに返します。クライアントとサーバーの間でオブジェクトを送受するためのオペレーションとして、i アプリ実行環境はアプリケーションに以下の 2 つを提供します。

- ・ GET オペレーション： OBEX クライアントが OBEX サーバーにオブジェクト送信を要求する
- ・ PUT オペレーション： OBEX クライアントから OBEX サーバーにオブジェクトを送信する

プロトコル自体がバイナリ形式で表現されることを除き、リクエストおよびレスポンスがヘッダやオブジェクト本体（ボディ）から構成されること、および規定されているレスポンスステータスの内容などは HTTP と類似しています。

OBEX 規格では、赤外線通信の packets 長の制限などに起因するオブジェクトの分割や再構成といった処理を行うための詳細な手順が規定されています。しかし i アプリ実行環境の提供する API がこれらの複雑な規定を隠蔽するため、開発者はより容易にアプリケーションを開発することができます。

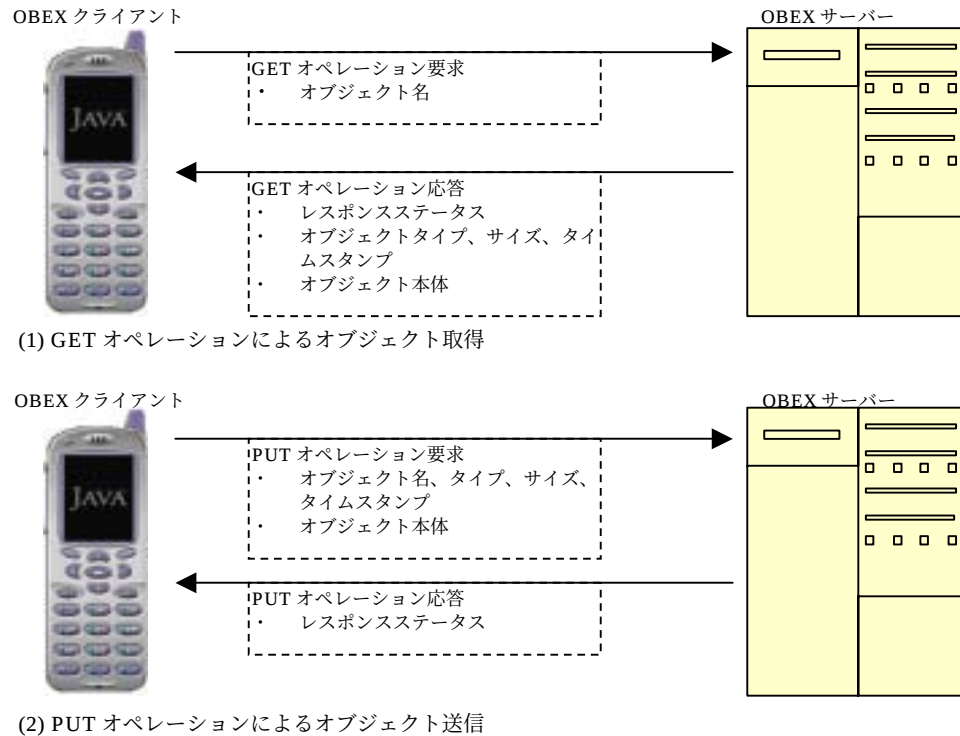


図 11: OBEX クライアントと OBEX サーバーの利用イメージ

10.2 OBEX 外部接続 API

OBEX 外部接続 API は、Generic Connection フレームワーク上に構築された、i アプリ独自の API セットとして提供されます。OBEX 外部接続 API の構成を以下に図示します。

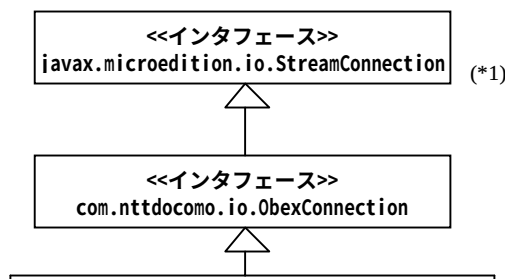


図 12: OBEX 外部接続 API の構成

ObexConnection インタフェースは OBEX クライアントと OBEX サーバーで共通に使用される定数やアクセスメソッドを定義するインタフェースであり、それを継承するサブインタフェースとして ClientObexConnection および ServerObexConnection が用意されています。これら 2 つのサブインタフェースでは、OBEX クライアントまたは OBEX サーバーに固有のメソッドが定義されています。

OBEX 外部接続で使用するコネクションオブジェクトは HTTP 通信や ScratchPad の場合と同様 Connector.open() メソッドを使用して取得しますが、このコネクションオブジェクトは ClientObexConnection または ServerObexConnection を実装しています。Connector.open() メソッドに指定する URL は、OBEX クライアントアプリケーションと OBEX サーバーアプリケーションでは異なります。

以降、OBEX クライアント API および OBEX サーバー API それぞれについて解説します。

10.2.1 OBEX クライアント API

OBEX クライアントが使用するコネクションオブジェクトは ClientObexConnection インタフェースを実装しています。このコネクションオブジェクトを取得するには、アプリケーションは URL 文字列 "obex:/irclient" を指定して Connector.open() メソッドを呼び出します。

以下に、アプリケーションで ClientObexConnection 実装インスタンスを取得、使用する例を示します。

例：ClientObexConnection の利用

```
ClientObexConnection coc;
int response;
try {
    // ClientObexConnection インスタンスを取得し、OBEX サーバーに接続する
    coc = (ClientObexConnection)Connector.open("obex:/irclient",
                                                Connector.READ_WRITE,
                                                true);

    coc.connect();

    // GET オペレーションの実行
    coc.setOperation(ObexConnection.GET);           // オペレーションに GET を設定
    coc.setName("userdata_1");                     // GET 対象オブジェクトの名前を設定
    coc.sendRequest();                               // オペレーションを送信

    // OBEX サーバーから返却されたレスポンスコードをチェックし、正常であれば受信した
```

```

// オブジェクト本体を読み込む
response = coc.getResponseCode();
if (response==ObexConnection.SUCCESS) {
    InputStream in = coc.openInputStream();
    // GET オペレーションで要求したデータの読み込み処理
    :
    in.close();
} else {
    throw new IOException();
}

// PUT オペレーションの実行
coc.setOperation(ObexConnection.PUT);           // オペレーションに PUT を設定
coc.setName("userdata_2");                       // PUT 対象オブジェクトの名前を設定
coc.setType("text/plain");                       // PUT 対象オブジェクトのタイプを設定
coc.setTime(System.currentTimeMillis());         // PUT 対象オブジェクトのタイムスタンプ
                                                    // を設定

OutputStream out = coc.openOutputStream();
// PUT オペレーションで送信するデータの書き込み処理
:
out.close();
coc.sendRequest();                               // オペレーションを送信
// レスポンスコードが正常であれば ClientObexConnection を閉じて処理終了
response = coc.getResponseCode();
if (response!=ObexConnection.SUCCESS) {
    throw new IOException();
}
coc.close();

} catch (IOException ioe) {
    // 例外処理
    // API がスローする IOException、ConnectionException、
    // アプリケーションがスローする IOException の処理を行う
    :
}

```

アプリケーションは `ClientObexConnection` を使用する際に、まず `connect()` メソッドを呼び出して OBEX サーバーとの間に赤外線通信のリンクを確立します。いったん確立された通信リンク上では、複数のオペレーションを連続して処理することができます。前出のサンプルコードでは、1 回確立した通信リンク上で GET オペレーションと PUT オペレーションを 1 回ずつ処理し、その後 `ClientObexConnection.close()` を呼び出して通信リンクを切断しています。

OBEX クライアントでは、オペレーションの処理は以下の手順に従って行います。

1. `setOperation()` メソッドを使用してオペレーション種別を設定します。
2. `ObexConnection` で定義されているアクセスメソッドを使用して、OBEX リクエストヘッダの設定を行います。アクセスメソッドを使用して設定することのできる OBEX リクエストヘッダには、Name ヘッダ、Type ヘッダ、Time ヘッダがあります。
3. PUT オペレーションの場合、送信するオブジェクトボディの内容を書き込みます。この操作は、コネクションオブジェクトの `openOutputStream()` または `openDataOutputStream()` メソッドで取り出した出力ストリームを使用して行います。
4. `sendRequest()` メソッドを使用してオペレーション処理要求を OBEX サーバーに送信します。このメソッドは OBEX サーバーから応答が返されるまでブロックします。このメソッド呼び出しから復帰すると、必要に応じて OBEX サーバーから返却された OBEX レスポンスコードや OBEX レスポンスヘッダの内容を参照することができます。OBEX クライアントが参照することのできる OBEX レスポンスヘッダには、Name ヘッダ、Type ヘッダ、Time ヘッダがあります。また、OBEX レスポンスコードは `ObexConnection` で定義されています。`ObexConnection` で定義されている OBEX レスポンスコードは、OBEX 規格におけるレスポンスコードの規定に準拠しています。各コードの意味などの詳細については OBEX 規格を参照してください。OBEX サーバーが OBEX

クライアントの要求を正常に処理できた場合、一般的には OBEX クライアントとして動作しているアプリケーションは、OBEX レスponseコードとして i アプリ実行環境 (CridentObexConnection.getResponseCode()) から ObexConnection.SUCCESS を受け取ります。

5. GET オペレーションの場合、返却されたオブジェクトボディの内容を読み出します。この操作は、コネクションオブジェクトの openInputStream() または openDataInputStream() メソッドで取り出した入力ストリームを使用して行います。オブジェクトボディの長さは、コネクションオブジェクトの getContenLength() メソッドを使用して取得することができます。

OBEX クライアントの処理は、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。例えばオペレーション処理要求を OBEX サーバーに送信していない状態で OBEX レスponseヘッダを取得しようとするると例外が発生します。

