XML Consortium

ProjectZeroの目指すWeb Oriented Architecture とは?

～Enterprise 2.0を目指すProjectZeroのご紹介～

XMLコンソーシアム Web2.0部会 SOA部会
日本アイ･ビー･エム　根本和郎

Disclaimer

XML Consortium

- ここに記載されている内容は、発表者個人の私的見解であって、IBMの見解を代表するものではありません。

Agenda

- Ruby on Railsの良さを個別に分解評価して、どのようなアーキテクチャー選択肢から選ばれたのか考察。
- その視点でProjectZeroの企画過程をなぞって考える。
- Enterprise化に対応可能なFull stack frameworkへの提言

Architectural decision

Ruby on railsの選んだもの

- 言語
  - **Ruby** / Java / Groovy / PHP / Python / ...
- O/R Mapping
  - Data Mapper / Row Data Gateway / Table Data Gateway / **Active Record**
- Convention over Configuration
  - **Convention**(設定) / Configuration(規約)
- オブジェクト指向 / DOA
  - **DOA**
- 定義体
  - **YAML** / XML
- Installのタイミング
  - **必要時にgemsでinstall** / 製品導入時にinstall

- 言語についての考察

Ruby

- Duck typing
  - 型宣言なし
- 動的決定性
  - Open class
  - 実行時決定による曖昧性
    instruction = "p 'hello cracked world'"
    eval(instruction) // 'hello cracked world'
- 趣味からスタートした言語
  - 言語仕様?
  - JCP (Java Community Process)相当なもの?

比較：Java, Groovy, Ruby

日付型オブジェクトの生成例

Java

```
import java.util.Date;
Date today = new Date();
```

Groovy　= Java – “ボイラープレート”

```
today = new Date()
```

Ruby

```
require ‘date’
today = Date.new
```


Boiler plate = 毎回同じ記述

Java

```
import java.util.*;
public class Date1 {
   public static void main(String[] args) {
        Date today = new Date();
        System.out.println(today);
   }
}                          // Mon Apr 21 08:15:38 JST 2008
```

Groovy　(クラスなしTop level statement表記可能)

```
today = new Date()
println today      // Mon Apr 21 08:23:35 JST 2008
```

Closure

- 「**1,2,3,4,5**」という数字の配列から、偶数を抜き出せ

Ruby

```
p [1,2,3,4,5].select {|i| i%2==0 }          // [2, 4]
```

Groovy

```
println ([1,2,3,4,5].findAll {it%2==0 } )   // [2, 4]
```

“動詞(find)＋目的語(All)” の Javaのメソッド表記ルールに準拠

it：クロージャー内 イテレータのデフォルト引数定義

Groovy Bean：JavaBeanのGroovy版

Foo = 型宣言なし　Bar = 型宣言あり　optional typing

```
class TypeSafeTest {
    static void main(args) {
    Foo f = new Foo()
            f.age = "taro"
    Bar b = new Bar()
            b.age = "hanako"
    println f.age
    println b.age
    }
}
class Foo {
    def name
    def age
}
class Bar{
    String name
    int age
}
```

Int型への文字列代入で エラーになる

def 型だと制約なし　Object型となる

堅く型宣言することもできる

Groovy,Java間の自然な連携

Groovy　(呼び出し側)

```
class Hello {
   static void main(args) {
   println “Hello"
   World wd = new World()
   wd.greet(“ Groovy! ")
   }
}
```

通常のメソッド呼び出しで
参照可能
ScriptEngimeManager必須ではない

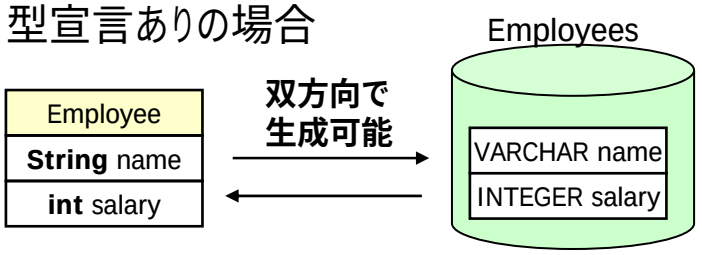
Java　(呼ばれる側)

```
public class World {
   public void greet(String arg) {
   System.out.println("world“ + arg);
   }
}
```

