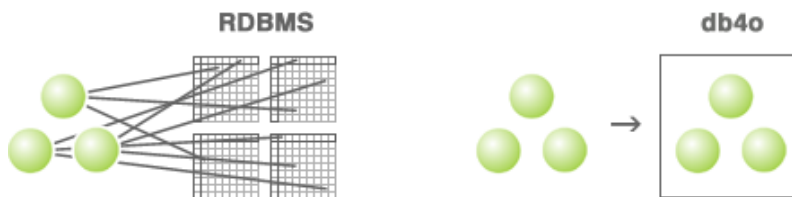


バージョン5.0 | Java and .NET

db4o オープンソースオブジェクトデータベース

db4oは、Javaと.NET技術者の開発時間とコストを大幅に削減し、これまでにない性能を実現するオープンソースオブジェクトデータベースです。db4oのネイティブなオブジェクトデータベースエンジンは、端末や機器、モバイルやデスクトップ上で動くパッケージソフトウェア、リアルタイム制御システムの組み込みデータベースとして最適です。簡単に言うと、DBA不在のJavaと.NET環境に最適と言えます。

リレーショナルデータベース (RDBMS) またはオブジェクトリレーショナルマッパーとdb4oオブジェクトデータベース

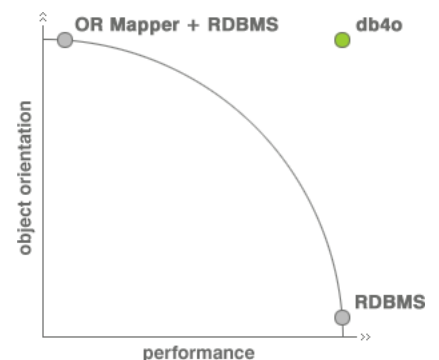


オブジェクト指向開発の技術者であれば、オブジェクト指向で考え、リレーショナルデータベースに当てはめる問題によく悩まされます。そのような時これまで、スピードとオブジェクト指向のどちらかの選択を余儀なくされてきました。SQLによるアクセスはスピードで勝りますが、大量の追加コードを書かなくてはなりません。オブジェクトリレーショナルマッパーはこの部分を代わりにやってくれますが、その分スピードを犠牲にしまいます。

db4oはオブジェクト指向とスピードの矛盾を解決するものです。非常に複雑なオブジェクト構造であっても、スピードを落とすことなく簡単に格納することができます。データベースのベンチマークでは、有名なオブジェクトリレーショナルマッパーとリレーショナルSQLデータベースである、HibernateとMySQLより44倍速いという結果が出ました。

今日リレーショナルデータベースが利用される大きな理由は、レガシーシステムです。例えば旧式の基幹システムデータと、それに依存する一連のアプリケーション保全のためです。しかしサーバー中心のシステムを超えて、従来のデータベース技術では不十分な、端末やモバイル、デスクトップ上で動作するアプリケーションが多数見受けられるようになってきました。このようなケースで、db4oはパフォーマンスや機能、費用対効果でワンランク上のソリューションを提供することができます。

db4oは、Javaと.NET技術者の開発時間とコストを大幅に削減し、これまでにない性能を実現するオープンソースオブジェクトデータベースです。db4oのネイティブなオブジェクトデータベースエンジンは、端末や機器、モバイルやデスクトップ上で動くパッケージソフトウェア、リアルタイム制御システムの組み込みデータベースとして最適です。簡単に言うと、DBA不在のJavaと.NET環境に最適と言えます。



製品 - アプリケーション、特徴、メリット

db4oは、オブジェクト指向開発で、端末やモバイル機器、またはデスクトップやサーバーへの、組み込みデータベースとして必要な機能を網羅するように設計されています。ゼロメンテナンス、省スペース、高性能、データ同期、リファクタリングといったリレーショナルデータベースに不足している機能が必要な場合、db4oがその答えです。Javaと.NETにネイティブなdb4oの単一ライブラリ(.jar/.dll)がアプリケーションへの組み込みを容易にし、非常に信頼性と拡張性のあるデータ操作を、どんなに複雑なオブジェクト構造であっても、一行のコードで行うことができます。