次の図に、ClientObexConnection の状態遷移を示します。

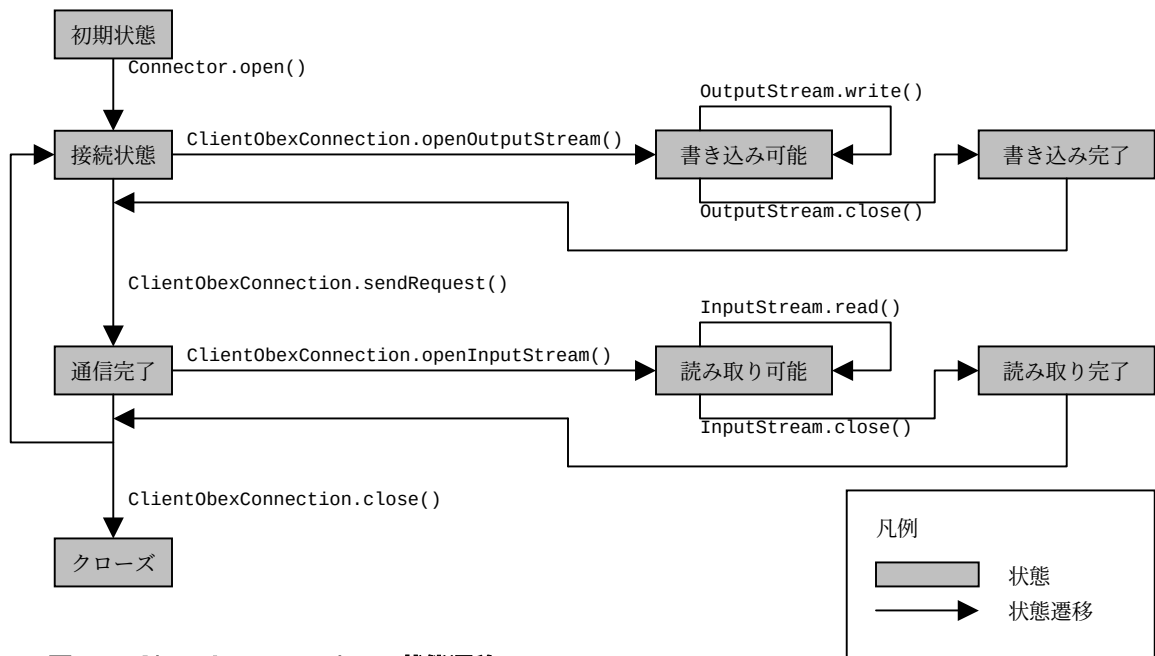


図 13: ClientObexConnection の状態遷移

注意事項：

- OBEX 外部接続においては、Connector クラスから直接 openInputStream()、openDataInputStream()、openOutputStream()、openDataOutputStream() の各メソッドにより入出力ストリームを取り出すことはできません。アプリケーションは Connector.open() メソッドで ClientObexConnection オブジェクトを取得し、そのオブジェクトから入出力ストリームを取り出す必要があります。
- 携帯電話の赤外線ポートへの通電は、アプリケーションから ClientObexConnection.connect() を呼び出した際に、システムにより自動的に行われます。
- OBEX サーバーに対し 1 度のオペレーションで、オブジェクトを送信するとともに OBEX サーバーからオブジェクトを受信することはできません。OBEX クライアントは、PUT オペレーションではオブジェクトの送信、GET オペレーションではオブジェクトの受信のみ行うことができます。
- OBEX クライアントにおけるコネクションオブジェクトのオープンモードは、READ モードが GET オペレーションに、WRITE モードが PUT オペレーションに対応します。1 回の接続で GET、PUT の両オペレーションを使用する場合は、READ_WRITE モードを使用するようにしてください。
- 1 つのアプリケーションから同時に確立可能な通信リンクは 1 つだけです。

- あるスレッドが `connect()` メソッドや `sendRequest()` メソッドを呼び出して OBEX サーバーのアクションを待っている時、他のスレッドが `ClientObexConnection` オブジェクトの `close()` メソッドを呼び出すと通信は切断されます。また、タイムアウト発生やユーザーの中断操作といった要因により、通信が切断されることがあります。
- GET、PUT の両オペレーションとも、1 回につき 30K バイトまでのオブジェクトを取り扱うことができます。この制限を越えたサイズのオブジェクトを送受信することはできません。
- アプリケーションが設定したオペレーション種別や OBEX ヘッダ、オブジェクトデータの内容は、1 回のオペレーションでのみ有効です。1 回の接続で連続してオペレーションを実行する場合、その都度これらを設定しなおすようにしてください。
- OBEX サーバーから返却されたレスポンスコードが異常を示す値 (0x20 番台以外のコード) であった場合、i アプリ実行環境は例外をスローします。
- DoJa-2.0 プロファイル対応の携帯電話では、外部との通信機能を一切停止して、通信を使用しない機能のみ利用可能とするセルフモード機能が搭載されます。ユーザーが携帯電話をセルフモードにしている状態では、アプリケーションの起動を行うことはできますが起動されたアプリケーションが OBEX 外部接続を行うことはできません。
- アプリケーションが赤外線通信を行っている間は、i アプリ実行環境は通信を実行中である旨のメッセージを表示します。また通信リンクの確立が行えなかった場合や、タイムアウト、ユーザー操作などにより通信が中断された場合には、i アプリ実行環境はその旨をユーザーに伝えるメッセージを表示します。これらのメッセージを非表示とすることはできません。

10.2.2 OBEX サーバーAPI

OBEX サーバーが使用するコネクションオブジェクトは `ServerObexConnection` インタフェースを実装しています。このコネクションオブジェクトを取得するには、アプリケーションは URL 文字列 "obex:/irserver" を指定して `Connector.open()` メソッドを呼び出します。

以下に、アプリケーションで `ServerObexConnection` 実装インスタンスを取得、使用する例を示します。

例：ServerObexConnection の利用

```
ServerObexConnection soc;
try {
    // ServerObexConnection インスタンスを取得する
    soc = (ServerObexConnection)Connector.open("obex:/irserver",
                                                Connector.READ_WRITE,
                                                true);

    // OBEX クライアントからの通信リンク確立要求を待ち合わせる
    soc.accept();

    // OBEX クライアントからのオペレーション要求を処理する
    while (true) {

        soc.receiveRequest();                // オペレーション要求を待つ
        int operation = soc.getOperation(); // 到着したオペレーションの種別を取得する

        if (operation==ObexConnection.GET) { // GET オペレーション要求
            // 要求されたオブジェクトの名前を取得
            String name = soc.getName();
            // 指定された名前に対応するオブジェクトのヘッダ情報を設定
            soc.setType("text/plain");
            soc.setTime(System.currentTimeMillis());
            // 指定された名前に対応するオブジェクトの書き込み処理
            OutputStream out = soc.openOutputStream();
            :
            out.close();
            // 正常終了を示すステータスコードを OBEX クライアントに返す
        }
    }
}
```

```

        soc.sendResponse(ObexConnection.SUCCESS);

    } else if (operation==ObexConnection.PUT) { // PUT オペレーション要求
        // 送信されたオブジェクトの各種情報を取得
        String name = soc.getName();
        String type = soc.getType();
        long time = soc.getTime();
        // 送信されたオブジェクトの読み出し処理
        InputStream in = soc.openInputStream();
        :
        in.close();
        // 正常終了を示すステータスコードを OBEX クライアントに返す
        soc.sendResponse(ObexConnection.SUCCESS);

    } else if (operation==ObexConnection.DISCONNECT) { // 切断要求
        break;
    }
}

// 通信リンクを切断し、通信を終了する
soc.close();

} catch (IOException ioe) {
    // 例外処理
    // API がスローする IOException、ConnectionException の処理を行う
    :
}

```

アプリケーションは `ServerObexConnection` を使用する際に、まず `accept()` メソッドを呼び出して OBEX クライアントからの赤外線通信のリンク確立要求 (OBEX クライアントの `connect()` メソッド実行) を待ちます。いったん確立された通信リンク上では、複数のオペレーションを連続して処理することができます。前出のサンプルコードでは、1 回確立した通信リンク上で、OBEX クライアントからの切断要求を受け付けるまで繰り返しオペレーションの処理を行っています。

OBEX サーバーでは、オペレーションの処理は以下の手順に従って行います。

1. `receiveRequest()` メソッドを使用して要求を受け付けます。このメソッドは、OBEX クライアントからの要求を受け付けるまでブロックします。
2. `getOperation()` メソッドを使用して要求のオペレーション種別を取得します。オペレーション種別として `ObexConnection.DISCONNECT` を受け取った場合は、OBEX クライアントが通信リンクの開放を通知したことを示しています。この要求を受け取った場合、OBEX サーバーもコネクションオブジェクトをクローズするようにしてください。
3. `ObexConnection` で定義されているアクセスメソッドを使用して、OBEX リクエストヘッダの内容を取得します。アクセスメソッドを使用して取得することのできる OBEX リクエストヘッダには Name ヘッダ、Type ヘッダ、Time ヘッダがあります。
4. PUT オペレーションの場合、OBEX クライアントから送信されたオブジェクトボディの内容を読み出します。この操作は、コネクションオブジェクトの `openInputStream()` または `openDataInputStream()` メソッドで取り出した入力ストリームを使用して行います。オブジェクトボディの長さは、コネクションオブジェクトの `getContentLength()` メソッドを使用して取得することができます。
5. `ObexConnection` で定義されているアクセスメソッドを使用して、OBEX レスポンスヘッダの内容を設定します。アクセスメソッドを使用して設定することのできる OBEX レスポンスヘッダには Name ヘッダ、Type ヘッダ、Time ヘッダがあります。
6. GET オペレーションの場合、OBEX クライアントが要求しているオブジェクトのオブジェクトボディを書き込みます。この操作は、コネクションオブジェクトの `openOutputStream()` または `openDataOutputStream()` メソッドで取り出した出力ストリームを使用して行います。

7. `sendResponse()` メソッドを使用して、OBEX クライアントにレスポンスを返します。このメソッドの引数には、`ObexConnection` で定義されている OBEX レスポンスコードを指定します。なお、`ObexConnection` で定義されている OBEX レスポンスコードは、OBEX 規格におけるレスポンスコードの規定に準拠しています。各コードの意味などの詳細については OBEX 規格を参照してください。OBEX サーバーが OBEX クライアントの要求を正常に処理できた場合、一般的には OBEX サーバーとして動作しているアプリケーションは、OBEX レスポンスコードとして `iアプリ実行環境 (ServerObexConnection.sendResponse())` に `ObexConnection.SUCCESS` を返します。

OBEX サーバーの処理は、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。例えばオペレーション処理要求を受け付けていない状態で OBEX リクエストヘッダを取得しようとするなど例外が発生します。

