レバレッジ指向アーキテクチャ

多言語・マルチパラダイム時代のシステム構築

2007年10月5日

NETCITYS株式会社
太田　渉

アジェンダ

- 今回の趣旨と目的
- Javaの良い点、悪い点
- Java環境の開発ライフサイクルサポート
- ソフトウェアの品質って？
- Webフロントエンド
- 参考アーキテクチャ
- デモ

今回の趣旨と目的

- レバレッジ　wikiペディア調べ
  - 【名】目的を達成するための力、行動力、影響力、勢力
  - 【他動】利用する、活用する
- トレードオフ　wikiペディア調べ
  - 一方を追求すれば他方を犠牲にせざるを得ないという二律背反の状態・関係のことである
  - トレードオフのある状況では具体的な選択肢の長所と短所をすべて考慮したうえで決定を行うことが求められる
- システム開発に置き換えると
  - 銀の弾丸は無いよ
  - 5つの世界＋α　http://japanese.joelonsoftware.com/Articles/FiveWorlds.html
    - コンテキストが違うと答えも異なる
- 道具は適材適所で使い分けても良いんじゃ？
  - XX陣営vsXX陣営
  - ひとつの言語　http://capsctrl.que.jp/kdmsnr/wiki/bliki/?OneLanguage

レバレッジ指向アーキテクチャ

- レバレッジ指向アーキテクチャとは
  - トレードオフを考慮しつつ
  - 最短で目的を達成するために
  - 適切な戦略（見積、契約）と戦術（開発方法）を
  - 組み合わせて導き出す

  上記の取り組み方やシステムアーキテクチャを表すために勝手に命名した造語です

Javaの良い点、悪い点

- 良い点
  - 静的型付のコンパイル言語
    - コンパイル時の厳密な型チェック
    - 手厚いツールサポート
  - 膨大な既存資産
    - 標準に加えOSSフレームワークやライブラリ（JDBC）
    - 無償で使える開発、実行環境とミドルウェア
  - 基幹系でも実績を積んだJVM
    - 仮想マシンモデルで過去にこれ程叩かれた環境は無い
    - Sun以外にもIBM、BEA、Gnu等選択肢も多い
  - マルチプラットフォーム＆オープンソース
    - ベンダーロックインの回避
    - 今後も消える事はまずなさそう

Javaの良い点、悪い点

- 悪い点
  - 互換性の呪縛
    - 設計ミスのAPI　Date、Calendar・BigDecimal等
    - 表面上のジェネリック等
  - コンパイル言語
    - 軽微な変更でもリコンパイルが必要
    - メタプログラミングが不可能もしくは難易度が高い
  - 組合せの多さと決定打の欠如
    - J2EEの失敗! JavaEEはどうか？
    - 数えきれない程のフレームワーク、ライブラリ、規格
    - UI系ライブラリとツールサポート環境の貧弱さ

    Seasar Project もそう見られてるかも知れないけど

私がJavaを避けるケース

- 不向きだと思われるケース
  - GUI系アプリケーション
    - Windows限定であれば.NETの方が楽です
  - プロトタイプ、使い捨てアプリケーション
    - RAD+スクリプト系の方が楽でしょう
  - ツール系
    - テキスト操作系はJavaは得意ではありません
    - MSOffice連携なんかも不向きです
  - 帳票系
    - ランタイムライセンス不要で安価なツールがありません
  - 外部デバイス連携
    - 一般的なシリアル、USB機器のドライバがありません

開発ライフサイクルサポート

- 各フェーズのサポート　無償でどんだけー！
  - 設計
    - UML　eclipse、JDeveloper、NetBeans
  - 実装
    - IDE　コード補完、コードナビゲーション、リファクタリング
  - テスト
    - ユニットテスト　JUnit、TestNG
    - テストカバレッジ DjUnit、Cobertura、EMMA、jcoverage
  - 品質検証
    - 規約、静的解析　　Checkstyle、FindBugs、PMD
    - メトリクス　　Eclipse Metrics、CAP、JDepend
  - ビルド、インテグレーション
    - ANT、maven2-Continuum、CruiseControl、hudson
  - モニタリング、プロファイリング
    - JConsole、MC4J、Netbeans Profiler、eclipse profiler

ソフトウェアの品質って？

- 公式定義＋α
  - 品質特性 ISO/IEC 9126（JIS X0129）
    - 機能性、信頼性、使用性、効率性、保守性、移植性
  - テスト容易性も付け加えて考える
  - 保守性(開発含む)とソフトウェアメトリクス
    - ソフトウェアメトリクス
      - 凝集度(コヒージョン)と結合度(カップリング)を数値化したもの

経理担当者、経営者のBS/PLと同じで
プログラマ、マネジャーは理解するべき
ほんとは顧客も知っててほしい

ソフトウェアの品質って？

- 規約・静的解析
  - ネーミングルール、コーディングスタイル
    - コミュニティで公開されている代表的なドキュメント
      - 株式会社電通国際情報サービス版 Javaコーディング規約
      - オブジェクト倶楽部版 Javaコーディング規約
    - チェックツール
      - IDEオプション、Checkstyle
  - 静的解析
    - FindBugs、PMD
      - コンパイルチェックは通っているが、言語仕様の理解不足や間違いやすい論理エラーを検証するツール
        メッセージがローカライズされているFindBugsが導入しやすい

