

# CACHE: Java アプリケーションに 柔軟で高パフォーマンスの永続性を与える

技術白書 : InterSystems Corporation

## はじめに

---

Java は、アプリケーション開発のための主力技術の1つであることは、疑う余地のないことです。その“一度書けばどこでも動く”という性質により、Java 環境は、Web ベースのアプリケーションおよび統合アプリケーションを作成する上で自然な選択となり、オブジェクト指向言語として、Java は迅速な開発を可能にしました。ただし、この Java のオブジェクト指向により、Java アプリケーションにデータの永続性を提供することがこれまで大きな課題となってきました。

このために 2 つの基本的なアプローチが開発されました。1 つは、データの格納と取得に JDBC を使用して、情報をリレーショナル・テーブルとしてモデル化する方法です。この方法には、Java アプリケーションを、現在依然として広く使用されているリレーショナル・データベースと互換性を持つようにできるという利点があります。ただし、複雑なオブジェクトと 2 次元テーブルの翻訳に必要な“マッピング”には、開発者の生産性とアプリケーション・パフォーマンスの両面で、代償が伴います。

Java ベースのアプリケーションにデータの永続性を提供するためのもう 1 つの基本的なアプローチは、データをオブジェクトとして格納および取得する方法です。このアプローチでは、オブジェクト/リレーショナル・マッピングにより生じる生産性とパフォーマンスの低下を回避することができますが、オブジェクト・データベースの使用を前提としていません。この方法の難点は、多くの組織では現時点でオブジェクト・データベースを使用していないということです。

同じ選択（データをテーブルとして格納するか、データをオブジェクトとして格納するか）は、Enterprise Java Beans (EJB) に永続性を与える上でも当てはまります。この場合、コンテナ管理パーシスタンス (CMP: Container-Managed Persistence) と、Bean 管理パーシスタンス (BMP: Bean-Managed Persistence) のどちらを使用するかを決定するという更なる問題も加わります。後者は通常アプリケーション・パフォーマンスの向上につながりますが、前者は実装がすばやく簡単であるという実績があります。

この白書では、Cache を使用して、データをリレーショナル・テーブルおよびオブジェクトの両方として、同時にまたシームレスにアクセス可能にする方法を説明します。また、Java 環境での Cache のさまざまな使用方法について、特に高性能の Bean 管理パーシスタンスを使用する J2EE エンティティ Bean の自動生成を可能にする、Cache の EJB プロジェクトに重点をおいて詳しく説明します。

## 多次元の柔軟性 – Caché の統一データ・アーキテクチャ

---

Caché では、Java 開発者はデータのモデル化をリレーショナル・テーブルまたはオブジェクトのいずれでしか実行できないように限定されることはありません。これは、Caché がリレーショナルでも、“純粋なオブジェクト” データベースでもないからです。むしろ、その基盤となるデータ・エンジンは情報をまばらな (スパース) 多次元配列として格納します。多次元構造は、データ・オブジェクトの複雑性を表現する上で十分表現力豊かでありながら、容易に 2 次元 (テーブルなど) 構造として投影することもできます。そのため、Caché の効率的な多次元配列は、開発者に対してリレーショナル・テーブルとしても、オブジェクトとしても同時に表現できます。

この二方向性の性質は、Cache 統一データ・アーキテクチャと呼ばれます。これにより、Java 開発者はアプリケーションに永続性を与えるために自分の好みの方法を選択できます。

<sup>1</sup> 実際のところ、三方向性の性質とも言うことができます。というのは、Cache では多次元配列の直接操作も可能だからです。

ただし、Java 開発者にとって、直接のデータ・アクセスは、Cache の SQL およびオブジェクト・アクセスに比べて意義が低いと思われる。

<sup>2</sup> 詳細は、Cache ドキュメントを参照してください。

## データをリレーショナル・テーブルとして処理する – Caché SQL および JDBC

Java 開発者の中には、SQL と JDBC を使用してデータを処理し、Cache をリレーショナル・テーブルのように扱うことを選択する人もいます。このアプローチは、既にリレーショナル・テーブルへの SQL 呼び出しを含む既存の Java アプリケーションにおいて Cache を使用する開発者にとって、特に効果的です。Cache では、リレーショナル・データ・アクセスがサポートされるため、そのようなアプリケーションは非常にわずかな変更を行うだけで、Cache に対して実行できます。