次の図に、`ServerObexConnection` の状態遷移を示します。

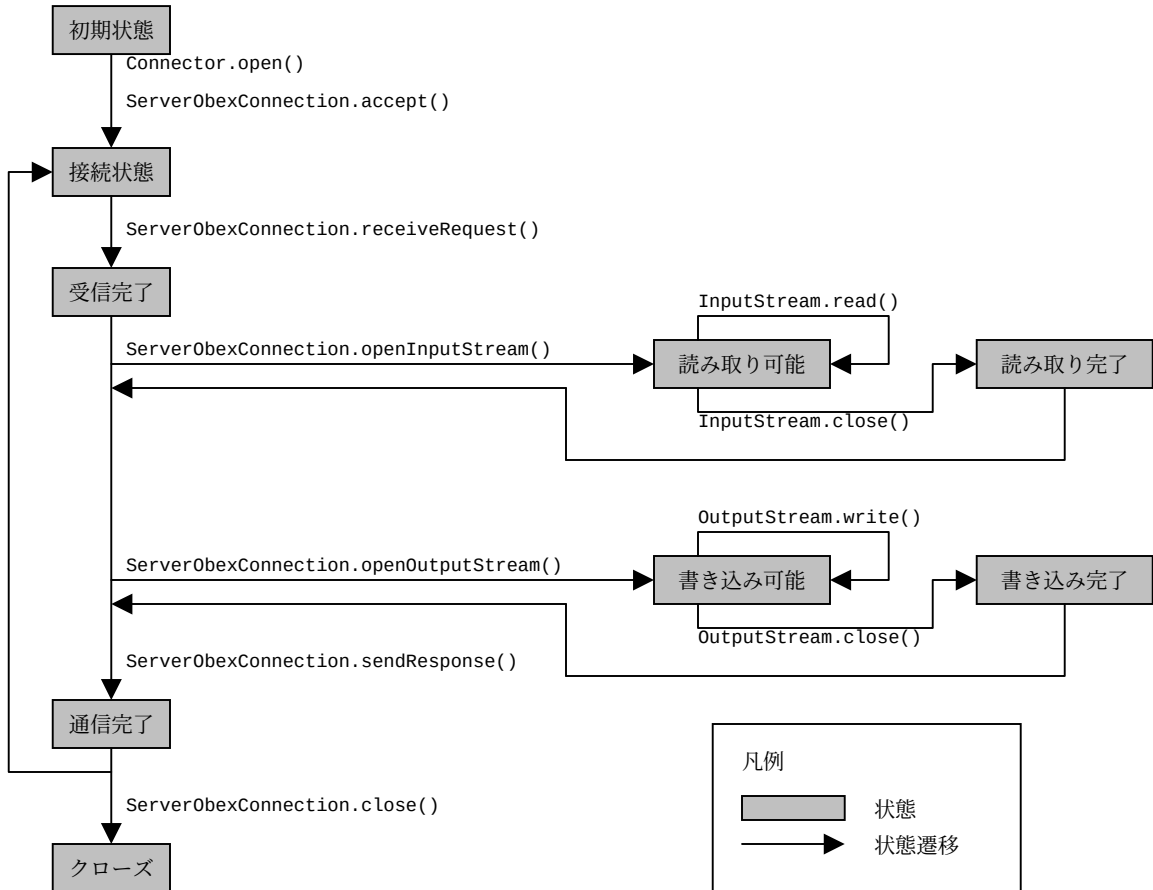


図 14: `ServerObexConnection` の状態遷移

注意事項：

- OBEX 外部接続においては、`Connector` クラスから直接 `openInputStream()`、`openDataInputStream()`、`openOutputStream()`、`openDataOutputStream()` の各メソッドにより入出力ストリームを取り出すことはできません。アプリケーションは `Connector.open()` メソッドで `ServerObexConnection` オブジェクトを取得し、そのオブジェクトから入出力ストリームを取り出す必要があります。
- 携帯電話の赤外線ポートへの通電は、アプリケーションから `ServerObexConnection.accept()` を呼び出した際に、システムにより自動的に行われます。

- OBEX クライアントからの PUT オペレーションに対して、OBEX サーバーがオブジェクトを送信することはできません。OBEX サーバーは GET オペレーションに対してのみオブジェクトを送信することができます。
- OBEX サーバーにおけるコネクションオブジェクトのオープンモードは、READ モードが PUT オペレーションに、WRITE モードが GET オペレーションに対応します。1 回の接続で GET、PUT の両オペレーションを受け付ける可能性がある場合は READ_WRITE モードを使用するようにしてください。
- 1 つのアプリケーションから同時に確立可能な通信リンクは 1 つだけです。
- あるスレッドが `accept()` メソッドや `receiveRequest()` メソッドを呼び出して OBEX クライアントのアクションを待っている時、他のスレッドが `ServerObexConnection` オブジェクトの `close()` メソッドを呼び出すと通信は切断されます。また、タイムアウト発生やユーザーの中断操作といった要因により、通信が切断されることがあります。
- GET、PUT の両オペレーションとも、1 回につき 30K バイトまでのオブジェクトを送受信することができます。この制限を越えたサイズのオブジェクトを送受信することはできません。
- アプリケーションが設定した OBEX ヘッダやオブジェクトデータの内容は、1 回のオペレーションでのみ有効です。1 回の接続で連続してオペレーションを実行する場合、その都度これらを設定しなおすようにしてください。
- DoJa-2.0 プロファイル対応の携帯電話では、外部との通信機能を一切停止して、通信を使用しない機能のみ利用可能とするセルフモード機能が搭載されます。ユーザーが携帯電話をセルフモードにしている状態では、アプリケーションの起動を行うことはできますが起動されたアプリケーションが OBEX 外部接続を行うことはできません。
- アプリケーションが赤外線通信を行っている間は、i アプリ実行環境は通信を実行中である旨のメッセージを表示します。また通信リンクの確立が行えなかった場合や、タイムアウト、ユーザー操作などにより通信が中断された場合には、i アプリ実行環境はその旨をユーザーに伝えるメッセージを表示します。これらのメッセージを非表示とすることはできません。

10.3 外部機器接続のヒント

以下に、DoJa-2.0 プロファイルにおける赤外線通信の下位レイヤ仕様、および携帯電話上のアプリケーションと携帯電話以外の外部機器との間で赤外線通信を行う上での注意点を示します。

携帯電話間で赤外線通信を行うアプリケーションを開発する場合はこれらを意識する必要はありません。携帯電話上のアプリケーションと他の外部機器との間で赤外線通信を行うシステムを構築するようなケースにおいて、外部機器側の設計を行う際の参考としてください。

なお、下記の記載内容に関する規定の詳細については IrDA 規格を参照してください。

<物理的条件>

通信速度： 共通的にサポートされる最大通信速度は 38.4Kbps です。

通信距離： 携帯電話に搭載される赤外線デバイスは、IrDA 規格における IrMC Low Power Option に対応しています。このため、通信距離は屋内環境で 20cm 程度に制限されます。

<IrLAP 接続パラメータ>

Baud Rate： 9600bps および 38400bps をサポートします。

Data Size： 64byte、128byte、256byte をサポートします。

Link Disconnect / Threshold Time : 3 秒 (Threshold=0 秒)、8 秒 (Threshold=3 秒) をサポートします。

<IrLMP デバイス名およびデバイスニックネーム>

デバイス名およびデバイスニックネームには、以下のような文字列が設定されます。

DoCoMo/<機種名>/j (例: DoCoMo/X504i/j)

<機種名>には、ユーザーエージェントに含まれている名前と同等のものが設定されます。外部機器からこの名前を参照することで、現在通信を行っている携帯電話の機種を判別することができます。

<注意点>

- OBEX で規定されているオペレーションのうち、i アプリ実行環境は OBEX クライアント、OBEX サーバーとも SetPath オペレーションをサポートしません。また、OBEX クライアントは ABORT オペレーションをサポートしません。
- i アプリ実行環境は、OBEX 認証をサポートしません。
- i アプリ実行環境において、OBEX クライアントおよび OBEX サーバーが受け付けることのできる OBEX ヘッダは Name、Type、Time、Body、End of Body の 5 つです。これら以外の OBEX ヘッダを使用することはできません。また、オブジェクトボディ (Body ヘッダおよび End of Body ヘッダ) は他のヘッダより後に送信してください。
- 携帯電話に搭載される赤外線通信機能は IrDA の相互接続認定を受けています。しかしこれは、IrDA 規格に準拠した全ての外部機器との接続性を保証するものではありません。使用する環境や相手側機器のハードウェアおよびソフトウェア特性などにより、接続することができない場合があります。

第 11 章

ネイティブアプリケーション連携

DoJa-2.0 プロファイルでは、i アプリと、ブラウザやメールなどの携帯電話上のネイティブアプリケーションとの連携動作をサポートしています。ネイティブアプリケーション連携機能を使用することにより、i アプリからブラウザを起動したり、メール（配信されたメール）から i アプリを起動したりすることが可能となります。

DoJa-2.0 プロファイルでは、以下の組み合わせの連携動作を行うことができます。

- ・ ブラウザから i アプリを起動
- ・ i アプリからブラウザを起動
- ・ メールから i アプリを起動
- ・ i アプリから音声発信機能（ダイヤラー）を起動
- ・ 外部機器（赤外線ポート）から i アプリを起動

本章では、ネイティブアプリケーション連携機能の利用方法、および各種制限事項などについて解説します。なお以下の解説では、ネイティブアプリケーションはブラウザなどのコンポーネント名で記載し、i アプリは単にアプリケーションと記載します。

11.1 ブラウザ連携

ブラウザ連携では、ブラウザからのアプリケーション起動およびアプリケーションからのブラウザ起動を行うことができます。なお、ブラウザからのアプリケーション起動においては、起動対象のアプリケーションはあらかじめ携帯電話にインストールされていなければなりません。

11.1.1 ブラウザからのアプリケーションの起動

ブラウザからアプリケーションを起動させるためには、アプリケーション開発者は以下の 2 つを行います。

- (1) インターネット経由でアクセス可能な Web サイト上にアプリケーション起動用の HTML ファイルを作成し、その中にアプリケーション起動タグを記述します。アプリケーション起動タグの記述内容により、そのタグを選択した際に携帯電話上で起動されるアプリケーションが決定されます。
- (2) ADF の LaunchByBrowser キーにて、アプリケーションが(1)で作成された HTML の URL からの起動を許可することを宣言します。LaunchByBrowser キーで許可を受けていない URL の HTML からは、そのアプリケーションを起動することはできません。なお、LaunchByBrowser キーで指定された URL と HTML の URL は前方一致で比較され、LaunchByBrowser キーに指定された内容が HTML の URL に全て含まれていれば、その HTML に対し起動許可が与えられます。

ブラウザからアプリケーションを起動するには、まずアプリケーション起動用 HTML をブラウザに表示させます。アプリケーション起動タグはリンク形式で表示されます。ユーザーがこのリンクを選択することにより、携帯電話は対応するアプリケーションを起動します。

以下に、アプリケーション起動用 HTML におけるタグの記述例を示します。

```
<OBJECT declare id="application.declaration"
data="http://www.nttdocomo.co.jp/java/abc.jam" type="application/x-jam">
<PARAM name="Param1" value="i-mode">
<PARAM name="Param2" value="i アプリ">
</OBJECT>
アプリケーションを起動するには
<A ista="#application.declaration" href="notapplicable.html">ここ</A>
をクリック。
```

アプリケーション起動のために使用されるタグは以下の通りです。

● A タグおよび ista 属性

A タグは、起動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、携帯電話にインストールされているアプリケーションが起動されます。A タグの ista 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。携帯電話上のどのアプリケーションが起動されるかは、後述の OBJECT タグに指定された ADF URL により決定されます。

ista 属性は DoJa-2.0 プロファイルにて新設された属性であり、DoJa-1.0 プロファイル以前の携帯電話では解釈されません。また、ista 属性と、アプリケーションをダウンロードするための ijam 属性を同時に有効にすることはできません。ista 属性と ijam 属性を同時に指定した場合、DoJa-2.0 プロファイルに対応した携帯電話では常に ista 属性を解釈し、ijam 属性は無視されます (DoJa-1.0 プロファイル対応の携帯電話では ista 属性を解釈できないため、常に ijam 属性が解釈されます)。

なお、ista 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。また、アプリケーションのダウンロード (ijam 属性) については、第 13 章を参照してください。