        上位プログラマのコードレビューは静的解析後に行う方が効率が良い

ソフトウェアの品質って？

FindBugsデモ

ソフトウェアの品質って？

- テスト
  - 単体（Unit）テスト
    - クラスのメソッドやプロパティに対する機能テスト
      単体テストのツールとしてはJunit、TestNG等がある
  - テストカバレッジ
    - 単体テストなどホワイトボックステストに分類されるテストの網羅性を示すツールとしてはDjUnit、Cobertura、EMMA、jcoverage等がある
      国産のプラグインであるDjUnitが導入しやすい
    - テストカバレッジは正しく使えば有効な物差しとなるが、100%のカバレッジと不具合0件はイコールでは無いため注意する

      参考となるリソースとしては以下がお勧め
      IBM developerWorks コード品質を追求する: カバレッジ・レポートに騙されないために

ソフトウェアの品質って？

DjUnitデモ

ソフトウェアの品質って？

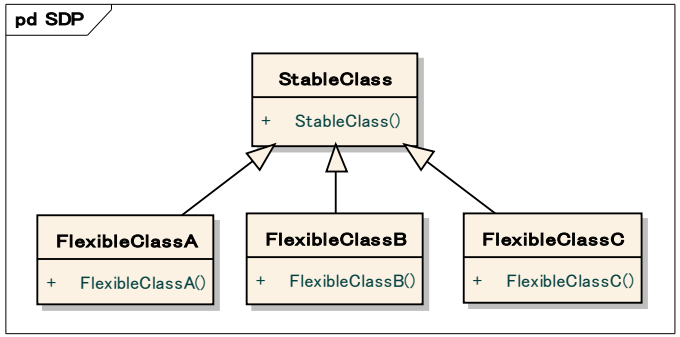
- 凝集度（コヒージョン）
  - プログラムのひとつのコンポーネントの中に含まれる機能の純粋度を表す尺度、コンスタンチンとヨードンが「構造化設計」において提案した、現在ではオブジェクト指向原則にも同概念が継承されている。オブジェクト指向原則は、RobertCecilMartin, BertrandMeyer, BarbaraLiskovを含む様々な人達により考え出され、RobertCecilMartinが纏めたもの。オブジェクト指向を用いて設計を行う際に有用となる概念集。
  - 関数レベル
    - 1つの関数は一つの機能しか実行しない
      - 関数名が機能を表していなかったり、XxxAndYyyとかXxxOrYyyなどの関数名は怪しい
  - クラスレベル
    - 責務単一原則　The SingleResponsibilityPrinciple(SRP)
  - パッケージレベル
    - 再利用開放等価原則　The ReuseReleaseEquivalencePrinciple(REP)
      - 再利用できる最小単位はリリースできる最小単位
    - 共通閉鎖原則　The CommonClosurePrinciple(CCP)
      - あるクラスの変更により、同時に変更されるクラス
    - 共通再利用原則　The CommonReusePrinciple(CRP)
      - 再利用できるのはクラスではなく一連のグループ

ソフトウェアの品質って？

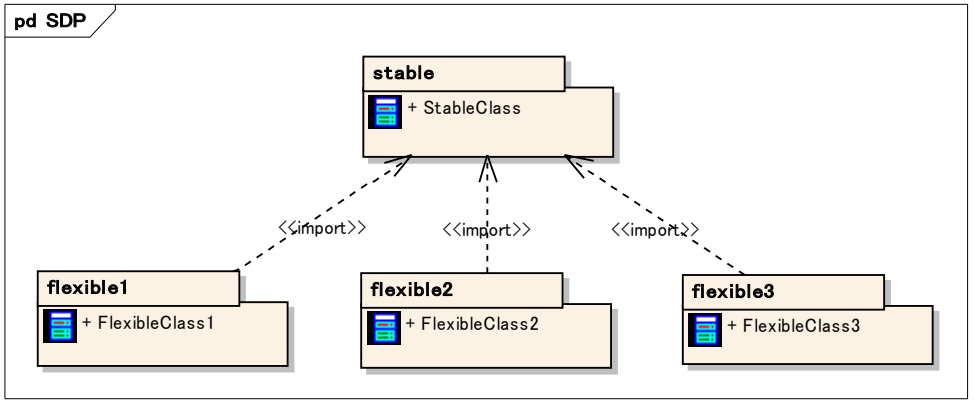
- 結合度(カップリング)
  - プログラムの中で呼びだし関係にある2つのコンポーネントの関係を表す尺度で、保守に伴う修正作業による副作用などを未然に防ぐために、情報の受け渡しが出来るだけ「疏結合」の状態を目指します。
  - 結合度
    - １:非直接結合　　何も関係ない(分析において使用する)
    - ２:データ結合　　データの「要素」よってのみやり取り
    - ３:スタンプ結合　　構造体やパラメータクラスを渡すが一部しか使わない
    - ４:制御結合　　制御フラグを渡す
    - ５:ハイブリッド結合　制御コードの数値などを渡す・返す
    - ６:共通結合　　グローバル変数
    - ７:内容結合　　参照渡しで更新されるなど内部データ構造を露呈
  - 結合度に関するオブジェクト指向原則
    - 安定依存原則　　The StableDependenciesPrinciple(SDP)
    - 非環式依存原則　The AcyclicDependenciesPrinciple(ADP)
    - 安定抽象概念原則　The StableAbstractionsPrinciple(SAP)

安定依存原則(SDP)

