データアクセス設計

WebSphere Application Server V4 EJB Solution Workshop

December 10-11, 2001

@ Makuhari Lecture Hall

Agenda

- **DB接続のパターンの比較**
  - Servlet直JDBC / EntityBean CMP / EntityBean BMP / SessionBean / AccessBean
    - 作る方法、管理容易性、生産性、パフォーマンス、参照系/更新系という比較
  - 実利用サンプル
    - WCS ：CMP & AccessBean
    - WCBE: + Session Bean
- **EJB-RDB Application Design Consideration**
  - CMP Entity Beanの粒度に対する考察
  - EJBのISOLATIONに関する考察
- **EJB Data Access将来動向**
  - EJB20のEntityBean機能強化
    - コンテナ管理リレーションシップ
    - EJBQL

目的
Web-RDBアプリケーション構築の上でEJBの適用の是非、
適用の際の種類選択時の考慮点を抽出し、デザイン時の参考とする

DB接続パターン

Agenda

- What's Web-RDB Application
- 接続パターン
  - JDBC直接
  - CMP Entity Bean
  - BMP Entity Bean
  - Session Bean
  - AccessBean
- 比較サマリー
- EJBアプリ実例
  - WCS　Entity Bean + Access Bean
  - WCBE　Session Bean

Web-RDB Applications

http req

Web Server

Web Application Server

?

DBMS

RDB

http res

Think Simple !

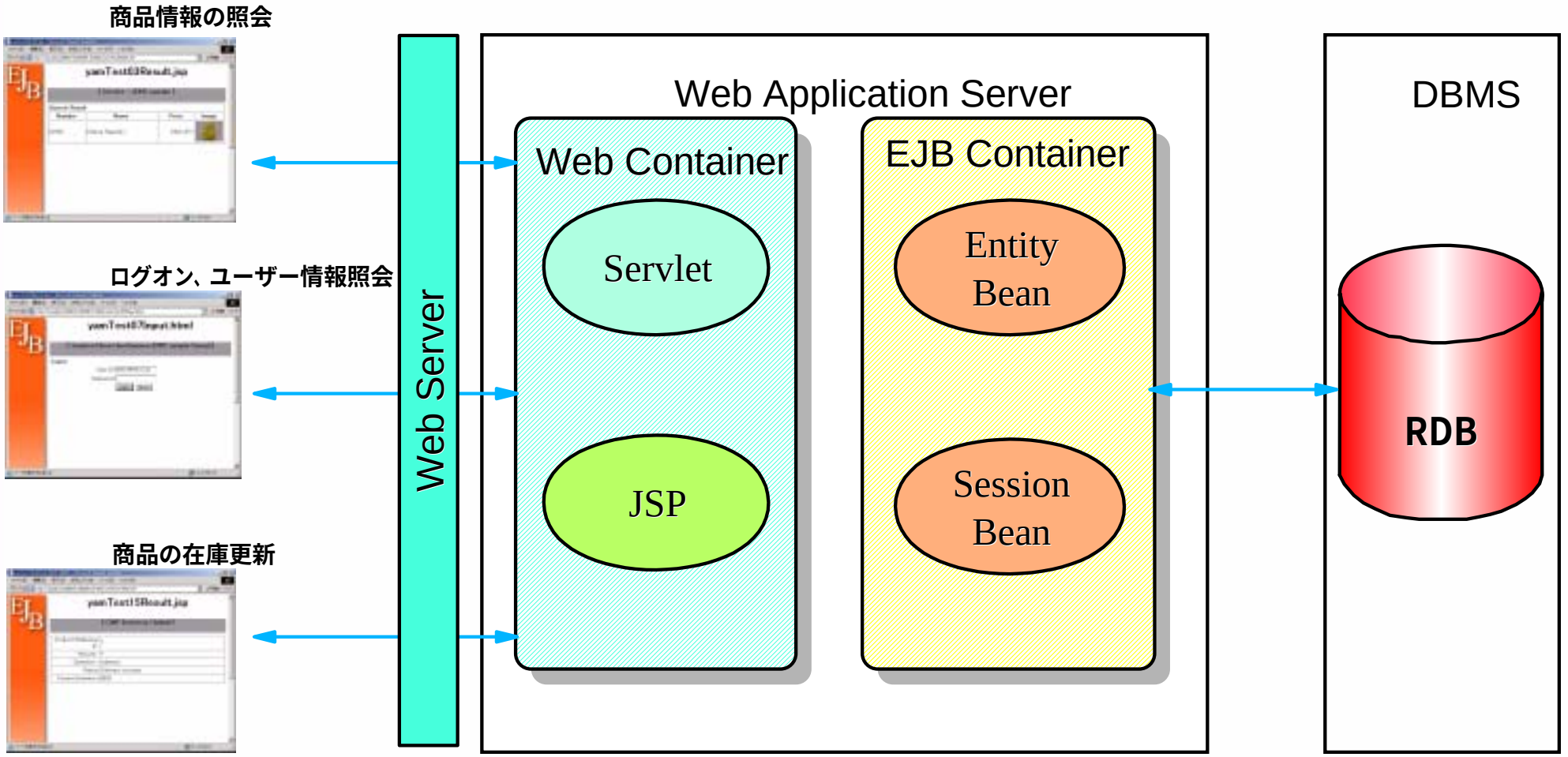
Web-RDB Applications

▶Web-DB連携アプリケーション

- 以下のコンポーネントより成り立つ

- このアプリケーション部分としてどんな選択肢が現実的に挙げられるか、それぞれの選択肢の利点、欠点、適用範囲は何かを実際のサンプルを元に以下の観点で比較検討
  - アプリケーションのタイプ (参照系か更新系か)
  - アプリケーション作成の生産性
  - 要求されるスキル
  - 管理負荷
  - アクセスするリソース
  - アプリの特性
  - パフォーマンス

接続パターンサンプル アプリケーションイメージ



- ECサイトを想定したアプリケーション

接続パターンサンプル 使用テーブル

▸PRODUCT(商品テーブル)

PRRFNBR	PRNUMBER	PRNAME	PRPRICE	PRVENT	PRPICT
1	00101	Kijitora Figure(S)	500	10	KijiFigureS.jpg
4	00201	2002 Kijitora Calendar(Desktop type)	800	10	KijiCalendarDesk.jpg

▸USERBASE (ユーザー基本テーブル)

USRFNBR	USLOGID	USLOGPW	USFLG1	USFLG2
10	wanco@wan.co.jp	wanco		
20	tora@kijitora.com	tora		

▸ADDRESS (ユーザー住所テーブル)

ADRFNBR	ADUSNBR	ADFLG	ADFNAME	ADLNAME	ADZIP	ADPREF	ADCITY	ADADDR1	ADADDR2	ADPHONE1	ADPHONE2
10	10	S	ワン吉	草原	261	千葉県	千葉市	美浜区中瀬1-1	MK-X15-C	043-123-4567	
11	10	O	ブウ	神野	380	長野県	長野市	南長野新田12-3	-	026-232-0000	

▸ORDER (オーダー表)

ORRFNBR	ORUSNBR	ORADNBR	ORTOTPRICE	ORTOTTAX	ORSTAT
10	10	10	2600	130	SBMT

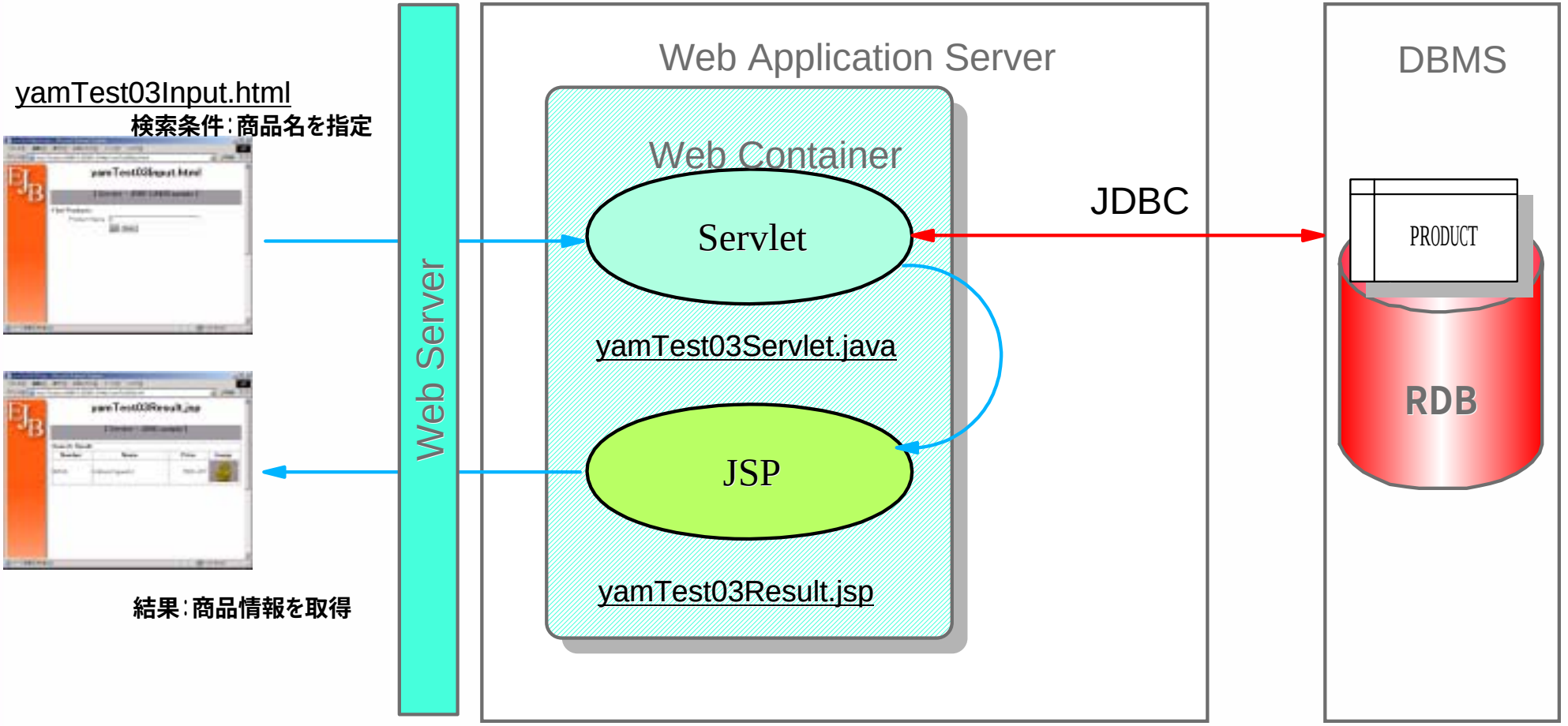
▸ORDERITEM (オーダー明細表)