● OBJECT タグ

OBJECT タグは、アプリケーションに対応する ADF を特定するために使用されます。OBJECT タグの id 属性には、A タグの ista 属性で参照される名前 (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。携帯電話はアプリケーションをダウンロードした際に、そのアプリケーションに対応する ADF の URL を記憶しています。ブラウザからのアプリケーション起動では、data 属性に指定された URL と同じ ADF を参照しているアプリケーションが起動対象となります。

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。

OBJECT タグの内部には、下記に示す PARAM タグを記述することができます。

● PARAM タグ

PARAM タグは、ブラウザからのアプリケーション起動時に、IApplication.getParameter () メソッドにより取得できるパラメータを指定するために使用されます。

PARAM タグは、OBJECT タグ中に最大 16 個含めることができます。また、1 つの OBJECT タグ中に指定された全 PARAM タグの name 属性の値の長さ と value 属性の値の長さの合計は最大 255 バイトに制限されます。name 属性の値および value 属性の値には、それぞれ SJIS で日本語テキストを指定することができます。

以下に、携帯電話にダウンロードされているアプリケーションと、アプリケーション起動用 HTML の内容の関係について図示します。

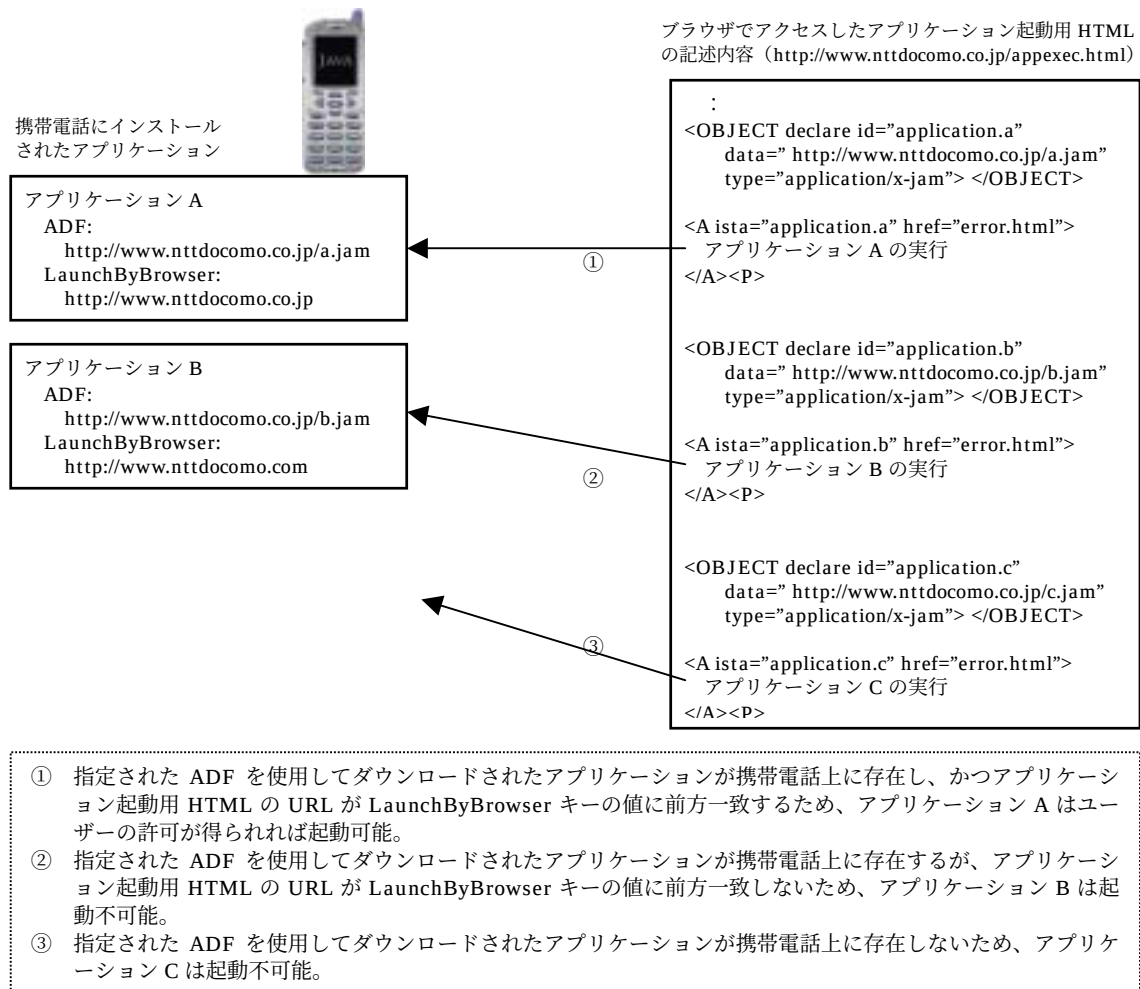


図 15: ダウンロード済みアプリケーションとアプリケーション起動用 HTML の記述内容との関係

注意事項：

- 携帯電話ユーザーは、携帯電話の設定によりブラウザからのアプリケーション起動を禁止することができます。またブラウザからアプリケーションを起動する前に、i アプリ実行環境はユーザーにアプリケーションを起動してよいか確認を行います。これらの機構によりユーザーの許可が得られなかった場合、そのアプリケーションの起動は行われません。
- ブラウザから起動されたアプリケーションの動作が終了 (IApplication.terminate())により終了) すると、アプリケーションを起動する前のブラウザの状態に復帰します。ただし、ブラウザから起動されたアプリケーションをユーザーが強制終了した場合、アプリケーション終了後にどのような状態に移るかはメーカーにより異なります。

11.1.2 アプリケーションからのブラウザ起動

アプリケーションからブラウザを起動するには、IApplication.launch()メソッドを使用します。このメソッドの第1引数にはIApplicationクラスで定義されている起動

種別（起動対象となるアプリケーションの種別）を、また第 2 引数には起動種別毎に規定されている起動引数を格納した String 配列を指定します。

DoJa-2.0 プロファイルでは、有効な起動種別は `IApplication.LAUNCH_BROWSER`（ブラウザの起動）のみとなっています。ブラウザを起動する場合、第 2 引数の先頭要素に接続先の URL（http スキームまたは https スキームをサポート）を設定します。URL は最長 255 バイトまで指定できます。

なおブラウザを起動するアプリケーションは、ADF の `UseBrowser` キーでブラウザ起動を行うことを宣言（値 `"launch"` を設定）する必要があります。この宣言を行っていないアプリケーションはブラウザを起動することはできません。

注意事項：

- 待ち受け実行中のアプリケーションからは、ブラウザを起動することはできません。
- `IApplication.launch()` メソッドを呼び出すと、アプリケーションはサスペンド状態に入ります。その後、以下の 2 点を満たすことでブラウザが起動されます。
 - ・ パケット通信が可能であること（携帯電話のディスプレイに、通信状態を示す "i" マークが点灯または点滅している状態）
 - ・ ブラウザ起動前、i アプリ実行環境がユーザーに求める起動確認に対しユーザーが同意することいずれかを満たしていない場合、アプリケーションはレジュームし、`IApplication.launch()` メソッドは例外をスローすることなく呼び出し元に復帰します。また、両方の条件を満たしてブラウザが起動される際、アプリケーションは終了します。
- アプリケーションから起動されたブラウザがユーザー操作により終了すると、待ち受け状態に復帰します。
- `HttpConnection` を使用した HTTP 通信時と異なり、アプリケーションからのブラウザ起動では、アプリケーションダウンロード元サイト以外のサイトにアクセスすることができます。ただしリクエストメソッドは GET に限定され、i アプリ実行環境によるユーザーへのブラウザ起動確認時に URL の内容がユーザーに提示されます。

11.2 メール連携

メール連携では、ユーザーに配信されたメールからアプリケーションの起動を行うことができます。アプリケーションからメールを起動したり、メールを作成・送信することはできません。なお、起動対象のアプリケーションはあらかじめ携帯電話にインストールされていなければなりません。

11.2.1 メールからのアプリケーションの起動

メールからアプリケーションを起動させるためには、アプリケーション開発者は以下の 2 つを行います。

- (1) ADF の `LaunchByMail` キーにて、対応するアプリケーションが、特定の送信元から受け取ったメールからの起動を許可することを宣言します。`LaunchByMail` キーで許可を受けていないメール送信元のメールからは、そのアプリケーションを起動することはできません。なお、`LaunchByMail` キーで指定されたメールアドレスとメール送信元アドレスは後方一致で比較され、`LaunchByMail` キーに指定された内容がメール送信元アドレスに全て含まれていれば、そのメールに対し起動許可が与えられます。
- (2) (1)の指定が行われたアプリケーションをダウンロードしているユーザーの携帯電話に対し、起動許可を得られるメールアドレスからメールを送信します。このメールには、起動するアプリケーションを決定するための情報やアプリケーションの起動パラメータなど（アプリケーション起動情報）を記述します。

アプリケーション起動情報が記述されたメールをメーラで閲覧すると、アプリケーション起動用のリンクなどが表示されます。ユーザーがこのリンクを選択することにより、携帯電話は対応するアプリケーションを起動します。

以下に、アプリケーション起動情報を含むメールの記述例を示します。

```
.....  
.....  
.....  
(ここまでメーラで閲覧可能なメール本文)  
--B:A  
TEXT=" i アプリ A "  
ADF="http://www.nttdocomo.co.jp/java/abc.jam"  
"Param1"="i-mode"  
"Param2"=" i アプリ "
```

(注) 各行とも行末は<CR><LF> (0x0d0a) です。

アプリケーション起動情報の記法は以下の通りです。

● --B:A

"--B:A"は、メール本文とアプリケーション起動情報の境界を示す識別子です。この識別子（アルファベット部は大文字である必要があります）のみを含む行以降はアプリケーション起動情報と見なされます。アプリケーション起動情報は、ユーザーがメールを閲覧する際にその内容を直接見ることはできません。また、メールの返信や転送などにおいても、アプリケーション起動情報は引用されません。

● TEXT キー

TEXT キーには、アプリケーション起動用リンクで表示される文字列の設定を行います。先の記述例ではメール閲覧時に" i アプリ A "という文字列がリンクとして表示され、ユーザーがそのリンクを選択することでアプリケーションが起動されます。

● ADF キー

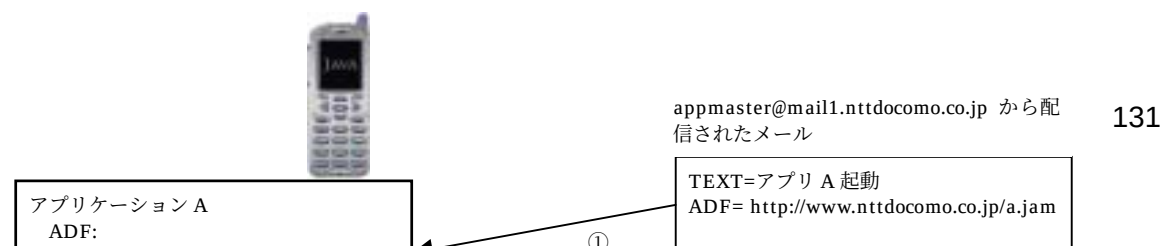
ADF キーには、起動したいアプリケーションに対応する ADF の URL を指定します。携帯電話はアプリケーションをダウンロードした際に、そのアプリケーションに対応する ADF の URL を記憶しています。メール連携においては、メールに記述された ADF キーの URL と同じ ADF を参照しているアプリケーションが起動対象となります。

