NICOLAS PONSINI

FLORIAN TOURNIER

Java Cardの概要

最小規模であり、もっとも広く普及しているJavaプラットフォームの1つについて、従来型のアプリの開発とプログラミング手法がどのように異なるのかを紹介

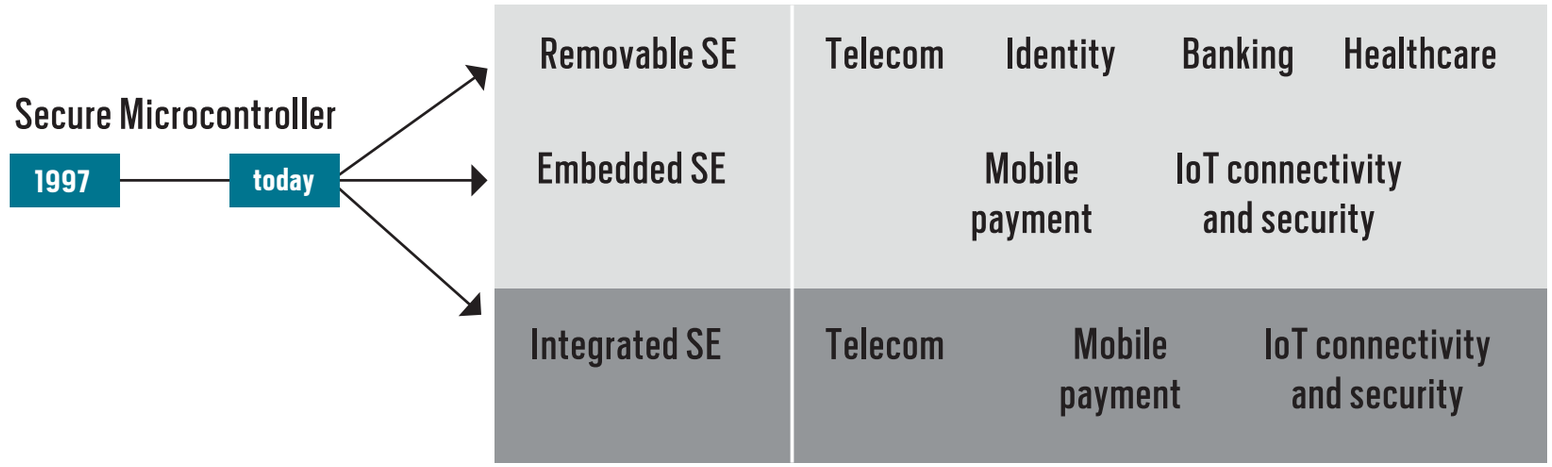
Java CardはJavaの最小限のサブセットであり、セキュア・エレメントのニーズに対応する機能が充実しています。セキュア・エレメント（SE）とは、耐タンパー性を持つハードウェア環境のことで、アプリケーションと暗号化された機密データをセキュアにホスティングすることができます。もっともよく使われているSEは、スマートカードに搭載されている1チップのセキュア・マイクロコントローラです。しかし、組込みSE（デバイス・ボードにはんだ付けされた、取り外しできないセキュア・マイクロコントローラ）から汎用チップに組み込まれる新しいセキュリティ設計に至るまで、新しいフォーム・ファクタも登場し始めています。図1に、一般的に利用されているSEの種類を示します。

これらのSEはすべて、一連の共通する重要な要件を持ち合わせています。

- **セキュリティ：**アプリケーションでは、トランザクションの原子性、暗号化のサポート、アプリケーション認証の署名、アプリケーションの独立性、ファイアウォールなどのセキュリティ属性を考慮する必要があります。
- **証明可能であること：**顧客は、内部のセキュリティ保証レベル基準に基づいて、コモンクライテリアやFIPSといった高水準のセキュリティ認定や、政府機関や決済機関などのドメイン固有の証明書を求めます。
- **コンパクトであること：**通常、SEはリソース（CPU、メモリ、帯域幅）に制約があるデバイスです。特に、メモリ構成では、使用可能な全メモリが1MBを超えることはほとんどありません。また、RAM仕様は2KBほどという低いものになる場合もあります。
- **標準ベースの管理性：**SEアプリケーションと、保存されている資格証明は、オープンな業界標準に基づいてセキュアに管理、更新する必要があります。

Java Cardは、Javaが提供するオープン性とコードの移植性を維持しつつ、こういった要件に適応しています。

Java Cardのエコシステム



Other secure hardware options: Trusted execution environments (TEEs), secure multipoint control units (MCUs), software environments