- 安定性-変更を加える際の影響範囲が多い状態
  - クラスの安定度
    - 子が多い＝安定　子が少ない＝不安定

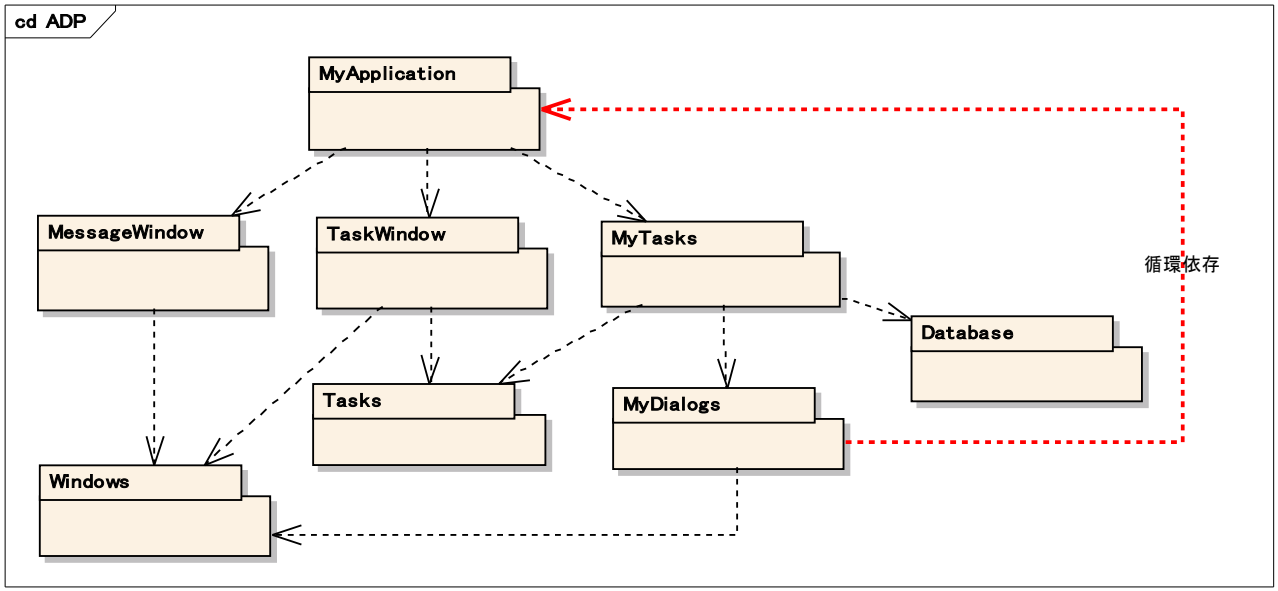


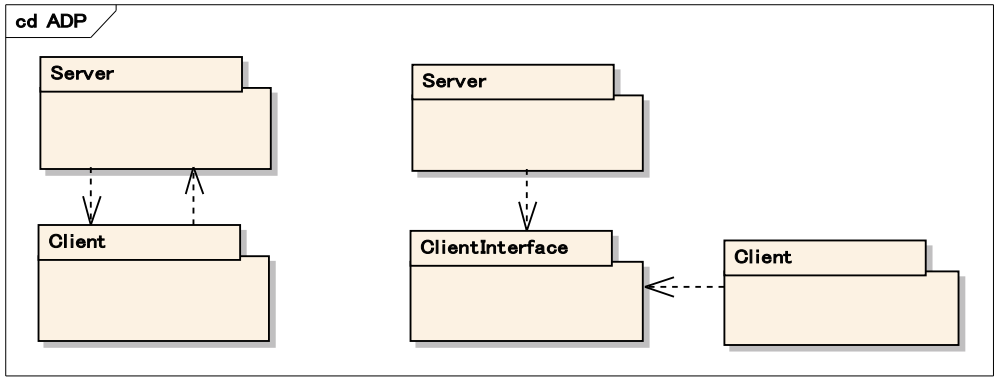
  - パッケージの安定度
    - 依存される＝安定　依存されない＝不安定



非循環式依存原則(ADP)

- 依存-Javaの場合Import、継承等





依存関係逆転の原則による変更例
此処で述べる依存がSeasar等のDIコンテナーのDI(依存性注入)がさす
Dependency

安定抽象概念原則(SAP)