ORITNBR	ORRFNBR	ORITPRNBR	PRITPRICE	ORITQUANT	ORITSUM
10	10	2	1000	1	1000
11	10	4	800	2	1600

接続パターン (1) 非EJB

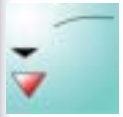
▶サンプル

- 商品情報の取得

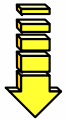


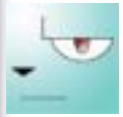
接続パターン (1) 非EJB

▶Servletから直接JDBCでデータベースにアクセス



- もっとも単純なDBアクセス方法であり、自由度が高い
  - Servletにデータアクセスを行うJDBCコード(SQL)を記載
  - 結果の取得･更新をServletプログラマーが制御
- 小規模開発ならば十分実用的な手法
- EJBと比較しJDBCの習得の方が容易で一般にスキルを求めることが可能

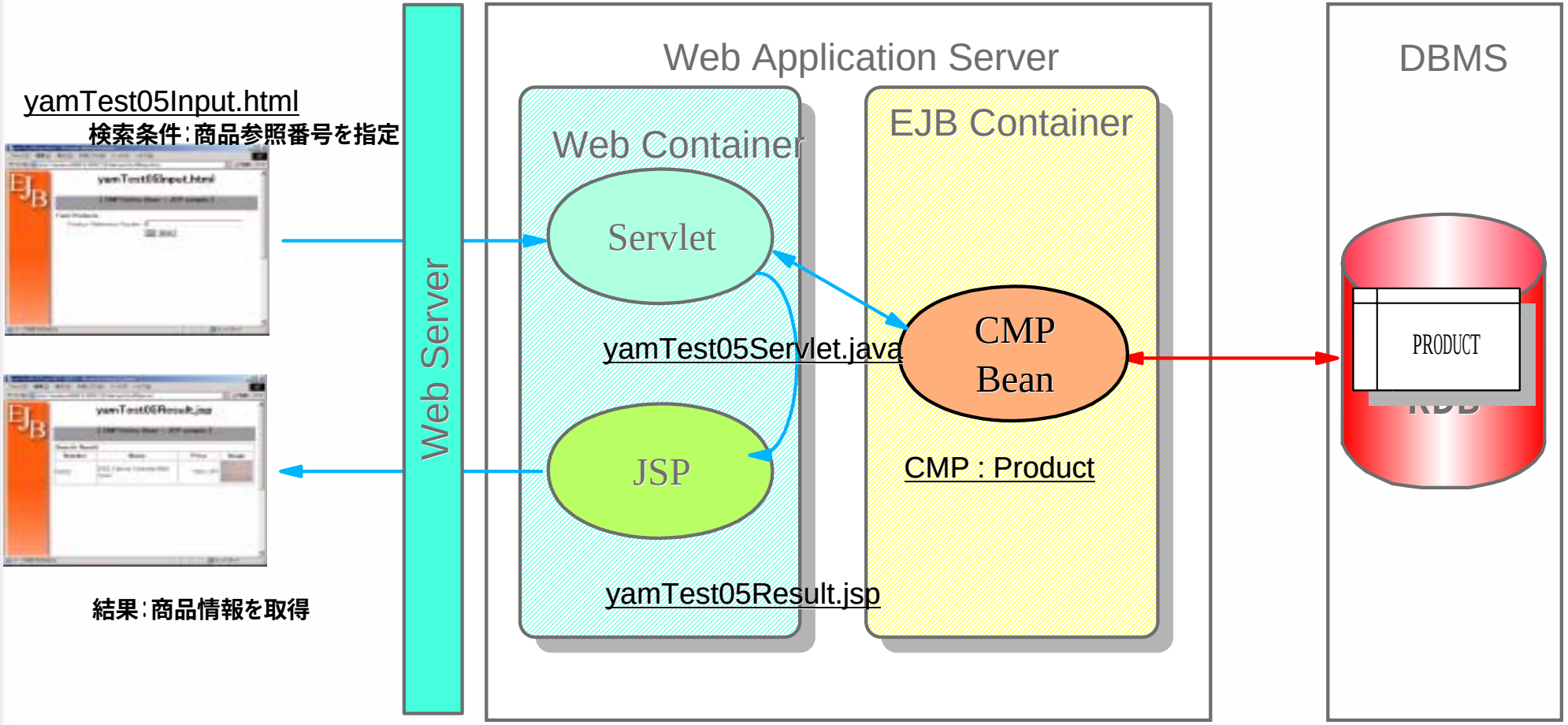




- データベースアクセスコードとアプリケーションが未分化
  - アプリケーションプログラマーはデータベーススキーマを知っておく必要がある
  - JDBCは汎用的なAPIだがその内部で記述されるSQLはDBMS依存の部分もあるため、ポータビリティが失われる
- オブジェクト指向デザイン、設計が難しい

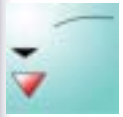
接続パターン (2) CMP Entity Bean

- サンプル
  - 同じく商品情報の取得

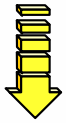


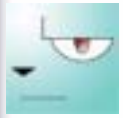
接続パターン (2) CMP Entity Bean

▶EJB DBアクセスの基本形態



- 再利用可能なビジネスオブジェクトを創出し、EnterpriseApplication間で共有することが目的
  - 前のSQLでの情報取得/更新部分をEJBのCMPを呼び出す形態で行うことでDBのテーブルをJavaのオブジェクトとして扱うことが可能となる
- データとロジック、SQLとプログラムの分離が実現されコードのポータビリティ、管理の自由度が向上
  - 大規模開発時などデータ管理側(Bean Provider)とビジネスロジック側に役割を分化できる



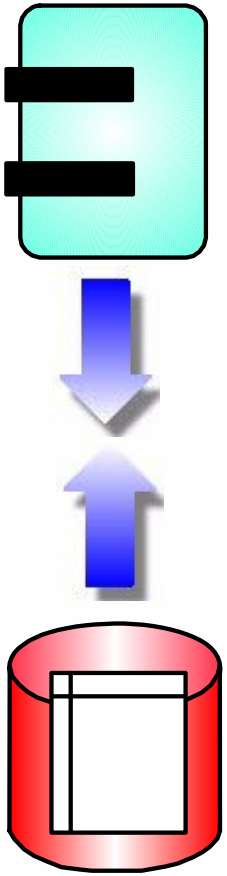


- Servlet直接に比較して作成コンポーネントが一つ増える
  - ただしCMP自身の作成はIDEなどの利用である程度自動化できる
- EJBをアクセスするServletなどのクライアントコードの開発者にEJBの知識が必要
- データベース直接操作ではなく間接的に操作することになるため、特有の考慮点がある
  - SQLを書かないで済む利点よりSQLを書けないことの制約のほうが多くなることも
  - ISOLATION LEVELへの理解が必要
- CMPがマップできるのは1表のみ
  - 複数表の関連を取り込むのが難しい→考慮点として後述
  - CMPフィールドがマップできるのは1カラムのみ
- データベースの1行が1インスタンスとなるため、大量検索に向かない

CMP作成の方法

- **TopDown**
  - EJBからテーブルスキーマを作成

- **Meet in the Middle**
  - 両者を作成しマッピング



- **Bottom Up**
  - テーブルからCMPを作成

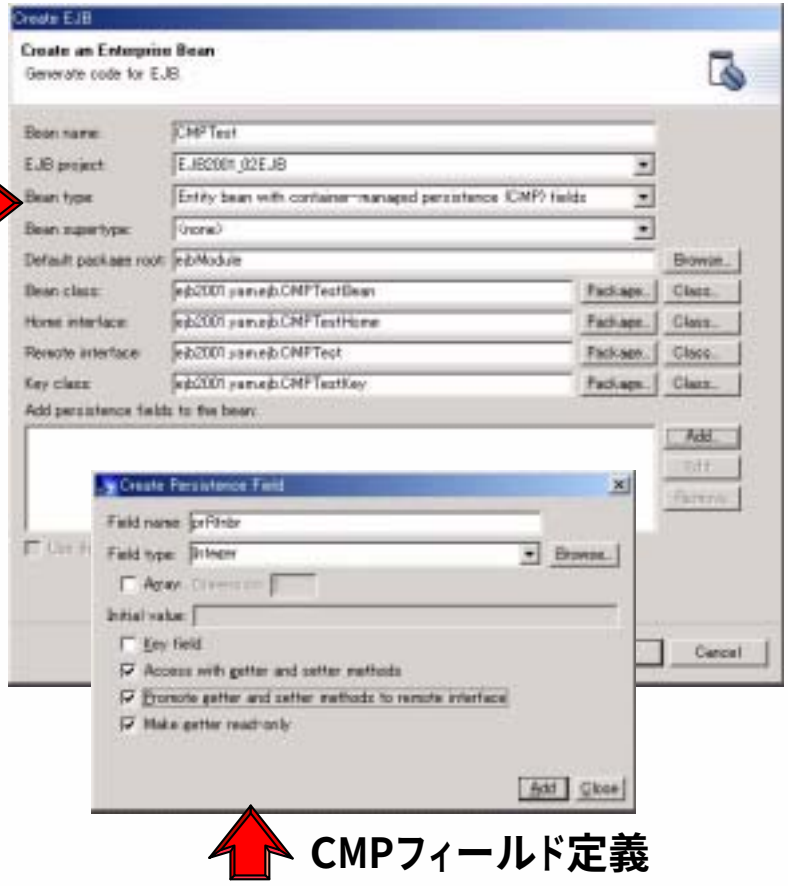
- 設計のアプローチ、コード修正量の軽重などから判断
- 妥当なのはMeet in the Middleか

