XMLデータベースを活用するアプリケーションのデザイン・パターン

植田 賢　山川 多美　山根 英彦

The Application Design Patterns for Utilizing XMLDB

Masaru Ueda, Tami Yamakawa, and Hidehiko Yamane

XMLデータの格納／再利用などを可能とするXMLデータベースが広がり始め，XQueryやDB設計，プログラミングなどの側面からの情報は増えつつあるが，アプリケーションのデザインについての情報は少なく，さまざまなプログラミング言語やライブラリーによる実装が考えられる一方で，アプリケーション・デザインは確立されていない．本論文では，XMLを扱う固有の処理についての整理，アプリケーション・デザイン上の選択観点とデザイン・パターンの提案，特許情報検索を想定したアプリケーションを用いたデザイン・パターンの性能検証や比較考察を通じて，XMLDBアプリケーション開発に対する提言を行う．

XML database, which enables to store and reuse XML data, is spreading gradually. The information about XQuery, database design and programming is also increasing. However, the information about the application design is very few while users can use various programming languages and libraries. In this paper, after listing XML unique processing and executing the performance test using patent inquiry applications, some application design patterns and selecting points which utilize XMLDB characteristics are proposed.

Words & Phrases：XMLデータベース，アプリケーション・デザイン，アプリケーション開発，DB2，XML
XML database, application design, application development, DB2, XML

1．はじめに

XMLデータを格納するデータベース（以下，DB）の需要の高まりとともに，リレーショナル・データベース（以下，RDB）・ベンダーを含む多くのベンダーからXMLデータベース（以下，XMLDB）がリリースされ，現在まで多数の製品が登場した［1］．近年では実際にXMLDBを利用したシステム構築事例も増加しており［2］，データ構造が柔軟なXMLデータの利用増大とともに，XMLDBの利用機会もさらに増えると考えられる．

XMLDB事例の増加とともに，XMLDB利用のためのDB設計，プログラミングなどの側面からの解説やガイドも増えつつあり［3］［4］［5］［6］，RDBを使い慣れたDB管理者や開発者がXMLDBを利用する際の障壁は低くなりつつある．

しかしながら，XMLDB利用時の最適なアプリケーション・デザインの選択，すなわち各アプリケーション・レイヤーでXMLデータをどのように処理し，データ構造の変化に伴うアプリケーション改修時にも変更の局所化を図るなどの観点から，開発言語やデザイン・パターンの適切な選択について論じた資料はまだ少ない．

提出日：2008年5月12日　再提出日：2008年9月25日

そこで本論文ではXMLDBアプリケーション開発を容易にするためのデザイン・パターンを提案する．

以下，2章でXMLDBの種類とソリューションの現状を整理し，3章でXMLDBアプリケーションのデザインを整理した上で，4章で4種類のデザイン・パターンを提案して，5章で実際に性能検証を行った結果をもとにパターンの比較・評価を述べる．

2．XMLDBの概要

2.1 XMLDBの種類と特徴

現在，XMLデータを格納できるDBは，主に以下の三つの種類に大別することができる［4］．

①RDB
②ネイティブXMLDB
③ハイブリッドXMLDB

①は，旧来から存在するRDBにXMLデータを格納する方式であり，XMLデータをテキストのデータとしてラージ・オブジェクト（LOB）方式で格納する方法と，リレーショナル表にマッピングして格納する方法がとられている．これに対して，より効率のよい専用のDBが必要と

なり，②が登場した．②では，XML データのパース（構文解析）を行い，XML の特徴であるツリー構造を保持したまま格納する．この方式は，XML データの要素や属性に対して直接索引を付与することが可能で，高速な検索を実現できる．また，XML データの構造が変化しても，DB 管理者作業がほとんど不要なので柔軟性が維持できる．