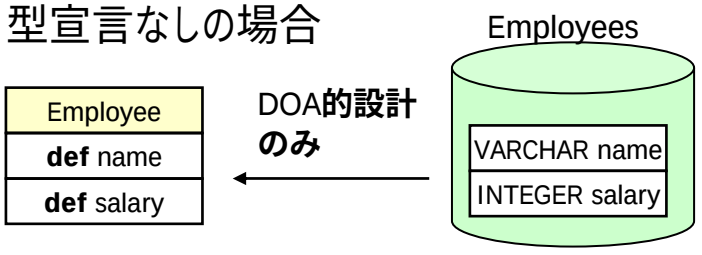
実行結果:
Hello world Groovy!

O/R Mapping

- 型宣言ありの場合



- 型宣言なしの場合



Rubyではなく Groovy

- 既存のJavaコード資産との親和性
- JSR241で、確定している言語仕様
- Javaプログラマーの習得効率
- Enterprise VM としてのJVMの資産価値
  Linux, AIX, Windows, zLinux, iSeries ...
- Optional typing：
  Duck typingと静的型宣言を必要に応じて使い分け可
- パフォーマンス：
  過激な動的決定性は求めない

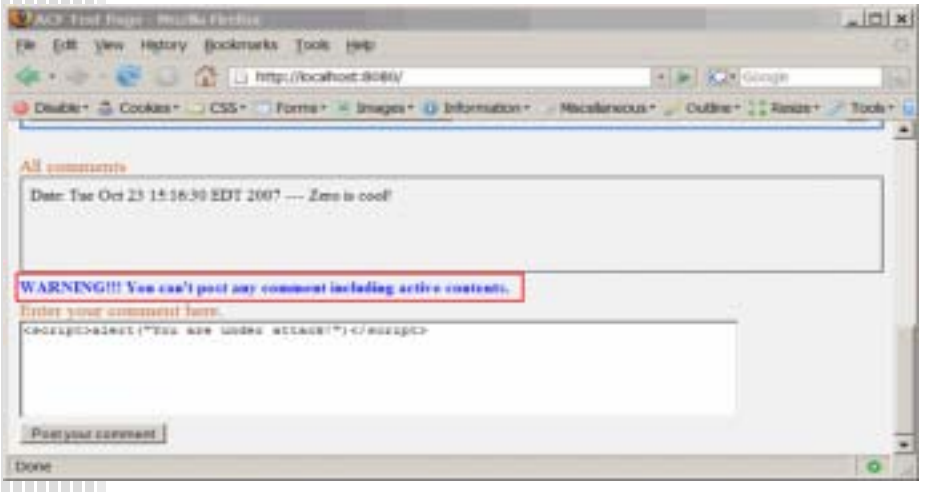
- セキュリティーについて考察

Securityは特別に重要

- HTTPSは通信を守るのみ
- WS-Security でのデータ保護は必要
- Active Content Filtering
  ユーザー入力からのスクリプトタグの自動削除
- Cookie管理はLTPAToken2 (Light weight Third Party Authentication) でCookie管理
- エンタープライズサービスに(Beta) はあり得ない、保証重要

