mod_rubyによる Apacheの拡張

日本Rubyの会 前田 修吾

本日のテーマ

- mod_rubyによるApacheの拡張
  - なのですが...その前に

スライドどうする?

- MagicPoint
  - 書き方忘れちゃったよ...
- Rabbit
  - RDで書ける!
  - でも使い方を覚える時間が...
- OpenOffice
  - よく出来てるよなあ
  - しかし、テキストエディタで書けない...
  - 何とかならないものか

OpenOfficeのファイル形式

- 拡張子は.sxiだが、どうもZIP形式のようだ
- 中身を見ると、XMLがいくつか入ってるみたい
  - というわけで...

RDをOpenOfficeのデータに変換しよう

- rd2sxiの概要
- RDとは?
- RDの処理系
- RDtoolの仕組み
- RD2SXIVisitorの仕組み
- サポートする要素

rd2sxiの概要

- RDをOpenOffice(Impress)のプレゼンテーションデータに変換
- コマンドラインツール
- もちろんRubyで実装

RDとは?

- プレーンテキストのようにシンプルに記述できるドキュメント形式
- HikiやRWikiなどのWikiや、tDiaryなどの日記ツールで広く利用されている
- もともと、Rubyのドキュメントを書くために設計された
- Rubyの日本語版リファレンスマニュアルはRD(RWiki)で書かれている

RDの処理系

- RDtool
  - 広く利用されている処理系
  - ライブラリとして利用可能
  - 拡張性が高い
- matz版
  - 幻の処理系
  - 今のRDとは若干仕様が異なる
    - タブではじまる行は無条件に整形済テキストとして扱う
    - HTMLタグが書ける
- 今回はRDtoolを採用

RDtoolの仕組み

- RDの解析はRDParserが行う
- 変換対象(HTMLとかLaTeXとか)ごとにVisitorを定義
- VisitorパターンでRDParserが解析したツリーを走査し、変換処理を行う
- つまり、RD2SXIVisitorを作ればいい

RD2SXIVisitorの仕組み

- apply_to_(RDの要素名)のようなメソッドをたくさん用意し、各要素をOpenOffice形式に変換
- RDを解析したツリーに対し、apply_to_XXXXを再帰的に適用
- ツリー全体にapply_to_XXXXが適用されれば、OpenOfficeデータの完成

サポートする要素

- 見出し
- 箇条書き
- 番号付き箇条書き
- 整形済みテキスト
- 強調
- リンク
- 画像

見出し

- 見出しは以下のように記述する

  = 見出し1

  ...

  = 見出し2

  ...
- 各見出しがページの区切りになる

箇条書き

- 箇条書きは以下のように記述する

  * 項目1
  * 項目2
  * 項目3

- 結果はこんな感じ
  - 項目1
  - 項目2
  - 項目3

数字付き箇条書き

- 箇条書きは以下のように記述する

  (1) 項目1
  (2) 項目2
  (3) 項目3

- 結果はこんな感じ

  1)項目1
  2)項目2
  3)項目3

整形済みテキスト

- 字下げされたテキストは整形済みテキストとしてそのまま出力

```
def hello
  puts "Hello World!"
end
```

- `*'などの特殊ではじまる行は整形済テキストとして扱われない
  - 全角の空白などでごまかす必要がある
  - RD自体の問題なのでしかたない

強調

- 強調したい部分は以下のように記述する

  ((*ここは強調*))
- 結果はこんな感じ
  - ここは強調

リンク

- OpenOfficeでリンクが使えるようなのでサポートしてみた

  ((<URL:http://www.ruby-lang.org/>))

  ((<日本Rubyの会|URL:http://jp.rubyist.net/>))

- 結果はこんな感じ
  - <URL:http://www.ruby-lang.org/>
  - 日本Rubyの会

画像

- 賢明なるみなさまはすでにお気付きの通り、画像はサポートしてません
- RDにはもともと画像を埋め込む機能なんてない
- 大人は「だったらOpenOfficeじゃなくていいじゃん」なんて言わない
- 大人は「リンクの方がよっぽどいらないじゃん」なんて言わない

おわりに

- RDはやっぱり書きやすい
- OpenOfficeはすばらしい
- ご静聴ありがとうございました

ではなくて

- ここからが本題

mod_ruby

- mod_rubyとは?
- mod_rubyの長所
- mod_rubyの短所
- 他の環境との比較
- mod_ruby用フレームワーク
- mod_rubyによるApacheの拡張

mod_rubyとは?

- ApacheにRubyインタプリタを組み込むためのモジュール
- Webアプリケーションの実行環境
- 作者は私

mod_rubyの長所

- CGIのようにリクエスト毎にプロセスを生成しないため高速
- CGI互換のインタフェイスを持つ

mod_rubyの短所

- 複数のアプリケーションが一つのRubyインタプリタを共有するため、グローバルな状態の変更の影響が大きい
- Apacheのプロセス毎にRubyインタプリタが動くため、メモリ消費量が多い
- Apacheでしか使えない
- Rubyインタプリタの実装に強く依存しているため、Ruby本体の変更の 影響が大きい
  - まあ、これで苦労するのは私だけなんですが

他の環境との比較

- CGIとの比較
- FastCGIとの比較
- WEBrickとの比較

CGIとの比較

- CGI
  - 1リクエスト毎に1つのプロセスを起動
  - 遅い
  - 後始末が楽
  - Apache以外でも利用可能
- mod_ruby
  - 1つのプロセスで複数のプロセスを処理
  - 速い
  - 後始末には多少気を使う
  - Apache以外では利用不可