- 依存する場合は具象ではなく抽象に依存する
  - 具象クラスに対してではなく、インタフェース・アブストラクトクラスに対してコーディングする

    Tips Ruby等の動的言語でDuckタイピングが有効な言語ではこの区別が無い

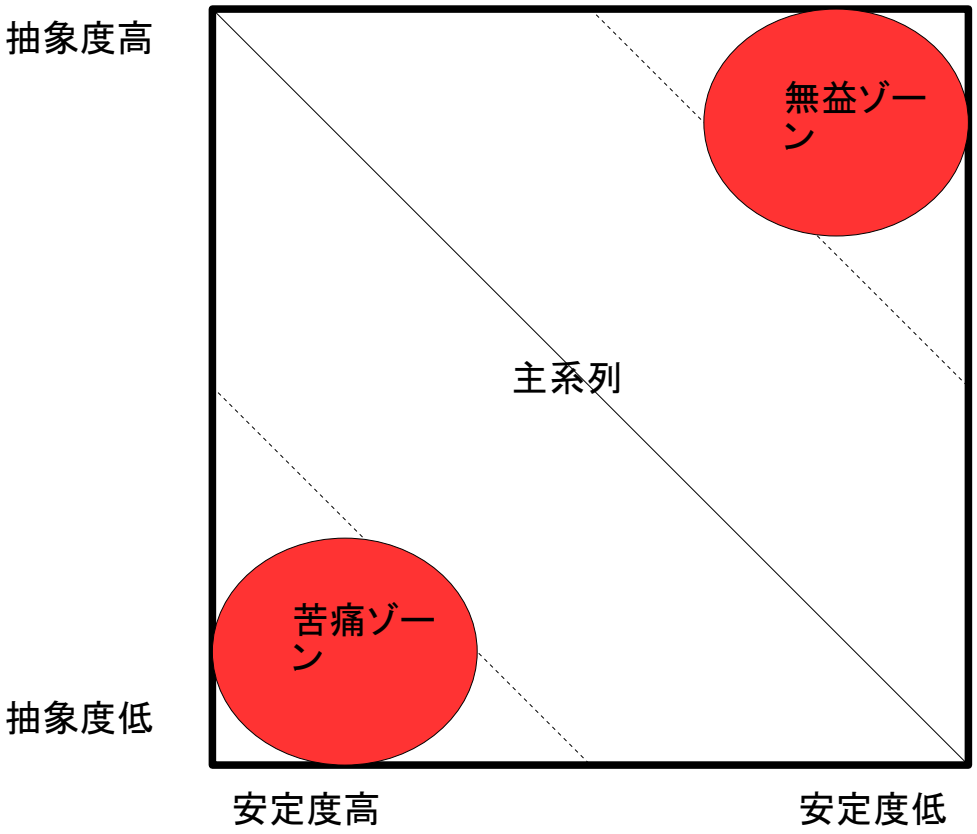
  - 安定性があるパッケージは変更が困難なため、抽象度が高くなければ拡張性が悪くなるためユーティリティパッケージなど特別な理由が無い限りは抽象度が高くなければならない

    Nc:パッケージ中のクラス数
    Na:パッケージ中の抽象クラス数、抽象クラスとは最低一つの純粋インタフェースを持ち、インスタンス化できないクラスである

    A「抽象度」=(Na / Nc)

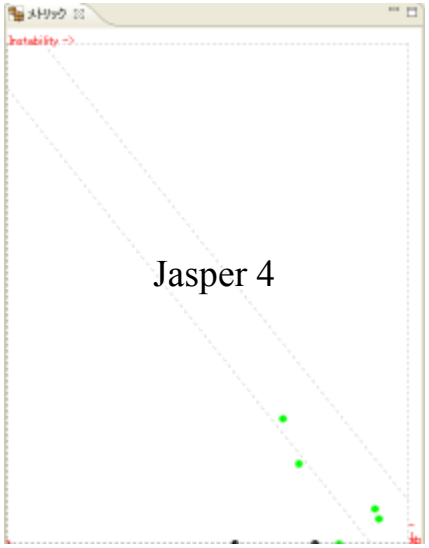
安定抽象概念原則(SAP)