この結果、開発者は以下のことができるようになります：

- コードを複雑にし、スピードとリファクタリング性能を低下させることが証明されている、オブジェクトリレーショナルマッピングに必要なコードまたはツールが不要になります。
- アプリケーションと完全に一体となったデータストレージによって、実行時のメンテナンスが不要となり、従来型のデータベースよりも、より高い信頼性とより速い性能を実現します。
- データベースに制約を受けない完全なオブジェクト指向開発によって、自然で豊富な機能を持つオブジェクトモデルをコストとマシンリソースを犠牲にせずに実現することができるので、より複雑なシステム開発を可能にします。
- 変更や修正、ソフトウェアコンポーネントの再利用が、旧コードの書き直しや大きなコストをかけずに行えます。その柔軟性によって、競合他社をリードすることができます。

db4oは合計25万ダウンロードを超えました。鉄道やネットワーク、自然科学や工業用、消費者と大企業向け、その他多数の業界のアプリケーションとして導入実績があります。db4oのユーザーと顧客は、現在168カ国、アルバニアからジンバブエまで多国籍にわたり、ボーイングやBosch、BMWやHertzなどの大企業から、非常にイノベティブなベンチャー企業にまでわたっています。

最新のオブジェクトデータベース技術を駆使し、db4oはJavaと.NETをネイティブにサポートする唯一のデータベースです。特定のデータベースに依存する高いライセンス体系からユーザーを開放する、クロスプラットフォームな機能を提供します。一連のユニークでオブジェクト指向なデータベース機能をサポートすることで、オブジェクト指向言語のメリットを最大限発揮できるよう後押ししています。オブジェクト指向レプリケーション、オブジェクト指向クエリ(ネイティブクエリ、QbEやS.O.D.A)、オブジェクトデータベースファイルの中身を閲覧するObjectManagerGUIは、データベース界で他に例の無い機能です。

db4o 導入事例

www.db4o.com/about/customers



AVE High Speed Trains



BOSCH's Packaging Robots

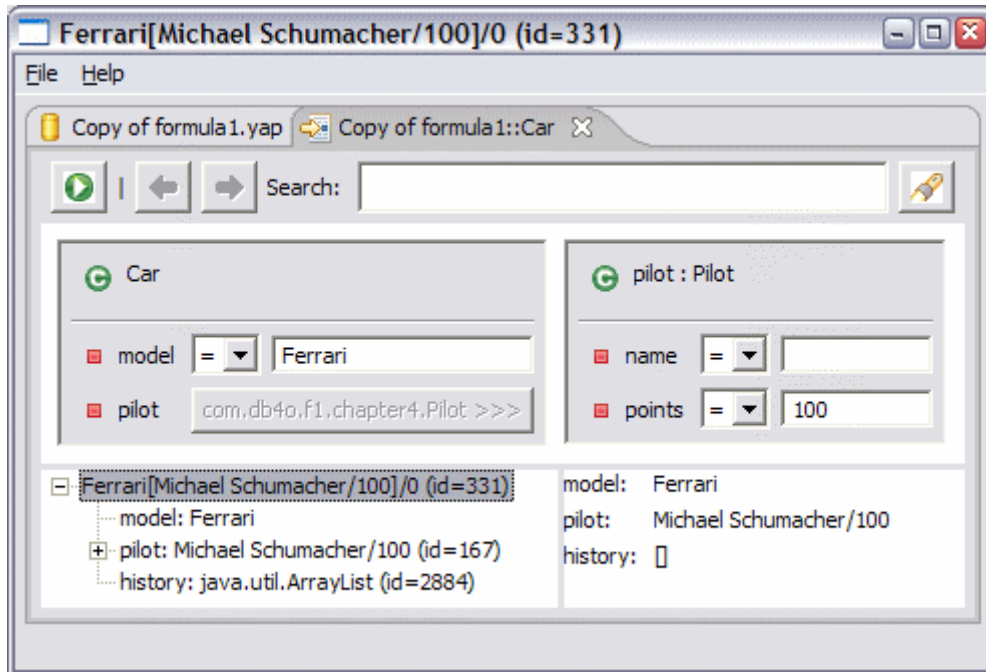


Clarity Medical's RetCam II



Eastern Data's Mobile Apps

www.db4o.com/about/customers



db4o ObjectManagerを使ってクエリの実行やファイル内の閲覧が可能

そして何よりも、db4oの学習、実装、使用が非常に簡単です。db4oの強力なデータベースエンジンによって、ユーザーはたった1行のコードでオブジェクトを格納することができますので、データの永続化に関する開発時間とコストを最小にします。