Active Content Filtering

スクリプト除去機能



- http://www.ibm.com/developerworks/jp/web/library/wa-pz-acf/

- O/R Mappingについて考察

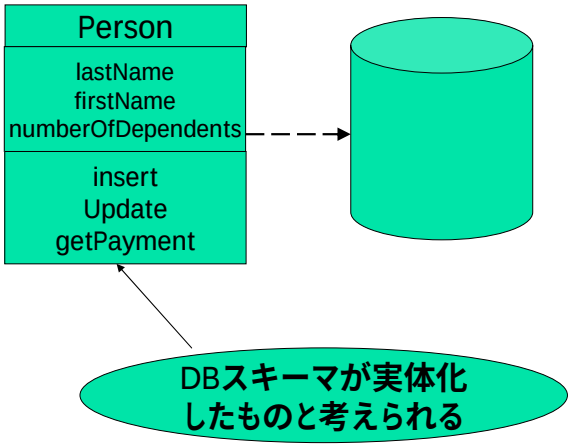
O/R Mapping

- Active Record
  1レコードに対して、一つのエンテティをCRUD操作メソッド付きで公開する
- Table Data Gateway
  テーブルそのものへのアクセッサーメソッドを提供する
  1テーブルに対して、一つのインスタンスと、一つのFinderクラスが存在
- Row Data Gateway
  ひとつのエンテティを生成し、そのフィールドを個々に操作することでデータをセットする。検索時にはFinderを使用する。
- Data Mapper
  Mapperを中間に挟み、両者のスキーマを完全に分離する

PofEAA “Pattern of Enterpirse Aplicaiton Arhitecture”参照

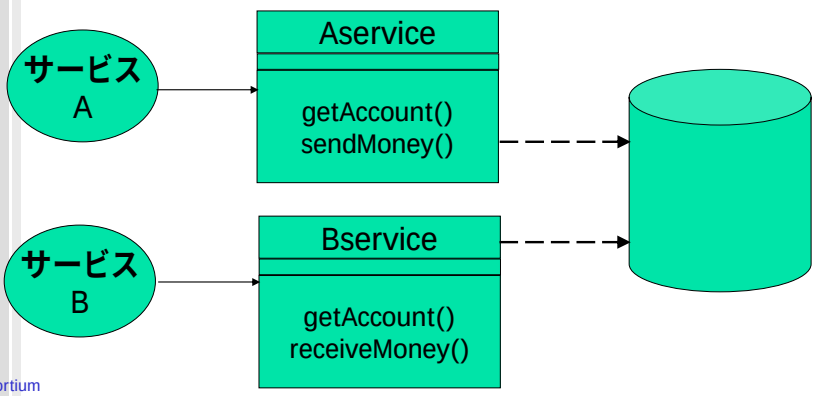
ActiveRecord

- データと振る舞いの両方を持つオブジェクトを生成する



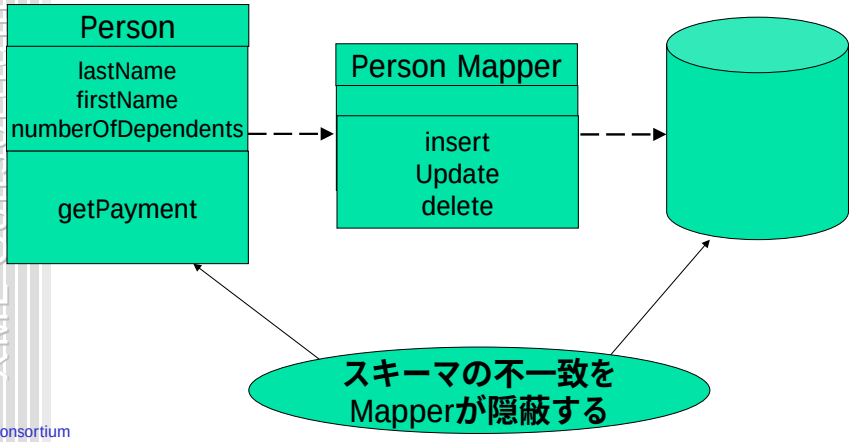
TransactionScript

- ビジネスロジックを必要なサービス事に個別に作成する。
- 開発は簡単だが似たようなビジネスロジック複数発生する
- Row data gateway, Table data gateway の2種類