③は，①と②を組み合わせたもので，リレーショナル列に加えて XML データ・タイプを持つ XML 列を定義可能とすることで実現されている．この方式では，RDB のための SQL と，XML のための XQuery や SQL/XML の両方の照会言語が，XML データとリレーショナル・データのどちらに対しても使用可能である．

XQuery は W3C により標準化された XML データ問い合わせ言語であり，XPath 表現式と FLWOR 表現式を使用してデータを検索する．SQL/XML は，SQL に対して XML データ問い合わせのための拡張が行われたもので，ISO により標準化されている．XQuery や SQL/XML の文法などはインターネット上の各種情報や参考文献［3］［4］［5］を参照されたい．

2.2 IBM DB2® pureXML™ の特徴と位置付け

DB2 では，2006 年のバージョン 9 以降にハイブリッド XMLDB 方式を実現する pureXML 機能が実装された．RDB とネイティブ XMLDB の両方の利点を享受しつつ，RDB で培ったバックアップ／リカバリーや高可用性対応などの DB 基本技術を利用することができる．

本論文では，DB2 に限らず，他社製の XMLDB にも応用できる XMLDB アプリケーション・デザインについて述べる．

2.3 XMLDB ソリューションの現状

XMLDB が必要となる場面として，主に以下が挙げられる．

- 既存の XML データを効率よく格納／取り出したい（例：新聞業界の NewsML など）
- 文書をメタデータとともに DB に格納して検索／再利用などをしたい（例：特許情報や財務情報など）
- スキーマー変更の多いデータを XML データとして柔軟に格納したい（例：商品ごとに仕様が異なり，頻繁に商品が追加される商品カタログやパーツ・カタログなど）

上二つはデータ交換で用いられる XML データやドキュメントとしての XML データなど，既存の XML データをそのまま格納し，索引や XQuery により効率よく取り出すことを目的としている．三つめは，業務ニーズによりスキーマー変更が頻発するような場合において，DB 保守のコストを削減することを目的としている．

また，情報の特性として XML に適しているという要件からだけでなく，適応的な開発を行うアジャイル開発の際に，スキーマーの柔軟性を生かして XMLDB が使用されることも多くなってきている．必然的に，開発言語については，Java™ に加えて PHP や Ruby などのスクリプト言語が使用されることも多くなってきている．

このような状況の中で，XMLDB を使用したアプリケーション・デザインに関しては，歴史の長い RDB アプリケーションに比べて，確立しているとはいいがたい状況である．

3. XMLDB アプリケーションのデザイン

3.1 XMLDB アプリケーション特有の処理

XMLDB アプリケーションではデータを XML 形式で格納するため，XML データの生成，パース，変換処理などをアプリケーション内で行う必要がある．さらに，XMLDB アプリケーションが DB から取得した XML データをどのような形式でクライアントとやりとりするかによって XML データの扱い方を考える必要がある．

例えば，Web サービスのように XML 形式でデータを出力する場合には，XML データから別の構造の XML データへの変換処理が必要になる．また通常の Web アプリケーションの場合には，XML データをパースして非 XML データとして出力するか，XML データを XHTML 形式に変換するなどの処理が必要になる．いずれの場合もアプリケーションへの入力を元にデータを更新／作成する際にはアプリケーション内で XML データを生成し，DB に格納する際の妥当性検証も行う．

本論文では，XMLDB 活用の主流である Web アプリケーションとしての XMLDB アプリケーション開発について考察を進める．

まず，アプリケーションを照会と入力・更新に分けて，それぞれに必要な処理について述べる．

（1）XML データの照会

アプリケーションで XML データを照会する際には以下の処理が必要である．

①検索条件の受け取り

②クエリーの生成

　XQueryもしくはSQL/XMLを生成

③クエリーの実行

④照会結果の取り出し
XMLもしくは文字列としての取り出し
⑤照会結果の出力
照会結果を加工せずに出力する場合や，照会結果をパースしてDOMやJavaオブジェクトなどに格納してから非XMLへ変換する場合，JSP™などを利用する場合，照会結果をXSLTによりXML変換して出力する場合など，多数の実装が考えられる．
以上の中でXML固有の処理は②④⑤である．

