Web関連技術の最新動向
フレームワーク編

株式会社ビーコンエヌシー
佐々木 健

自己紹介

- 所属
  - 株式会社ビーコンエヌシー(BeaconNC)
  - アンカーテクノロジー株式会社からデータセンター事業部が独立
  - 2008年11月～ (生まれたての会社です)
  - ウェブページはまだありませんが、データセンターは今日もちゃんと動いてます

    ※ウェブできました → http://www.benc.jp/

アジェンダ

- ウェブフレームワークとは？
- ウェブフレームワークの歴史
- 最近の話題
- その他

ウェブフレームワークとは？

- ウェブサイト
- Webアプリケーション
- Webサービス

の開発をサポートする

- 言語ライブラリ
- ツールセット
- アプリケーション

※厳密な定義は意外と困難。

具体的には？

- CGI.pm、Template-Toolkit、PEAR、Smarty、ASP.NET、JavaEE、OpenACS、Ruby on Rails、Struts、Zope、Django、Spring MVC、Tapestry、JBoss Seam、Velocity、CakePHP、PRADO、Qcodo、symfony、Zoop Framework、Zend Framework、Maypole、Catalyst、Jifty、TurboGears、Pylons、Quixote、Karrigell、web2py、Nitro、Merb、Kahua、Seaside、MODx、Drupal、Joomla、TYPO3 、Plone、MovableType、WordPress、、、、、、、、、

- **とにかく沢山ある。多種多様。**
- **今も増殖中。**
- **すべてを把握するのはまず不可能。**

- (**参考**)wikipedia
- http://en.wikipedia.org/wiki/List_of_web_application_frameworks

ウェブフレームワークが提供するもの

機能
- **データベースアクセス** (**データベース接続**, O/R**マッパー**)
- URL**マッピング**
- **表示支援** (**テンプレートシステム**, Ajax)
- **速度向上機能** (**キャッシュ**)
- **自動構成**
- **外部インターフェース** (Web**サービス**)
- **標準プロトコル対応**
- **セキュリティサポート**

効能
- **コーディング作法の統一化**
- **アーキテクチャ(モデルやインターフェースの考え方)の共通化**
- **コード量の低減**
- **テスト量の低減**
- **メンテナンスコストの低減**

ウェブフレームワークの歴史 (メインストーリー 1)

- **ターニングポントは** Struts **と** Rails
  - Struts**前**(before Struts) **〜**2000**年**
  - Struts**後**(after Struts、before Rails) **〜**2004**年**
  - Rails**後** (after Rails) **〜現在**

Struts**前**(before Struts) **〜**2000**年**

- **ウェブ向けスクリプト言語**(Perl、PHP、JSP、ASP **等**)**を利用**
- **ウェブ用ライブラリ**(cgi-lib.pl、CGI.pm、**テンプレートライブラリ**)**を活用**

Apache Struts

- 2000年5月～

- **特徴**
  - MVC **アーキテクチャを元に実装**
  - **デザインとロジックの分離**
  - **美しいアーキテクチャ、統一性**
  - **必要充分な機能**

Struts**後**(after Struts、before Rails)
～2004**年**

- Struts**を真似たフレームワークを他言語で実装**

- Struts**の問題点に対する回答、改良**
  - XML**地獄への対応**
  - DB**アクセスの提供**
  - **ライブラリの拡充によるフルスタック化**

Ruby on Rails

- 2004年7月～

- 特徴
  - MVC アーキテクチャ
  - DRY:Don't Repeat Yourself
  - 設定よりも規約
  - O/R マッパー(ActiveRecord)
  - フルスタック
  - DSL - Domain Specific Language(ドメイン特化言語)
  - Ajax への対応

  - XML地獄からの開放
  - 圧倒的なコード量の少なさ
  - すぐできる感
  - 便利な O/R マッパー
  - 規約重視、柔軟さに対する妥協
  - 開発環境込みの完全なセット

Rails後 (after Rails)
～現在

- **フルスタック指向**
- Rails **を真似たフレームワークを他言語で実装**
- Rails **から良いとこ取り**

- Rails **での問題点への回答**
  - **柔軟さより保守性への再評価**
  - **より高速なDBアクセスの模索**
  - **大規模サービス対応の模索**

ウェブフレームワークの歴史 (サイドストーリー)

- CMSツール、ブログツールの流行
  - Xoops 等の CMS による簡単なウェブ作成
  - Mobavle Type を始めとするブログツールの流行
  - Wiki の流行
- スクリプト層の進化
  - Ajax の登場
  - JavaScriptライブラリの進化
  - ActionScript、Shilverlight
  - JavaScript によるフレームワーク実装(Progression等)
- アーキテクチャ的に興味深い実装
  - Zope - Python のオブジェクト指向ウェブフレームワーク
  - Seaside - Smalltalk によるオブジェクト指向ウェブフレームワーク
  - Kahua - Scheme によるウェブフレームワーク実装。遅延評価。