DataMapper

- Mapperクラスが、インピーダンスミスマッチを吸収する

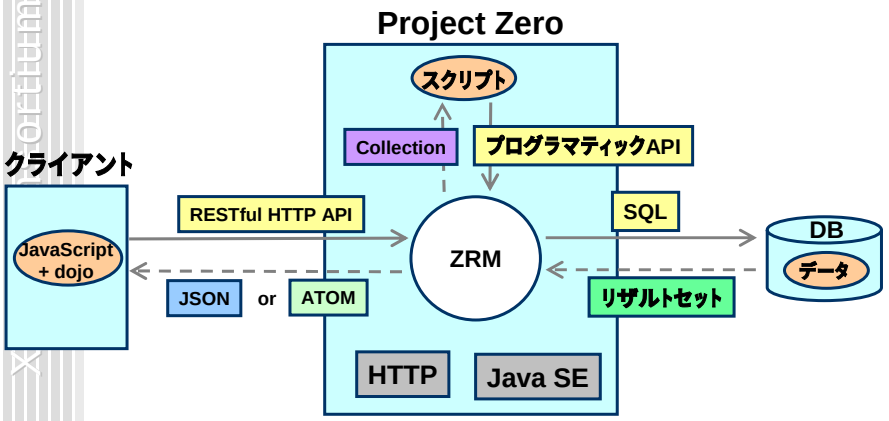


O/R Mapping

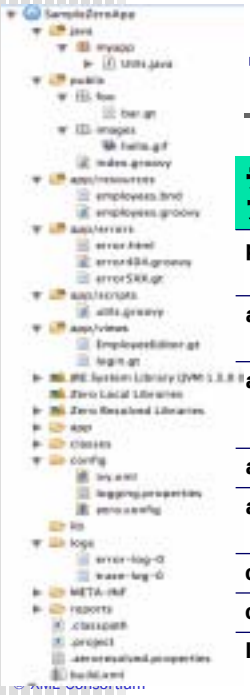
- Active Recordは
  - 小規模開発の迅速なスタートが可能
  - DBありき、DOA的アプローチ
  - 複雑なビジネスロジックが先にあると対応しにくい
  - 過去の資産継承は苦手
- Mapper は
  - ビジネスとリソースの分離が可能
  - O/R Mappingは双方向
  - Zero resource model ? (後述)

Zeroリソース・モデル – ZRM -

Databaseに対して双方向
DOA(Data指向アーキテクチャー)の制約を受けない
JavaSEベース (JavaEEではない)



- Convention over Configuration



ディレクトリ構成

自動生成のConvention

ディレクトリまたはファイル名	説明
public	Webからアクセス可能なルート・ディレクトリ(ドキュメント・ルートに相当)
app/resources	RESTfulリソースのリソース・ハンドラ(例.リソース名.groovy)を配置するディレクトリ
app/errors	エラー表示する静的ファイル(例 .html)または動的ファイル(例.groovy, .gt)を配置するディレクトリ
app/scripts	共有されるスクリプトを配置するディレクトリ
app/views	ビューを実装するスクリプト(例.gt)を配置するディレクトリ
config/ivy.xml	依存する拡張モジュールの定義
config/zero.config	構成ファイル
logs	ログ用ディレクトリ

*.gt = Groovy Template

zero.config (zero)　database.yml (RoR)

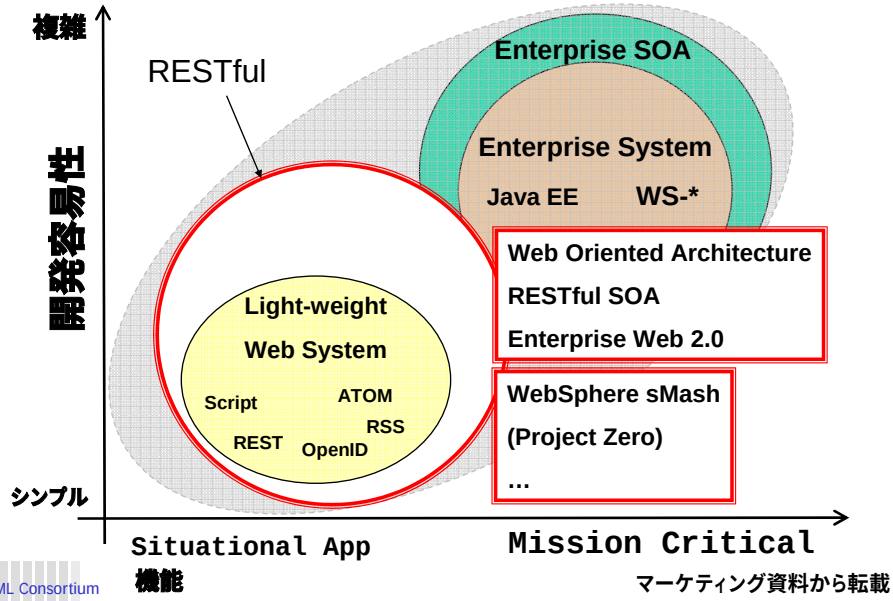
Zeroリソース・モデル(ZRM)