(2) XMLデータの入力・更新

アプリケーションでXMLデータの入力・変更を行う際には以下の処理が必要である．
①更新条件と更新内容の受け取り
更新後XMLデータをそのまま受け取る場合，ノード単位に更新前と更新後の数値を受け取る場合などが考えられる．XMLデータの部分更新を行う場合は，どのノード（要素や属性）をどう更新するかという情報を画面などから受け取る必要がある．
②更新後XMLの生成
aで文字列として受け取った更新内容に対して検証を行い，事前にJavaやDOMオブジェクトに格納しておいた更新対象XMLデータを組み合わせて，更新後XMLデータのオブジェクトを生成する．
③更新クエリーの生成
XMLデータの部分更新を行う場合は，どのノード（要素や属性）をどう更新するかという情報をもとに更新クエリーを生成する．
④更新クエリーの結果確認
以上の中でXML固有の処理は②③となる．①はXML固有の処理というよりも，②で選択した方式に応じて決まる処理である．

3.2 XMLDBアプリケーションの実装

主要なプログラミング言語では，**表1**のようにXML処理のためのライブラリーが用意されており，アプリケーションから利用することが可能となっている．

一つの言語においても，XPathおよびスキーマーへの対応など，どのようなXML処理を行うかにより複数の実現方法があり，ライブラリー選定の際に考慮が必要となる．このように多くの選択肢が存在することも，アプリケーション・デザインが確立されていない理由の一つであるといえる．

表1. プログラミング言語のXML対応一覧

言語	ライブラリー	機能・特長	適用個所
Java	JAXP	SAX/DOM XSLT XPath StAX（JavaSE 6で導入）	パース 生成 変換 検証
	JAXB	Java/XML変換 （XMLスキーマ利用）	パース 生成
PHP	SAX	プッシュ型	パース
	DOM	ツリー型	パース 生成 検証
	SimpleXML	利用しやすい （PHP5で導入）	パース 生成
Ruby	REXML	XPathサポート	パース 生成 検証

4. XMLDBアプリケーションのデザイン・パターン

本章では以下の三つの観点に着目してアプリケーションのデザイン・パターンの提案を行う．

- XMLデータの処理（変換・パース・生成・検証）を行うレイヤー
- XMLデータの処理パターン
- XMLデータの保持方法

4.1 XMLデータの処理を行うレイヤー

XMLデータの処理を行うレイヤーを，Webアプリケーションのデザイン・パターンとして代表的なMVC（Model-View-Controller）パターンにあてはめてみる[7]．MVCパターンではModelでDB入出力・データ処理・ビジネス・ロジック実行，Viewで表示・入出力，Controllerで ModelとViewの制御を行う．

XML構造の変化に対する柔軟性を考慮するとXMLデータに関わる処理の局所化が望ましいため，MVCの役割に沿って，XMLデータの処理を割り当てると以下のようになる．

XMLパース：Model
生成：Model
検証：Model
XML変換：Modelもしくは View
DBに対するクエリー生成・発行：Model

図1にXMLデータの処理を配置したMVCモデル基本形を示す．

4.2 XML データの処理パターン

XML データの処理パターンは，以下の 3 種類に分類できる.

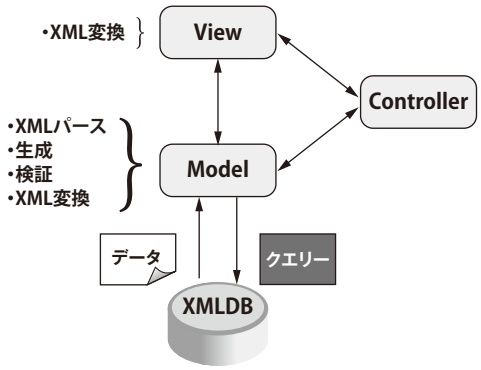

図 1. XMLDB アプリケーションの MVC モデル基本形

A1: XML 変換を伴うアプリケーション（End-to-End で XML を利用する）

DB から取得した XML データを別形式の XML データに変換して出力する. Web サービスのように入力が XML の場合には入力 XML を変換して DB に格納する.