最近の話題 1

- ユーザの趣向の二極化
  - 安定した変わらないフレームワーク v.s. 進化の早いフレームワーク
  - 2008/7/30 に行なわれた LL Future にて、どちらのフレームワークを求めるか、とアンケートをとったところ、ほぼ同数
  - 二極化ではなく多極化？
- フルスタック型 v.s. ライブラリ型
  - モジュールを組み合わせてフルスタックにするものもある。
- アーキテクチャ、デザインパターン
  - ビジネスロジックの場所をどこに置くか？
  - MVC v.s. 多層アーキテクチャ
  - DIコンテナ
- フレームワークの多種多様化
  - ベース言語の多様化
  - 言語on言語の動き
  - DSLの流行
  - オレオレフレームワークのオープン化

最近の話題 2

- フレームワークの思想の移転
  - 規約ベースコンフィギュレーション
  - O/Rマッパー、データベースアクセス手法
  - アーキテクチャ
- フレームワークの相互作用による強化
  - 他のフレームワークにある機能はとりあえず実装
  - 開発プロセスまで取りこむ
- アプリケーション内言語のフレームワーク化
  - 最近の、CMSやブログはテンプレート言語で高度なプログラミングが可能
    - MovableType 等
  - SaaS(ASP)として提供されたアプリケーション内の言語でも高度なプログラミングが可能
    - Salesforce APEX

最近の話題 3

- クラウドサービスへの対応
  - Google Apps
  - Salesforce
  - Amazon EC2 & S3
- セマンティックウェブに向けて
  - W3C標準
  - RESTful
- インターネットをまたぐアーキテクチャへの対応
  - OpenID
  - Gadget
- 端末の多様化
  - 携帯対応
  - ブラウザの多様化
- 強力なJavaScript
  - JavaScriptでサーバサイドも実装

Appendix 1

フレームワークの負の側面

- フレームワークを使う ＝ 新しい技術を導入する
  - フレームワークを使えばスキルが低くても大丈夫というのは嘘
- 導入時の問題
  - インストールでハマる
  - そもそも使い方を覚えられない
  - フレームワークのバグにハマる
  - 機能、パフォーマンスの不足
- 運用上の問題
  - フレームワークのバグ
  - 使っていたフレームワークが保守されなくなった
  - バージョンアップによるフレームワークの仕様が変わる
  - OSバージョンアップによりライブラリが変更されて動かなくなる
  - 一部のライブラリのバージョンを上げたら挙動が変わる

フレームワークは導入時にきちんと評価を行ないましょう

Appendix 2

フレームワークの選び方

- ステップ0
  - そもそもフレームワークが必要かどうか良く考える
  - 今の世の中、スクラッチ開発をしなくても大抵のものは利用しやすい形で存在する
  - Google Apps とか便利なものを知らないだけでは？
  - でも、なかったらやっぱり作るしかない→ステップ1へ
- ステップ1
  - システムを運用する期間中、ちゃんと保守できるものを選定すること
  - 保守できる人、開発できる人を確保できるものを選定すること
  - ある程度流行っているものに絞られるはず
  - 知名度がある程度あって衰退していないものを選ぶ。(Google Trends便利)
- ステップ2
  - ちゃんと評価はしよう。中身ぐらいは見てみよう。

Appendix 3
フレームワークの勉強方法、捉え方

- 勉強に楽な方法はない
  - フレームワークは定型処理を楽に行なうためのもの。
  - 定型処理の中身は知っておかないといけない。
  - フレームワークの巨大化＆ウェブ技術の広がり⇒覚えるべきことは沢山。
- 最低限知っておかなければいけないこと
  - 今的なウェブの通信の仕組み。⇒HTTP、セッション、Webサービス等
  - 今的なウェブコンテンツの構成要素。⇒HTML、CSS、JavaScript等
  - 今的なデータベースの知識⇒RDBの考え方、SQL等
  - 今的なプログラミング知識⇒オブジェクト思考、デザインパターン、開発プロセス
- フレームワークをどう捉えるか
  - 思想、アーキテクチャはどうなっているか？
  - テンプレートシステムはどうなっているか？
  - データオブジェクトをどう定義しているか？
  - 通信に対するサポートはどうなっているか？
  - 設定ファイルはどうなってるか？

Appendix 4
お勧めフレームワーク(勉強したい方向け)

- Ruby on Rails → http://www.rubyonrails.org/
  - 今的なフレームワークがどういうものか良く分かる。
  - 万能じゃないこと、長期運用に向かないこと、を良く理解して使うこと。
  - ついでに Ruby の勉強もできる。
- Django → http://www.djangoproject.com/
  - 世界のLLの主流はPythonっぽい。
  - Google App Engine もDjangoベース。
- SAStruts → http://sastruts.seasar.org/
  - Javaベース、手堅く、わかりやすく、長期サポートもある。
  - 先進を不要、必要なものを実直に作りたい人向け。
  - 作ってるのが日本人(ひがやすを氏)。
- Kahua → http://www.kahua.org/
  - Scheme(Gauche)ベース。
  - 変なものが好きな人向け。
  - 作ってるのが日本人。