DB構成

- デフォルトでは、ZRMがApache Derbyを使用し、DB、テーブルを作成し、利用する
- デフォルトのDB構成を置き換える場合、次の例のような構成をzero.configに定義する
  - ***/config/db/zero-resource = {***
  - ***"class" : "org.apache.derby.jdbc.EmbeddedDataSource",***
  - ***"databaseName" : "db/resource",***
  - ***"createDatabase" : "create"***
  - ***}***
- 既存のDBを使用する場合、次の例のような構成をzero.configに定義する
- ***/config/resource/dbKey = "my-database"***

- SOA + RESTful について考察

複雑/シンプル/業務/Web

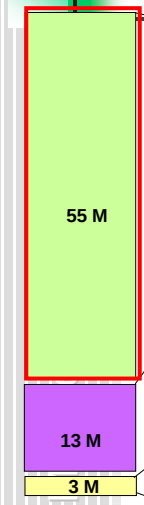


マーケティング資料から転載

RESTful SOA

- プロトコル：HTTP (SOAP over JMSも)
- ビジネスオブジェクト：JSON, ATOM, XML
- インターフェイス：ブラウザ上でAJAX
- 動詞：GET, PUT, POST, DELETE
- セキュリティー：HTTPS + WS-Security

エンタープライズ開発者+Web開発者



開発者分類

スキル

アプリケーション・アセンブラ
•Those using computers at work who can automate business tasks that are not already addressed (e.g. working with databases and dynamic spreadsheets).
•Shielded from integration issues.

UI and Content Manipulation Skills

Spreadsheets, Databases, Wikis, Blogs, Web Content Management

Webデベロッパ
•Service consumers.
•Those using computers at work who say that they "program"
•Focused on empowering the Assemblers.

Web Development. Skills for content integration and application logic.

商用 + OSS App Server + web開発者

トラディッショナル・プログラマ
•Creating services, wrapping services
•Deploying these applications. Discovery of feeds & services, mgmt of the environment, etc

Skills for backend Integration and

商用App Server + 企業開発者

•1M are sophisticated professional application programmers, 2M are less sophisticated