CMP作成の方法 (Meet in the Middle)

▶WSADでの作成手順(1)

1. EJB WizardでCMPの作成
   - CMPフィールド＝テーブルのカラムを指定して作成
   - 以下のファイルがベースとして作成される
     - ❑Beanクラス　<EJBName>Bean.java
     - ❑Homeインターフェース　<EJBName>Home.java
     - ❑Remoteインターフェース <EJBName>.java
     - ❑PrimaryKeyクラス　<EJBName>Key.java
2. Beanの開発
   - ejbCreate()、ejbPostCreate()メソッドの引数変更
     - ❑テーブル定義のNOT NULLカラム分に合わせ、メソッドに引数を追加
     - ❑メソッド内でプロパティ値をセット
   - ビジネスメソッドの実装
     - ❑その他EntityBean上に実装するビジネスメソッドをコーディング
3. Home Interface定義
   - create()メソッド引数変更
     - ❑ejbCreate()/ejbPostCreate()にあわせて引数変更
   - Primary Key以外での検索のためのFinderを定義
     - ❑ExtensionEditorでFinder検索条件ロジックを追加
4. RemoteInterface定義
   - 必要なビジネスメソッドやAccessorをBeanからPromote
5. DDの編集
   - EJB Editor、Extension Editorなどで定義情報を指定

CMP



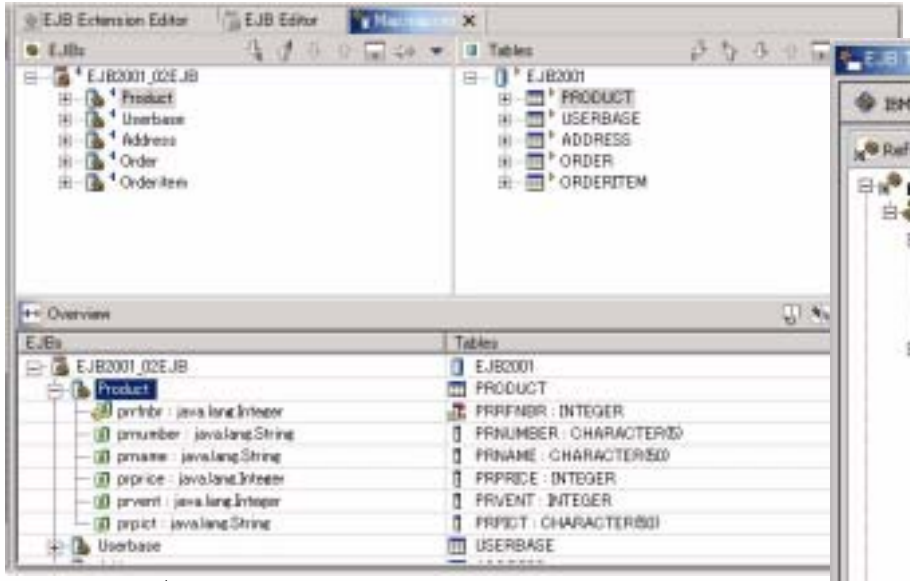
CMPフィールド定義

CMP作成の方法 (Meet in the Middle)

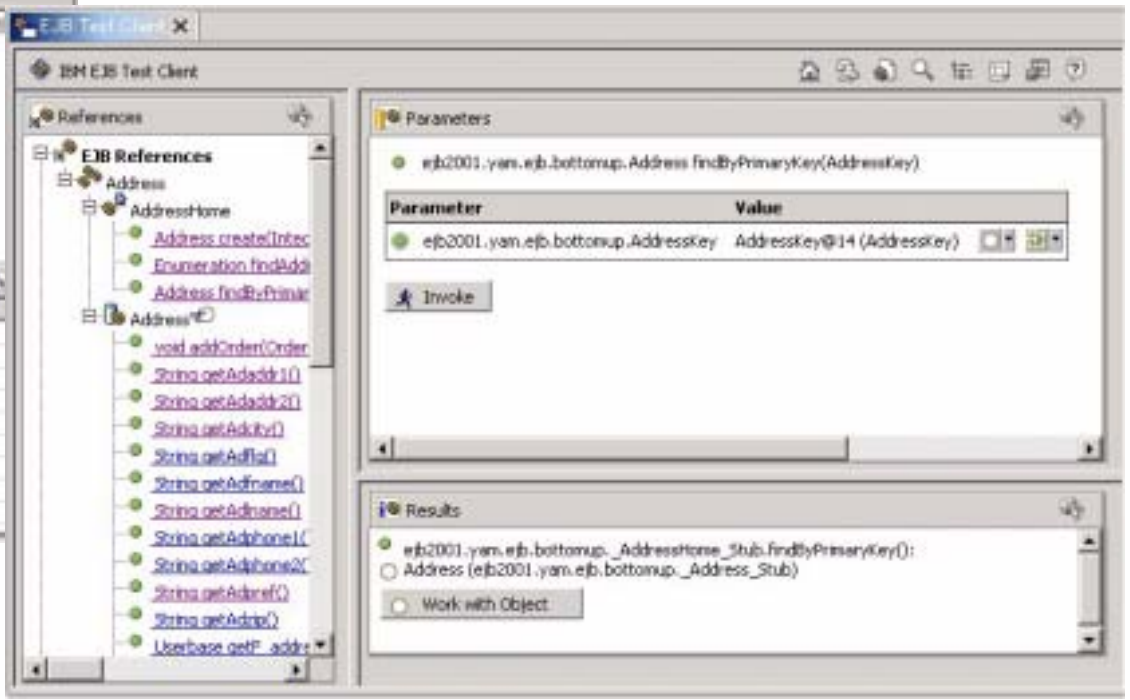
▶WSADでの作成手順(2)

6. テーブルへのマッピング
   - CMPとテーブル、CMPフィールドとカラムのマッピング
   - マッピングエディターが提供される
   - SQLデータタイプとCMPフィールドの列のタイプが互換である必要性
7. Deployed Codeの作成
8. テスト
   - テスト環境にてEJBテストクライアントが起動
   - メソッドコールが適切にできるかを確認。

EJB Test Client



マッピング
エディター



EntityBeanでのデータベースアクセス SQL VS EJBクライアント


```
select prname,prprice,prvent,prpict
from product
where prrfnbr = 3;
```

```
productHome = (ProductHome)
javax.rmi.PortableRemoteObject.narrow(ctx.lookup("ejb/Product",
ProductHome.class);
    ..................
Product product = productHome.findByPrimaryKey(new
ProductKey(prrfnbr));
    ..................
String row[] = new String[3];
row[0] = product.getPrName();
row[1] = String.valueOf(product.getPrPrice());
row[2] = String.valueOf(product.getPrVent());
row[3] = product.getPrPict();
```

EntityBeanでのデータベースアクセス SQL VS EJBクライアント


```
select prname,prprice,prvent from
product
where prrfnbr = 3
```

(1) PrimaryKeyでの検索

```
select prname,prprice,prvent from
product
where prname like '%tora%'
```

(2) PrimaryKey以外の値で検索

```
insert into product (prrfnbr,
prnumber,prname)
values (6, '00301' 'Kijitora CD-R')
```

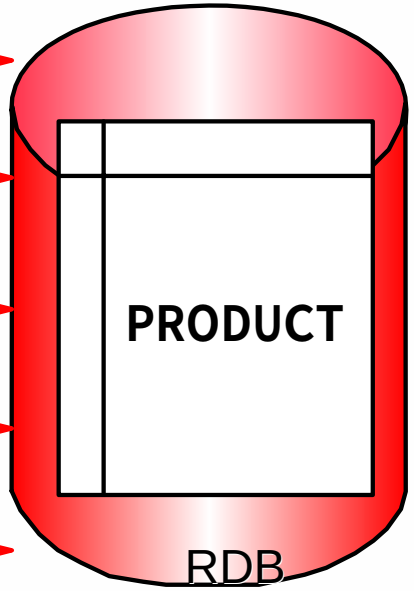
(3) 新規行の挿入

```
update product set prvent = 100 where prrfnbr =
5
```

(4) 行の更新

```
delete from product where prrfnbr =
4
```

(5) 行の削除



```
Product product =  productHome.findByPrimaryKey(prrfnbr);
String prname = product.getPrName("tora");  ....
```

(1) PrimaryKeyでの検索

```
Product product =  productHome.findByPrName(prrname);
Integer prprice = product.getPrPrice();   ...
```

(2) PrimaryKey以外の値で検索

```
Product product =  productHome.create(prrfnbr,
prnumber,prname);
```

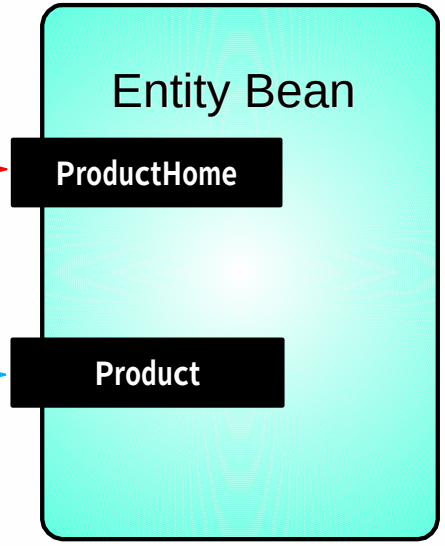
(3) 新規行の挿入

```
Product product =  productHome.findByPrimaryKey(prrfnbr);
product.setPrVent(prvent);
```