図1：一般的に利用されるセキュア・エレメント

ここ数年で、Java CardベースのSEの数は着実に増加しています。現在、Java Cardは世界でもっとも普及しているアプリケーション・プラットフォームであり、2017年の時点で、60億を超えるJava CardベースのSEが利用されていると見積もられています。その大半は、スマートカード（SIMカード、銀行のキャッシュ・カード、IDカード）が占めていますが、近年では、モバイル機器に組み込まれたJava CardベースのSEが急成長を遂げています。Internet of Things（IoT）ネットワーク用の末端のデバイスには、新しいタイプのSEも導入されています。

Java Cardテクノロジーを使用するアプリケーションはさまざまで、垂直市場によって大きく異なっています。Java Cardは主として以下のような用途で使われています。

- ID、認証、アクセス制御
- 接触型、非接触型決済などのセキュアな取引
- 資格証明の保管やコンテンツのセキュリティ
- サブスクリプション管理
- デバイスの整合性チェックや構成証明
- デジタル著作権管理（DRM）

Java Cardテクノロジーのエンドユーザーは、携帯電話事業者、金融機関、政府、モバイル機器メーカー、医療関係団体、一般企業、交通機関などです。ETSI、GlobalPlatform、ISOなどの標準化団体は、仕様の一部にJava Cardを活用しています。Java Cardの開発者コミュニティは専門的で、JavaCardのアプリケーションを開発する上で専門的

な知識と一般的なJavaの知識が必要です。それでも現在、このコミュニティは急速に拡大しています。エンドユーザーは社内でJava Card開発の専門性を磨いており、学術機関は研究やオープン・ソース・コンテンツでの貢献を深めています。

プラットフォームの進化

Java Cardは、1996年に登場しました。当時としては最先端の考え方（まだJava MEが登場する前のことで、これが最初の超小型Javaフレームワークでした）で、まだ不完全（仮想マシン（VM）仕様はなく、あったのは基本的な暗号化フレームワークのみでした）でもありました。続くバージョンでは、アーキテクチャが変わってAPIが拡張され、最初のバージョンのユーザーは認識できないだろうと思われるほどの変化を遂げた、最新のJava Cardフレームワークへと生まれ変わりました。

この進化を促し、大きく貢献することになったのがJava Card Forum（JCF）です。JCFは、Java Cardの最初のリリース後まもなく設立されました。JCFのWebサイトによると、この団体は「スマートカード、セキュア・オペレーティング・システム、セキュア半導体業界の企業が連携し、マルチアプリケーション・スマートカードやセキュア・デバイスの優先プログラミング言語としてJavaの推進と開発を行うために協力し合う」ものです。

JCFは、Java Card仕様を進化させるために、オラクルに提言を行っています。その提言は、プラットフォームに主要なイノベーションをもたらすことに貢献してきました。バージョン2.0でJava Card VMと実行時条件の仕様策定が始まってから、バージョン2.1で相互運用可能なファイル形式が導入されました。バージョン2.2では、欧州電気通信標準化機構（ETSI）との整合性がとられ、非接触型スマートカードのサポートが可能になりました。バージョン3.0では、仕様に2つの亜種（ClassicおよびConnected）が登場し、暗号化のサポートも拡大されました。現在、JCFはバージョン3.1の定義に向かってオラクルと共同作業を行っています。このバージョンでは、IoTの領域で新しいユースケースが実現できるようになる見込みです。バージョン3.1は、2018年にリリースされる予定となっています。

Java Cardの内側

オラクルは、仕様、開発ツール、セキュリティ文書など、Java Cardアプリケーションを開発するための要素を幅広く提供しています。

Java Card仕様：図2にJava Card開発プラットフォームを示します。ご覧のとおり、Java Card仕様には、アプリケーション開発をサポートする3つの主な要素が含まれています。

- Java CardプラットフォームのVM仕様では、Java Card VMの命令セット、サポートされているJava言語のサブセッ

ト、Java Cardテクノロジー対応デバイスにアプレットやライブラリをインストールするために使うファイル形式が提供されています。

- Java Cardプラットフォームのランタイム環境（RE）仕様では、Java CardテクノロジーのREの実装すべてに求められる動作が定義されています。REには、Java Card VMの実装、Java Card APIクラスのほか、アプレットの選択/選択解除といったランタイム・サポート・サービスが含まれています。
- Java CardプラットフォームAPIは、Java Card RE仕様を補完するもので、Java Card VMおよびJava Card REをサポートするクラス定義が含まれています。

Java Card Development Kit：このキットは無料でダウンロードでき、Java Cardプラットフォーム向けに書かれたアプリケーションを開発できる完全なスタンドアロン開発環境が含まれています。また、アプリケーションのテストを容易にするために、完全なJava Cardシミュレータと、Eclipseとの統合も含まれています。

