D-034

オブジェクトの永続化と RDB 設計からの解放

Object Persistence and Free From RDB Design

東海林　健志†　久住　貴史†　西　宏昭†　宮川　治‡　大山　実†

Takeshi Shoji† Takashi Kusumi† Hiroaki Nishi† Osamu Miyakawa‡ Minoru Ohyama†

1. はじめに

Web が配信サービスに使用されるようになると，サービスプロバイダは動的なコンテンツの作成が必要になってきた．初期のころは，Common Gateway Interface (CGI) が動的コンテンツを生成するための主なテクノロジーとして広く利用されたが，Platform への依存性や拡張性の欠如などのいくつかの欠点が存在した．

これらの限界を解決するために開発されたのが Java Servlet テクノロジー[1]である．Java Servlet テクノロジーが従来の CGI と異なるのは，リクエスト毎に Process が起動するのではなく，Thread が起動するという点である．この機構により Application はメモリ内での「データの共有」を可能にした．しかし，「データの共有」によるメリットを活かしきれていないのが現状である．

「データの共有」を最大限に活かした Application は効率よく軽量に動作するはずである．さらに，「データの共有」という機能を抽出することにより，開発を容易にし，開発時間や開発コストを削減することが可能となる．そこで，「データの共有」すなわちオブジェクトの永続化を行う Framework の開発を行った．また，近年 Web アプリケーションの信頼性及びパフォーマンスが注目され始め，Web アプリケーションのパフォーマンスは業務の生産性に直結し，Web アプリケーションの停止は業務の停止を意味しているため非常に重要視されている．そのため検証として，実際に作成した 1 つの Web アプリケーションを対象に，永続化処理にこの Framework の適用前，適用後のパフォーマンスの違いについて測定を行い，信頼性について考察した．

2. 永続化

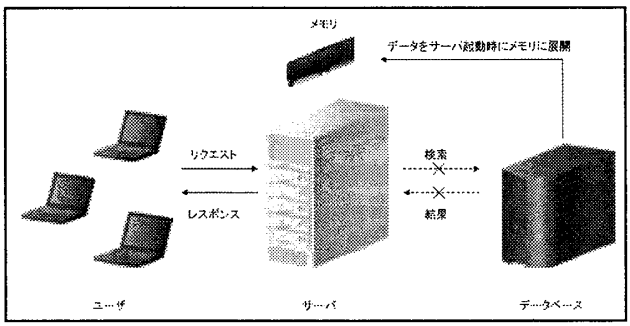

図 1　永続化模式図

永続化とは，メモリ上にあるインスタンスの状態をディスクなど(不揮発性記憶媒体)に保存することにより，プログラムが終了してもインスタンスの状態を保持し，再び必要なときインスタンスの状態を復元させることができるようにするための考え方である(図 1)．サーバ起動時にメモリ上に Database(DB) のデータを読み込むことにより，サーバは毎回 DB へのアクセスがなくなり，ユーザのリクエストに対しすぐにレスポンスを返すことが可能になる．

2.1 複雑性

一般的に永続化を行うためのデータの格納には Relational Database(RDB)を用いるが，この際オブジェクト指向と RDB のインピーダンスミスマッチが問題となる．また，DB を扱うには DB 設計の知識，SQL 文法の知識，SQL インジェクションなど固有のセキュリティの知識，パフォーマンスチューニングに関する知識，など必要とする知識が多く，複雑になりやすくフォールトが生まれやすい．

また DB へのアクセス速度は，メモリに比べ大幅に遅く，パフォーマンスのボトルネックとなる．これらはキャッシュを行うことで大幅な改善が望めるが，この場合マルチスレッド環境においてスレッドセーフを保つことは難しい．

2.2 複雑性の排除による性能及び信頼性の向上

この Framework は，この永続化処理をプログラミングレスにすることを目的としており，DB 固有のコーディングは必要なくなる．

これにより全体のコーディング量が減ることで，永続化処理のモジュールだけでなく，相対的に Application 全体でフォールトを内包する可能性が減り，信頼性の向上に繋がる．また，今までのデータのキャッシングなどパフォーマンス向上のための処理も，この Framework が自動で処理するため，パフォーマンスの高い Web アプリケーションを迅速に開発することが可能である．

