第3回設計勉強会
事例紹介:YakiBiki(Wiki)

2009/08/25

坂本昌彦
sakamoto-gsyc-3s@glamenv-septzen.net

自己紹介

- 7月までメーカ系受託開発のSE(Java, PHPなど)
- 9月よりWebポータルサイトの開発職
- 先週墨田区に引っ越してきたばかりです。(徒歩5km圏内だと家賃補助も出るので)

事例紹介：YakiBiki

- 2007/10月～マイペースで作っているWiki
- 既存のフレームワークは使ってない
  - 自前の状態遷移ライブラリ
- RDBも使ってない（自前のDB/INDEXライブラリ）
- SimpleTest+Stagehand_TestRunnerによるユニットテストの自動化をDB部分など中心に採用
-
- ・・・一般的(?)なWebアプリではないけど、OSSとしてWikiを作ろうとした時に悩んだり考えた事などを。

悩んだ事1:どのフレームワーク使う？

- 2007年当時、1stユーザである坂本自身が使っていたレンタルサーバはPHP4だった。
- 「まだまだPHP4の環境はある筈なので、PHP4にも対応させよう。」
  - →symfony, PieceProjectなどPHP5only系除外
- CakePHP? Ethna?
- →実際は出来レースで、その前に作っていた自前の状態遷移ライブラリが実用に耐えうるか試してみたかった為、既存のフレームワークは使わない事が最初から心の中では決まってた。

フレームワークを使わずに・・・

- 既存のFWを参考にしたディレクトリ構成で、簡単なディスパッチャ形式の作りで統一。
- ユーザやグループ、ACLなど本格的(?)な入力画面では状態遷移を管理する自前ライブラリ(Xhwlay)を使って画面遷移やvalidationの作り方を統一。
- 典型的な三層レイヤーにしている:
  - 1) ディスパッチャ(=ActionとControllerのないまぜ)
  - 2) データ処理レイヤー(=Model)
  - 3) Smartyラッパクラス(=View)

悩んだ事2:どのDB使う？

- レンタルサーバなどで動かすとなると、MySQL/PostgreSQL
- MySQL3/4/5, PostgreSQL6/7/8 に対応させる
  - →自分の実力じゃ無理。
- データ構造が落ち着くまで大分かかりそう
  - アクセス制御をどう作り込むかまるで見えてなかった
- 色々対応させようとすると、データ構造が変わる時変更箇所が多くなり、確認も大変
- →既存のRDBは使わない。

既存のRDBを使わずに・・・

- 「データ構造が定まらない」→「テーブルスキーマを決めなくてもデータを操作できるようなライブラリが欲しい」
  - → "Id-Field-Value"形式のデータを操作するライブラリ"grain"を自作。
- 検索でどのカラムを使うのかはほぼ分かり切っている(タイトル/日付/カテゴリ)
  - →INDEXデータの操作もgrainに実装。
- トランザクション的な処理で異常発生時は？
  - エラーで落としちゃう。特に回復処理などは行わない。
- むしろキャッシュ処理周りで結構手こずった。

悩んだ事3:アクセス制御どうする？

- 「検索結果の一覧処理」にもアクセス制御を的確に反映させたいのがボトルネックだった。
- ユーザを束ねるグループに対しても読み書き制御をしたい→ユーザのconflictの対処も必要。
-
- 記事データ1件毎にアクセス制御をチェックしていたのでは、検索結果の一覧取得や表示が間に合わない。
- 記事データ1件毎に大量のチェックボックスをマウスでクリックしたくない。

一ヶ月近く悶々とした末の"ACL"

- 予めユーザとグループのアクセス情報の一覧を定義しておく→"Access Control List"
- 記事データに対しては「どのACLを適用するか」だけを選択すればOKなようにしておく。
- ユーザのconflictについては優先ポリシーで回避する。
- ACLの定義時や更新時に、Read/WriteできるユーザIDとそのACLのIDを展開して、ユーザIDとRead/Write操作から対応するACLIDを取得できる逆索引を用意する。等

TDDとユニットテストについて

- PHPUnitは早い段階で却下(当時の最新がPHP5前提、PHPUnit1/2/3で変更が大きいように見えたりしたなど)
- SimpleTest + Stagehand_TestRunnerでユニットテストを自動化
- 内部ライブラリとして切り出せる範囲についてはTDDを実施→「どう使うか」というインターフェイスを先に考える事で、かなりメソッド名や引数が洗練された。
- テスト対象としてはModel層や内部ライブラリ、ユーティリティメソッドが中心。Web側は諦めた。

TDDとユニットテストの感想

- SingletonはxUnitの敵
- キャッシュはクリアできればxUnitと何とか付き合える。
- 何でもかんでもユニットテストにする必要は無い。
- 「入力」「出力」をどうシミュレートし易くするかがポイント。
- 入力/出力パターンの網羅は大変。ロジックの抜けが無いように注意するに留める。パターンの網羅でassertが多くなると、I/Fの変更時が大変。
- ACL絡みの部分はTDDして良かった！

symfonyとCakePHPも触った事がありますので、それも踏まえて。

- 「業務ロジック」処理はHTTP環境に依存させず、xUnitなどで単独で実行できるようにしておきたい。
  - "$_GET"や"$_SESSION"を直接触らせない、あるいはそれを取得するFWのメソッドを直接呼ばない、など。
- 「仮想レコード」みたいな概念を実装しておくと、複数画面を前後して渡り歩く入力画面を作りやすい。
  - 状態遷移系のFWならデフォルトで用意しているかも。
  - 最近はAjaxを連携させる場合も多いと思うので、そうした場合にも「処理中の仮想レコード」みたいなのをセッション/DB中に持っていると楽かも・・・。

独断と偏見を添えて

- 「裏側：業務系」→入力フローが複雑だったり、サブ入力フローやサブ入力画面が必要だったり。
  - →状態遷移を管理できるステートフル系のフレームワークがよさげ。Pieceとか。
- 「表側：フロント系」→複雑な入力フローは無くて、データを取得して整形して表示するだけ。
  - →非ステートフル系のフレームワーク(symfony/cake/ethnaなど)で問題ない。場合によっては、フレームワークは使わず自前のディレクトリ構造やディスパッチャで作りを統一するだけで事足りる？

まとめ

- 自前フレームワーク、自前DBライブラリでもなんとかなった。
- TDDあるいはxUnitによるユニットテストはやはり効果がある。
- 複雑な入力フローを持つ画面群を正しく遷移させたい場合は、状態遷移を管理してくれるライブラリやフレームワークを使うと便利。
-
- ご静聴ありがとうございました。