A2: XML パースを行うアプリケーション

DB から取得した XML データをパースし，アプリケーションが出力するための非 XML データを取り出す. アプリケーションで非 XML データから XML データを生成して，DB に格納する.

A3: XML 処理を行わないアプリケーション

DB からデータを取得する際に RDB データ型として取り出す. DB に格納する際には，アプリケーションでは XML データを直接生成せずにクエリー操作で行う.

4.3 XML データの保持方法

XML データのアプリケーション内での保持方法は，以下の 3 種類に分類できる.

B1: DOM ツリーとして XML データを保持

DB より取得した XML データを DOM パーサーで処理し DOM ツリーをメモリー上に保持する. 要素・属性の値を要求された際には DOM API を用いて結果を返す.

B2: 文字列形式で XML データを保持

DB から取得した XML データを文字列としてメモリー上に保持する. 要素・属性の値を要求された際には DOM，SAX，StAX などの API を用いてパースを行い，結果を返す.

B3: プログラム言語のオブジェクト形式に変換して保持.

DB から取得した XML データをパースし，文書に対応したオブジェクトを生成してメモリー上に保持する.

B1 の場合は，XML インスタンスが必要とされる間はその DOM ツリーがメモリー上に存在するため，より多くのメモリー・リソースを使用する.

B2 の場合は，XML インスタンスの値が要求される都度パースを行う必要があるため，より多くの CPU リソースを必要とする可能性がある.

B3 の場合は，最初のオブジェクト生成で CPU リソースを使用するが，いったんオブジェクト形式に変換されるとその後の値の参照は通常のオブジェクト参照と同程度のリソース使用となる.

また，B1，B2 の場合にはデータを利用する側（View，Controller）において DB に格納される XML データの構造を意識する必要がある.

それに対し，B3 の場合には XML インスタンスに対応するオブジェクトを作成し，それを利用するため Model 開発者のみが XML データ構造を意識すればよい. ただし XML データ構造と 1:1 に対応するクラスを開発する必要があるが，Java で開発する場合には JAXB を利用して省力化を図ることもできる.

メモリー・CPU 使用量に関しては XML インスタンスのライフサイクルに依存し，ライフサイクルが長く，オブジェクトが頻繁に再利用される場合に違いが出てくる.

XML データのパースに関しては，SAX や DOM パーサーを用いて開発するのに比べ，Java では J2SE™ 5.0 から導入された XPath を利用することで開発の効率化を図ることもできる.

4.4 4 つのデザイン・パターンとその位置付け

ここまではデザイン・パターンを決める三つの観点ごとにバリエーションを考えてきたので，次にそれらを組み合わせた XMLDB アプリケーション全体としてのデザイン・パターンを提案する.

具体的には，図 1 の ‘MVC モデル基本形’ に ‘XML データの取り扱い’ のバリエーション A1，A2，A3 および XML データの ‘アプリケーション内での保持方法’ のバリエーション B1，B2，B3 を組み合わせて，どの組み合わせがデザイン・パターンとして提案できるかを考えてみる.