そして、リファクタリングや新しい機能の追加、ソフトウェアコンポーネントの再利用をしようとするとき、これらのメリットは4倍になります。db4oはネイティブでないAPIやハードコーディングした文字列を使用していないので、開発環境を通じてリファクタリングを自動化しています。もしインストールされている環境に更新があれば、自動バージョンングオブジェクトがオブジェクトモデルの変更を自動で反映させます。変換作業は不要です。技術者はエンハンサーやプリコンパイルング、またはポストコンパイルングをしなくてもいいのです。変更を反映することが不可能にさえ思えるレプリケーションでさえ、分散データ構造を壊すことはありません。

db4oは、ダウンロードしてわずか5分ほどで使い始めることができます。ダウンロードされたファイルには、リレーショナルデータベースからオブジェクトデータベースへの移行を容易にできるようにした、開発者用チュートリアルも含まれています。

db4oホワイトペーパー フリーダウンロード可

- The Database Behind the Brains, Rick Grehan 著
- Complex Object Structures, Persistence, and db4o, Rick Grehan 著
- Native Queries for Persistent Objects, Willian Cook, Carl Rosenberger 共著
- オブジェクトデータベースのアジャイル開発手法, Scott Ambler

www.db4o.com/about/productinformation/whitepapers

ベンチマーク

www.db4o.com/about/productinformation/benchmarks



主な機能とメリット

主要機能	メリット
<p>1行データベース あらゆるオブジェクトの1行格納 クラス = オブジェクト構造 効率的な製品化</p>	<p>データ永続化コストを90%削減</p>
<p>組込み可能 ゼロメンテナンス 自動バージョンング 400 KBの省スペース</p>	<p>製品の市場投入10%スピードアップ</p>
<p>複数プラットフォーム Javaと.NETにネイティブ 携帯端末、PC、サーバー クロスプラットフォーム</p>	<p>従来のデータベースに比べ44倍速い</p>
<p>オブジェクト指向をデータベースへ オブジェクト指向レプリケーション ネイティブクエリ ObjectManagerブラウザ</p>	<p>ゼロメンテナンスなので大量配布可能</p>
	<p>無駄のない完全なオブジェクト指向ソフトウェアの開発</p>
	<p>分散環境で、同期を必要とするデータ構造の構築</p>
	<p>最小エラー、リファクタリング、長寿命</p>

1行データベースによる開発時間の短縮

これは次のように簡単です。db4oの単一ライブラリ(.jar /.dll)を開発環境に入れ、データベースファイルを開き、どんなに複雑なオブジェクトであっても、任意のオブジェクトを1行のコードで格納します。以下にJavaによるコードを示します。

```
public void store(Car car){
    ObjectContainer db =
        Db4o.openFile("car.yap");
    db.set(car);
    db.commit();
    db.close();
}
```

他に例を見ないほど簡単なこの操作性が、開発時間の短縮を実現します。

データベース構造の設計、実装、メンテナンスは不要です。クラス構造がデータベース構造だからです。従来データベースに必要な作業、例えばハードコーディングした文字列やXML、プリコンパilingやポストコンパiling、エンハンサーを必要とするその他のネイティブでないファイルなど結果的に製品化作業を遅らせていたものも不要となります。

評価の高いF1を例にした対話式チュートリアルを使いながら、ここまでわずか5分ほどです。

このように、ソフトウェア開発で大幅な時間削減が可能になります。

さらに大幅な時間削減を可能にするのは、ソフトウェアを変更する時です。リファクタリング、新しい機能の追加、ソフトウェアコンポーネントの再利用などです。オブジェクトモデルの変更が完全に反映されるだけでなく、ネイティブで操作容易性を追及するdb4oは、開発環境に全ての作業を実行させますので、間違えようがないほど簡単です。デバッガーやビルドは不要ですし、実装済みのアプリケーションを気にする必要もありません。インストールされている環境へのあらゆるオブジェクトモデルの変更はdb4oが対処するからです。これまでにない生産性を手にし、ソフトウェアの変更はもはや悪夢ではなく喜びとなるでしょう。