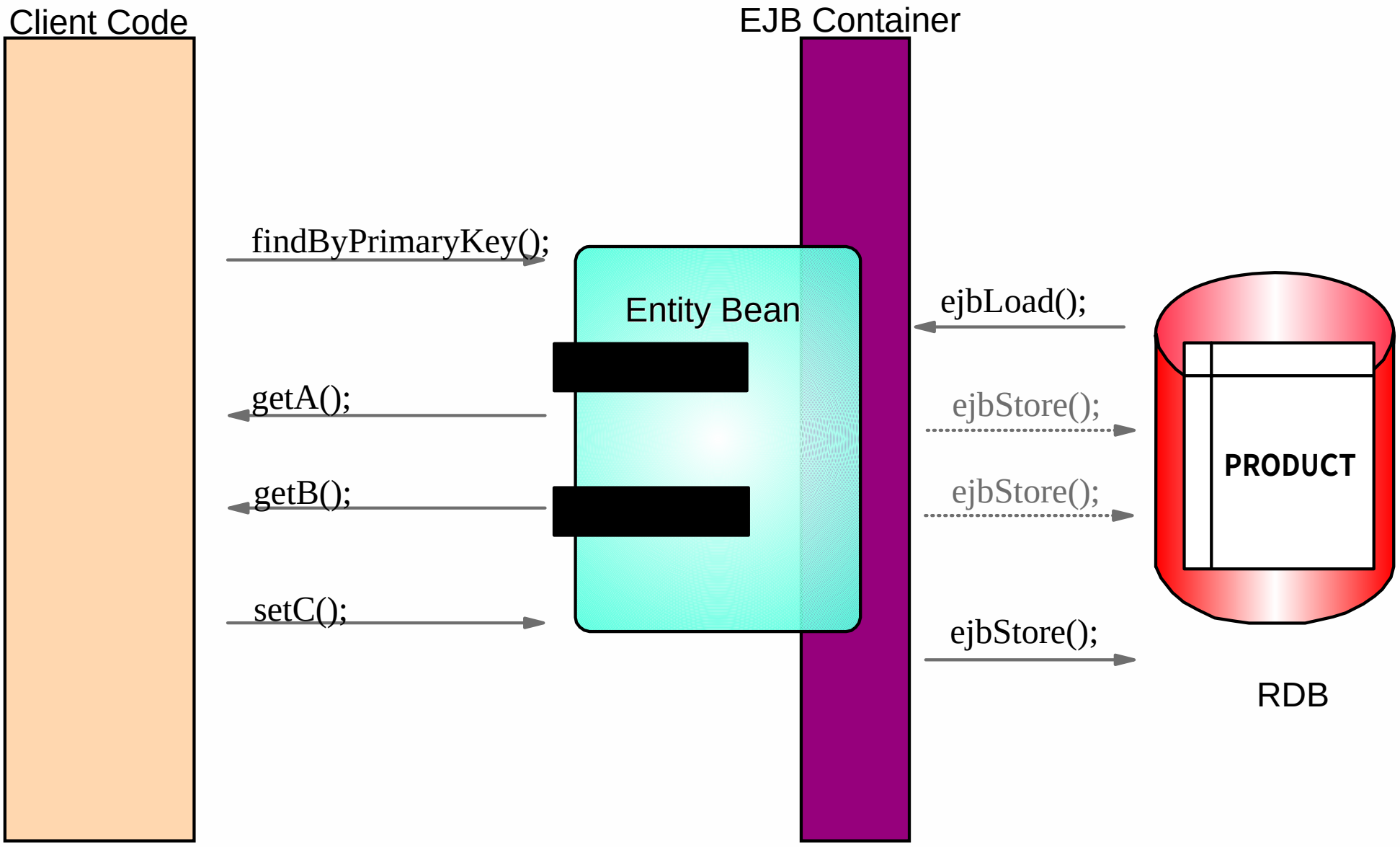
(4) 行の更新

```
Product product =  productHome.findByPrimaryKey(prrfnbr);
product.remove();
```

(5) 行の削除



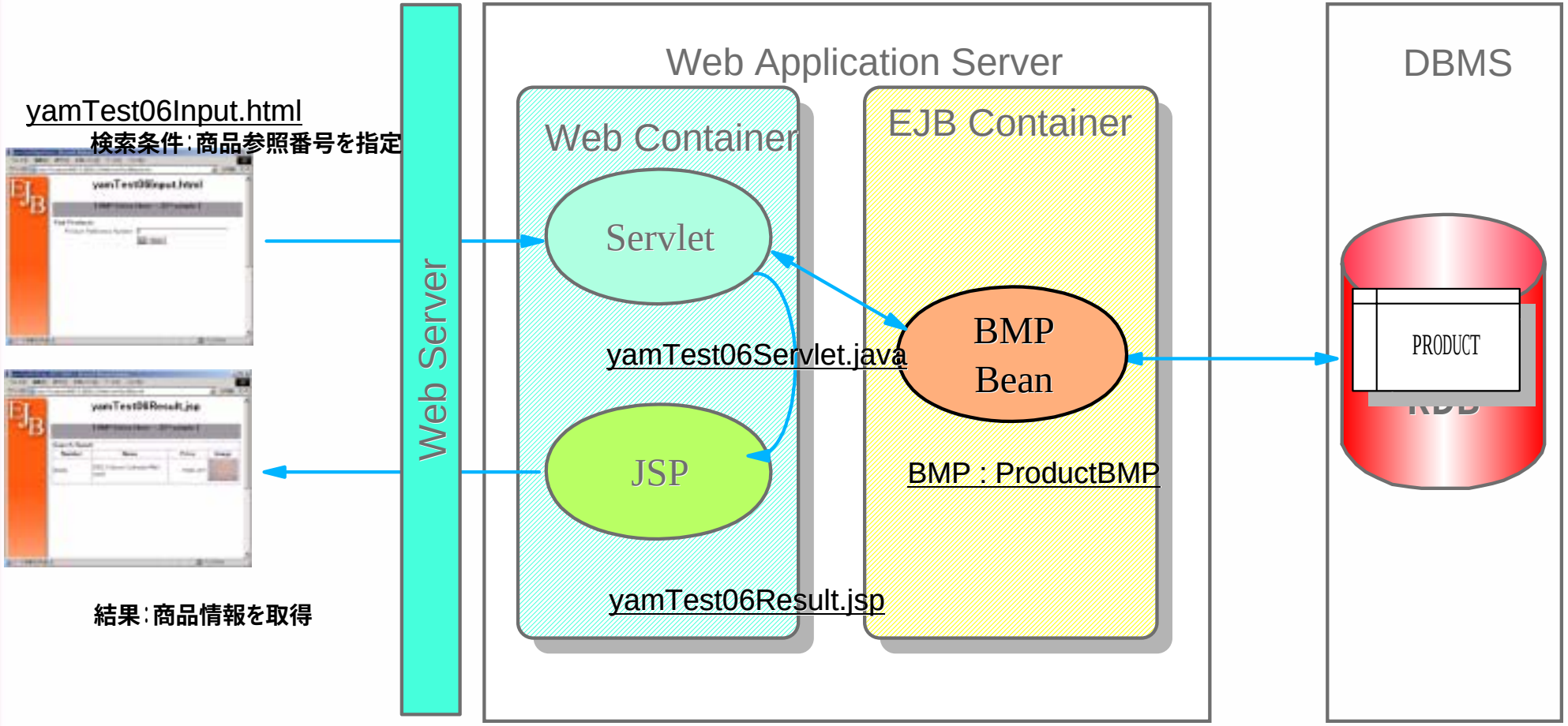
EntityBeanでのデータベースアクセス
データ同期



EntityBeanでのデータベースアクセス データ同期

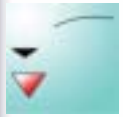
- EJBクライアントが実行するメソッドとコンテナが実行するメソッド種別と相互のタイムラグに注意
  - クライアントアプリケーションではどの時点でデータベース表に対しSelect/Updateが出るかのコントロールは出来ない
  - finder
    - キーを返すのみ
    - キーを元にした検索はその後ejbLoad()で行われる
  - getter
    - getter実行ではインスタンス上に取得された値を返す
    - <READONLY>指定がない場合、ejbLoad()で更新される
  - setter
    - setter実行ではインスタンス上に取得された値を更新する
    - その後ejbStore()で実際のDB値が更新される。
- CMPではejbLoad()、ejbStore()、ejbCreate()、ejbRemove()といったメソッドで内部的にコンテナにより実行されるSQLのコントロールは出来ない
  - SQL文はコンテナによりDeploy時点で作成される
- 1getterメソッドのアクセスのためにもejbLoad()=全カラムの取得が行われる
  - CMPフィールド（=列）が100個あるうち1個だけ取得したくても全カラム分の取得が行われる

接続パターン (3)
BMP Entity Bean

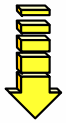


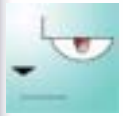
接続パターン (3)
BMP Entity Bean

▶どうしてもCMPでは出来ないときに



- CMPではEJBコンテナ任せのSQLがBMPでは自分で書ける
  - 複数表へのマップ、CMPではコントロール不可能なejbStore()やejbLoad()などを制御可能
  - 痒いところに手が届くSQL
    - 要らないカラムは取得しない、複数行の関数処理など
  - SQLの実行にPrepared Statementが使用できるのでキャッシュにより処理を高速化できる
- リソースがRDB以外の場合や複数データソースの場合でも使用可能
- CMPは多くのことを実現しようとしているが、現時点で実装可能な部分の限界がある
  - EJBのバージョンが進み多くの必要とされる機能の実装を待つ間、BMPでカバーしていくと言うのが基本的なスタンス





- CMPで出来ることをなぞるためのものではない
  - だから実はここで取り上げる例はあまりよくない
- 書くのはすごく大変
  - 通常のJDBCアプリと同じコードを書く必要
    - JDBCと同じ苦労＝実行時にしかSQLのエラーは分からない
  - でもEJBコンテナの気持ちになれる
    - CMP作成者は一度経験しておくべきかも
    - 一個書くまでが大変で、最初の一個が書ければ後の開発負荷は小さくなる
- メンテも大変
  - CMPなら構成の書き換えで済むところを再コーディングになる可能性
  - SQLを書く＝DBMSからの独立度合いが低いため移植が厳しい