A1 の End-to-End で XML を利用する場合には，

XML - 非 XML 変換を省いて開発およびリソース負荷の軽減を図るという理由から，アプリケーション内データも XML を利用することが望ましい．よって A1 には B3 ではなく，B1 もしくは B2 を利用することで XSLT などによる XML 変換を可能とする．また，B1，B2 に関してはデザインよりも実装面の性格が強いため同列に扱う．

従って XMLDB アプリケーション・デザインとしては**表 2** の四つのパターンを提案する．

表 2. 組み合わせによるパターン

	B1	B2	B3
A1	I	I	
A2	II	II	III
A3			IV

四つのパターンを**図 1** の MVC モデル基本形にあてはめると**図 2** のようになる．

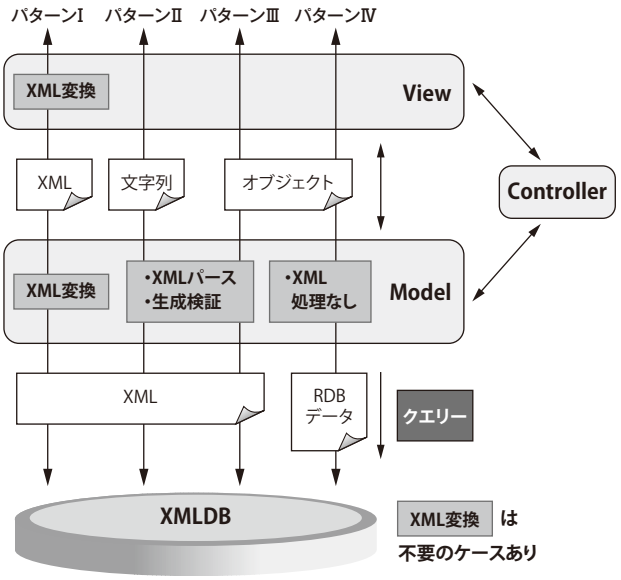

図 2. XMLDB アプリケーションのデザイン・パターン

パターンIの拡張として，DB2 の場合には XML 変換に用いる XSLT 処理をアプリケーション内ではなく，DB の提供する XSLT 関数を利用するパターンも考えられる．

ここでは，XMLDB アプリケーションの拡張性や適用の容易さなどの観点から，前項で提案した四つのパターンの位置付けを述べる．

パターンIは，アプリケーションの入出力が XML であるという点から，XML データを異なる XML 構造で再利用する場合，他システムと XML データを用いて連携する場合に推奨できる．Web サービスとの親和性も高い．

パターンIIは，プログラム内部で扱うデータがリレーショナルから XML 形式に変わった以外は従来の Web アプリケーション・デザインと同様であり，既存の Web アプリケーションに対してデータソースとして XMLDB を追加する場合や，Web アプリケーション開発スタイルがすでに確立されている環境においても適用しやすい

パターンIIIは，DB 入出力以外は従来の RDB アプリケーションと同じであり，上のパターンに加え，それほど XML の習熟度が高くない開発者を抱える開発プロジェクトにおいても適用できる．

パターンIVは，データが XML であることを意識するのはクエリーだけであり，それ以外は RDB アプリケーションと同じである．よって，RDB 対応のパッケージ製品や既存アプリケーションを XML データに対して稼働させる場合，リレーショナル・データ対応の開発フレームワークを流用したい場合，などに推奨できる．

実際のプロジェクトに適用する場合には，メンバーの保有スキルやプロジェクト方針などから言語を選定し，前述のデザイン・パターンを選択することで，自然に XML 処理に利用するライブラリー（**表 1** 参照）と配置が決まり，アプリケーション開発方針の決定が可能となる．

5. デザイン・パターンの検証

ここでは，提案する四つのデザイン・パターンが実際に XMLDB アプリケーションへ適用できること，処理性能やリソース負荷の確認を行うために，特許情報照会アプリケーションの開発および検証を行った結果と考察を述べる．

5.1 検証用 XMLDB アプリケーション

性能検証用にアプリケーションを作成した．検索対象として，特許庁により提供されている公開特許広報の XML 構造のデータを用いた［8］．まず，DB から先頭の 500 件を照会してリストを作成し，リスト中の任意のデータに対し ID を指定して詳細表示を行う．

パターンIでは XML から XHTML への XSL 変換をアプリケーション，DB のそれぞれで行った．パターンIIでは DOM，XPath で XML パースを行い，パターンIVでは DB2 の XMLTABLE 関数を利用して SQL データ型で値を取得した．