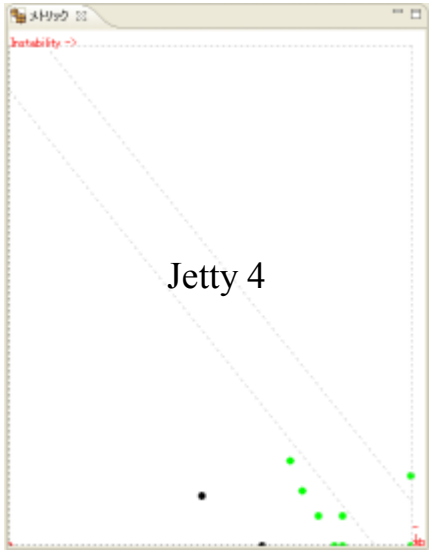
- 抽象度と安定度をテロップした図

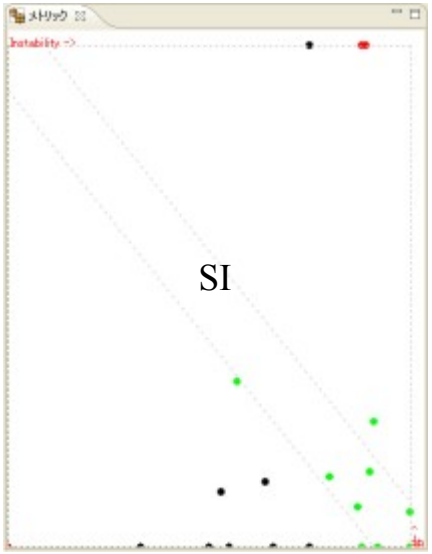


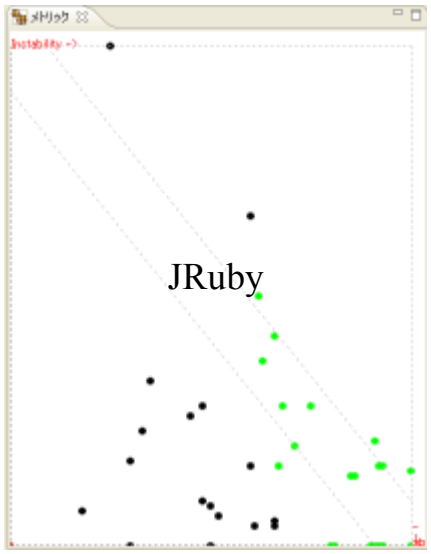
JDependデモ

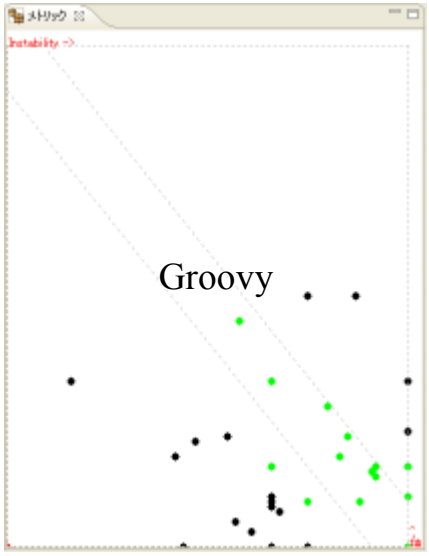
JDependメトリクス





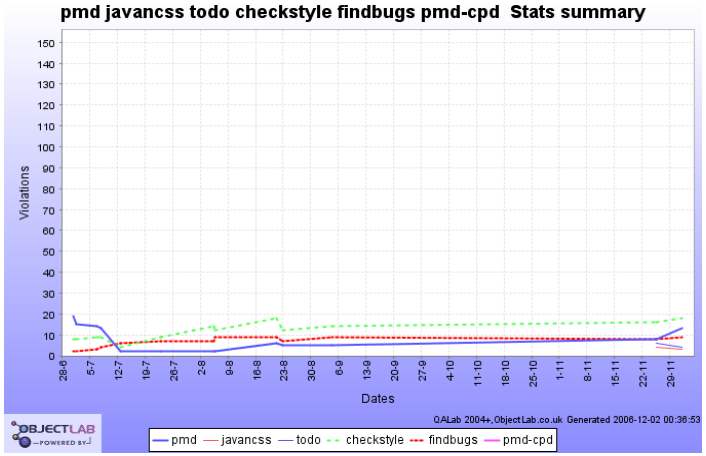






ソフトウェアの品質って？

- 開発ライフサイクルへ継続的に取り入れる
  - 開発ライフサイクルでは常に変化が起こっているため継続的なチェックが必要
  - 単独のツールを統合するのは結構大変
  - 有償ツールは高価過ぎて導入出来ない
  - そこでQALab＋Limy Eclipse Plugin
    Checkstyle、PMD、findbugs、JUnit、Cobertura、JavaNCSS等各種品質検証ツールを統合し、時系列でのレポートを採取できる



ソフトウェアの品質って？

QALab＋Limy Eclipse Pluginデモ

ソフトウェアの品質って？

- その他のメトリクス
  - QALab＋Limy Eclipse Pluginでは測定できない物に以下のようなメトリクスが存在する
    - クラス中の静的メソッドの数
    - クラス中のフィールド数
    - クラス中のすべてのメソッドのサイクロマチック数（VG）の和
    - 親クラスのメソッドをオーバーライドしたメソッド数
    - クラス中の静的フィールド数
    - メソッド中の最大のネスト数
    - クラス中のメソッドの凝集性欠如の度合い
    - メソッドのサイクロマチック数
    - メソッドのパラメータ数
    - クラスの持つサブクラスの数
    - 継承の深さ
  - まずは簡単に導入出来るSourceMonitorがお勧め

SourceMonitorデモ

Javaプラットフォームに対する結論

- 従来工数がかかるとされていたデータアクセスレイヤはO/Rマッピングフレームワークの普及とジェネレータ等の補助によって大きな問題ではなくなった
- 硬いと言われるJava言語仕様もDIとAOPをうまく使う事で、定型的で冗長なコードを排除可能になった
- 開発ライフサイクル全般をカバーするツールがOSSや無償で揃うためどのようなプロジェクトにおいても同一の環境で高度な作業が可能になった
- 実行・モニタリング環境もOSSや無償で揃うため制限なく利用が可能になった
- 蓄積された資産や経験は依然有効でスクリプティングへの対応も進められている

データベースをバックエンドにしたチームで開発・保守される
企業システムにおいてサーバーサイド技術の選択肢としては
レガシーと揶揄される事があるが現在もっとも有効なプラット
フォームと位置づけられる
JavaからXXXへとか言わないでJavaもXXXも使い分けましょう！