BMPとSQLの対比

- Beanクラスメソッドと内部で実装するSQL
  - ejbCreate()
    - NOT NULLカラムの値を引数にINSERTを行う
  - ejbRemove()
    - Keyを元にしたDELETEを行う
  - ejbFindByPrimaryKey()
    - PrimryKeyによるSELECTを行い、Keyの有無を返す
  - ejbFindByXxx()
    - PrimaryKey以外でのSELECTを行い、Key（複数のケースもあり）を返す
  - ejbStore()
    - setterやビジネスメソッドで変更されたプロパティ値を保管するためのUPDATEを行う
    - UPDATE対象を何処まで絞るかでCMPのREADONLYメソッドに相当するコーディングを行いうる
      - 次項参照
  - ejbLoad()
    - findしたオブジェクトの値を埋めるためにKeyを元に取得の必要なフィールド分のSELECTを行う

BMPでのREADONLYの実装

▶EJB W/S 2000資料より

- BMPでのread-onlyメソッドの実装
  - CMPはコンテナに対して定義が可能だがBMPでは定義による切り替えは出来ない
  - getter call時のフラグを判別しejbStore()でDBをUPDATEしないようなロジックを追加する

❑dirtyフラグ

```
private boolean dirty = true;
```

❑ejbStore()のカストマイズ

```
public void ejbStore() throws java.rmi.RemoteException {
        if(isDirty()){
           データベース･アクセス･コード
        }
        setDirty(true);
}
```

❑アクセサー(getter)のカストマイズ

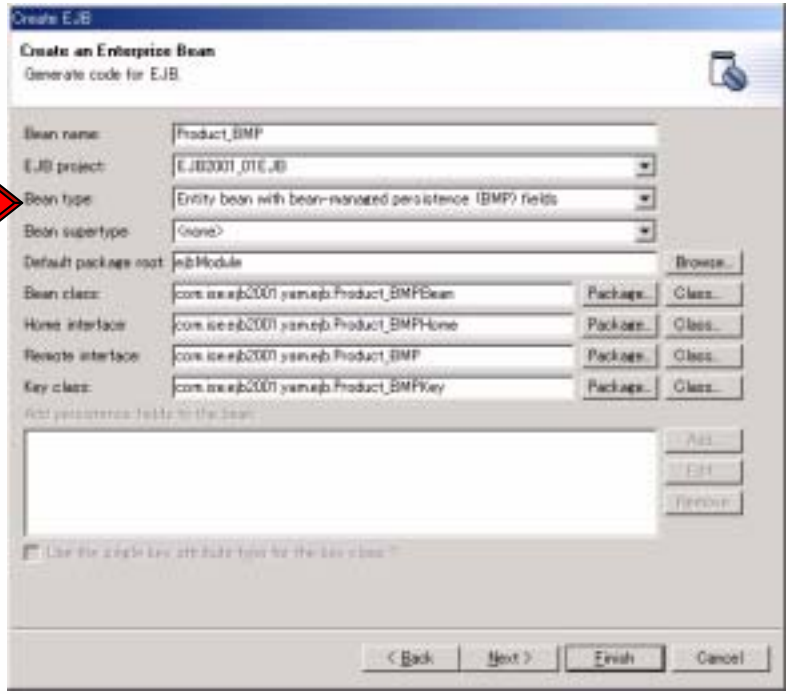
```
public java.lang.String getBooknum() {
        setDirty(false);
        return booknum;
}
```

BMP作成手順

▶WSADでの作成手順(1)

1. EJB WizardでBMPの作成
   - 以下のファイルがスケルトンとして作成される
     - ❑Beanクラス　<EJBName>Bean.java
     - ❑Homeインターフェース　<EJBName>Home.java
     - ❑Remoteインターフェース <EJBName>.java
     - ❑PrimaryKeyクラス　<EJBName>Key.java
2. Beanの開発
   - ejbCreate()、ejbPostCreate()メソッドの引数変更
     - ❑テーブル定義のNOT NULLカラム分に合わせ、これらのメソッドに引数を追加
   - ejbCreate()にロジックを追加
     - ❑INSERT文を実行
   - ejbFindByPrimaryKey()にロジックを追加
     - ❑SELECT <PKカラム> ... WHERE <PKカラム> = 引数を実行
   - ejbLoad()にロジック追加
     - ❑SELECTし変数に値を格納する部分を実行
   - ejbStore()にロジックを追加
     - ❑変数をUPDATEする部分を実行
   - ejbRemove()にロジックを追加
     - ❑DELETE部分を実行
   - Accessorメソッド(getter/setter)を追加
     - ❑これは変数の格納と取り出し (実際はLoadとStoreで実行)
   - Cutsom Finderを実装
     - ❑Primary Key以外での検索を行うSELECT部分
   - ビジネスメソッドの実装
     - ❑その他オペレーション

BMP



BMP作成手順

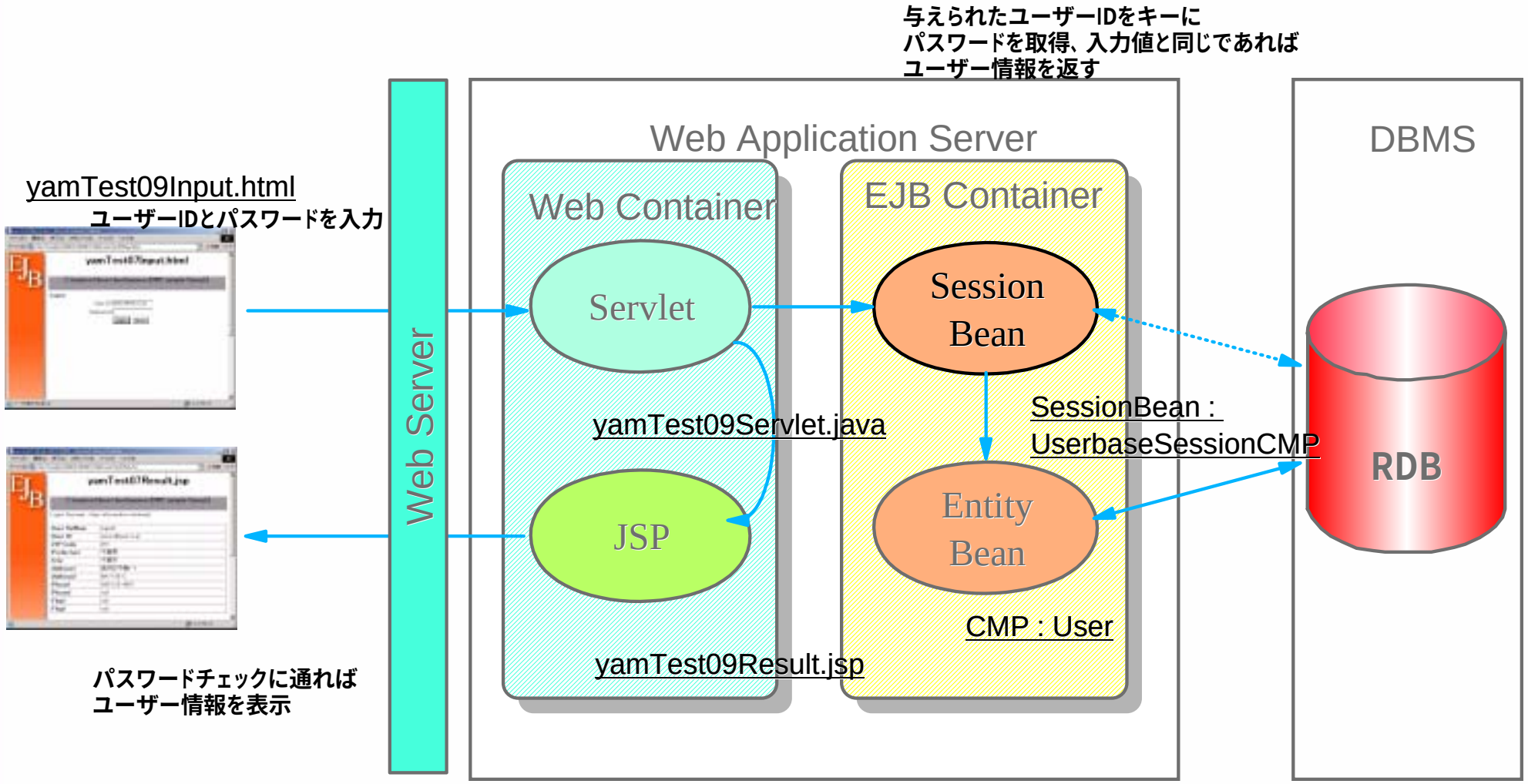
►WSADでの作成手順(2)

3. Home Interface定義
   - create()メソッド引数変更
     - ejbCreate()/ejbPostCreate()にあわせて引数変更
4. RemoteInterface定義
   - 必要なビジネスメソッドやAccessorをBeanからPromote
5. PrimaryKey Class定義
6. DDの編集
   - EJB Editor、Extension Editorなどで定義情報を指定（JNDI名など）
7. テスト
   - テスト環境でメソッドコールが適切にできるかを確認。

CMP or BMP ?