2.3 O/R Mapping ツール

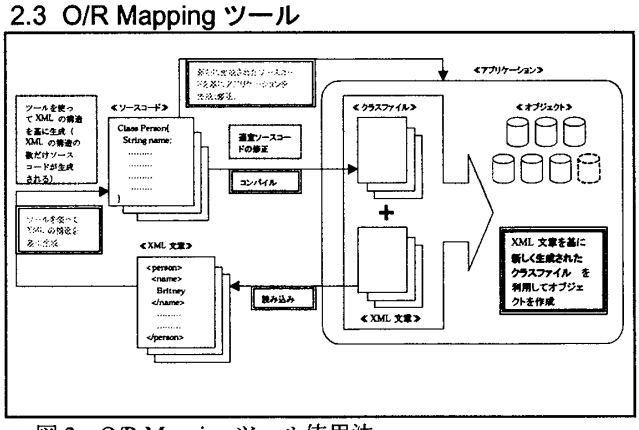

図 2　O/R Mapping ツール使用法

† 東京電機大学大学院　情報環境学研究科，Tokyo Denki University, Graduate School of Information Environment

‡ 東京電機大学　情報環境学部，Tokyo Denki University, The School of Information Environment

現存している O/R Mapping を行うための主なツールの使用方法の概要を以下に示す．(図２)

- 永続化するためのデータ構造の設計・作成
- リレーショナルデータベースの設計
- XML で Mapping コードを定義
- ツールを利用して Java のソースコードを自動生成
- 必要によっては生成されたソースコードに微調整 (主に SQL 関連)
- コンパイル
- Application が XML を読み込んで Instance を生成

これらの O/R Mapping ツールは有用かつ実用的であるがいくつか問題を含んでいる．一つは XML が Mapping コードを定義するために使用されるということである．多くの O/R Mapping ツールは自動的に SQL 文を発行する JDBC 層を隠蔽することができるが，それはオブジェクトーリレーショナル間におけるインピーダンスミスマッチの解決には至らない．Mapping コードを作成するにはオブジェクト指向設計技術だけではなく，RDB 設計の知識が必要とされ高度な能力が不可欠となる場合がある．そのため Application の開発において O/R Mapping の設計に割かれる時間は少なくない．もう一つは，ツールによって SQL 文を含んだ Java のソースコードを生成するため，膨大なソースコードの保守・管理が必要となる．また，パフォーマンスの向上を求める場合には修正のためのスキルが必要となる．

3. 解決方法

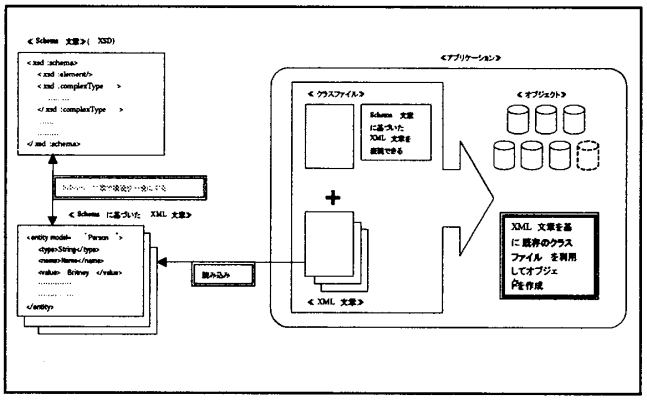

図 3　解決法

「データ構造」または「オブジェクトの状態」を表現するための XMLSchema を一意にし，インピーダンスミスマッチを解決する．その際の使用方法を以下に示す．(図３)

1. 永続化するためのデータ構造の設計
2. XML でデータ構造を定義する (直接 Java オブジェクトを生成することも可能)
3. Application が XML を読み込んで Instance を生成

この解決方法では O/R Mapping のために必要とされる Mapping コードは 1 行も必要ではない．それは XML に O/R Mapping のための Mapping コードを記述するのではなく，「データ構造」または「オブジェクトの状態」を記述することで可能となった．この Framework は O/R Mapping を行う前に O/X Mapping (Object/XML Mapping) を行い，その XML を RDB に保存することでインピーダンスミスマッチの問題を解決した．

さらには新たにソースコードを生成しないため，そのための保守・管理が不要である．さらに，Framework が自動的に永続化を行うためパフォーマンスを向上させるためのスキルも不要となる．