Java Card Protection Profile：Java Card Protection Profileは、Java Cardベースの製品の作成者をサポートしてくれるもので、セキュリティ評価が行われる際に、銀行、政府などのカード発行者のセキュリティ要求を満たす助けになります。ここで提供されているのは、Java Cardプラットフォームの特徴に合わせて設計されたセキュリティ要件を備えるモジュール・セットです。Java Cardベース製品の開発者は、情報技術セキュリティ評価のためのコモンクライテリア（Common Criteria for IT Security Evaluation）を使って、セキュリティ評価を完了する時間とコストを削減できます。

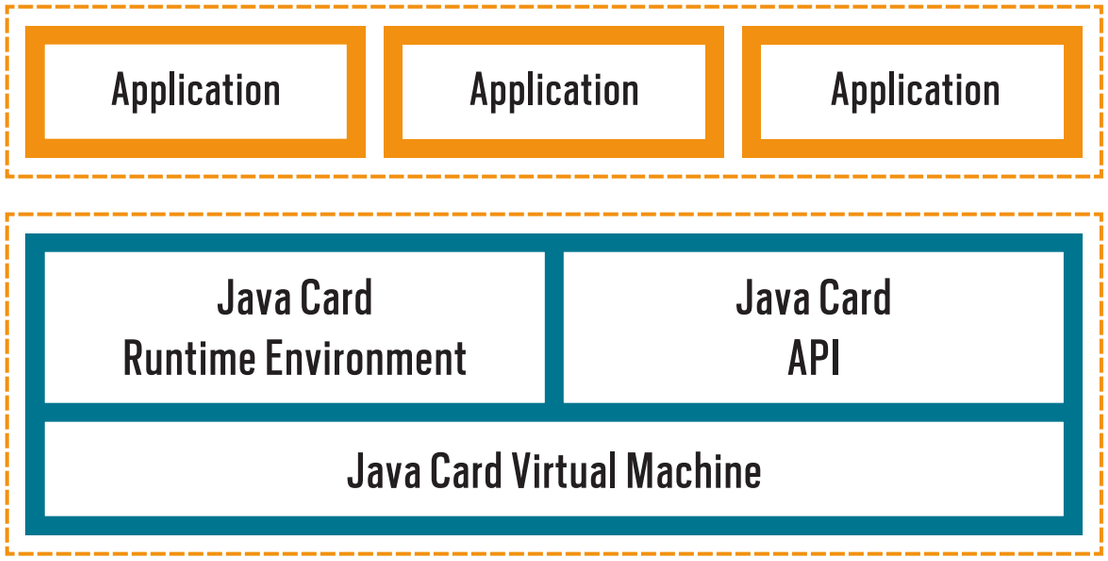


図2：Java Card開発プラットフォーム

Java Cardの開発に特有な事項

Java Cardの特異性について触れる前に、避けるべき2つの考え方をお伝えしておきます。

1つ目は、Java Cardは単なるスマートカードだというものです。Java Cardの「カード」という言葉は誤った名称です。もう少し正確に言うとすれば（マーケティング的な魅力は劣りますが）、「リソースが制約されたデバイスでセキュアにJavaテクノロジー・ベースのアプリケーションを実行するためのJava」となるでしょう。Java Cardのフレームワークとランタイムは、セキュリティ指向に基づいています。ハードウェア攻撃への対策もとられており、限られたメモリしか搭載されていないデバイスでもセキュアなSE上で実行されるように設計されています。しかし、その他のフォーム・ファクタも、さまざまなマーケットでJavaCardを実行するのに適しています。たとえば、現在のJavaCardは、あらゆる種類のセキュリティ・デバイスのセキュリティ・フレームワークになっています。

2つ目の考え方は、豊富な知識を持つJava開発者はJava Card開発者になれるというものです。技術力の高いJava開発者であっても、Java Cardのプログラミングには戸惑うかもしれません。Java CardはJavaのサブセットであるものの、オブジェクトの永続性や原子性などの点で、いくつかの特異性があります。そのため、Java Cardの開発は、従来、その分野に特化したスペシャリストの作業となっています。長期間（物理的にカードが使われなくなるまで）デプロイできることが正式に立証されたアプレットの開発、ソフトウェアおよびハードウェアへの攻撃に対する保護を行うためには、強力なセキュリティ・スキルが必要になります。Java Card Development Kitは誰でも使ってみることができますが（そうしてみることをお勧めします）、実際の場で使うアプレットをデプロイするのに必要な専門知識を獲得するためには、時間と経験が必要になるでしょう。

本記事の以降の部分では、他のJava環境と比べてJava Cardに特有な考え方のうち、主なものについて説明します。

Java言語のサブセット

Java Cardは、Java言語（JDK 7）のサブセットです。以下に、主要な言語機能について、サポートされているものとサポートされていないものを示します。

Java Cardでサポートされている JVM機能には、次のようなものがあります。

- クラス、インタフェース、継承、例外、パッケージなどのJavaのオブジェクト指向機能
- 小さなプリミティブ・データ型：boolean、byte、short、int（オプション）
- 1次元配列
- ObjectとThrowableの一部のメソッド