● パラメータ指定キー

アプリケーション開発者は、キーの名前と値をそれぞれダブルクォートで囲むことにより、アプリケーションに引き渡すことのできるパラメータを定義することができます。このように指定されたパラメータは、`IApplication.getParameter()`メソッドで取得することができます（ブラウザからのアプリケーション起動における PARAM タグと同等）。

パラメータ指定キーは、1つのメール内に最大 16 個含めることができます。また、1つのメール内で使用されている全パラメータ指定キーの名前の長さとの合計は最大 255 バイトに制限されます。キーの名前および値には、それぞれ日本語テキストを指定することができます。パラメータに日本語テキストを使用する場合、長さの制限は SJIS 表現に対し適用されます。

以下に、携帯電話にダウンロードされているアプリケーションと、メールに記述されたアプリケーション起動情報の内容の関係について図示します。



携帯電話にインストール
されたアプリケーション

図 16: ダウンロード済みアプリケーションとアプリケーション起動情報の記述内容との関係

注意事項：

- 携帯電話ユーザーは、携帯電話の設定によりメールからのアプリケーション起動を禁止することができます。またメールからアプリケーションを起動する前に、i アプリ実行環境はユーザーにアプリケーションを起動してよいか確認を行います。これらの機構によりユーザーの許可が得られなかった場合、そのアプリケーションの起動は行われません。
- メールから起動されたアプリケーションの動作が終了（`UIApplication.terminate()`により終了）すると、アプリケーションを起動する前のメールの状態（元のメールが表示されている状態）に復帰します。ただし、メールから起動されたアプリケーションをユーザーが強制終了した場合、アプリケーション終了後にどのような状態に移るかはメーカーにより異なります。
- メールにメロディ添付とアプリケーション起動情報の両方を設定することはできません。両方設定した場合は、両方とも無効なものとして無視されます。

11.3 音声通話連携

音声通話連携では、アプリケーションから携帯電話の音声発信機能（ダイヤラー）の起動を行うことができます。

11.3.1 アプリケーションからの音声発信機能の起動

アプリケーションから音声発信機能を起動するには、`com.nttdocomo.util.Phone` クラスの `call()` メソッドを使用します。このメソッドを、引数に電話番号を表現する文字列を指定して呼び出すことにより音声発信機能が起動します。

電話番号文字列には、0～9 までの数字、'#'、'*'、',' (ポーズ：1 秒間)、'/' (ポーズ：キー入力待ち) を指定することができます。ポーズ指定は電話番号文字列中に連続して指定することはできません。また、電話番号文字列中に現れる '-' は無視されます。

なお、音声発信機能を使用するアプリケーションは、ADF の `UseTelephone` キーで音声発信機能を使用することを宣言 (値 "call" を設定) する必要があります。この宣言を行っていないアプリケーションは音声発信機能を使用することはできません。

注意事項：

- 非活性化状態にある待ち受けアプリケーションは、音声発信機能を起動することはできません。
- `Phone.call()` メソッドを呼び出すと、アプリケーションはサスペンド状態に入ります。その後、i アプリ実行環境がユーザーに求める発信確認に対しユーザーが同意することによりダイヤル操作が行われます。音声通話が終了するとアプリケーションはレジュームし、`call()` メソッドは呼び出し元に復帰します。

11.4 外部機器連携

外部機器連携では、外部機器から赤外線ポートを介して携帯電話に起動指示を行うことにより、携帯電話上のアプリケーションの起動を行うことができます。なお、起動対象のアプリケーションはあらかじめ携帯電話にインストールされていなければなりません。

11.4.1 外部機器からのアプリケーションの起動

外部機器からアプリケーションを起動させるためには、アプリケーション開発者は以下の 2 つを行います。

- (1) 外部機器上に、赤外線ポートを介して携帯電話へアプリケーションの起動を指示するための OBEX アプリケーションを作成・配置します。外部機器側の赤外線通信に関する要件は 10.3 項を参照してください。外部機器からのアプリケーション起動指示は、後述の `vTrigger` オブジェクトを携帯電話に送信することにより行います。
- (2) 外部機器から起動させたいアプリケーションに対応する ADF の `AllowPushBy` キーにて、アプリケーションが特定の起動コマンドを含んだ `vTrigger` オブジェクトを送信する外部機器からの起動を許可することを宣言します。`AllowPushBy` で宣言する起動コマンドと外部機器が送信する起動コマンドが完全に一致していない場合、そのアプリケーションを起動することはできません。`AllowPushBy` キーは以下のように指定します。ADF の記述方法の詳細については 12.5.1 項を参照してください。

`AllowPushBy = Irda:<起動コマンド>`

携帯電話の赤外線ポートは、ユーザーによるメニューやボタンなどの操作により通電されます。この状態でユーザーが外部機器と携帯電話の赤外線ポートを互いに対向させ、`vTrigger` オブジェクトを送受することによりアプリケーションが起動されます。

vTrigger とは、i アプリ対応携帯電話において、外部機器連携をサポートするために規定されたオブジェクトフォーマットです。vTrigger は複数のプロパティ(属性名と値のペア)の集合として表現されるテキストデータです。赤外線ポートを介して vTrigger オブジェクトを携帯電話に送信することにより、携帯電話上のアプリケーションに起動指示を出すことができます。

以下に、vTrigger の書式について解説します。

```
BEGIN:VTRG
VERSION:<vTrigger フォーマットバージョン>
ADFURL:<起動対象アプリケーションの ADF URL>
COMMAND:<起動コマンド>
<パラメータ>
END:VTRG
```

※行末は<CR><LF> (0x0d0a) としてください。

記法は以下の通りです。

● BEGIN:VTRG

vTrigger オブジェクトの開始を表します。

● VERSION:<vTrigger フォーマットバージョン>

使用している vTrigger オブジェクトのフォーマットバージョンを指定します。DoJa-2.0 プロファイルでは、vTrigger フォーマットバージョンとして"1.0"を指定します。

● ADFURL:<起動対象アプリケーションの ADF URL>

この vTrigger オブジェクトで起動させたいアプリケーションに対応する ADF の URL を ASCII 形式で指定します (最大 255 バイト)。

● COMMAND:<起動コマンド>

起動コマンドを ASCII 形式で指定します (最大 255 バイト)。ここで指定されている起動コマンドと、アプリケーションの ADF の AllowPushBy キーで起動許可が与えられている起動コマンドが完全に一致していなければ、アプリケーションを起動することはできません。

● <パラメータ>

起動されたアプリケーションに引き渡すパラメータを指定します (オプション)。1 つのパラメータは以下のフォーマットを取ります。

PARAM [;<エンコード指定>][;<文字セット指定>]:<パラメータ名>;<パラメータ値>

※[] は省略可能であることを示します。なお、エンコード指定と文字セット指定の位置は入れ替わっていても構いません。

パラメータ名およびパラメータ値は、アプリケーション起動時に `IApplication.getParameter()` メソッドにより取得できるパラメータを指定するために使用されます。<パラメータ>は 1 つの vTrigger オブジェクトに最大 16 個まで指定することができます。ただし、全ての<パラメータ>の名前と値の合計長は 255 バイトに制限されます。

エンコード指定および文字セット指定は以下のように指定します。

ー エンコード指定

ENCODE=<エンコード名> の形式で指定します。エンコード名には"QUOTED-PRINTABLE"または"BASE64"のいずれかが指定できます。指定がない場合はエンコード指定が行われていないものと見なされます。

ー 文字セット指定

CHARSET=<文字セット名> の形式で指定します。DoJa-2.0 プロファイルでは有効な文字セット名は"SHIFT_JIS"のみです。また、文字セット指定がない場合も"SHIFT_JIS"が指定されたものと見なされます。

なお vTrigger オブジェクトでは、ボディの内容は7ビットコードで記述されている必要があります。パラメータの名前や値に日本語 (SHIFT_JIS) を指定する場合は、必ず QUOTED-PRINTABLE 形式または BASE64 形式でエンコードするようにしてください。エンコードされた日本語パラメータは、指定されたエンコード種別に従って i アプリ実行環境で自動的にデコードされ、さらに Java の String に変換された上で IApplication.getParameter() メソッドに渡されます。

● END:VTRG

vTrigger オブジェクトの終了を表します。

以下に、vTrigger オブジェクトの記述例を示します。

```
BEGIN:VTRG
VERSION:1.0
ADFURL:http://www.nttdocomo.co.jp/java/a.jam
COMMAND:startup Application A
PARAM:Param1;i-mode
PARAM;ENCODING=BASE64;CHARSET=SHIFT_JIS:Param2;gomDQYN2g4o=
END:VTRG
```

OBEX レイヤにおける vTrigger オブジェクトの送信とは、Body ヘッダに vTrigger オブジェクトの内容を設定し、PUT オペレーションにて送信することをいいます。携帯電話は外部機器からのアプリケーション起動要求に対し、アプリケーションの起動可否を判断して以下の OBEX レスポンスコードを返します。

レスポンスコード	OBEX における規定	外部機器連携における規定
0xA0	OK, Success	指定されたアプリケーションの起動が可能である
0xC4	Not Found	指定したアプリケーションが存在しない
0xE1	Database Locked	指定したアプリケーションの外部機器からの起動が禁止されている、または AllowPushBy キーの起動コマンドと vTrigger オブジェクトで受信した起動コマンドが一致しない

レスポンスコード 0xA0 が外部機器に返された時点では、携帯電話上ではまだアプリケーションは起動されていません。アプリケーションの起動は、携帯電話がレスポンスコードを返却し、赤外線通信が終了された後に行われます。