4. 設計

4.1 データ構造の定義

```
<xsd:complexType name="xentity">
  <xsd:sequence>
    <xsd:element name="variable" type="variable" minOccurs="1"
      maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="model" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="variable">
  <xsd:sequence>
    <xsd:element name="type" type="xsd:string"/>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="value" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="final" type="xsd:boolean" use="required"/>
</xsd:complexType>
```

図 4　XMLSchema (データ構造)

XMLSchema (図４) を一意に定義しても，データ構造は一意にならないことを以下に示す．

- variable 要素はいくつも存在してよい
- variable 要素は type, name, value 要素を一つずつ持つ
- variable 要素はプログラム中に変数の宣言を示すクラスのオブジェクトとなる
- type 要素はプログラム中に『変数の型』となる
- name 要素はプログラム中に『変数の名前』となる
- value 要素はプログラム中に『変数の値』となる

この設計により基本的な「データ構造」をこの XMLSchema から生成される XMLInstance を作成することで，Framework が自動的にオブジェクトに変換し，その「データ構造」を永続化することが可能となった．

4.2 オブジェクトの状態の定義

```
<xsd:complexType name="xentity">
  <xsd:sequence>
    <xsd:element name="variable" type="variable" minOccurs="1"
      maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="model" type="xsd:string"/>
  <xsd:attribute name="key" type="xsd:string"/>
  <xsd:attribute name="state" type="xsd:boolean"/>
</xsd:complexType>
<xsd:complexType name="variable">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="value" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

図 5　XMLSchema(オブジェクトの状態)

「オブジェクトの状態」の永続化を行う場合に必要な情報を表現するために XMLSchema を以下のように定義した．(図 5)

- model 属性はデータ構造への参照を示す
- key 属性はオブジェクトの識別を示す
- name 属性はデータ構造の変数を一意に特定する
- value 属性は特定されたデータの値

この設計によりオブジェクトを識別し，データ構造に適した情報を安全に永続化することが可能となった．また，この XMLInstance は基本的に，Framework が自動的に生成・

保存するため，ユーザはオブジェクトレベルで変更・削除を行うだけでそのオブジェクトの永続化が可能となる．

4.3 Object 構造

XML で「データ構造」と「オブジェクトの状態」を分割することにより，Application で使用される永続化に必要なクラス・インスタンスを表現することが可能となった．しかし，この XML をそのまま Java Object へと変換することは困難である．そのため Type Object パターンを Framework の設計に適用した．

Type Object パターン[2]はクラスとインスタンスを切り離し，クラスを表現するための型クラスと，インスタンスを表現するための型オブジェクトに分けられる．型クラスをインスタンス化したオブジェクトが本来のクラスとなり，型オブジェクトは一つの型クラスから生成されたインスタンスとなる．この Type Object パターンを適用すると，実行時にクラスを生成することが可能になり，設計時に予測不可能だったデータ構造を動的に追加することが可能となる．よって，この Framework に対する追加・変更なしに Application で無制限に作成される膨大なクラス郡を管理することが可能になった．