- クラス
- ガベージ・コレクタ（オプションであり、存在する場合も存在しない場合もあります）

Java Cardでサポートされていないjvm機能には、次のようなものがあります。

- 動的クラス・ローディング
- セキュリティ・マネージャ
- スレッド
- クローニング
- 大きなプリミティブ・データ型（long、double、float）
- 文字と文字列
- 多次元配列
- キーワード（native、synchronized、transient、volatileなど）

Java Card API

Java Card APIでは、以下で説明するパッケージが主要なライブラリです。

java.ioでは、標準Javaプログラミング言語のjava.ioパッケージのサブセットが定義されています。このパッケージには、java.io.IOExceptionクラスが含まれており、標準Javaプログラミング言語と同じ例外階層が維持されるようになっています。

java.langでは、Javaプログラミング言語のJava Cardテクノロジー・サブセットの設計に欠かせないクラスを提供しています。このパッケージのクラスは、標準Javaプログラミング言語のjava.langから派生したもので、Java Card VMが必要とする中核機能を表しています。代表的な中核機能として挙げられるのは、すべてのJava言語クラスのベース・クラスであるObjectクラスと、例外および実行時例外のクラスのベース・クラスであるThrowableクラスです。

javacard.frameworkでは、Java Cardテクノロジー・ベースのアプレットの構築、通信、実行に使うクラスとインタフェースのフレームワークを提供しています。ここでは、APDU（Application Protocol Data Unit）クラス、Applet（Java Cardアプレット）クラス、JCSystem（Java Card System）クラス、PIN（Personal Identification Number）インタフェースや、さまざまなJava Card固有の例外など、Java Cardの中核をなす考え方が定義されています。

javacard.securityでは、対称（AES）および非対称（RSA、ECC、DHなど）の操作、メッセージ・ダイジェスト、署名などに使うさまざまな種類の鍵やアルゴリズムを含むセキュリティAPIが定義されています。

javacardx.cryptoは、Java Cardプラットフォームでのセキュリティや暗号化を扱うフレームワークを実装した機能を含む拡張パッケージです。このパッケージは、輸出規制の対象となる可能性があります。プラットフォームは、実装に暗号化や復号化の機能が含まれる場合にのみ、このオプション・パッケージをサポートする必要があります。このパッケージには、CipherクラスとKeyEncryptionインタフェースが含まれています。Cipherは、メッセージの暗号化および復号化に使うメソッドを提供します。KeyEncryptionは、セキュアなエンド・ツー・エンド方式で鍵を更新できる機能を提供します。

javacardx.biometryとjavacardx.biometry1toNは、Java Cardプラットフォームでバイオメトリック・フレームワークを実装するための機能を含む拡張パッケージです。

javacardx.externalは、Java Cardプラットフォーム上のJava Card REが直接アドレス指定できないメモリ・サブシステムにアクセスするメカニズムを提供する拡張パッケージです。

完全を期すため、Java Card APIのパッケージのリストを示します。

- java.lang（サブセット）
- java.io（サブセット）
- javacard.framework
- javacard.framework.service
- javacard.security
- javacardx.crypto
- javacardx.annotations
- javacardx.apdu
- javacardx.biometry
- javacardx.biometry1toN
- javacardx.external
- javacardx.framework.math
- javacardx.framework.string
- javacardx.framework.tlv
- javacardx.util
- javacardx.security

仮想マシンのライフサイクルと永続性

Java Card VMとそれに関連する実行時条件は、Java Card仕様の独特な側面です。VMは、ホスト・オペレーティング・システム上の終了可能なプロセスとして実行されるわけではありません。VMは恒久的に実行され、ホスト・ハードウェアの寿命がVMのライフタイムとなります。つまり、VMを終了させることはできず、関連アプリケーションやオブジェクトが自動的に破棄されることはありません。

次回リセットされたときにVMが再起動し、永続ストレージから以前のヒープがリカバリされます。

VMや作成されたオブジェクトは、永続するアプリケーション情報を表すために使われ、電源が失われても消えることはありません。電源が失われても、その時点で停止するだけです。次回リセットされたときにVMが再起動し、永続ストレージから以前のヒープがリカバリされます。この設計は、スマートカードの世界から継承されたものです。スマートカードの世界では、電源が外部にあり、カード受付デバイス（CAD）に依存するため、電源が失われてもトランザクションが残っている必要があります。ここでのCADとは、接触型I/Oインタフェースを持つ従来型のカード受付デバイスと、非接触型インタフェースを持つ近接結合デバイス（PCD）の両方のカード・リーダーを指しています。