DB への同時接続数は 20 とし，Apache JMeter により詳細表示に対して同時 10 スレッドで 50ms のウェイトをかけて 120 秒間負荷を与えて測定した．DB のバッファー・プールは 500 件の XML データを十分格納でき

る 2GB とし，DB サーバーでのディスク I/O 処理がなくなるように調節した．こうすることで，純粋な XML 処理に関する CPU 使用率を得ることができる．

検証環境として，DB サーバーに P550Q の LPAR（AIX® 5.3ML06，1.5GHz*4，16GB） 上の DB2 V9.5，アプリケーション・サーバーとして BladeCenter® HS20（Windows® 2003 Server x64 SP2，2.66GHz*4，5GB）上の Tomcat 6.0 を使用した．

5.2 XMLDB アプリケーションの検証結果

検証結果を**表 3** に挙げる．**表 3** から分かるように，パターンⅡの DOM，XPath などによる XML パース処理はすでに知られている通り［9］，オーバーヘッドが大きいためアプリケーション・サーバーでの負荷が高くなる．このパターンを利用する場合には，アプリケーション･サーバーの CPU リソースを多く必要とすることを考慮に入れる必要がある．

表 3. 特許情報検索アプリケーションの検証結果

パターン	パターンの特徴	検索スループット /sec	アプリケーション・サーバー CPU 使用率	DB サーバー CPU 使用率
Ⅰ	XSL 変換	85.1	10%	7%
Ⅰ	DB で XSL 変換	89.0	5%	78%
Ⅱ	DOM	68.6	40%	6%
Ⅱ	XPath	64.6	60%	6%
Ⅲ	Java オブジェクト変換	66.0	50%	6%
Ⅳ	XMLTABLE 関数	144.7	1%	11%

パターンⅠの XSL 変換については，アプリケーション・サーバーで行うか，DB サーバーで行うかにより，負荷の発生個所が異なるが，パターンⅡよりもスループットが若干高くなる．ただし，DB サーバーで変換を行う際の CPU 利用率がほかよりもかなり高くなる．

アプリケーション・サーバーはスケールアウトで対応することが可能であるが，DB サーバーではスケールアウトできないためベンチマークを行って，CPU リソース見積もりを早い段階で行う必要がある．

特筆すべきはパターンⅣの DB2 の XMLTABLE 関数を用いた場合のパフォーマンスである．この場合はアプリケーション・サーバー，DB サーバーともに CPU 使用率が低く，かつスループットが高くなる．理由としては，DB2 の pureXML 実装では XML データの格納時にパースが行われるため，XMLTABLE 関数によるデータ取得ではほかのパターンのように，XML データのシリアライズ，アプリケーションでの再パースの必要がなく，効率的に処理が行われるためである．

さらに XML データ全体に対する必要なデータの割合が小さければ小さいほど，パフォーマンスが高くなると考えられる．しかし，アプリケーション内部でのデータは XML ではなく，結果として XML を活用しているとはいえない．

6．おわりに

本論文では，XMLDB アプリケーション固有の XML 処理とその実装方法を整理し，Web アプリケーションを想定した四つのデザイン・パターンを，その位置付け・使い分けとともに提案した．さらに，検証アプリケーションを利用した実機検証によって，パターンごとの性能やリソース負荷の特性を提示した．

デザイン・パターンを意識することで，XML 特有の処理の実装個所が明確になり，メンテナンス性の高い XMLDB アプリケーション開発につなげられる．

ただし，デザイン・パターンは必ずしも一つに固定する必要はなく，従来の RDB を利用した開発から，開発者の XML に対する習熟度が上がるにつれパターンⅣ→Ⅲ→Ⅱ→Ⅰへと変化させていき，サービス指向なアプリケーション開発へとステップアップを図ることができる．