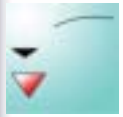
- **CMPかBMPか？**
  - Persistent ObjectはCMPで行こう（作る＆メンテが楽だから）、ただし下の条件を満たさない場合
    - テーブルへのマッピングで問題がある
      - Relational DBのRelationをどのように現すか → 後述
      - アクセスすべきリソースが多岐にわたる、データソースが複数ある
    - パフォーマンスをもっと追求
      - BMPで処理速度を追求
    - Beanの数が少ない
    - 『楽だけど束縛される人生』より『道は険しくても自由な人生』を求めるタイプのプログラマーばかりがなぜかプロジェクトにたくさんいる
  - CMPで作った後BMPに変えても、Home/Remoteが変わらなければクライアントに影響は与えない

接続パターン (4) Session Bean

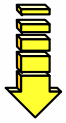


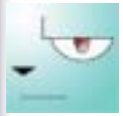
接続パターン (4) Session Bean

▶Session Beanには基本ビジネスロジック



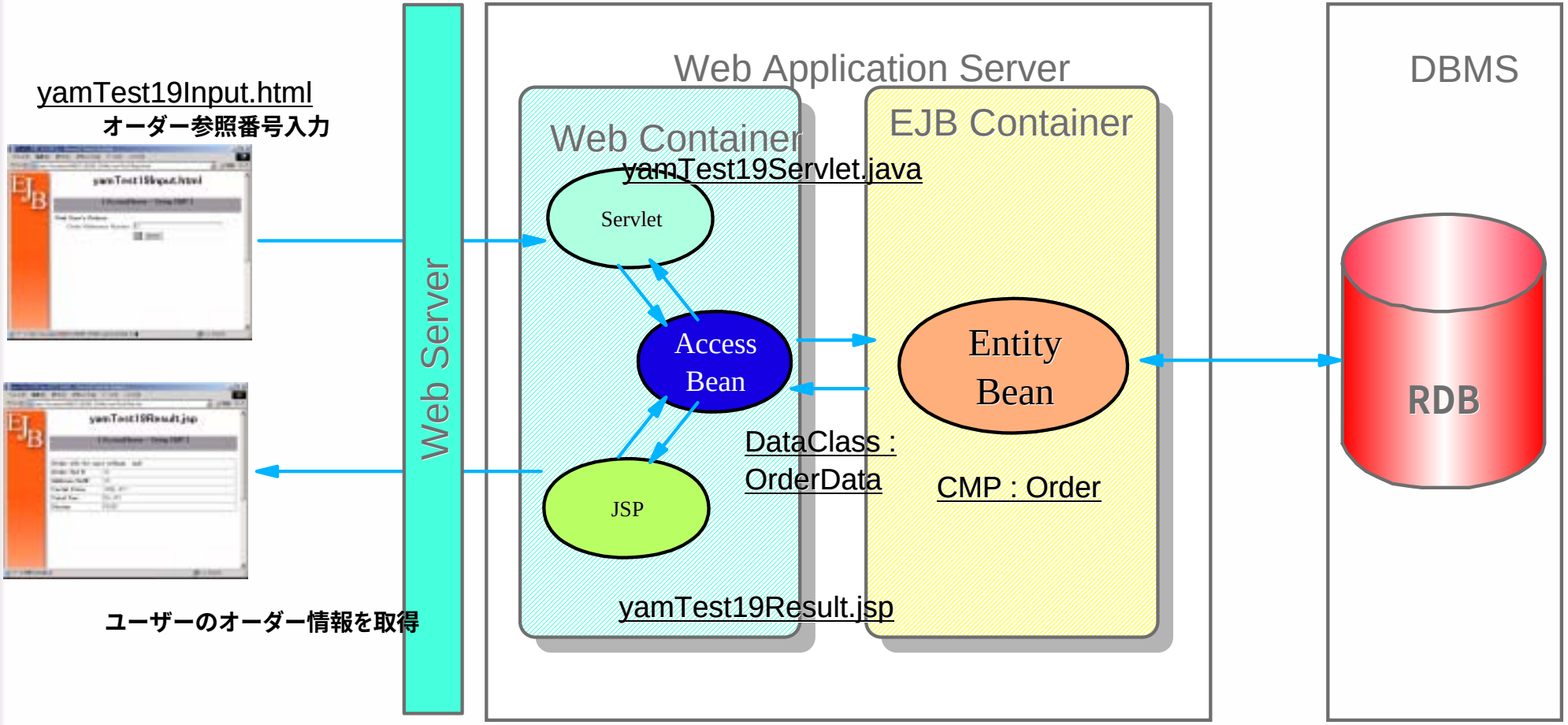
- 通常Enitity Beanに対してはビジネスロジックをどこで持つかという観点での比較対象
- データアクセスの観点からSession Beanを理解すると2通り
  - データアクセス部分をJDBCで直接書き込んだSession Bean
  - データアクセスをCMPを介して行うSession Bean
- クライアントのアクセス方法が簡略化
  - CMPをアクセスする部分をカプセル化し、SessionBeanのビジネスメソッドを触ることに注力できる
  - トランザクションコントロールなどもこの辺りのレイヤーでやるのが通常
- CMPの苦手な大量検索、複雑な条件での検索などに適用可能





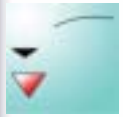
- SessionBeanベタ書きはDBリソースのコンポーネント化をあきらめる部分
  - 設計などに関わってくる
  - でもどうしてもEntityBeanでは出来ない場合に補完的に使用か

接続パターン (5) Access Beans

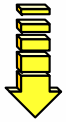


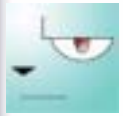
接続パターン (5) Access Beans

▶EJBクライアントの悩み解決



- コードの複雑さの隠蔽
  - EJBのlookup→Home→Remoteという手順
  - EJB独自の作法がクライアントコーディングを行うプログラマーにEJBのスキルを要求する
    - ❑AccessBeansの使用により通常のJavaBeans使用と同様のアクセスが可能
    - ❑EJB開発者とクライアントコード開発者でRole分けが可能
- パフォーマンスの改善
  - EnterpirseBeanのProxy Objectへの毎度のRemoteコールがパフォーマンスを落とす
  - ローカルで完結する構成でも毎回Remoteコールを行う必要性
    - ❑AccessBean使用によりAccessBeanにまずプロパティ値をキャッシュ、各getter/setterはこのキャッシュにアクセスすることでremoteコールを極力減らす

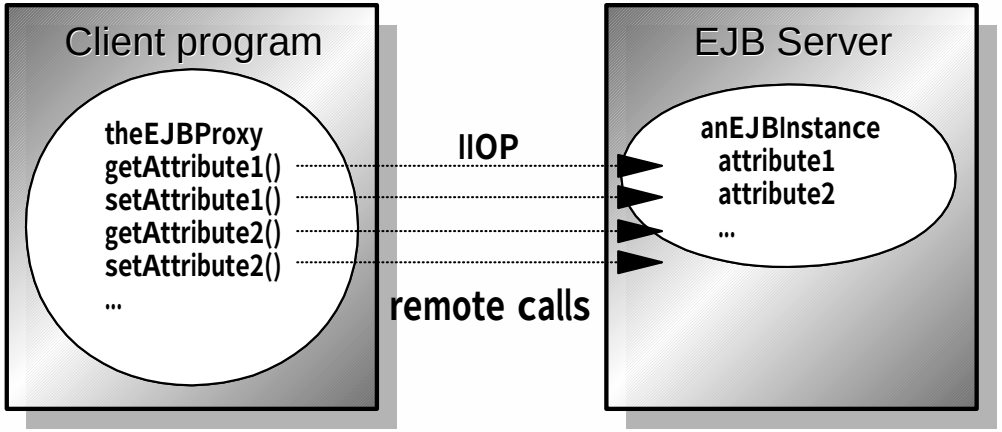




- ベンダー独自インプリ
  - 互換性の消失、移行時の考慮
- 新たなものを覚える必要性
  - ま、EJBに比べれば. . .
- Access Bean独自の考慮点
  - DB、EntityBeanからさらに離れた場所に情報をキャッシュするため特に更新系アプリではデータ整合性を維持する考慮が必要

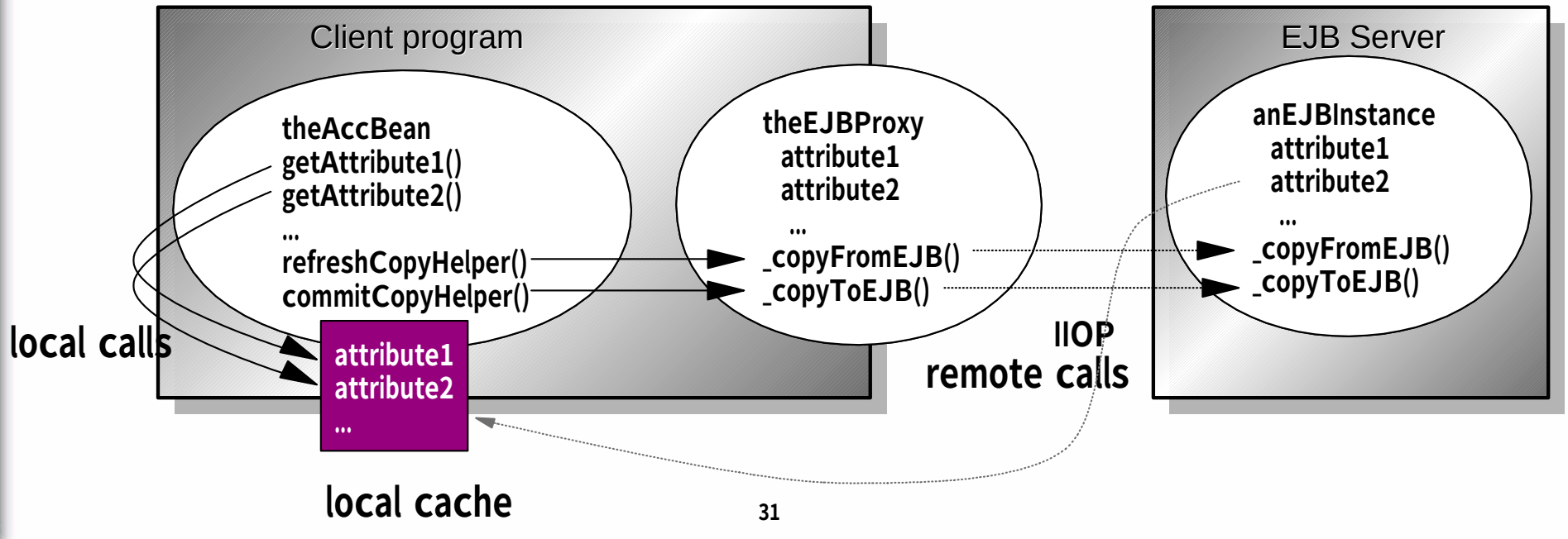
Access Beans Advantages

►AccessBeansを使用する利点



Enterprise Beans Client

Access Beans Client



AccessBeanの種類

VisualAge for Java

WSAD

Java Bean Wrappers

Copy Helpers

Rowset Access Beans

EJB Factories

Data Classes

- InitialContextとHome Interfaceの機能をカプセル化して提供
- 他のタイプのAccessBean作成で自動作成

- プロパティ値をローカルにキャッシュしアクセスすることでパフォーマンス改善

Type	WSAD	VAJ	Session Bean	Entity Bean	Recommend
Bean Wrapper	✓	✓	✓	✓	✗
Copy Helper	✓	✓	✗	✓	✗
Row Set	✗	✓	✗	✓	✗
EJB Factory	✓	✗	✓	✓	✓
Data Class	✓	✗	✗	✓	✓