Webフロント

評価するだけでも大変なフレームワークの乱立

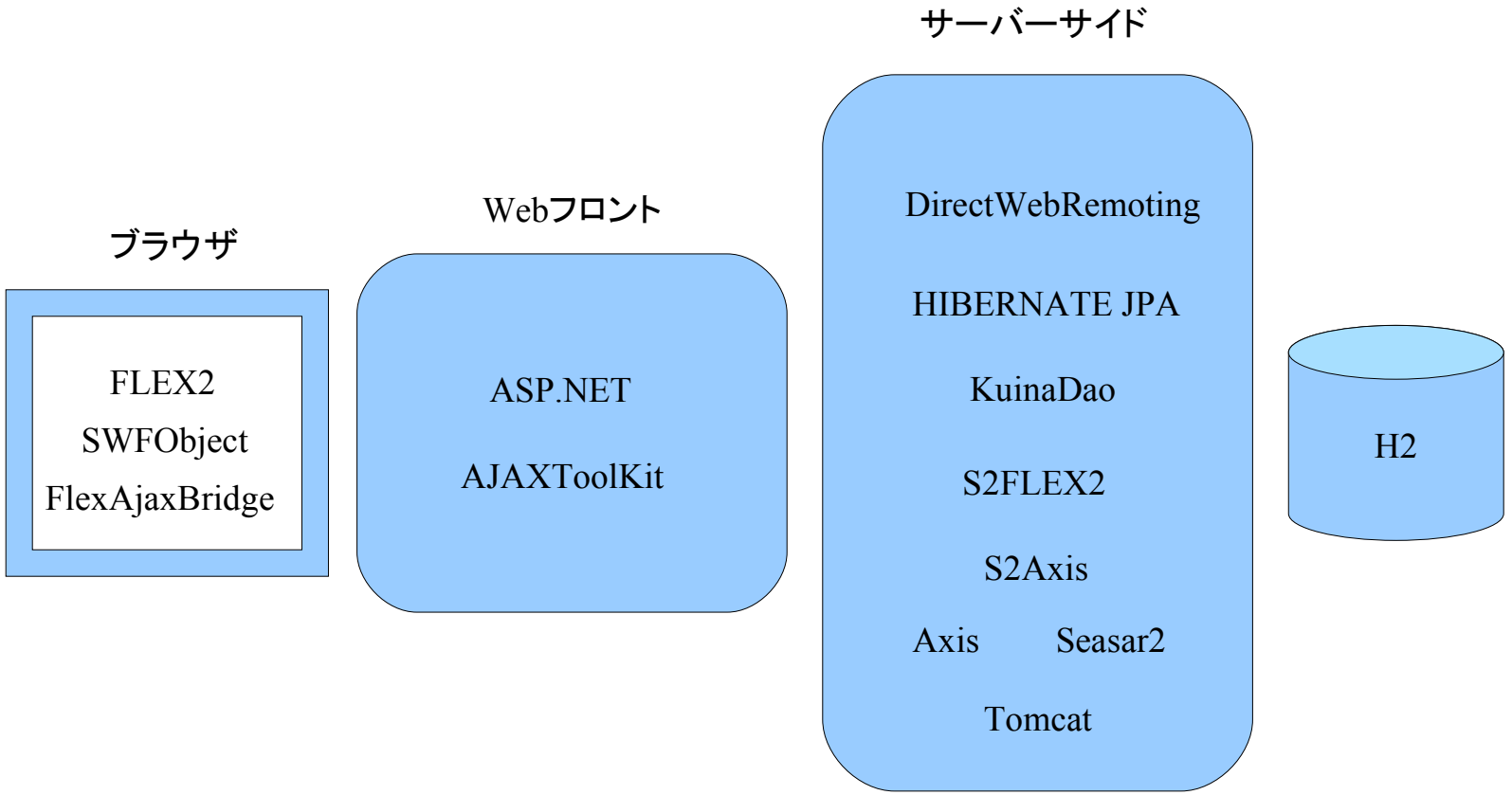
系統が似ている物は一長一短で決定打無し！！
レッドオーシャンの争い

- 現状での判断
  - 前提条件
    - 企業向けシステム(データ項目、入力項目が多い)
    - 玉石混淆チームでの開発
    - 充実した開発環境がある
    - 情報(日本語)が豊富で入手し易い
    - 生産性、保守性が高い

一般的な機能はASP.NET＋AJAXToolkitがオススメ

ブラウザの限界を突破する必要がある個所は
FLEX2(将来的にはSilverlightも)がオススメ

参考アーキテクチャ



参考アーキテクチャ

- プロダクトの選定理由
  - サーバーサイド
    - PostgreSQL、MySQL等のOSSDB
    - Javaとする事でオープンソースDBに対する親和性を確保できる
    - OSSで構成されるためライセンス費用が抑えられる
    - サービスをS2Axisで公開することでシステム構成の柔軟性とシステム寿命を確保する、WFと組み合わせればSOAプラットフォームに拡張可能
  - Webフロント
    - 非常に容易に高度なコンポーネントとAjaxを用いた画面を作成できるASP.NETをベースとする
    - 表現力が足りない部分は埋め込みFlex2で補う事でRIA的要素も無理なくこなせるようにする
  - 全てFLEX2を選択しない理由
    - ページ遷移の概念が無いのでSEO、パーマリンクがサポートしにくい
    - モジュールが増えるにつれコンパイル時間が増加する
    - 明示的にアンロード不可能なので巨大なモジュールは避けたいが、分割ロードの仕組みを作りこむのが面倒