実行

Java Cardのアプリケーションは、変換済みアプレット（CAP）ファイルという形式です。CAPファイルによって、Java Cardプラットフォームでのバイナリ互換性が保証されます。CAPファイルには、実行可能な、クラスのバイナリ表現が含まれています。まず、他のJavaアプリケーションと同じように、Java Cardアプレットのパッケージを構成するJavaプログラミング言語のクラス・ファイルが生成されます。これが終わると、パッケージをCAPファイルに変換するコンバータ・ツールによって前処理されます。コンバータは、エクスポート・ファイルを生成することもできます。

Java Cardにはクラス・ローダーがないため、VMはプラットフォーム上のインストーラがプラットフォームにインストールしたコードを解釈します。この処理は、リンク処理に似ています。エクスポート・ファイルは、アプリケーションがインポートするパッケージに関する情報を変換前に取得するという用途でも、アプリケーションが後で使う可能性のあるパッケージに関する情報を変換後に出力するという用途でも使われます。たとえば、アプレットのコードがCAPファイルに変換されるとき、エクスポート・ファイルが入力として使われ、その際にJava Card APIパッケージが参照されます。

インストーラは、CAPファイルをダウンロードしてインストールするためのランタイム・メカニズムです。インストーラは、CADインストール・プログラムから実行可能バイナリを受け取り、バイナリをスマートカードのメモリに書き込み、カード上の他のクラスとリンクさせ、Java Card REが内部的に使用するデータ構造を作成して初期化します。

Java Card VM仕様では、ランタイム検証機能は必須ではありません。アプレットがCAPファイルに変換された後、対応するアプレットのインストールが行われる前に、検証ツールがVM外でCAPファイルのバイトコードの検証を行います。この処理は、アプレットの検証とインストールの間のパスがセキュアであることを意味します。Java Cardプラットフォームにインストールされる CAPファイルの整合性と信頼性（多くの場合は機密性も含まれます）は、保証される必要があります。この処理は、いかなる意味でも、実行時に追加の検証を行うことを妨げるものではありません。しかし、そのような追加のアクションは実装に依存するものであり、仕様によって強制されるものではありません。

図3は、アプレットの開発とデプロイのモデルを示しています。

Java Cardプラットフォームでは、単一の永続的オブジェクト・フィールドまたは単一のクラス・フィールドに対するどのような更新も原子性を持つことが保証されています。

原子性とトランザクション

Java Cardプラットフォームでは、単一の永続オブジェクト・フィールドまたは単一のクラス・フィールドに対するどのような更新も原子性を持つことが保証されています。さらに、Java Cardプラットフォームは、永続配列での単一要素の原子性も提供しています。すなわち、CADセッション間で保持される必要があるデータ要素（オブジェクトまたはクラスのフィールド、または配列要素）の更新中にスマートカードの電源が失われても、そのデータ要素は以前の値にリストアされます。メソッドの中にも、複数データ要素の一括更新での原子性を保証しているものがあります。たとえば、Util.arrayCopyメソッドでは、原子性によって、すべてのバイトが正しくコピーされることが保証されています。そうでない場合、コピー先の配列には以前のバイト値がリストアされます。アプレットでは、配列を更新する際の原子性は必須ではありません。その場合に使うことができるように、Util.arrayCopyNonAtomicメソッドが提供されています。

アプレットから、複数の異なるオブジェクトに含まれる、複数の異なるフィールドまたは配列要素を原子的に更新することが必要な場合もあるでしょう。つまり、すべての更新が正常かつ一貫性を持って行われるか、すべてのフィールドおよび要素が以前の値にリストアされるかのいずれかになる必要があるということです。Java Cardプラ