AccessBeans クライアントコードサンプル

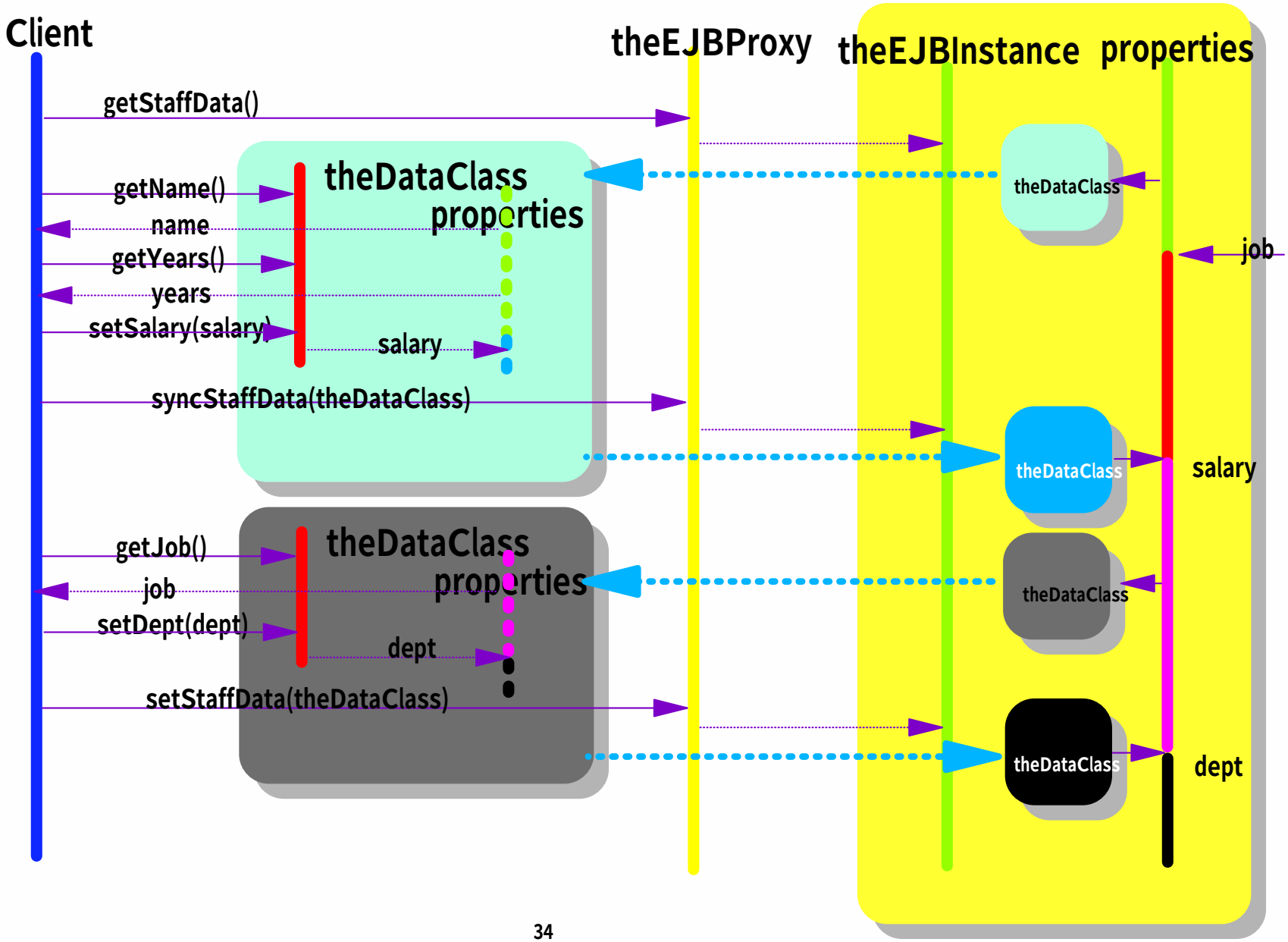
▶通常のEJBクライアント

```
Hashtable parms = new Hashtable();
parms.put(Context.INITIAL_CONTEXT_FACTORY,
                  "com.ibm.websphere.naming.WsnInitialContextFactory");
Context ctx = new InitialContext(parms);
orderHome = (OrderHome) javax.rmi.PortableRemoteObject.narrow(
ctx.lookup("ejb/Order"),OrderHome.class);

Integer orderrefnum = new Integer(request.getParameter("orderrefnum"));
Order order = orderHome.findByPrimaryKey(new OrderKey(orderrefnum));

String row[] = new String[6];
     row[0] = orderrefnum.toString();
     row[1] = order.getOrUsnbr().toString();
     row[2] = order.getOrAdnbr().toString();
     row[3] = order.getOrTotprice().toString();
     row[4] = order.getOrTottax().toString();
     row[5] = order.getOrStat();
```

▶Factory使用

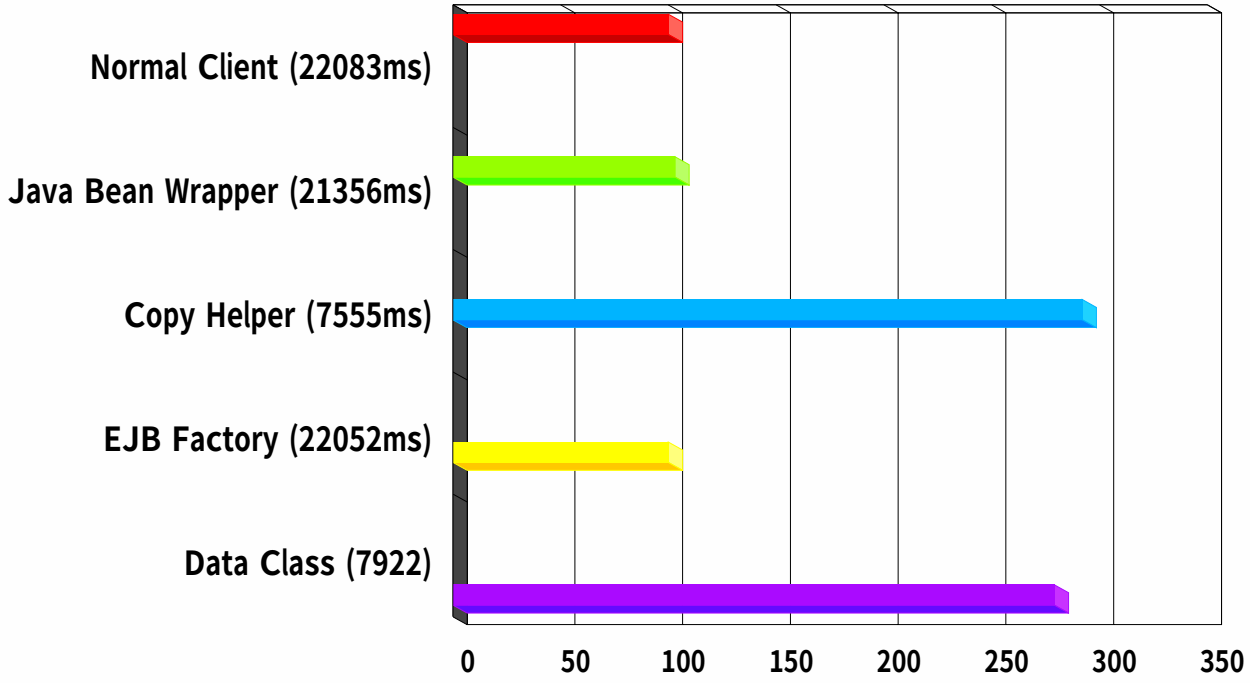
```
OrderFactory factory = new OrderFactory();
Order order = factory.findByPrimaryKey(new OrderKey(orderrefnum));

String row[] = new String[6];
     row[0] = orderrefnum.toString();
     row[1] = order.getOrUsnbr().toString();
     row[2] = order.getOrAdnbr().toString();
     row[3] = order.getOrTotprice().toString();
     row[4] = order.getOrTottax().toString();
     row[5] = order.getOrStat();
```