要するに、db4oは通常のシリアル化と同じように簡単にオブジェクトを永続化するだけでなく、クエリなどのデータベースに必要な機能を全て提供しながら、驚くべきことに、シリアル化を損なうことなくオブジェクトモデルの変更を可能にするということなのです。

組込みデータベース

db4oは、ユーザーからは完全に分からないように、クライアントやソフトウェアコンポーネントに組み込めるように設計されています。そのため、アプリケーションの他に別途インストールする必要はなく、わずか400KBのライブラリとして提供されています。また、db4oはアプリケーションと同一のスレッド上で動作するので、メモリ管理、スピードプロファイリングやデバッグなどを全て制御することが可能です。アプリケーションが起動済みであれば、データベースも起動されています。例外はありません。

最も重要なのは、インストールされている環境でのオブジェクトモデルの変更に対し、db4oは極めて柔軟性があるということです。db4oは常に管理者不在を想定していますので、アプリケーションは旧モデルから新モデルへの移行を日常的に行っています。他のデータベースのソリューション、リレーショナルやネイティブでないオブジェクトデータベースとは違って、db4oは変更を管理するために特に何も必要としません。すべてのパッケージを用意する必要性や、ソースのエラーは無くなります。

ポータビリティとクロスプラットフォーム実装

これほど多くのオブジェクト指向プラットフォーム上でネイティブに動作する組込みデータベースはほとんどありません。ましてや混合環境で動作するものではありません。db4oを使えば、様々なプラットフォーム上や、WindowsクライアントとJavaサーバーの混合環境上に実装することが可能です。

db4oはJava JDK 1.1.xから5.0をサポートし、J2EEとJ2SE上で動作することが可能です。また、リフレクションをサポートする、例えばCDCやPersonalProfile、Symbian、SavajeやZaurusといったJ2MEダイアレクトを利用して動作することも可能です。顧客の需要に応じて、db4oはリフレクションをサポートしていない、RIM (Blackberry) とPalm OSを含むCLDCやMIDPをサポートする予定です。

db4oはさらに、全ての.NETプラットフォーム上、.NETやコンパクトフレームワーク、monoで動作し、全てのマネジドコード、例えばC#やVB.NET、ASP.NET、Boo、マネジドC++をサポートしています。

一時接続クライアントと分散データ構造

db4oは主にスタンドアロンクライアント、例えば自動車や携帯電話などで利用されますが、それらはいわゆる、少なくとも一時的にでも、ミドルウェアやサーバーまたは他の装置に接続されます。そのためdb4oは、組込みモードだけでなく、エンタプライズシステムやサーバーサイドソフトウェアにも組み込めるよう、クライアントサーバーモードでも利用可能です。

db4oのユニークな機能であるオブジェクト指向レプリケーションを使えば、クライアントサーバー間やピアtoピアでのデータ同期を簡単に行うことができます。2つのデータベースに接続し、同期が必要なオブジェクトを検索し、1行のコードでオブジェクトを同期します。親子関係にあるクラスは必要なものを自然に記述しているので、親オブジェクトを同期するときには必要な子オブジェクトも同期されます。また、オブジェクトのデータが競合した場合に、競合を処理するビジネスロジックを記述することができます。これにより、複数のチームによる異なるモジュール間のような分散環境下で動き回る、“スマートオブジェクト”を作成することが可能です。またリファクタリングにも柔軟に対応します。

全体的に見ると、db4oの多様な動作モードとユニークなオブジェクト指向レプリケーションによって、特にサービス指向コンピューティングに要求される、非常にパワフルで効率的な分散データ構造を構築することが可能です。

ネイティブクエリ

db4oバージョン5で、db4objects社は世界で初めて、ネイティブクエリ(NQ)を実装しました。ネイティブクエリは、データベースクエリをプログラミング言語で記述するという、Microsoft社のLINQ(.NET Language Integrated Queries)プロジェクトによってその妥当性が証明された、業界のトレンドに先駆けて実装されました。