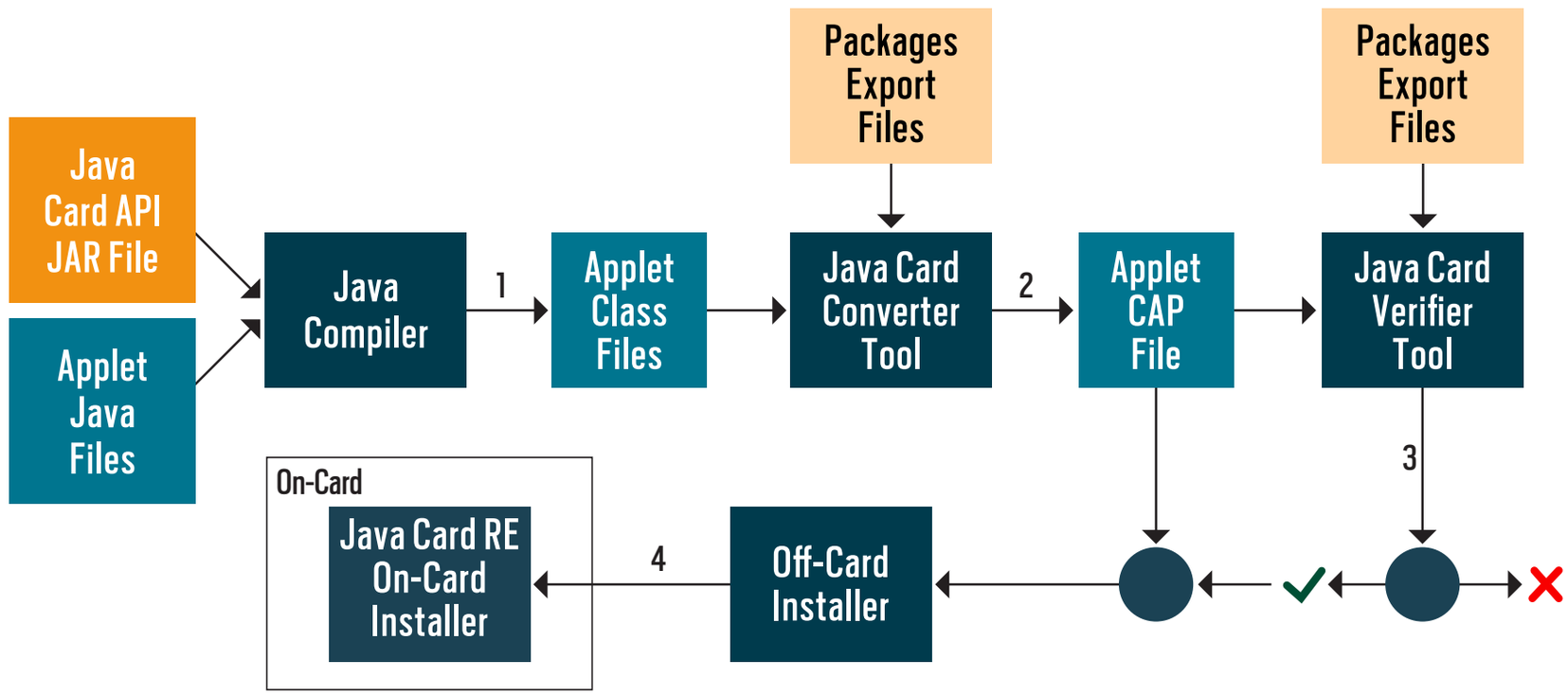


図3：アプレットの開発とデプロイのモデル

ットフォームは、トランザクション・モデルをサポートしています。アプレットは、JCSystem.beginTransactionメソッドを呼び出して、原子性を持つ一連の更新の始まりを指定できます。それ以降のコードでの各オブジェクト更新は、暫定的に行われます。フィールドや配列要素は更新されたように見えますが（フィールドや配列要素を読み取ると、最新の暫定値が返されます）、更新はまだコミットされていません。アプレットがJCSystem.commitTransactionを呼び出すと、すべての暫定的更新が永続ストレージにコミットされます。JCSystem.commitTransactionが完了する前に電源が失われた場合や、その他のシステム障害が発生した場合、暫定的に更新されたフィールドや配列要素はすべて以前の値にリストアされます。アプレットで内部的な問題が発生した場合や、トランザクションをキャンセルすることにした場合は、JCSystem.abortTransactionを呼び出してプログラム的に暫定的更新を取り消すことができます。

アプレットの独立性とファイアウォール

Java Cardのコンテキストとは、各アプレットのパッケージとJava Card REに関連付けられている、保護されたオブジェクト領域のことです。アプレットが所有するすべてのオブジェクトは、アプレットのパッケージのコンテキストに属

しています。

Java Card REの実装は、すべてコンテキストとアプレットの独立性をサポートしています。独立性とは、あるアプレットが別のコンテキストにあるアプレットのフィールドやオブジェクトにはアクセスできないという意味です。ただし、他のアプレットが明示的にアクセス用のインタフェースを提供している場合は除きます。

Java Card Forumは、新しいバージョンのJava Card仕様をリリースするための作業を行っています。IoTマーケットや新しいフォーム・ファクタのセキュア・エレメントに対応する予定です。

Java Cardの重要なセキュリティ機能の1つが、アプレットのファイアウォールです。このテクノロジーは、実行時に強制的に適用される保護で、Javaプログラミング言語での保護（こちらも変わらずJava Cardのアプレットに適用されます）とは別のものです。これによって、強力な型付けと保護属性が強制的に適用されることが保証されます。

Java Card VMでは、アプレットのファイアウォールが常に強制的に適用されます。そのため、実行時にVMが自動的に追加のセキュリティ・チェックを行うことができます。