参考 DataClass



AccessBeansパフォーマンス比較

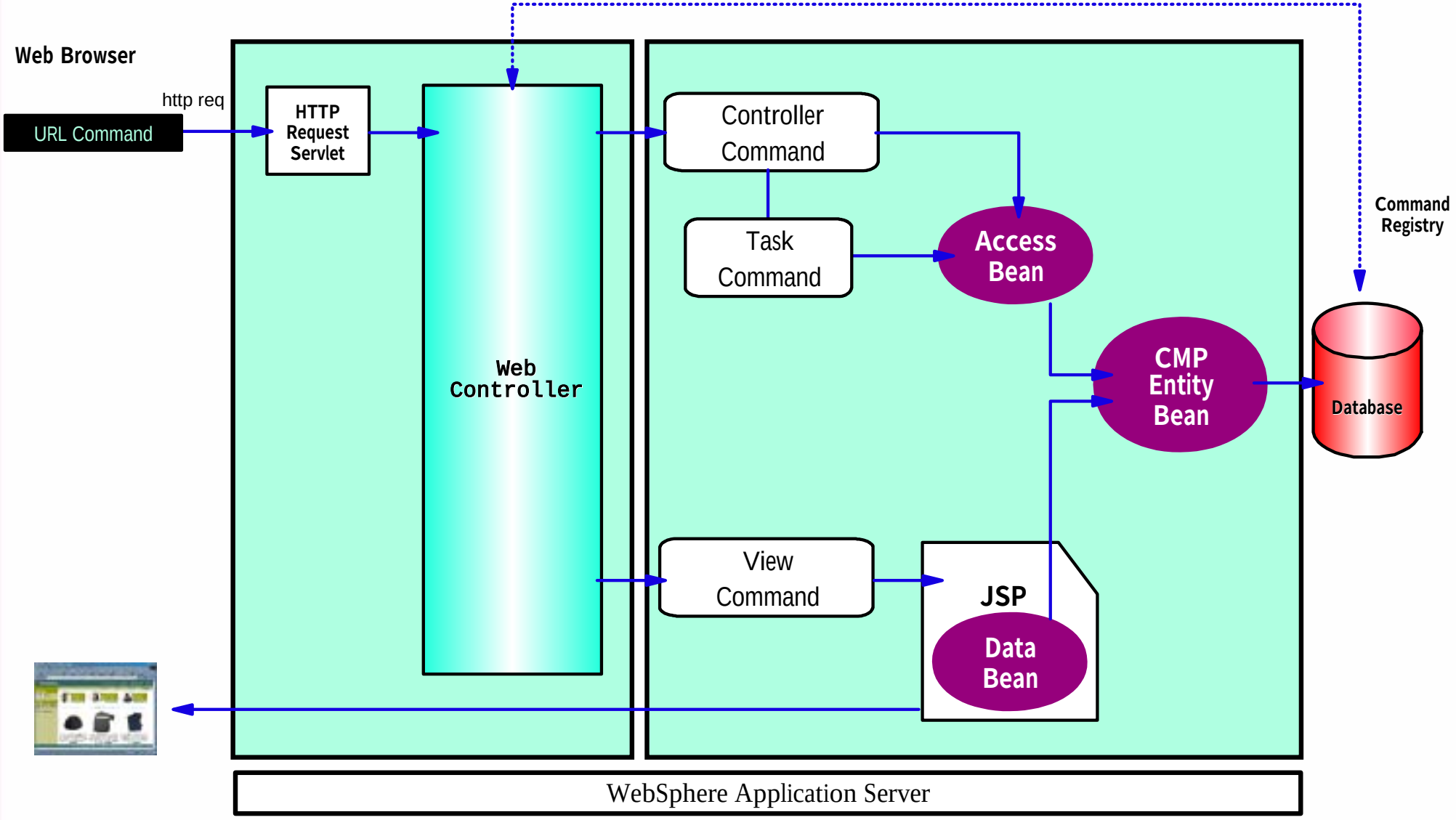
- パフォーマンス比較
  - JavaBean Wrapper、CopyHelper、EJB Factory、Data Classで通常クライアントと比較
    - ServletのserviceメソッドHomeの取得から複数のgetメソッドの実行を1000回ループ
  - CopyHelper/Data Classは2倍以上のパフォーマンス改善が望みうる
    - 機能的な拡張がなされた分Data Classが若干遅い
- Normal EJB Clientを100とした場合のパフォーマンス相対値



比較サマリー

- **EJB使用**
  - この議論は完結していると言う前提
  - アプリケーションの設計がオブジェクト指向設計であればなおEJBとの親和性高し
    - データベース設計が完了していなくてもクライアントコーディングが出来る
- **EntityBean使用**
  - 基本的にEJBのPersistentObjectはEntityBeanで扱う
  - データオブジェクトの量が大量になると現行コンテナではキャパシティ的に扱えない場合もある
  - 検索条件が定まらない、大量リターンの平均値を取るなどEntityBeanが苦手とする部分にSessionBeanなどのオプションを取りうる
- **CMP vs BMP**
  - コーディングの負荷を考え極力CMPで済ませたいがRelationの取り方、パフォーマンスなどの要件によりBMPを使用することも検討
  - BMPを使用する場合はCMPの模倣ではなくアドバンテージが出せる形で使う
  - BMP使用に踏み切ることでポータビリティを犠牲にすることになる
- **Access Bean使用**
  - EntityBeanのパフォーマンス改善を図りたい
  - コーディング量の軽減、開発者のRole分けが可能
  - これもポータビリティの犠牲と移行の際の考慮が必要になる

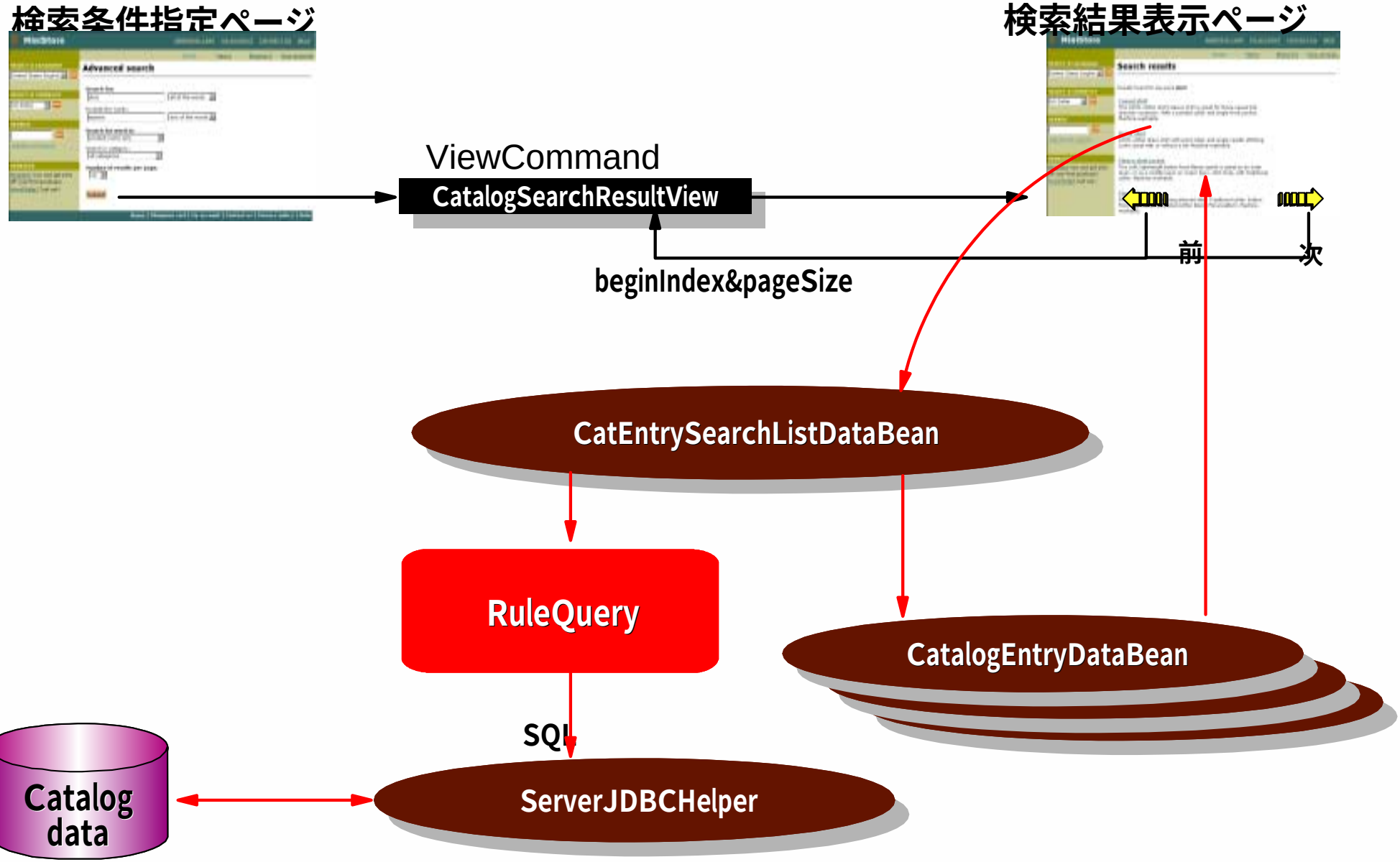
Web-RDB & EJB使用例 (1)
WebSphere Commerce Suite 5.1



Web-RDB & EJB使用例 (1) WebSphere Commerce Suite 5.1

- CMPを使用したECアプリケーションをツールとして提供
  - WCS5.1は現状WAS3.5上で稼動
  - Servletは全体で一つ
    - リクエストを解析しWebControllerに送る
  - WebControllerがトランザクション制御などコントロールを一手に引き受ける
    - URLリクエストから定義情報のデータベースを参照し、ControllerCommandクラスを呼び出す
  - ControllerCommandがビジネスメソッドを実装
    - データアクセス部分はCopyHelperのAccessBeanを介して行われる
    - カストマイズ時のControllerCommandのコーディングは使用すべきAccessBeanのJavaDocを元に行う
    - 必要に応じてサブルーチン的にTaskCommandを呼び出すが、その内部もデータアクセスに関してはAccessBeanを使用している
    - 正常終了時点でViewNameを返し、WebControllerが呼び出すべきJSPのAliasを指定
  - ViewNameを元にWebControllerはViewCommandを介してJSPを呼び出す
    - これもViewNameをもとに定義情報のデータベースを参照し、適切なJSPを呼び出す
    - データアクセスは基本的に参照系に限定されるため、AccessBeanをさらに拡張したDataBeanと呼ばれるJavaBeansを使用する
  - WebControllerがトランザクション制御を行うため、ControllerCommand/TaskCommand/JSPの処理は1UOW内で行われる

Web-RDB & EJB使用例 (2)
WebSphere Commerce Business Edition 5.1



Web-RDB & EJB使用例 (2)
WebSphere Commerce Business Edition 5.1

- **基本はWCSと同じ、商品検索機能をSessionBeanで実現**
  - WAS 4.0ベースとなるWCBE 5.1 GAは来年1Q予定、現行Dev EditionはWAS 3.54ベース
  - 基本構成や使用クラスはWCSを元にしているが、EntityBeanが苦手な複雑な検索条件指定や戻り件数のコントロールなどが必要となる商品検索機能にSessionBeanを使用している
  - 参照系なので、呼び出すのはControllerCommandではなくViewCommand
    - 検索条件を指定して呼び出されるViewCommand→JSPの中にCatEntrySearchListDataBeanというDataBeanが入っている
  - CatEntrySearchListDataBeanは受け取る条件となるパラメーターが決まっており、そのパラメーターをRuleQueryに渡す
    - RuleQueryは入ってきた値を元にSQL WHERE Clauseを作成、ServerJDBCHelperというSessionBeanにSQLの処理を依頼し条件に合致した商品のキー値を受け取る
  - CatEntrySearchListDataBeanは通常の商品表示と同様に受け取ったキーから商品情報を取得するDataBeanを呼び出し、商品情報を表示する