Java クライアント側で変更が必要なのは、Cache データ・サーバに接続する必要があった場合のみです。まず、システム環境変数 CLASSPATH に、CacheDB.jar ファイルのロケーションを含めるよう変更する必要があります。CacheDB.jar (InterSystems により提供され、既定の Cache インストールに含まれています) には、Java サポートに必要なすべての Cache システム・クラスの“純粋な Java”バージョンが含まれています。

次に、CacheDB.jar の以下のパッケージを Java アプリケーションにインポートする必要があります。

***com.intersys.objects*** Caché サーバと通信するために必要な接続とキャッシュのメカニズムを実装します。  
***com.intersys.objects.reflect*** アプリケーションで Java のリフレクション機能を活用できるようにします。  
***com.intersys.jdbc*** JDBC 接続を供給します。

これらのパッケージがインポートされると、Java 開発者は標準的なメソッドを使用して Caché サーバに接続し、SQL クエリやコマンドを実行できるようになります。Caché には、Caché データベースへの方向接続のために JDBC 呼び出しを Caché にネイティブなプロトコル・フォーマットに変換する、タイプ 4 JDBC ドライバが含まれています。Caché の JDBC ドライバは、JDBC 2.0 準拠で、いくつかの拡張があります。

もちろん、Caché の対象データベースには適切なデータ・テーブルが含まれている必要があります。既存の Java アプリケーションが Caché に変換される場合は、使用中のリレーショナル・データ・スキーマが既に存在する必要があります。それらの DDL テーブル定義は、Caché にインポートしてコンパイルできます (その際 Caché の統一データ・アーキテクチャは、自動的にそれらのリレーショナル・データ構造をオブジェクトとしても使用可能にします)。新規の開発には、Caché は必要なデータ構造をすばやく容易に作成できるウィザード方式の IDE を用意します。

Caché データベースにアクセスするために SQL と JDBC を使用することの利点は何でしょうか。現在リレーショナル・データベースにアクセスする既存の Java アプリケーションにとっては、パフォーマンスの向上とシステム管理機能であると言えます。Caché はリレーショナル・データベースであるかのようにアクセスできますが、情報は実際には効率的な多次元配列に格納されます。多次元データ構造を使用すると、リレーショナル技術に付きものの“ジョイン”と“テーブル・ホッピング”が排除され、SQL クエリに対応する応答が高速化します。実際、Caché の SQL 応答が、リレーショナル・データベースに比べて 5 倍も高速になることはよくあります。

新規の Java 開発にとって、永続性を提供するために SQL と JDBC を使用することの主な利点は、データベース独立性です。アプリケーションは、Cache に対してだけでなくリレーショナル・データベースに対しても実行可能にする必要がある場合があります。また、JDBC に精通した開発者は、他のアプローチよりも JDBC の使用を好むかもしれません。ただし、多くの場合、Java 開発者は選択肢が与えられるなら、情報をオブジェクトとして格納して操作することを選ぶでしょう。

## データをオブジェクトとして処理する – Caché の Java バインディング

---

Caché をオブジェクト・データベースとして使用すると、Java 開発者はオブジェクト/リレーショナル・マッピングによって生じる生産性とパフォーマンスの低下を経験する必要がありません。このアプローチは、開発者がデータをモデル化する方法を選択できる場合に、新規の Java アプリケーションに対して最もよく使用されます。