4.4 Dependency Injection

現在，Dependency Injection(DI)[3]と呼ばれるパターンが注目されている．この DI の考え方は上位レベルのモジュールが下位レベルのモジュールに依存せず，「具象」が「抽象」に依存することによりその再利用が可能になるというものである．現存している大部分の O/R Mapping ツールは DI パターンが適用され，Interface Injection，Setter Injection，Constructor Injection と呼ばれる形式[4]などで実装されている．

この Framework もまた DI パターンの原則に従って設計されている．しかし，現存する O/R Mapping ツールで使用されている実装形式とは異なる．それは前述した Type Object パターンを使用して実装しているという点である．現存している O/R Mapping ツールは主に上記の 3 つ形式で実装されているが，これらの実装は「データ構造」を定義するためのものではなく状態または関連を保持するものである．しかし，この Framework には Java プログラム上だけではなく XML の設計に関しても Type Object パターンを適用した．これは Java プログラム上における型の依存性をなくし，そのデータ構造・状態・依存関係を維持するための情報を XML に抽出することにより，Application が型に縛られずにデータを永続化することができる．この設計による最大の利点は動的なデータ構造の定義が可能になることである．よって，最コンパイル・再起動なしにデータの定義・永続化を行うことができる．

5. 動作

この Framework を動作させるには，ユーザは Application の名前，DB，DB の URL，DB の Username・Password を Configuration ファイルとして Framework に読み込ませる．次に，ユーザは Application で必要となるデータ構造を設計し，図 4 の XMLSchema に従い設計したデータ構造を XMLInstance で記述し Framework に読み込ませる．これで Framework がデータ構造・オブジェクトの状態の永続化に必要な情報はすべて揃う．そこで，Framework は受け取ったデータ構造の XMLInstance を Java Object へと変換する．ユーザはこのデータ構造を示す Java Object を利用することで，状態を示すインスタンスを生成することが可能となる．さらに，状態を示すインスタンスはすでに永続化されてメモリ上に存在し，オブジェクトレベルでの変更・削除を行うと，Observer パターン[5]の実装により自動的に Framework へ通知される．通知された Framework はその Java Object の状態の Memento[5]となる XMLInstance を自動生成し，不揮発性記憶媒体へと保存する仕組みである．

不揮発性記憶媒体への保存に関しては，変更されるオブジェクトを一度 Queue に溜め，瞬間的に同一オブジェクトへの変更が複数回認められた場合，最後に通知されたオブジェクトの状態を変更対象として保存することで DB への負荷を軽減し軽量化を図った．これは，これまでの不揮発性記憶媒体での一元化を行う形態から，永続化を行うことによるメモリ上での一元化を行うことにより可能となった方法である．また，Queue に溜まったオブジェクトは Worker Thread パターン[6]により遅延書き込み処理されるため，分散化を図りサーバへの負荷を軽減させている．

6. 測定環境

この Framework では中・小規模な Application を想定としているため以下のようなハードウェア構成も用いた．

6.1 ハードウェア構成

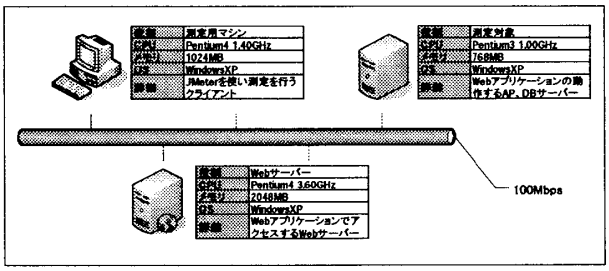

図 6 ハードウェア構成図

測定用マシン，測定対象，Web サーバの 3 つのコンピュータを使用した．Web アプリケーションを動作させるのに必要な，アプリケーションサーバ(Tomcat)，DB サーバ(MySQL)は共に測定対象のマシン上で動作させ，計測を行った．このため Web アプリケーションの DB へのアクセスに関しては，ネットワークのトラフィックが生じない．測定に用いたデータは，正確な負荷状況を再現するため，別の Web サーバ上に用意した．

6.2 ソフトウェア構成

Web アプリケーションの開発，動作，計測に使用したソフトウェアを表 1 に示す．

表 1. ソフトウェア構成

Java 仮想マシン	J2SE build 1.5.0-b64
パフォーマンス測定	Apache JMeter 2.0.2
アプリケーションサーバ	Apache Tomcat 5.5.4
Web サーバ	Apache 2.0.52-3
DB サーバ	MySQL 5.0.2-alpha

6.3 適用した Web アプリケーション