US Bureau of Labor Statistics
Estimate for US - 2012

http://www.cs.cmu.edu/~cscaffid/papers/eu_20050923_vlhcc.ppt

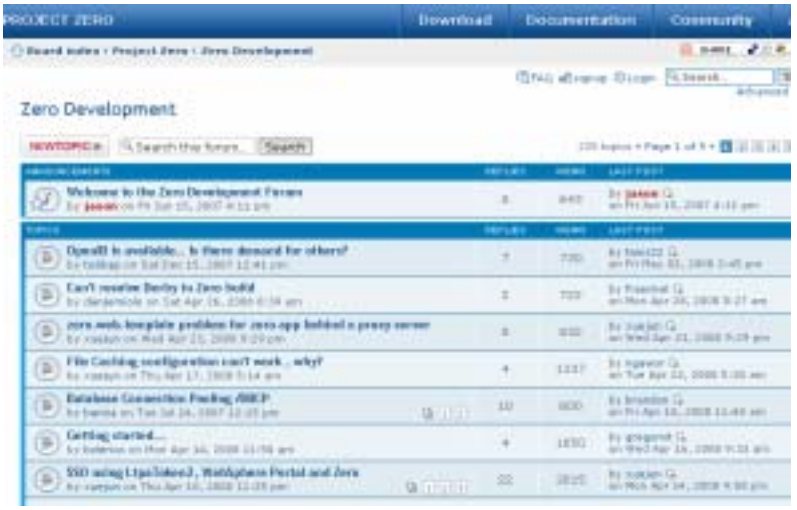
Zeroのコンセプト

- Prototype.jsでなく dojo toolkit
- Rubyでなく Groovy + PHP
- Web技術とEnterprise技術をブレンド
- スレッド型でなくプロセス型
- 1JVMプロセス = nリクエスト (メモリリーク回避)
  高速起動型JVMを採用
- WOA (Web Oriented Architecture)
  SOA資源にWebでアクセス
- 動的決定 開発時決定

www.projectzero.org

- 充実したドキュメントよりも、Wiki

Forum



Assembler

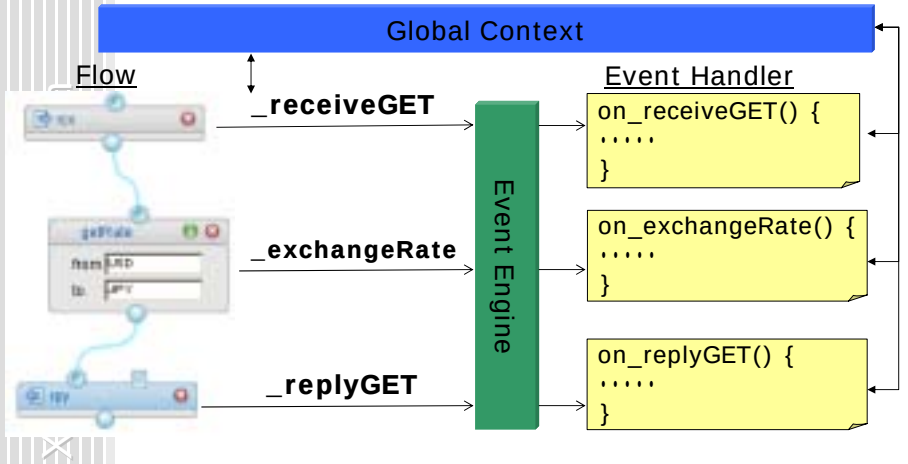
- ブラウザ上で開発
- XMLが流れて処理が進む
- 定義が追加可能

Groovy DSL: Flow builder

```
def flow() {
  def builder = new FlowBuilder(writer);
    builder.process( name:"feedGroovyExample") {
      receiveGET( name:"rssRcv")
      feed( name:"YahooFeed", url:"http://rss.news.yahoo.com/rss/topstories") {
        control( source:"rssRcv")
      }
      feed( name:"CNNFeed", url:"http://rss.cnn.com/rss/cnn_topstories.rss") {
        control( source:"rssRcv")
      }
      aggregateFeeds( name:"aggregate" ) {
         input( value: '${YahooFeed}')
         input( value: '${CNNFeed}')
      }
      sortFeed( name:"sort", orderBy:"-title") {
         input( value: '${aggregate}')
      }
      replyGET( name:"rssRply") {
         input( value: '${sort}' )
      }
    }
}
```