Caché は、カプセル化、多重継承、多態性、参照オブジェクトおよび埋め込みオブジェクト、コレクション、リレーションシップなど、Java 開発者にとって馴染み深いオブジェクト・モデリングの概念を完全に実装しています。Caché クラスは、Caché スタジオで容易に定義および編集できます。Caché スタジオは、共通の開発タスクを素早く完了するためのウィザードを含むグラフィカル IDE です。ウィザードの 1 つに、Caché クラスを (他のオブジェクトの中で) Java クラスまたは Enterprise Java Beans として迅速に投影するための、Projection ウィザードがあります。

クラス定義に Java プロジェクションが含まれる場合、Caché はコンパイル時に Caché クラスと対応する Java プロキシ・クラスの両方を生成します。

プロキシ・クラスは純粋な Java で記述されます。プロキシ・クラスは、CacheDB.jar に含まれる純粋な Java システム・クラスから、その (クラスの格納、取得、保存のための) 永続メソッドを継承します。Java クラスとして、プロキシ・クラスのすべてのプロパティはプライベートであると見なされます。アクセサ・メソッドが Caché により自動的に生成され、クラス定義に含められます。

プロキシ・クラス (純粋な Java クラスとしての) は、それ自身がコンパイルされなければ、実行可能になりません。プロキシ・クラスは、一度コンパイルされると、Caché サーバに接続可能なすべての Java アプリケーションに対して使用できるようになります。接続を作成するためのクライアント・サイドの要件は、JDBC を使用する際と同じです。つまり、CacheDB.jar のロケーションをアプリケーションの CLASSPATH 宣言に追加し、CacheDB.jar の適切なパッケージをインポートする必要があります。Caché サーバ・接続では、Caché データベースに対する JDBC アクセスとオブジェクト・アクセスを同時にサポートします。

Java プログラマが Caché データベース・エンジンに真のオブジェクト・アクセスを実行できるようにすることによって、Java と Caché の間でずっと効率的な仕事の配分が可能になります。一般的には、データベースにアクセスする必要のあるすべてのメソッドが、Caché サーバでより高速に実行できます。ただし、Caché のランタイム環境には、現在 Java 仮想マシンは含まれません。Caché サーバ・サイドのメソッドは、Basic または Caché ObjectScript で記述する必要があります。

開発者が Caché クラスから派生した Java プロキシ・クラスにクライアント・サイドのビジネス・ロジックを追加したい場合は、最初にその Java プロキシからすべてのプロパティとメソッドを継承する新しいクラスを作成し、次にその新しいクラスに対して変更や追加を行うことをお勧めします。そのようにすると、何らかの編集がデータ・サーバ上の Caché クラスに対して実行され、対応する Java プロキシの新しいバージョンが作成された場合に、クライアント・サイドで変更を繰り返す必要がなくなります。クライアント・サイドのビジネス・ロジックを追加するもう 1 つのオプションがあります。Caché では Caché クラス定義に Java コードを含めることを許可していますが、Java コードは Caché クラスの実行可能な形式にコンパイルすることはできません。その代わりに、コードは Java プロキシ・クラスに (書かれたとおりに) 含められます。Java プロキシがコンパイルされると、コードは Java クライアント上で実行されます。

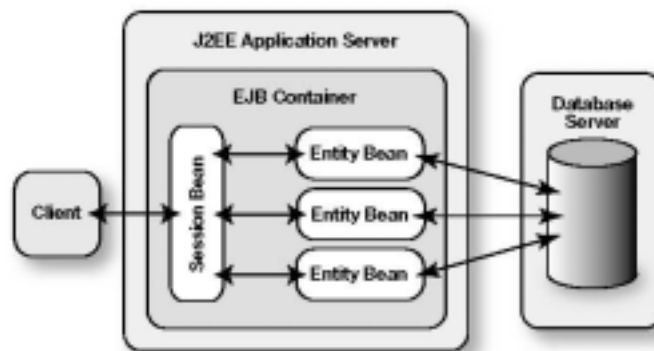
## アプリケーション・サーバで作業する – Enterprise Java Beans