以下に、携帯電話にダウンロードされているアプリケーションと、vTrigger オブジェクトの記述内容の関係について図示します。

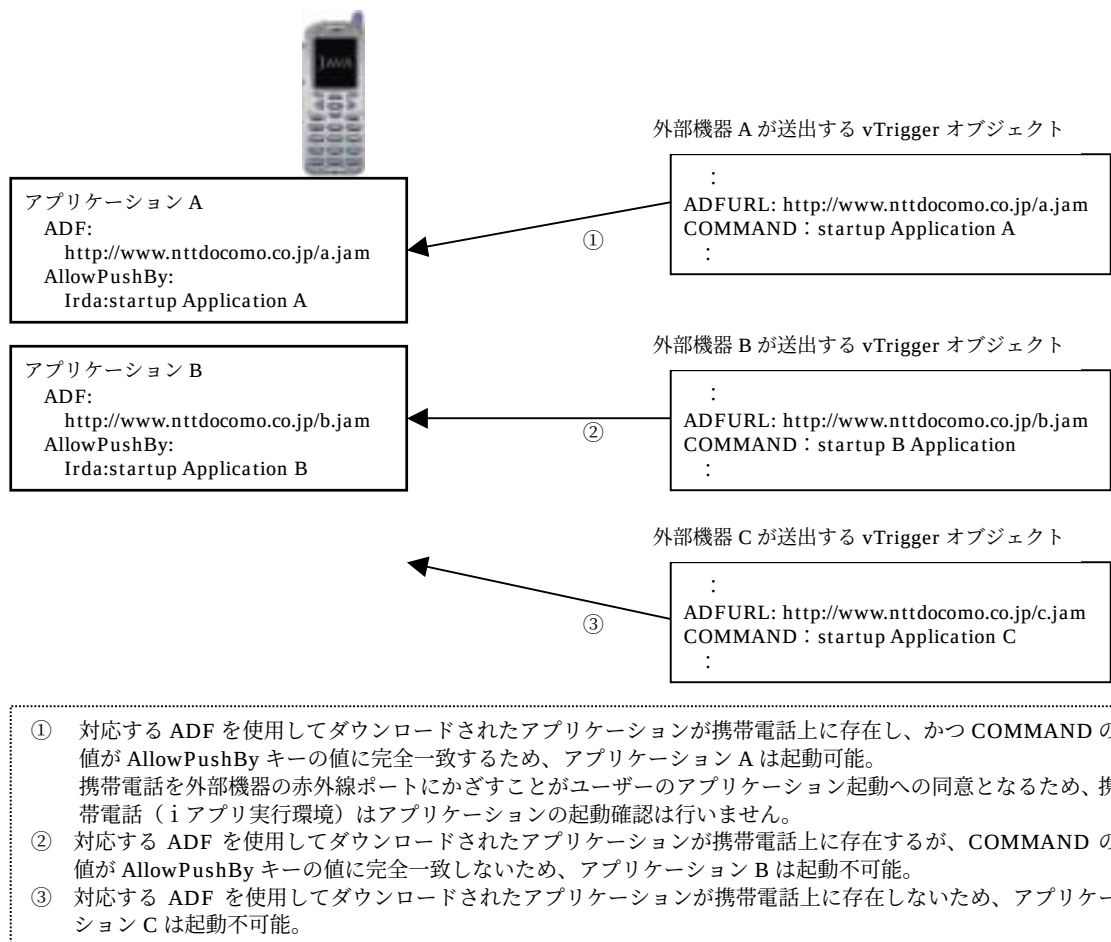


図 17: ダウンロード済みアプリケーションと vTrigger オブジェクトの記述内容との関係

注意事項：

- 携帯電話ユーザーは、携帯電話の設定により外部機器からのアプリケーション起動を禁止することができます。外部機器からの起動が禁止されている場合、vTrigger オブジェクトを受信してもアプリケーションの起動は行われません。
- 外部機器から起動コマンド（vTrigger オブジェクト）を受信するためには、ユーザーはあらかじめ携帯電話の赤外線ポートへの通電操作を行っている必要があります。赤外線ポートへの通電操作の方法はメーカーにより異なります。
- 外部機器から起動されたアプリケーションの動作が終了すると、待ち受け状態に復帰します。ただし、外部機器から起動されたアプリケーションをユーザーが強制終了した場合、アプリケーション終了後にどのような状態に移るかはメーカーにより異なります。
- 外部機器から vTrigger オブジェクトを携帯電話に送信する際、その vTrigger オブジェクトの Name ヘッダの値は末尾（拡張子）が".vtg"となるようにしてください。携帯電話は、受信したオブジェクトの名前の拡張子が".vtg"である場合に、そのオブジェクトを vTrigger オブジェクトと認識します。

第 12 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、i アプリ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 i アプリを開発するためのアプリケーション開発環境（以降 i アプリ開発環境とします）を提供していますが、ここでは i アプリ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。i アプリ開発環境の詳細については、i アプリ開発環境の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 SDK, Standard Edition（バージョン 1.3.1 以降 J2SE SDK とします）を利用したことがあり、J2SE 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 13 章を参照してください。

12.1 i アプリ開発環境のインストール

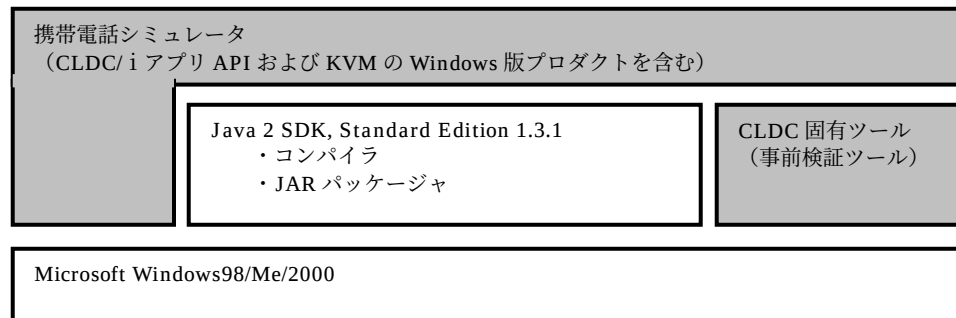
i アプリ開発環境を使用するには、NTT ドコモが配布する i アプリ開発環境のアーカイブファイル入手し、PC 上でインストール作業を行います。i アプリ開発環境のインストール手順の詳細については、i アプリ開発環境のユーザーズガイドを参照してください。

なお、i アプリ開発環境をインストールするには、i アプリ開発環境のほかに J2SE SDK を用意する必要があります。J2SE SDK をインストールした上で i アプリ開発環境をインストールしてください。

12.2 i アプリ開発環境のコンポーネント

i アプリ開発環境は、PC 上で i アプリ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 アプリをビルド、シミュレート実行することができます。

i アプリ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 18: i アプリ開発環境の構成

以下に i アプリ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
i アプリ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 アプリ実行環境では、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。

【DoJa-2.0】

DoJa-2.0 プロファイル向けに提供される i アプリ開発環境は、DoJa-1.0 プロファイル向けの i アプリ開発環境と若干コンポーネント構成が異なります。

12.3 開発サイクルの概要

次に、一般的な開発サイクルを示します。

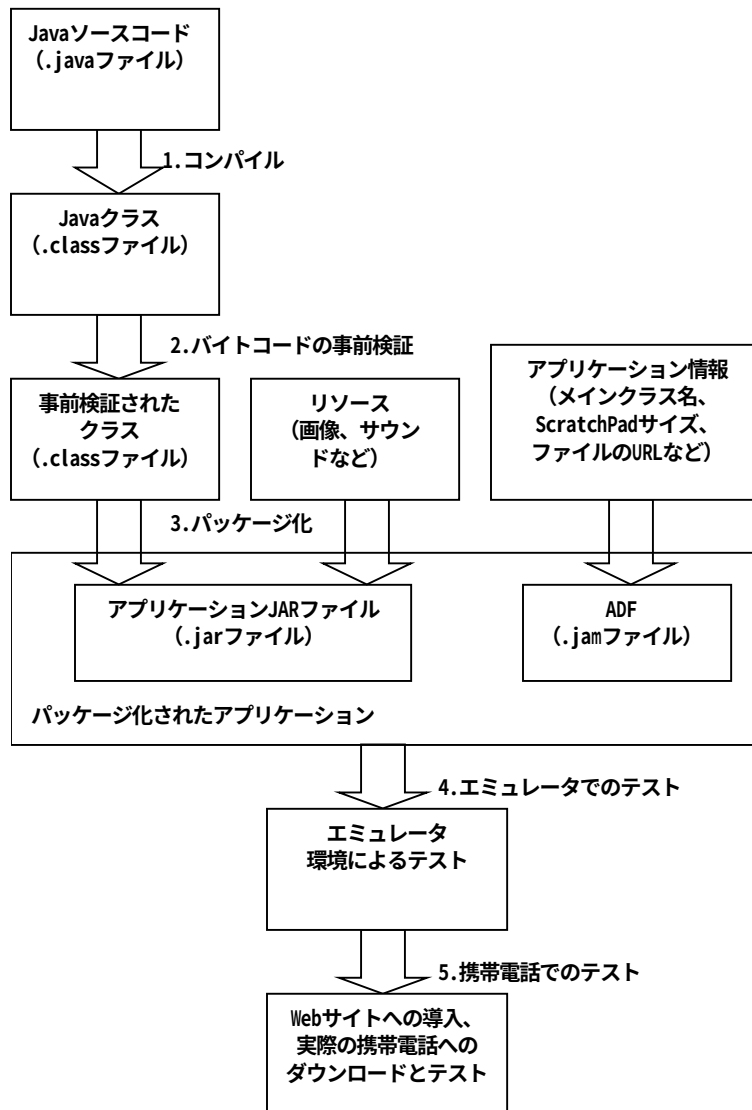


図 19. 開発サイクル

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、J2SE SDK 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 アプリ実行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。

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のバイトコード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、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。

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、J2SE SDK 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 SDK 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 i アプリ対応携帯電話でのテスト – 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 サイトに配置し、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、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して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 アプリ開発環境のユーザーズガイドを参照してください。

12.4 通信先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、i アプリ対応携帯電話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、クライアント側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。

またアプリケーションが OBEX 外部接続を使用する場合、通信先となる OBEX アプリケーションを用意する必要があります。i アプリ対応携帯電話間で OBEX 外部接続を利用するケースでは、i アプリ API を使用して OBEX クライアント、OBEX サーバーとも開発することができます。携帯電話とその他の機器を接続する場合、通信先アプリケーションの開発については、通信先となるプラットフォームで利用する OBEX アプリケーション開発環境のドキュメントを参照してください。

12.5 アプリケーションのパッケージ化

図 19 に示したように、アプリケーションは、ADF と JAR ファイル (アプリケーション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。

12.5.1 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: オプションルキー

アプリケーションのバージョン（最大 10 バイト）。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 30720 以下となります。なお、DoJa-1.0 プロファイルでは AppSize キーの値の上限は 10240 です。

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・最大 255 バイト）。ASCII 文字のみ指定可能です。

AppParam: オプションルキー

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（最大 255 バイト）。これらのパラメータを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。

ConfigurationVer(KvmVer): オプションルキー

J2ME コンフィグレーションバージョン。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。ConfigurationVer と KvmVer は同じ意味を持ちますが、1 つの ADF に両者を同時に指定することはできません。KvmVer は DoJa-1.0 プロファイルとの互換性のために用意されています。このキーの設定値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)

ProfileVer: オプションルキー（DoJa-2.0 プロファイルにて新設）

i アプリ実行環境のプロファイルバージョン。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 i アプリ実行環境のすべてのバージョンで動作することを示します。このキーの設定値は以下のフォーマットに従います。

DoJa-x.x (x は 0~9 の数字。本バージョンでは"1.0"および"2.0"をサポート)