参考アーキテクチャ

- デモ
  - TODOリストサンプル標準

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
        DeleteMethod="Delete"
        InsertMethod="Insert" OldValuesParameterFormatString="original_{0}"
        SelectMethod="GetTasksByStatus"
        UpdateMethod="Update"
       TypeName="TaskDataSetTableAdapters.TasksTableAdapter"
        OnUpdating="ObjectDataSource1_Updating">
       <DeleteParameters>
          <asp:Parameter Name="Original_TaskId" Type="Int32" />
       </DeleteParameters>
       <UpdateParameters>
          <asp:Parameter Name="Name" Type="String" />
          <asp:Parameter Name="Complete" Type="Boolean" />
          <asp:Parameter Name="Original_TaskId" Type="Int32" />
       </UpdateParameters>
       <SelectParameters>
          <asp:ControlParameter ControlID="DropDownList1" Name="IsComplete" PropertyName="SelectedValue"
             Type="Boolean" />
       </SelectParameters>
       <InsertParameters>
          <asp:Parameter Name="Name" Type="String" />
          <asp:Parameter Name="Complete" Type="Boolean" />
       </InsertParameters>
</asp:ObjectDataSource>
```

参考アーキテクチャ

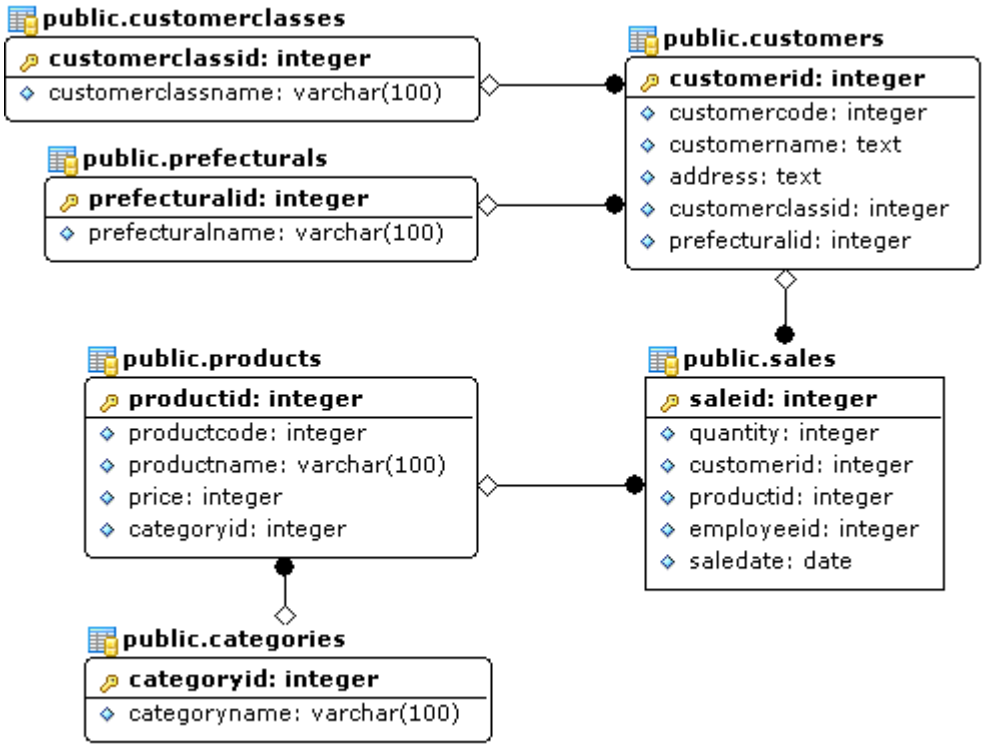
- TODOリストサンプル拡張

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
        SelectMethod="findDataTableByComplete"
        InsertMethod="persist"
        UpdateMethod="update"
        DeleteMethod="remove"
        OldValuesParameterFormatString="original_{0}"
        TypeName="ToDoList.AxisProxy.AxisGatewayService"
        OnUpdating="ObjectDataSource1_Updating"
        DataObjectTypeName="ToDoList.AxisProxy.Tasks">
        <SelectParameters>
            <asp:ControlParameter ControlID="DropDownList1" Name="complete"
                PropertyName="SelectedValue"
                Type="Boolean" DefaultValue="Complete" />
        </SelectParameters>
</asp:ObjectDataSource>
```

参考アーキテクチャ

- ASP.NET　標準実装の問題
  - TableAdapterでのDBアクセス
    - データアクセスパターンがTableModuleだけ
    - トランザクションスコープとADO.NETプロバイダ
      - » MSSQLやOracle以外ADO.NET2.0以上を正式に対応していない
    - 動的なクエリの生成が難しい
    - 標準で提供される排他制御の選択肢が少ない
    - 発行したSQLの確認が面倒
    - ユニットテストが実行しにくい

– グリッドデモデータのDBスキーマ



参考アーキテクチャ

- グリッドサンプル「GridView＋ObjectDataSource+DataTable」
  - マスタデータ等のデータ量が少ない場合は簡単に済ませる。DataTableの場合はCacheも可能

```
(partial class AxisGatewayService)
 /// <summary>
 /// GridViewの自動ソート、ページング機能は配列やコレクションでは
 /// サポートされないためWebサービスで取得したCustomer[]の値を
 /// DataTableへ変換して返す
 /// </summary>
 public DataTable getAllCustomerDataTable()
 {
     IDataExchange<Customer[], DataTable> dxo =
       new DataExchangeImpl<Customer[], DataTable>();
     Customer[] customers = this.getAllCustomers();
     return dxo.Convert(customers);
 }
```

参考アーキテクチャ

– グリッドサンプル「GridView＋ObjectDataSource+DataTable」

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server" SelectMethod="getAllCustomerDataTable"
    TypeName="web_sample.AxisGatewayReference.AxisGatewayService"
    EnableCaching="False"></asp:ObjectDataSource>
    <asp:GridView ID="GridView1" runat="server" AllowPaging="True" AllowSorting="True"
                AutoGenerateColumns="False" DataSourceID="ObjectDataSource1" CellPadding="4"
                        EnableSortingAndPagingCallbacks="True" ForeColor="#333333" GridLines="None">
    <Columns>
        <asp:BoundField DataField="customerid" HeaderText="顧客ID" SortExpression="customerid" />
        <asp:BoundField DataField="customername" HeaderText="顧客名" SortExpression="customername" />
        <asp:BoundField DataField="prefecturalname" HeaderText="都道府県"
                SortExpression="prefecturalname" />
        <asp:BoundField DataField="address" HeaderText="住所" SortExpression="address" />
    </Columns>
    <FooterStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
    <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />
    <EditRowStyle BackColor="#999999" />
    <SelectedRowStyle BackColor="#E2DED6" Font-Bold="True" ForeColor="#333333" />
    <PagerStyle BackColor="#284775" ForeColor="White" HorizontalAlign="Center" />
    <HeaderStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
    <AlternatingRowStyle BackColor="White" ForeColor="#284775" />
</asp:GridView>
```

参考アーキテクチャ

- グリッドサンプル「GridView＋ObjectDataSource+Custom」
  - データ量が多い場合はCustomPaging
    - 条件に適合するレコード数と返すメソッド
    - ページ単位のデータを返すメソッド