また，さらに開発生産性を上げるために PHP や Ruby などのスクリプト言語に限らず,新しいライブラリー（JAXB など）や,今後普及する J2SE6 で導入される JDBC™ 4.0 の SQLXML 型（DB との XML 入出力の効率化が可能）などの活用を意識して言語を選択することも必要である．

本論文では，XMLDB の特長であるスキーマーの柔軟性を生かすために必要となる，スキーマー変更時のアプリケーション対応とスキーマー管理にまで言及することができなかった．今後，本論文で提案するデザイン・パターンに対して，スキーマーを DB で持つのか，アプリケーション内で持つのか，JAXB などのライブラリーへバインドするのか，などの点について考察を深めて，さらなる XMLDB アプリケーション開発における提案を行いたいと考えている．また既存のフレームワークとの連携についても考察を進めたい．

謝辞

本論文の作成にあたっては，社内タスク JTO2008 ‘XMLDB が提供できるもの’ でのディスカッションを参考にさせていただきましたので，タスク・メンバーの方に深謝いたします．

参考文献

［1］ wikipedia: XML database, http://en.wikipedia.org/wiki/XML_database（2008.4.15）.
［2］ XMLコンソーシアム: XMLDB事例紹介, http://www.xmlconsortium.org/seminar/06/070123/070123-prog.html（2008.3.31）.
［3］ 日本アイ･ビー･エム: プロジェクトの流れで理解するXMLDBデザイン徹底解説, 日経BP, ISBN978-4822262211（2008）.
［4］ 下佐粉 昭他: XML-DB開発 実技コース, 翔泳社, ISBN 978-4798116259（2008）.
［5］ 菅原香代子/米持幸寿: XQuery+XMLデータベース入門, 日経BP社, ISBN978-4822282929（2006）.
［6］ 山川 多美: “ハイブリッド･データベースにおけるXMLデータモデリング,” ProVISION, No. 52, pp. 86-91（2007）.
［7］ wikipedia : Model View Controller, http://ja.wikipedia.org/wiki/Model_View_Controller（2008.3.31）.
［8］ 特許庁: 公報仕様 特許, 実用新案 第3.1版,http://www.jpo.go.jp/（2007.6.30）.
［9］ 二上哲也: “XMLによるコンポーネント間通信,” ProVISION, No. 31, pp. 44-53（2001）.

日本アイ･ビー･エム株式会社、
ハイ･バリュー･ソリューション･センター
ベータプログラム推進
主任ITスペシャリスト

植田 賢 Masaru Ueda

［プロフィール］
過去数年にわたりサービス部門でオープン系のシステム開発プロジェクトにおいてアプリケーション設計，開発に従事 . 現在は IT スペシャリストとしてオープン・プラットフォームにおける Information Management 製品のベータ･プログラム推進および技術支援を行う .
uedam@jp.ibm.com

日本アイ･ビー･エム株式会社、
ハイ･バリュー･ソリューション･センター
ベータプログラム推進
ICP-コンサルティングITスペシャリスト

山川 多美 Tami Yamakawa

［プロフィール］
オープン・プラットフォームにおける Information Management 製品のベータ・プログラム推進を担当するチーム・リーダー . 製品品質向上／事例創出／スキル育成に力を注ぐと同時に，先進プロジェクトにおいて ,DB2 を中心とした技術支援活動を行っている .
tami@jp.ibm.com

日本アイ･ビー･エム株式会社
ハイ･バリュー･ソリューション･センター
ベータプログラム推進 ITスペシャリスト

山根 英彦 Hidehiko Yamane

［プロフィール］
2003 年ソフトウェア製品のベータ・プログラムを推進するエンジニアとして日本 IBM 入社 . 現在 IT スペシャリストとして Information Management 製品を担当し , 新機能を中心に検証 , 構築支援を実施 . 最近では DB2 9 の新機能 pureXML を用いた XMLDB の性能検証を行う .
yamane3@jp.ibm.com