GroovyのBuilder機能を利用して、Flow専用のDSLを提供

Flowアーキテクチャー



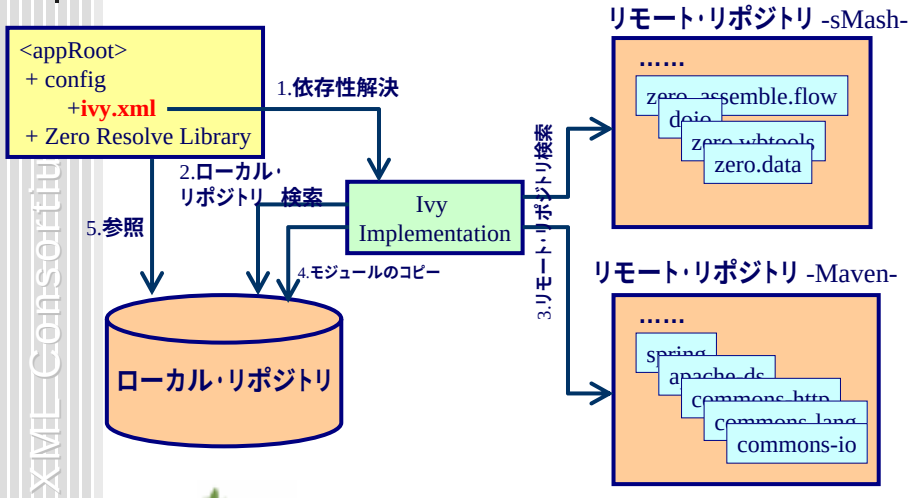
Flow定義

- Flow定義の方法は二通り
  - XML(Flow Language)
    - 手書き or Flow Editor
  - Groovy(Flow DSL)
    - 手書きのみ

単純なFlowはGUIで手っ取り早く
複雑なFlowはGroovyで効率良く

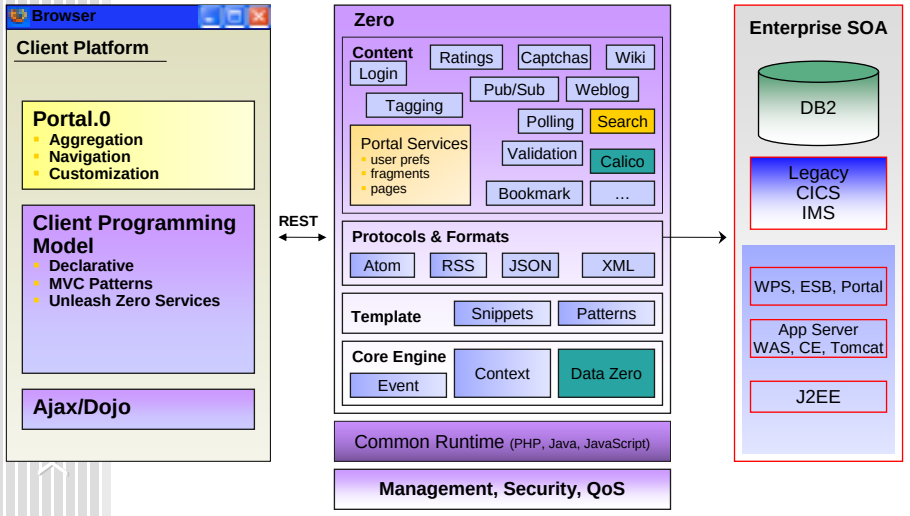
- 「構造型と手順型」
  - XMLは構造を表すのに適している
    - ツール化しやすいが、複雑な制御は苦手
  - プログラムは手順を表すのに適している
    - 制御構造はリッチだが、ツール化は困難

モジュラー・アーキテクチャー



Apache Ivy project (**依存性管理、リポジトリー管理**)

Web SOA + Enterprise SOA



SOAとWeb2.0比較

SOA風設計

OrderService
+getAllOrders()
+updateOrderList()
+addNewOrder()
+getOrderDetail()
+isOrderValid()
:

CustomerService
+getCustomers()
+getCustomer()
+addCustomer()
+updateCustomer()
+deleteCustomer()
:

多数のメソッドと大きな汎用資源

Web2.0風設計

<<Interface>>
Resource
GET
PUT
POST
DELETE

4種のメソッドと多数の資源(URI)

/orders/
GET – list all orders
PUT – N/A
POST – add a new order
DELETE – cancel all orders