さらに、Java Card REは独自のJava Card REコンテキストを保持しています。このコンテキストは、アプレットのコンテキストとよく似ていますが、アプレットのコンテキストでは拒否される操作を行えるように、特別なシステム権限が付与されています。図4に、マルチアプレット・アーキテクチャのセキュリティを示します。

アプレットの独立性は重要なセキュリティ機能ではあるものの、アプレットが相互に連携する必要がある状況では、アプレット間でオブジェクトを共有する仕組みが必要です。Java Card REでは、共有可能インタフェース・オブジェクトという考え方を用いてこのような共有が実現されています。これは、あるアプレットのオブジェクトを他のアプレットから使えるようにする唯一の方法です。共有可能インタフェース・オブジェクトを使用する方法について詳しくは、javacard.framework.Shareableインタフェースの説明を参照してください。

クライアント/サーバーAPDUモデル

Java Cardアプレットとの通信インタフェースは、アプリケーション・プロトコル・データ・ユニット（APDU）と呼ばれる、パケットによるメカニズムです。関連する仕様は、ISO 7816 Part 1～Part 6のドキュメントに含まれています。Java Cardアプレットを開発するという目的では、もっとも関連性の高いドキュメントはISO 7816-4（アプリケーション・レベル）です。物理レベルなどの低レベルではさまざまな場合があります。たとえば、接触型I/Oインタフェース向けのシリアル・ポート、I2C、SPIや、非接触型インタフェース向けのISO 14443、SWPなどです。

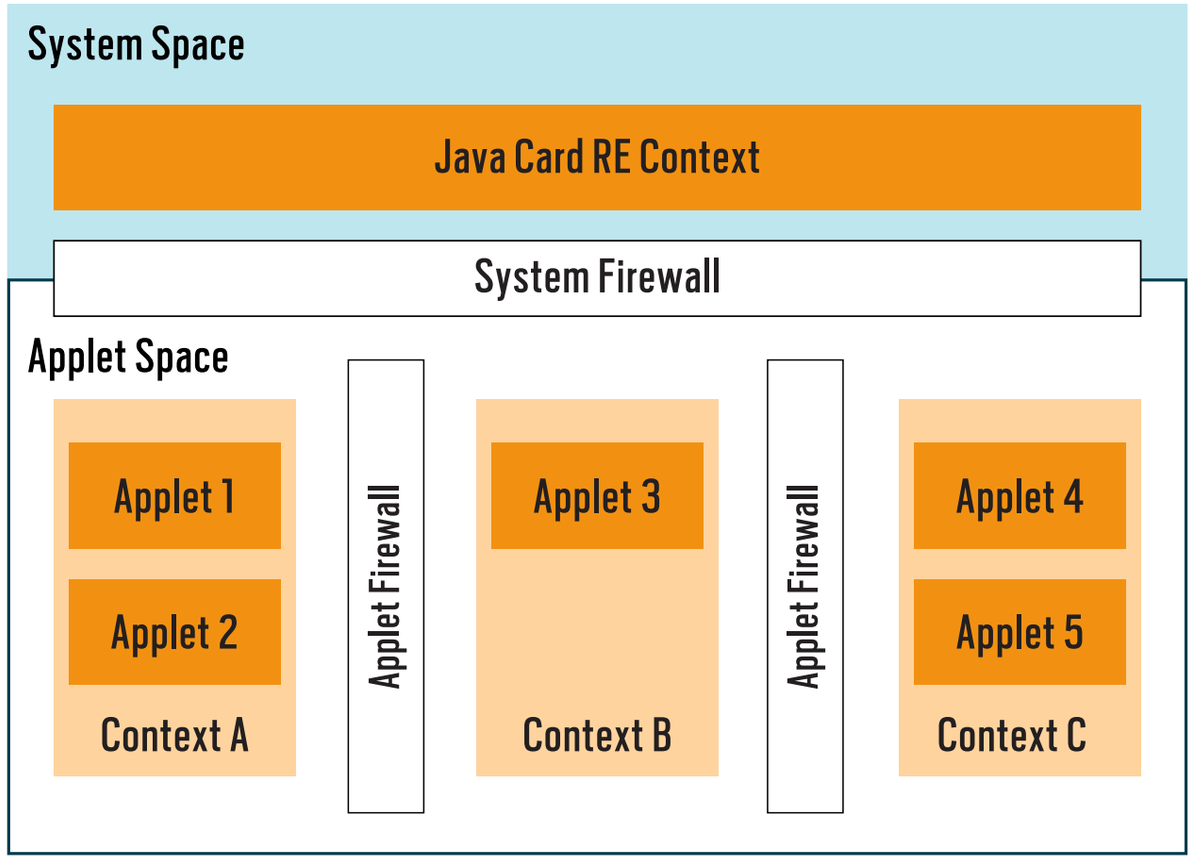


図4：マルチアプレットJava Card実装でのセキュリティ

通信モデルは、コマンド・レスポンス・モデルです。Java Cardアプレットは、CAD内で実行されているクライアント・アプリケーションからリクエストを受信するサーバーとして機能します（図5参照）。Java Card VMは、一度に1つのコマンドを処理します（スレッドはサポートされていません）。しかし、ランタイムは、あるアプレットのセッションと別のアプレットのセッションを同時に管理できます。