この Web アプリケーションは，Web ページのアーカイブを行うアプリケーションである．これは日々更新される Web ページについて，指定した時点でのスナップショットを保存し，その時点での Web ページへ永続的にアクセスを可能にするものである．単純な Web ブラウザでの保存とは違い，アーカイブを行ったページを複数人で簡単に共有でき，Web ページのヴァージョン管理にも応用が可能である．

この Web アプリケーションの永続化を適用したものと，一般的な JDBC を使い永続化処理を実装したものの，2 つを作成し計測を行った．

6.4 測定方法

Web アプリケーションの書き込み及び読み込みの性能の変化と特性を測定した．書き込みでは 50KB のデータを複数回書き込ませ，書き込み時の負荷を計測した．読み込みでは書き込まれた 50 KB のデータを複数回読み込ませ，読み込み時の負荷を計測した．測定時の条件は下記の表を参照．

- 同時に読み込ませる数または書き込ませる数(スレッド数) を変化させ，計測を行った
- 5 秒の間隔で 10 回繰り返した平均を計測した
- 上記の条件で計測を 3 回行い，平均値を記録した

7. 測定結果

図 7 に書き込み時，図 8 に読み込み時のパフォーマンス測定の結果を示す．図の Before は一般的な JDBC を使い永続化処理を実装したもので，After は永続化フレームワーク適用後のものである．

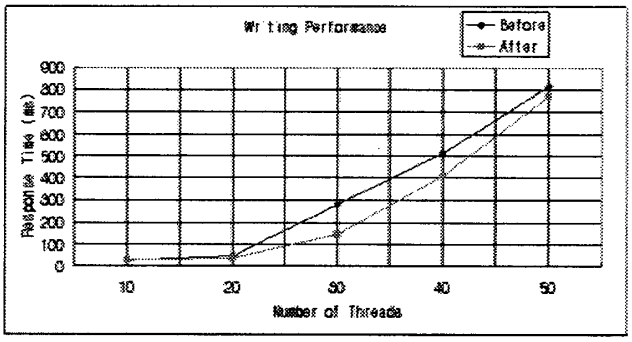

図 7　書き込み時のパフォーマンス

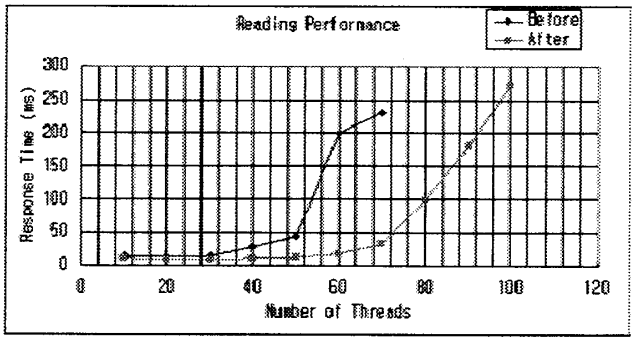

図 8　読み込み時のパフォーマンス

書き込みではそれほどパフォーマンスの向上は見られなかった．しかし，読み込みでは永続化フレームワーク適用前のものが，スレッド数が 80 の時点で，コネクションのエラーが発生し測定が不可能になったのに対し，適用後は 100 スレッドの負荷にまで耐えられている．またスレッド数 70 の時点では，適用前(232ms)に比べ，適用後(33ms)では約 700%のパフォーマンスの向上が確認できた．

8. 評価

この Framework を使用した際の利点を以下に示す．

- 複雑かつ煩雑な Mapping コードが不要
- 動的なデータ構造の追加が可能
- SQL レスで Application 開発が可能
- DB アクセスの軽減
- CPU・ネットワーク負荷の軽減によりコスト削減

ユーザはこれまでのデータの永続化における複雑で煩雑な作業から開放され，これまでに必要とされた高度なスキルなしでパフォーマンスの向上も得ることができる．したがって，Application の最大の焦点であるユーザインタフェースやビジネスロジックに専念することができ，より Agile な開発[3]が可能になると考える．

9. まとめ

XML に「データ構造」と「オブジェクトの状態」を記述することにより，従来の O/R Mapping ツールにおける問題を解決し，基本的なデータ構造のオブジェクトを永続化することが可能になった．また，オブジェクトの永続化による利点を最大限に活かすことにより，非常に軽量かつ高速に動作することが可能になった．また，マルチスレッドの遅延書き込みを利用することにより，瞬間的な DB への負荷を分散し安全性を高めることも可能となった．

また，測定においてはこの永続化 Framework が，実際に読み込みにおいて大幅なパフォーマンスの向上があることを実証できた．全体のコーディング量はフレームワーク適用前が 630 行であるのに対し，適用後は 410 行と約 65%の行数に削減できたことも評価できる．

今後の展望としては，より複雑なオブジェクト構造 である Composite パターン[5] の定義や汎用的なコレクションである Map や List などの実装を行う．また，現在のシリアライズ機構では復元困難な「オブジェクトの参照」を復元可能とする仕様について考えていきたい．

参考文献

1. http://java.sun.com/products/servlet/index.jsp
2. PLoPD Editors : Pattern Languages of Program Design Addison Wesley Longman, Inc. 2000
3. Robert C. Martin : Agile Software Development Pearson Education, Inc. 2003
4. http://martinfowler.com/articles/injection.html
5. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides : Design Patterns Addison-Wesley Publishing Company 1995
6. 結城　浩 : Java 言語で学ぶデザインパターン入門 マルチスレッド編 ソフトバンクパブリッシング 2002