```
/// <summary>
/// GridViewで表示するページ毎のデータを取得する
/// 検索条件として顧客名と前方一致で指定可能
/// sortExpression, startRowIndex, maximumRowsのパラメータは
/// GridView既定の名称で暗黙で設定される
/// </summary>
public CustomerSearchDto[] findCustomersByName(
    string customerName, string sortExpression,
            int startRowIndex, int maximumRows)
{
    CustomerSearchArgDto dto = new CustomerSearchArgDto();
    dto.customerName = customerName;
    dto.orderBy = sortExpression;
    dto.offset = startRowIndex;
    dto.limit = maximumRows;
    return this.findCustomersByName(dto);
}
```

参考アーキテクチャ

```
(Daoメソッド)
public List<CustomerSearchDto> findByCustomerName(CustomerSearchArgDto dto);

(sqlファイル)
select
  c.customerid,
  c.customername,
  c.address,
  p.prefecturalname
from customers c
  inner join prefecturals p on c.prefecturalid  = p.prefecturalid
/*BEGIN*/where
  /*IF dto.customerName == null*/1=1/*END*/
  /*IF dto.customerName != null && dto.customerName != ""*/
            and customername like /*dto.customerName*/'タマ'
  /*END*/
  /*IF dto.orderBy == "customerid"*/order by customerid/*END*/
  /*IF dto.orderBy == "customerid DESC"*/order by customerid desc/*END*/
  /*IF dto.orderBy == "customername"*/order by customername/*END*/
  /*IF dto.orderBy == "customername DESC"*/order by customername desc/*END*/
  /*IF dto.orderBy == "address"*/order by address/*END*/
  /*IF dto.orderBy == "address DESC"*/order by address desc/*END*/
  /*IF dto.orderBy == "prefecturalname"*/order by prefecturalname/*END*/
  /*IF dto.orderBy == "prefecturalname DESC"*/order by prefecturalname desc/*END*/
  /*IF dto.limit != null*/limit /*dto.limit*/100/*END*/
  /*IF dto.offset != null*/offset /*dto.offset*/0/*END*/
/*END*/
```

参考アーキテクチャ

– グリッドサンプル「GridView＋ObjectDataSource+Custom」

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server" SelectMethod="findCustomersByName"
    TypeName="web_sample.AxisGatewayReference.AxisGatewayService"
    EnablePaging="True" SelectCountMethod="countCustomerByCustomerName"
    SortParameterName="sortExpression">
    <SelectParameters>
        <asp:controlparameter ControlID="txtCustName" DefaultValue=""
            Name="customerName" PropertyName="Text" Type="String" />
    </SelectParameters>
</asp:ObjectDataSource>
<asp:Label ID="lblCustName" runat="server" Text="顧客名：">></asp:Label>
<asp:TextBox ID="txtCustName" runat="server" Width="250px">></asp:TextBox>
<cc1:TextBoxWatermarkExtender ID="txtCustName_TextBoxWatermarkExtender"
    runat="server" Enabled="True" TargetControlID="txtCustName"
    WatermarkText="顧客名を前方一致で指定してください">
</cc1:TextBoxWatermarkExtender>
<asp:Button ID="btnSearch" runat="server" Text="検索実行" />
            ・・・略
</asp:GridView>
```

参考アーキテクチャ

– Flex2連携

FlexAjaxBridgeとSWFObjectを使って簡単に相互連携できます

```
<script type="text/javascript" src="bridge/FABridge.js" ></script>
<script type="text/javascript" src="js/swfobject.js"></script>
<script type="text/javascript">
var flexApp;
function loadSalesData(customerId) {
    if(!customerId) return;
            var root = FABridge.SalesChart.root();
            root.loadSalesData(customerId);
}
</script>

<div id="flashoutput">
    swfが表示されない時に、代わりに表示される内容。
</div>
<script type="text/javascript">
    var so = new SWFObject("http://localhost:8080//server-
sample/SalesChart.swf",
        "flexApp", "350", "200", "9", "#336699");
    so.addVariable("bridgeName", "SalesChart");
    so.write("flashoutput");
    if (typeof(flashoutput) != 'undefined')
        flexApp = flashoutput.firstChild;
</script>
```

参考アーキテクチャ

- 参考アーキテクチャの注意点
  - 環境構築、障害対応には前提知識と経験が必要
    - 複数のプロダクトを組合わせるため
    - 複数の言語間でのデータ型の対応関係と制限
  - リモート呼出が前提なのでサービスの粒度に注意
  - デザイナとの連携はやりにくい
  - 開発にはそれなりのPCスペックを要求
  - ASP.NETそのものはクローズドソース
  - Web層のスケールアウトにはライセンスが必要

Thank you!

ありがとうございました

参考文献・リソース

- Seasarプロジェクト
- 書籍
  - アジャイルソフトウェア開発の奥義 ロバート・C・マーチン
  - Code Complete第2版〈上、下〉—完全なプログラミングを目指して スティーブ マコネル
  - Code Quality コードリーディングによる非機能特性の識別技法 Diomidis Spinellis
  - エンタープライズ アプリケーションアーキテクチャパターン マーチン・ファウラー
- オブジェクト指向原則
- 設計におけるオブジェクトの責務分配に有効なものさし — 凝集度と結合度 —
- 初めてのソフトウェアメトリクス
- Eclipseで使えるテストツールカタログ
- 開発における「品質」をどう考えるか