Enterprise Java Beans は、n 層の分散アプリケーションの一部としてすべての EJB 互換アプリケーション上で実行するよう設計された Java クラスです。アプリケーション・サーバでは、トランザクション・モニタリング、セキュリティ、スレッディングなどのシステムレベルのサービスと、永続化エンジンに対するインタフェースが提供されます。

図 #1 に汎用 EJB アーキテクチャを示します。以下の 2 種類の基本的な Bean があります。

- ・ セッション Bean は永続的ではなく、ステートレスです。通常ビジネス・ロジックを含み、1 つ以上のエンティティ Bean と相互作用します。Cache では、クラスをセッション Bean として投影できます。(前述の、Java で記述されたビジネス・ロジックを実装する方法を参照してください)。
- ・ エンティティ Bean は永続的であるため、データベースと対話する必要があります。すべての Java クラスと同様、エンティティ Bean は JDBC または Cache の Java バインディングを使用して Cache にアクセスできます。

図 #1: 典型的な J2EE アーキテクチャ



エンティティ Bean についてさらに考慮すべき事項は、アプリケーションでコンテナ管理パーシスタンス (CMP) と Bean 管理パーシスタンス (BMP) のどちらを使用するかという点です。

コンテナ管理パーシスタンスとは、アプリケーション・サーバ・ソフトウェアが、すべてのエンティティ Bean に対するデータベースとの対話を処理することを意味します。CMP の利点は、Java アプリケーション開発者が、各エンティティ Bean に対して永続化メソッドを書く必要がないということです。CMP の不利な点は、パフォーマンスの低下を招く恐れがあることです。CMP では Bean とデータベースの間に処理用の余分なレイヤを追加するだけでなく、アプリケーション・サーバ・ソフトウェアが永続性を提供するために常に SQL と JDBC を使用します。これにより J2EE アプリケーションがほとんどの一般的に使用されているデータベースと互換性を持つようになりますが、Java オブジェクトとリレーショナル・データベースの間の“インピーダンス・ミスマッチ”を解決するために必要なプロセスのオーバーヘッドにより、パフォーマンスが低下する可能性があります。

高パフォーマンスを必要とする J2EE アプリケーションのために、多くの Java 開発者は Bean 管理パーシスタンスを使用することを選びます。これは、各エンティティ Bean にカスタムの永続メソッドを記述して、各 Bean が最も効率的な仕方で“それ自身を格納する方法を知っている”ようにする必要のあることを意味します。通常、BMP はエンティティ Bean をオブジェクトとして格納することができるデータベースで使用され、インピーダンス・ミスマッチによるパフォーマンスの低下を回避できるようにします。BMP の利点は、高パフォーマンスです。不利な点は、通常 BMP では多量のハンド・コーディングが必要となり、結果として開発と保守のサイクルに、より長期の時間がかかることです。

## Caché の EJB バインディング

---

Caché では、標準的な Java クラスとしてクラスを投影すると本質的に同じ方法で、クラスを Enterprise Java Beans としても投影します。投影に必要なのは、下の図 #2 に示すように、プロジェクト文を含めることだけです。Caché クラスのコンパイル時には、必要とされるすべての J2EE 互換 java bean コード、必要なすべてのインタフェース、テスト・サブレット、テスト Web ページ、およびアプリケーションで必要とされるすべてのデプロイメント記述子が生成されます。

生成されるコードは J2EE 互換ですが、デプロイメント記述子など、いくつかのアイテムは各 EJB アプリケーション・サーバに固有となります。このドキュメントの作成時点では、Caché では以下の EJB アプリケーション・サーバがサポートされます。