SPSize: オプションルキー

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。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 TimeZone

Dow:曜日 DD:日 Mon:月 YYYY:年
HH:時間 MM:分 SS:秒 TimeZone:タイムゾーン

タイムゾーンは省略可能です。DoJa-1.0 プロファイルではタイムゾーンの指定はサポートされないため、DoJa-1.0 プロファイル対応の携帯電話と DoJa-2.0 プロファイル対応の携帯電話で ADF を共有する場合は、タイムゾーンは省略するようにしてください。

タイムゾーンは"GMT"および"JST"に対応しています。また、"+0900"というようにタイムゾーンを時差で指定することもできます。タイムゾーンが指定されていない場合、"JST"が指定されたものとして扱われます。

指定例: Fri, 25 Feb 2000 12:04:25 JST

UseNetwork: オプションルキー

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 アプリ実行環境では"http"が唯一の有効な値です（HTTPS 使用時も"http"を指定します）。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: オプションルキー

ターゲットとする携帯電話の機種名を指定します（最大 128 バイト）。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。このキーには以下のいずれかの形式で値を設定します。

- ターゲットとなる機種名の個別指定。機種名には、ユーザーエージェントに含まれている名前と同等のものを使用します。ターゲットとなる機種が複数存在する場合は、コンマ区切りで列挙します。

例: X503i, Y503i, Z503i

- ターゲットとなるシリーズ全機種の一括指定。"all1504"のように、キーワード"all"の後にシリーズ名を指定します。ターゲットとなるシリーズが複数存在する場合は、コンマ区切りで列挙します。なお、この指定方法は DoJa-1.0 プロファイルのみに対応した携帯電話では解釈できないため注意してください。

LaunchAt: オプションルキー

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 アプリ実行環境では、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。

MyConcierge: オプションルキー (DoJa-2.0 プロファイルにて新設)

待ち受けアプリケーションであることを宣言します。値に"Yes"が指定されている場合、そのアプリケーションは待ち受けアプリケーションとして利用可能であることを示します。この指定がないアプリケーションは、待ち受けアプリケーションとして携帯電話に登録することはできません。

UseTelephone: オプションルキー (DoJa-2.0 プロファイルにて新設)

音声通話機能を利用するアプリケーションであることを宣言します。値に"call"が指定されている場合、そのアプリケーションは音声発信 (`com.nttdocomo.util.Phone.call()` メソッド) を使用することができます。この指定がないアプリケーションは、音声発信を利用することはできません。

UseBrowser: オプションルキー (DoJa-2.0 プロファイルにて新設)

ブラウザ機能を利用するアプリケーションであることを宣言します。値に"launch"が指定されている場合、そのアプリケーションは `com.nttdocomo.ui.IApplication.launch()` メソッドを使用してブラウザを起動することができます。この指定がないアプリケーションは、ブラウザを起動することはできません。また、アプリケーションが待ち受け実行されている場合は、このキーの設定に関わらずブラウザを起動することはできません。

LaunchByMail: オプションルキー (DoJa-2.0 プロファイルにて新設)

このアプリケーションがメールからの起動を許可することを宣言します。値には、起動を許可するメールの送信元メールアドレスを 1 つだけ ASCII 形式で指定します (最大 50 バイト)。このキーの値が、後方一致比較の結果として送信元メールアドレスに全て含まれていた場合、そのメールからアプリケーションを起動することを許可します。なお、このキーに値"any"が指定されている場合、送信元メールアドレスの制限は行いません。

LaunchByBrowser: オプションルキー (DoJa-2.0 プロファイルにて新設)

このアプリケーションがブラウザからの起動を許可することを宣言します。値には、起動を許可する Web ページの URL を 1 つだけ ASCII 形式で指定します (最大 255 バイト)。このキーの値が、前方一致比較の結果としてブラウザで表示中の URL に全て含まれていた場合、その Web ページからアプリケーションを起動することを許可します。なお、このキーに値"any"が指定されている場合、URL の制限は行いません。

AllowPushBy: オプションルキー (DoJa-2.0 プロファイルにて新設)

このアプリケーションが、赤外線ポートからの vTrigger オブジェクト受信による起動を許可することを宣言します。値には、以下の形式のパラメータを 1 つだけ ASCII 形式で指定します (最大 255 バイト)。

`Irda:<command>`

`command` には、アプリケーション起動を許可する vTrigger オブジェクトに指定される起動コマンドを指定します。このキーの値が、アプリケーションに対して起動要求を行った vTrigger オブジェクトの起動コマンドと完全一致した場合、その vTrigger オブジェクトによってアプリケーションを起動することを許可します。

なお、`command` には空白およびタブを含めることはできません。

AppTrace: オプションルキー (DoJa-2.0 プロファイルにて新設)

アプリケーション開発のフェーズにおいて、携帯電話上で簡易なデバッグ機能を提供するためのキーです。このキーに値"on"を指定すると、携帯電話上で以下の機能が有効となります。

- キャッチされない例外によりアプリケーションが異常終了した場合、どのような例外によりアプリケーションが異常終了したかを表示します。
- アプリケーションから標準出力または標準エラー出力に出力した文字列をロギングし、アプリケーション終了後に表示させることができます。ただしロギング可能な文字列長には制限があり (最大 512 バイト)、新しく書き込まれたものから順に制限の範囲内のものが表示されます。なお、`java.lang.Throwable.printStackTrace()` を呼び出した結果はこのログには保存されません。
- JAM のアプリケーション情報表示機能が詳細モードとなります。

アプリケーションの開発が完了した際には、本キーは ADF から削除してください。

DrawArea: オプションルキー (DoJa-2.0 プロファイルにて新設)

このアプリケーションが前提としている画面サイズ（アプリケーションの描画エリアサイズ）を指定します。大きな描画エリアを持つ携帯電話で小さい描画エリアを前提としたアプリケーションを実行すると、ディスプレイに対しセンタリングされた、指定サイズの矩形領域内でアプリケーションが動作します。矩形領域の外側の部分にアプリケーションから描画を行うことはできません（その部分の表示色はメーカーにより異なります）。

キーの値は以下のように指定します。

`<width>x<height>` （width は横方向、height は縦方向のドット数 "x"は小文字のみ有効）

例： `DrawArea = 120x130`

width の設定可能最小値は 96、height の設定可能最小値は 72 です。また、このキーで指定された描画エリアサイズは、アプリケーションをダウンロードしようとしている携帯電話の描画エリアサイズ以下である必要があります（横または縦のいずれも、ダウンロードしようとしている携帯電話のサイズ以下でなければなりません）。これらの条件を満たさない設定が行われている場合、そのアプリケーションをダウンロードすることはできません。

なお、このキーが指定されていない場合は、その携帯電話の描画エリアを全て使用してアプリケーションを実行します。

GetSysInfo: オプショナルキー（DoJa-2.0 プロファイルにて新設）

このアプリケーションが、携帯電話のアイコン情報（DoJa-2.0 プロファイルではメール・メッセージの受信状態のみ）を参照することを宣言します。このキーに値"yes"が指定されているアプリケーションがダウンロードされると、携帯電話はアプリケーションにアイコン情報の参照を許可するかユーザーに確認を求めます。このキーが指定されていないアプリケーションやユーザーから許可を得ていないアプリケーションは、アイコン情報を参照することはできません。

なおメール・メッセージの受信状態の取得には、`PhoneSystem.getAttribute()` メソッドを使用します。

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
ConfigurationVer = CLDC-1.0
ProfileVer = DoJa-2.0
AppClass = banking.BankingDemo
AppVer = 1.0
PackageURL = http://aserver.com:8080/BankingDemo.jar
AppSize = 17556
LastModified = Fri, 25 Feb 2000 12:04:25
```

12.5.2 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。

12.6 アプリケーションのテスト

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 i アプリ対応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 i アプリ対応携帯電話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 アプリ開発環境の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、ConfigurationVer キー（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、開発コンピュータのハードディスク（エミュレータのアプリケーションディレクトリ）にあるアプリケーション JAR ファイルを読み込んで実行するため、PackageURL に指定された値は使用されません。実際に携帯電話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 を配置する必要があります。これらの詳細については第 13 章を参照してください。

注意事項：

- DoJa-1.0 プロファイル対応の i アプリ開発環境では、メディアサウンドのデータとして MIDI 形式ファイルを使用する必要があります。このため、DoJa-1.0 プロファイル対応の i アプリ開発環境を使用してメディアサウンドを使用するアプリケーションを開発する場合、実際の携帯電話を使用したテストを行う際に、MIDI 形式ファイルを 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。

なお、DoJa-2.0 プロファイル対応の i アプリ開発環境ではメディアサウンドのデータとして i メロディ形式ファイルおよび MIDI 形式ファイルを使用することができます。

- 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 13 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。

13.1 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、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">
<PARAM name="Param1" value="i-mode">
<PARAM name="Param2" value="i アプリ">
</OBJECT>
ダウンロードするには
<A ijam="#application.declaration" href="notapplicable.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 タグおよび ijam 属性

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 i アプリ対応携帯電話向け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、i アプリ対応機種では ijam 属性を解釈するようになります。また ijam 属性と、ブラウザからアプリケーションを起動するための ista 属性を同時に有効にすることはできません。ijam 属性と ista 属性を同時に指定した場合、DoJa-2.0 プロファイルに対応した携帯電話では常に ista 属性を解釈し、ijam 属性は無視されます (DoJa-1.0 プロファイル対応の携帯電話では ista 属性を解釈できないため、常に ijam 属性が解釈されます)。

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。

【DoJa-2.0】

ブラウザからのアプリケーション起動および A タグの `ista` 属性は DoJa-2.0 プロファイルで追加されました。これらの詳細については、第 11 章を参照してください。

● 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 タグの `ijam` 属性で参照される名前 (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。

OBJECT タグの内部には、下記に示す PARAM タグを記述することができます。

● PARAM タグ

PARAM タグは、アプリケーションをダウンロードして初回の実行時(*1)に、`IApplication.getParameter()` メソッドにより取得できるパラメータを指定するために使用されます。

Param タグは、OBJECT タグ中に最大 16 個含めることができます。また、`name` 属性の値の長さ `value` 属性の値の長さの合計は最大 255 バイトに制限されます。`name` 属性の値および `value` 属性の値には、それぞれ SJIS で日本語テキストを指定することができます。

- (*1) アプリケーションをダウンロードして初回の実行がネイティブアプリケーション連携によるもの (ブラウザ、メーラ、外部機器からの起動) である場合、ここで指定された PARAM タグの内容はアプリケーションに渡されません。初回実行がネイティブアプリケーション連携によるものである場合、ネイティブアプリケーションから渡されるパラメータの方が有効となりアプリケーションに渡されます。ネイティブアプリケーション連携の詳細については第 11 章を参照してください。

【DoJa-2.0】

PARAM タグによるパラメータ指定および `IApplication.getParameter()` メソッドは DoJa-2.0 プロファイルにて追加されました。

i アプリ対応携帯電話は、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。

【DoJa-2.0】

DoJa-2.0 プロファイルでは、ADF の拡張子に制限はありません。これに対し、DoJa-1.0 プロファイルでは、ADF の拡張子は `".jam"` である必要があります (ADF の URL の実体が CGI などの場合は、その CGI の URL 末尾を `".jam"` とします)。

13.2 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。

13.3 特定メーカーの i アプリ対応携帯電話へのダウンロード

i アプリオプション機能または i アプリ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、i アプリ対応携帯電話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

FOMA 携帯電話の i アプリ実行環境

ー FOMA02 仕様モデル、 FOMA03 前期仕様モデル

ここでは、2002 年冬以降に発売予定の FOMA 携帯電話（以降、FOMA02 仕様モデルとします）および 2003 年春以降に発売予定の FOMA 携帯電話（以降、FOMA03 前期仕様モデルとします）に搭載される i アプリ実行環境についての特記事項を記載します。

これらの FOMA 携帯電話には、以下のプロファイルバージョンに準拠した i アプリ実行環境が搭載されます。これらのプロファイルは、一部機能において DoJa-2.0 プロファイルから拡張されています。

- ・ FOMA02 仕様モデル： DoJa-2.1 プロファイル
- ・ FOMA03 前期仕様モデル： DoJa-2.2 プロファイル

また、FOMA サービスにおける無線ネットワークの大容量化およびサービス内容の拡大に伴い、FOMA 携帯電話の i アプリ実行環境には一部仕様の最適化が図られています。

A.1 DoJa-2.1 プロファイル

DoJa-2.1 プロファイルにおける DoJa-2.0 プロファイルからの変更点は以下の通りです。

A.1.1 携帯電話の個体識別情報の取得

DoJa-2.1 プロファイルでは、携帯電話の個体識別情報を参照するための API が追加されています。携帯電話の個体識別情報には以下の 2 種類があり、それぞれ同じ値が複数の個体に割り当てられることはありません。

- ・ 携帯電話本体の識別情報（製造番号）
- ・ UIM カードを搭載する携帯電話では、その UIM カードの識別情報（製造番号）

個体識別情報の参照には、com.nttdocomo.util.Phone クラスの以下の API を使用します。