アプレットがサポートするプロトコルを定義するためには、処理するAPDUを定義する必要があります。これが、（完全に最初の手順ではないにしても、）アプレット開発における最初の手順の1つになります。

アプリケーション・モデル

すべてのJava Cardアプリケーションは、javacard.framework.Appletクラスを拡張する必要があります。以降では、実装する代表的なメソッドを示します。

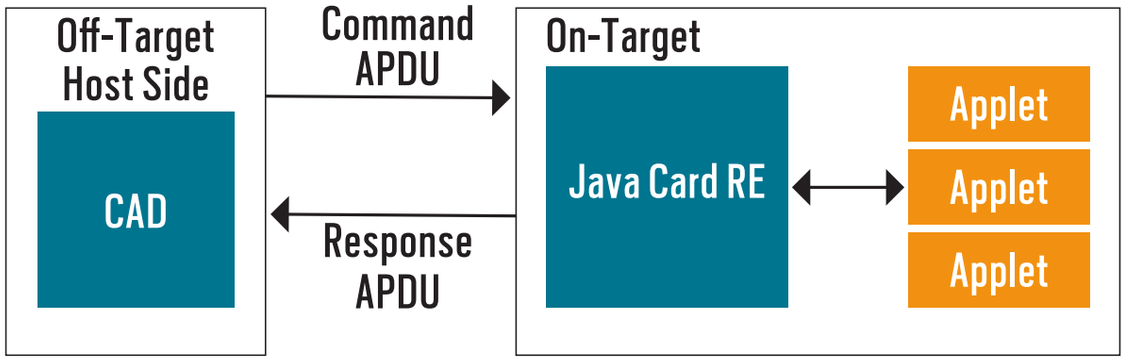


図5：Java Cardと外部デバイスの通信

アプレットのコンストラクタは、install()メソッドによって一度だけ呼び出されます。コンストラクタは、アプレットがメモリ不足にならないように、アプレットのライフタイム全体にわたって使用されるオブジェクトを割り当てる役割を果たします。

install()静的メソッドは、アプレットのインストール処理の際にアプレットのインスタンスを作成するもので、Java Card REによって呼び出されます。アプレットのインストール処理を終えるには、必要な初期化を行ったうえで、いずれかのregister()メソッドの呼出しに成功する必要があります。register()メソッドは、後でアプレットの選択に使用できるように、アプレット識別子（AID）を指定します。AIDは、ISO 7816-5で定義されています。

select()メソッドは、アプレットとセッションを確立しようとしているCADアプリケーションによって、アプレットが選択されたことをそのアプレットに通知するもので、REによって呼び出されます。trueを返すことで、アプレットがAPDUコマンドを受信する準備ができていることを示します。

deselect()メソッドは、選択されていたアプレットが選択解除されたことをそのアプレットに通知するもので、REによって呼び出されます。ここで、アプレットはセッションのクリーンアップを行うことがあります。

process()メソッドは、受信したAPDUコマンドを処理するもので、Java Card REによって呼び出されます。アプレットは、リクエストされたアクションを実行し、存在する場合はレスポンス・データを返すことになっています。

まとめ

本記事では、現在のJava Cardの動作の仕組みを大まかに説明しました。おわかりのように、プログラミングは単純です。経験やスキルが求められるのは、セキュリティの側面です。オラクルやJava Card Forumのパートナーは、新しいバージョンのJava Card仕様であるバージョン3.1をリリースするための作業を行っています。そこには、IoTマーケットや新しいフォーム・ファクタのセキュア・エレメントに対応する機能が導入される予定です。2018年にリリースされる見込みであり、今後のJava Magazineの記事で取り上げる予定です。</article>

Nicolas Ponsini：オラクルのセキュリティ・ソリューション・アーキテクト。セキュリティ、暗号化、IoT、Trusted Execution Environmentの専門家で、関連分野で9つの特許を保有。

Florian Tournier：オラクルの開発部門のシニア・ディレクター。小型デバイスやIoTセキュリティ向けのJava製品の構築や提供を行うチームを統率。Sun MicrosystemsでJava MEおよびJava Cardの製品管理を率いていたが、買収に伴い2010年にオラクルに入社。HEC Parisで経営学の修士号を、Ecole Centrale de Lyonでコンピュータ・サイエンスの学位を取得している。