文字列ベースのAPI(SQLやOQL、またはJDOQLやEJBQL、SODAなど)ではなく、NQは開発者がプログラミング言語(例えばJavaやC#、VB.NETなど)そのものを、データベースアクセスに使うことができます。これによって、生産性を低下させる要因となる、プログラミング言語とデータアクセスAPIの切り替えをする必要がなくなります。

それではネイティブクエリで記述された以下のクエリを考えてみましょう:

```
List<Student> students = database.query<Student>( new Predicate(){
    public boolean match(Student student){
        return student.getAge() < 20
            && student.getGrade().equals(gradeA); }});
```

同じクエリがJDOQLだと次のようになります:

```
String param = "String gradeParam"
String filter = "age < 20 & grade == gradeParam";
Query q = persistenceManager.createQuery(Student.class, filter);
q.declareParameters(param);
Collection students = (Collection)q.execute(gradeA);
```

ご覧になって分かるとおり、ネイティブクエリはすべての文字列をクエリから排除することが可能です。従って100%タイプセーフ、100%リファクタブル、100%オブジェクト指向であると言えます。例えば上記の例では、たとえageフィールドが日付型だったとしても、タイプミスマッチによるエラーは、実行されるまでスローされません。IDEを利用すれば、NQは完全にタイプセーフですから、コーディングの段階でタイプミスマッチのエラーがスローされません。

結果的に、NQを利用してデータアクセスを行えば、ソフトウェアの生産性を向上させることができます。また、頻繁にリファクタリングやオブジェクトモデルの変更を行うことが容易になります。例えばわずかな一度の変更で、上記のNQにあるageフィールドを、_ageフィールドに変更することができます。同じ事をJDOQLやその他の文字列ベースのAPIで行うと、クエリ内は変更されないの、アプリケーションは正常に動かなくなるでしょう。

その他のAPIとUIブラウザ

Query By ExampleとS.O.D.A.という2つのAPIがdb4oには用意されていて、特定のケースや後方互換性のために利用することができます。

Query by Example(QBE)を利用すると、既存のセッターメソッドを利用してクエリテンプレートを作成し、非常に簡単にクエリを実行することができます。

```
Car car = new Car();
car.setName("Ferrari");
List cars = db.get(car);
```

S.O.D.A.は、ランタイム時に動的にクエリを生成することができる、ノードベースのクエリAPIです。サーバー上でカスタムコードを実行させることが可能なので、ネットワーク上を移動するオブジェクトを少なくし、クエリの実行速度をスピードアップさせることができます。

Xstreamのようなライブラリを利用すれば、簡単にJavaオブジェクトをXMLに書き出すことが可能です。このようなライブラリを使用すれば、db4oをメッセージ指向のアーキテクチャで利用することができます。

需要に応じて、データベース内のデータを出力し、既存のレポートングツールで利用するために、SQL-92インターフェースを追加する予定です。

ObjectManager は、db4oデータベースファイルにアクセスし、閲覧、検索、そして編集ができるビジュアルツールです。たとえ必要なクラスが無くても可能です。本来デバッグ用に開発されましたが、オブジェクトデータベース初心者の方々が、リレーショナルデータベースからオブジェクトデータベースへ考え方をシフトさせる際の学習ルーツとしても使われています。

データシート | db4o バージョン 5.0

プラットフォーム	
Java	<ul style="list-style-type: none"> • J2EE • J2SE • J2ME(リフレクション) : CDC、PersonalProfile、Symbian、SavajeとZaurus 準備中: J2ME(書き込み専用リフレクション) : CLDC/MIDP (RIM/Blackberry とPalm OSを含む) • Servlet/JSP • Java Web Start
.NET	<ul style="list-style-type: none"> • .NET 1.x • .NET 2.0 • コンパクトフレームワーク • Windows Mobile 5.0 • Mono
プログラム言語	
Java	<ul style="list-style-type: none"> • JDK 1.1.x - JDK 5.0
.NET	<ul style="list-style-type: none"> • 全ての.NETマネージドコード (C#, VB.NET, ASP.NET, Boo, Managed C++ 等)
コマンド	
セッション	<ul style="list-style-type: none"> • 開始、終了
データベースファイル	<ul style="list-style-type: none"> • 新規作成、オープン、クローズ、削除
トランザクション	<ul style="list-style-type: none"> • コミット、ロールバック
オブジェクト	<ul style="list-style-type: none"> • 保存、取得、更新 (カスケード含む)、レプリケーション、削除 (カスケード含む)
メッセージング	<ul style="list-style-type: none"> • TCP/IP
透過性	
言語構成	<ul style="list-style-type: none"> • プリミティブ • 文字列 • 1次元配列 • 多次元配列 • インナークラス • Java/C# コレクション • Publicコンストラクタを持たないクラス • .NET 構造体 • Blobs (データベースファイルとは別に保存)