BEA WebLogic Server 8.1  
JBoss 3.2  
Pramati 3.5  
WebSphere 5.1

各 EJB プロジェクトには、開発者が設定する必要のある複数のパラメータがあります (Projection ウィザードでは、EJB を生成するために Cache が必要とする情報のリストが提供されます。経験のある開発者は、プロジェクトを Cache クラス定義に“ハンド・コーディング”することができます)。このパラメータの 1 つは、PERSISTENCETYPE パラメータで、EJB がコンテナ管理パーシスタンスと Bean 管理パーシスタンスのどちらを使用するかを指定します。

図 #2: 典型的な EJB プロジェクト文

**BEA WebLogic 8.1 で BMP プロジェクトを作成するこのプロジェクト文は、Caché クラス定義に含まれます。**

```
Projection BeaWebLogic8 As %Projection.EJBWebLogic(APPLICATIONDIR =
"C:¥bea¥user_projects¥domains¥mydomain¥applications", APPSERVERHOME =
"c:¥bea¥weblogic81¥server", BEANNAME = "MyBean", CLASSLIST =
"MyPackage.MyClass", JAVAHOME = "c:¥bea¥jdk141_05", PACKAGE =
"MyPackage", PERSISTENCETYPE = "BMP", ROOTDIR =
"C:¥path_to_projected_files", SERVERTYPE = "WEBLOGIC8");
```

<sup>3</sup> 最新のリストについては、[www.intersystems.com](http://www.intersystems.com) を参照してください。

CMP が選択された場合は、Cache は標準の JDBC 互換 Java コードを生成します。BMP の場合は、Cache は自動的に (オブジェクト・データ・アクセスを使用して) 以下の複数の永続メソッドを EJB 内に実装します。

```
ejbCreate( )
ejbLoad( )
ejbStore( )
ejbRemove( )
ejbFindByPrimaryKey(string key)
```

これらのメソッドは、他のテクノロジーではコーディングに数時間かかる場合がありますが、Cache では数秒で自動的に作成されます。Cache を使用すると、J2EE 開発者は迅速かつ容易に Bean 管理パーシスタンスを実装できます。アプリケーションは、BMP によくある長期間の開発サイクルを回避しながら、高パフォーマンスを享受できます。Cache では、各 Bean のテストと配置に必要なすべてのファイルを自動的に生成することにより、(PERSISTENCETYPE でどちらが選択されたかどうかに関わらず) 開発サイクルをさらに短縮します。

## まとめ

---

Java アプリケーションにデータの永続性を与えるための唯一の“正しい”方法などはありません。アプリケーションの要件と利用できるデータベース・テクノロジーに応じて、Java 開発者はデータを単純なテーブル構造としてモデル化するか、複雑なデータ・オブジェクトとしてモデル化するかを決定します。J2EE アプリケーションには、コンテナ管理または Bean 管理パーシスタンスを選択するかもしれません。永続性を実装する方法を決定するには、通常データベース独立性、アプリケーションのパフォーマンス、および開発に要する時間と労力のいずれを選ぶかという妥協が求められます。

Cache ではこの妥協を最小限にします。Cache の統一データ・アーキテクチャでは、データがリレーショナル・データベースとしても、表現力豊かなデータ・オブジェクトとしても、マッピングを必要とせずに同時にアクセスできるようになります。Bean 管理パーシスタンスによる Enterprise Java Beans の自動生成により、開発者は高パフォーマンスの J2EE アプリケーションを、コーディングに多大の労力を要することなく作成できるようになります。Java 開発者がデータの永続性を実装するという課題に対してどのアプローチを選ぶかに関わらず、Cache は最小限の開発努力で最大限のパフォーマンスを提供できるよう設計されています。

2004年12月

インターシステムズジャパン株式会社  
〒160-0023 東京都新宿区西新宿6 - 10 - 1  
日土地西新宿ビル17F  
<http://www.InterSystems.co.jp/>

INTERSYSTEMS

InterSystems Caché および Ensembleは、米国インターシステムズ社の登録商標です。その他の製品名は該当各社の登録商標または商標です