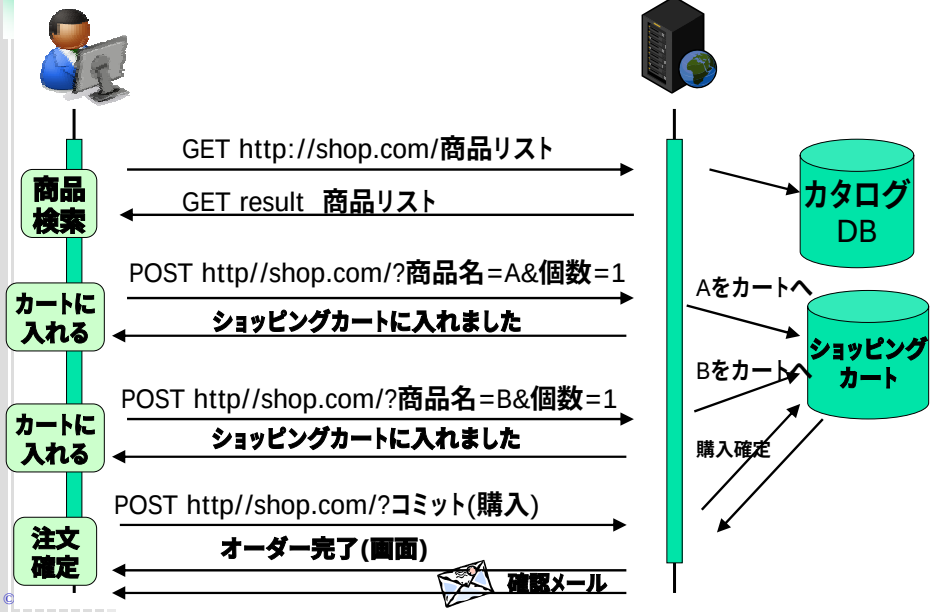
/orders/{id}
GET – get order detail
PUT – update order
POST – add item
DELETE – cancel order

/customers/
GET – list all customers
PUT – N/A
POST – add a new customer
DELETE – delete all customers

/customers/{id}
GET – get all orders fro customer
PUT – update customer
POST – add order
DELETE – cancel all customer orders

/customers/{id}/orders/
GET – get all orders for customer
PUT – N/A
POST – add order
DELETE – cancel all customer orders

REST的使い方ユースケース



REST的な背景

- Client-Server：2点間通信
- HTTPSで十分なセキュリティ
- その場の使い捨てのJSON、スキーマ定義不要
- Schema validate不要
- 短時間 (1分) 以内で完了
- 参照中心、更新処理は単純な処理

インピーダンスミスマッチ

- オブジェクト指向 ←→Database
  - O/R Mapper
- Web指向　←→ SOA(エンタープライズ)
  - ProjectZero
    - JSON ←→ SOAP, Business Object
    - HTTP ←→ JMS など
    - REST ←→ Business ObjectのVerb(動詞)
    - Web開発(Yahoo pipesの様なもの) ←→Eclipse
    - Wiki ←→ マニュアル
    - Etc...

まとめ

Web2.0 ←→ SOA

Web	ProjectZero	Enterprise
▪ HTTP ▪JSON	▪両方の架け橋	▪ HTTP JMS MQ FTP ▪XML (Business Object)
▪ブログ　コミュニティー ▪Wiki	▪www.projectzero.org	▪マニュアル　スペック ▪ヘルプデスク
▪REST(ステートレス) ▪参照系	▪両方採用	▪ステートフル ▪更新系
▪試行錯誤 (beta)	▪ProjectZero + ▪sMash (商用)	▪SI企業が保証
▪HTTPS ▪Same origin policy	▪両方採用	▪WS-Security ▪ESB
▪みんなで開発	▪CDCD : コードはオープン 開発は社内エンジニアのみ	▪企業内エンジニアが開発
▪単純、実行時決定	▪Radical simplification	▪複雑、設計時決定
▪Ruby,PHPなど多様	▪Groovy + PHP	▪Java
▪Web上(Wiki上)	▪両方採用	▪Eclipse
▪Gems, CPAN, PEAR	▪Ivy で依存性解決	▪最初にすべてinstall
▪DRY CoC	▪DRY CoC	▪作り込み

- ご静聴ありがとうございました。