手間要らず	<ul style="list-style-type: none"> 特定のクラスから派生不要 特定のインターフェース実装不要 コードの変更不要 Serializableインターフェース不要
Privateフィールド	<ul style="list-style-type: none"> 保存可
ファイルI/O	<ul style="list-style-type: none"> 独自クラス化
Reflector	<ul style="list-style-type: none"> 独自クラス化 Generic
クエリ / APIs	
オブジェクト指向	<ul style="list-style-type: none"> ネイティブクエリ(NQ) Query By Example S.O.D.A.
SQL	<ul style="list-style-type: none"> SQL-92 読み取り専用(準備中)
XML	<ul style="list-style-type: none"> サードパーティ製品利用(例 Xstream)
モード / Concurrency	
動作モード	<ul style="list-style-type: none"> ローカル クライアントサーバー
スレッド	<ul style="list-style-type: none"> 複数
トランザクション	<ul style="list-style-type: none"> 複数、並列
Semaphores	<ul style="list-style-type: none"> 利用可
Read-Only Mode	<ul style="list-style-type: none"> 利用可
拡張性とパフォーマンス	
ベンチマーク	<ul style="list-style-type: none"> Hibernate/MySQLの44倍以上速い
拡張性の例	<ul style="list-style-type: none"> 毎秒200,000オブジェクト保存 20,000クラス保存 PDAに300,000オブジェクト保存
インメモリモード	<ul style="list-style-type: none"> 利用可
クライアント	<ul style="list-style-type: none"> 単一プロセス実行可
サーバー	<ul style="list-style-type: none"> サーバー上でのクエリ実行可
コレクション	<ul style="list-style-type: none"> 利用可
キャッシュ	<ul style="list-style-type: none"> 利用可
インデックス	<ul style="list-style-type: none"> 利用可

レプリケーション	
UUID	<ul style="list-style-type: none"> 全ての格納済みオブジェクトに割り当て
オブジェクトの移動	<ul style="list-style-type: none"> データベース間で容易な移動またはコピー可能
同期化	<ul style="list-style-type: none"> クエリ 更新 競合処理
セキュリティと暗号化	
ファイル保護	<ul style="list-style-type: none"> パスワード
ファイル暗号化	<ul style="list-style-type: none"> 簡易暗号化 独自の暗号化によるI/O可
可用性、信頼性、ゼロメンテナンス	
トランザクション	<ul style="list-style-type: none"> ACID システムエラー時のコミット復旧
スレッドセーフティ	<ul style="list-style-type: none"> 利用可
自動復旧	<ul style="list-style-type: none"> 利用可
オンラインバックアップ	<ul style="list-style-type: none"> 利用可
空き容量制御	<ul style="list-style-type: none"> 利用可
国際化	
Unicode	<ul style="list-style-type: none"> 利用可
リファクタリング	
バージョンング	<ul style="list-style-type: none"> 自動認識とメンテナンス
スキーマの統合	<ul style="list-style-type: none"> 利用可
名称変更	<ul style="list-style-type: none"> クラス、フィールド APIから削除された値へのアクセス可能
メモリとファイルサイズ	
DB容量	<ul style="list-style-type: none"> ~400 KB
最小RAM容量	<ul style="list-style-type: none"> ~1 MB
最大DBファイル容量	<ul style="list-style-type: none"> 254 GB / 1データベースファイル
クラスサイズ	<ul style="list-style-type: none"> 保存時増大なし
インスタンスサイズ	<ul style="list-style-type: none"> 保存時増大なし