FastCGIとの比較

- FastCGI
  - アプリケーション毎に1つのプロセスを起動
  - アプリケーション全体で利用するデータをメモリに保持できる
  - Apache以外でも利用可能
- mod_ruby
  - 1つのプロセスで複数のプロセスを処理
  - アプリケーション全体で利用するデータはファイルなどに保存
  - Apache以外では利用不可

WEBrickとの比較

- WEBrick
  - アプリケーション毎に1つのプロセスを起動?
  - アプリケーション全体で利用するデータをメモリに保持できる
  - Apache以外でも利用可能(スタンドアロンのhttpd)
- mod_ruby
  - 1つのプロセスで複数のプロセスを処理
  - アプリケーション全体で利用するデータはファイルなどに保存
  - Apache以外では利用不可

mod_rubyについての結論

- 何か旗色がわるいような...
- うちのサイトそんなにアクセスないから、CGIで十分かも...
- mod_rubyステ?

mod_ruby用フレームワーク

- そのまま使うのではなく、フレームワークを使えばいい
  - テンプレートエンジンなどのプレゼンテーション層はmod_rubyで動かす
  - ロジックはFastCGIのように別プロセスで動かす
  - コンポーネントベース
  - イベントドリブン
  - 継続を使ってセッションを管理してもいいかもしれない

mod_ruby用フレームワークの問題

- まだ実装されてない
- そもそも、実装されるのか
- だれが実装するのか

mod_ruby用フレームワークについての結論

- 気が向いたら作るかもしれません
- 気が向いたらだれか作ってください

気を取り直して

- 他の環境ではできないことをやろう

mod_rubyによるApacheの拡張

- mod_rubyの機能
- いろいろなハンドラ
- サンプルによるデモ

mod_rubyの機能

- Rubyオブジェクトをハンドラとして登録できる
  - 必要なのはRubyスクリプトとApacheの設定だけ
  - ハンドラの種類毎に呼び出すメソッドが異なるため、一つのオブジェクトを複数のハンドラとして登録可能
- Apache APIをRubyから利用可能

いろいろなハンドラ

- RubyHandler
- RubyTransHandler
- RubyAccessHandler
- RubyLogHandler
  - などなど

RubyHandler

- リクエストに対する処理を行い、レスポンスを返す
- CGI互換のApache::RubyRunやeRuby用のApache::ERubyRunはこのハンドラ

```
def handler(r)
  r.content_type = "text/plain"
  r.send_http_header
  r.print("Hello World!")
  return Apache::OK
end
```

RubyTransHandler

- リクエストURIを実際のリソースにマッピング
- たとえばAliasとかDirectoryIndexとか

```
def translate_uri(r)
  r.filename = URI2FILENAME[r.uri]
  return Apache::OK
end
```

RubyAccessHandler

- アクセスの可否をチェックするためのハンドラ
- アクセスを禁止する場合、Apache::FORBIDDENを返す

```
def check_access(r)
  if /NG word/.match(r.headers_in["Referer"])
    return Apache::FORBIDDEN
  end
  return Apache::OK
end
```

RubyLogHandler

- レスポンスを返した後で呼ばれる
- ログを取るためのハンドラ
- 地味
- でも、あるものと組み合わせるとおもしろいかも
- というわけで、今回のサンプルの主役に決定

サンプル(自動サムネイル生成)

- 画像のサムネイルを作成するツールを作ろう
- 名前はThumbnailGenerator
- 画像の登録にはWebDAVを使う
  - ドラッグ･アンド･ドロップでOK
- 登録時にindex.htmlも自動生成

WebDAVとは?

- Sambaみたいなもの
- HTTPに対する拡張
- オープンな仕様
- Apache用のモジュールあります(mod_dav)
- 某社のクライアントの実装はバグバグらしい

処理の流れ

1)mod_davがWebDAVの処理を行う

2)ApacheがThumbnailGeneratorの
log_transactionメソッドを起動

3)ImageMagickを呼び出してサムネイルを生成

4)index.htmlを生成

- ほとんど他人まかせに見えるのは気のせい

プログラムの構成

- ThumbnailGeneratorクラスのみによって構成される

```
class ThumbnailGenerator
  def initialize; ... end
  def log_transaction(r); ... end
  ...
end
```

  - パブリックなメソッドはlog_transaction一つだけ

log_transaction

- 引数にApache::Requestオブジェクトを取る
- リクエストに関する情報はApache::Requestオブジェクトから取得
- リクエストに応じて処理を行う
  - PUTが成功したらサムネイルを生成
  - DELETEが成功したらサムネイルを削除
  - ほんとはMOVEとかも考慮しないといけない(今回は手抜き)

Apacheの設定

- 以下のように、使いたい場所にハンドラを登録する

```
RubyRequire thumbnail-generator
<Location /dav>
  DAV on
  RubyLogHandler ThumbnailGenerator.instance
</Location>
```

デモ

- ではデモをこちらから

応用

- ファイルの更新をメールで通知
- 自動バックアップ

まとめ

- mod_rubyを使うと、簡単にApacheを拡張できる
- Rubyはすばらしい
- Apacheはすばらしい
- mod_rubyも捨てたもんじゃないですよ

おわり

- ご静聴ありがとうございました