```
public static final String getProperty( String key )
```

指定されたキーに該当するプロパティ値を取得します。取得できるプロパティ値は以下の 2 つです。

- ー public static final String Phone.TERMINAL_ID 携帯電話本体の識別情報
- ー public static final String Phone.USER_ID UIM カードの識別情報

無効なキーが指定された場合は null を返します。

パラメータ：

key – プロパティを指定する文字列を指定します。

例外：

SecurityException – このメソッドを呼び出したアプリケーションが、ADF の GetUtn キーにより
個体識別情報を参照するための適切な宣言を行っていない場合に発生します。

IllegalArgumentException – 引数に空文字列を指定して呼び出した場合に発生します。

NullPointerException – 引数に null を指定して呼び出した場合に発生します。

また、個体識別情報を参照するアプリケーションは、ADF において以下の宣言を行う必要があります。

GetUtn： オプショナルキー（DoJa-2.1 プロファイルにて新設）

このアプリケーションが、携帯電話の個体識別情報を参照することを宣言します。このキーに値
"terminalid"が指定されている場合は携帯電話本体の識別情報を、また値"userid"が指定され
ている場合は UIM カードの識別情報を参照することができます。携帯電話本体の識別情報と UIM
カードの識別情報を両方参照する場合は、これら 2 つをカンマ区切りで両方指定します。

このキーが指定されたアプリケーションがダウンロードされると、携帯電話はアプリケーションに
個体識別情報の参照を許可するかユーザーに確認を求めます。このキーが指定されていないアプリ
ケーションやユーザーから許可を得ていないアプリケーションは、個体識別情報を参照することは
できません。

A.1.2 ADF へのプロファイルバージョン（ProfileVer キー）の設定

DoJa-2.1 プロファイルでは、ADF に設定するプロファイルバージョン（ProfileVer キー）として
"DoJa-2.1"を設定することができます。プロファイルバージョンとして"DoJa-2.1"を設定した場合、そ
のアプリケーションは DoJa-2.1 プロファイル対応携帯電話でのみダウンロードすることができます。

DoJa-2.1 プロファイル対応携帯電話では、ProfileVer キーに"DoJa-2.0"または"DoJa-1.0"が指定され
たアプリケーションもダウンロードすることができます。アプリケーションを他のプロファイルバージョン
の携帯電話と共用する場合は、ProfileVer キーに適切な値を設定するか、または ProfileVer キー全体
の記述を省略するようにしてください。

A.1.3 携帯電話からのテレビ電話の発信

com.nttdocomo.util.Phone.call()メソッドでは、i アプリから通話発信を行う機能を提供していま
す。DoJa-2.1 プロファイルでは、テレビ電話をサポートする携帯電話において、このメソッドを使用して i
アプリからテレビ電話発信を行なうための機能が追加されています。

Phone.call()メソッドの引数には発信先の電話番号を表す文字列を指定しますが、テレビ電話を発信する
場合には、その電話番号文字列の前に接頭辞"tel-av:"を付加（例："tel-av:090xxxxxxx"）します。

A.2 DoJa-2.2 プロファイル

DoJa-2.2 プロファイルは DoJa-2.1 プロファイルの後継プロファイルですが、i アプリの
開発者向け基本機能に関する変更点はありません。変更点は ADF へのプロファイルバー
ジョンの記法のみです。

A.2.1 ADF へのプロファイルバージョン（ProfileVer キー）の設定

DoJa-2.2 プロファイルでは、ADF に設定するプロファイルバージョン（ProfileVer キー）として
"DoJa-2.2"を設定することができます。プロファイルバージョンとして"DoJa-2.2"を設定した場合、そ
のアプリケーションは DoJa-2.2 プロファイル対応携帯電話でのみダウンロードすることができます。

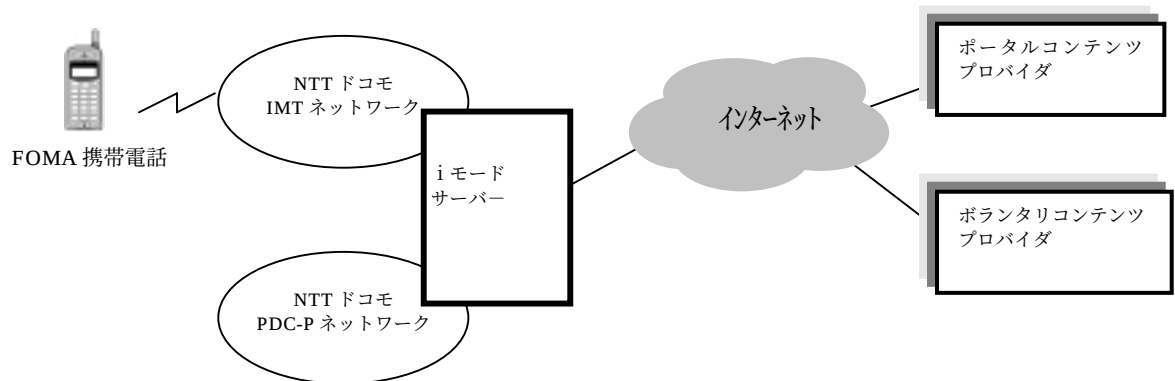
DoJa-2.2 プロファイル対応携帯電話では、ProfileVer キーに"DoJa-2.1"、"DoJa-2.0"、"DoJa-1.0" が指定されたアプリケーションもダウンロードすることができます。アプリケーションを他のプロファイルバージョンの携帯電話と共用する場合は、ProfileVer キーに適切な値を設定するか、または ProfileVer キー全体の記述を省略するようにしてください。

A.3 FOMA に向けた i アプリ実行環境最適化

次に DoJa-2.0 プロファイル規定中、FOMA 携帯電話向けに最適化された項目について説明します。

A.3.1 ネットワークアーキテクチャ

FOMA サービスでは、次世代移動通信システムの世界標準規格 IMT-2000 に採用されている W-CDMA 方式の無線ネットワーク網（以降 IMT ネットワークと呼びます）を使用します。アプリケーションのダウンロードやアプリケーションからの HTTP 通信はこの IMT ネットワークを介して行われます。



IMT ネットワークでは携帯電話からのデータ送信時に最大 64Kbps、データ受信時に最大 384Kbps（いずれもベストエフォート型）の高速なパケット通信を行うことができ、アプリケーションからの通信処理も高速に行うことができます。

なお、従来の i モード対応携帯電話が HTTP/1.0 ベースで実装されているのに対し、FOMA 携帯電話では HTTP/1.1 ベースで実装されています。これは、i アプリから HTTP 通信を行う場合にも適用されます。

A.3.2 i アプリ実行環境

次に、FOMA 携帯電話に搭載される i アプリ実行環境の 504 シリーズからの変更点を示します。以下の変更点は、主に FOMA におけるネットワークの大容量化およびサービス拡大内容を i アプリ実行環境に反映したものです。

(1) ScratchPad サイズの拡大

FOMA 携帯電話の i アプリ実行環境では、1つのアプリケーションが利用可能な ScratchPad の最大サイズが 100KByte から 200KByte に拡大されます。

(2) メロディフォーマットの追加

FOMA 携帯電話では、サポートするサウンドデータのフォーマットとして i メロディ形式 (MFi) に加え SMF (Standard MIDI File) が追加されます。なお、FOMA サービスでは MFi に代わり SMF が標準的なサウンドデータフォーマットとなります。今後発売される機種によっては MFi をサポートしない可能性がありますので注意してください。

(3) VisualPresenter による i モーションの再生

FOMA 携帯電話では、VisualPresenter による i モーションの再生が標準的にサポートされます。アプリケーションから i モーションデータを再生するには、i モーションデータを示すロケーション（URL）から MediaImage を作成し、それを VisualPresenter にセットして再生（play()）します。

アプリケーションから i モーションの再生を行うと、アプリケーションは一時中断して i モーションプレイヤーが起動し、セットされているデータの再生が行われます。ユーザー操作により i モーションプレイヤーが終了するとアプリケーションはレジュームし、再生指示以降の処理を続行します。

なお、アプリケーションから再生可能な i モーションのファイルサイズは最大 150KByte までです。また、非活性化状態の待ち受けアプリケーションから i モーションを再生することはできません。非活性化状態の待ち受けアプリケーションから i モーションを再生しようとした場合の振る舞いはメーカーにより異なります。

(4) HTTP 通信時の送受信データサイズ

FOMA 携帯電話では、HTTP 通信 1 回当たりの送受信データサイズが以下のように拡張されます。

- 送信データサイズ： 80KByte
- 受信データサイズ： 150KByte

(5) HTTP 通信時における HEAD メソッドの利用について

FOMA 携帯電話では、HTTP 通信時に指定するリクエストメソッドのうち HEAD メソッドは使用することができません。FOMA 携帯電話で指定可能なリクエストメソッドは GET メソッド、POST メソッドの 2 つです。

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 アプリ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 Sockets 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。