Фрэймворки веб-приложений

Кулаков Кирилл Александрович

Подходы к разработке веб-приложений

- Программно-ориентированный подход
  - веб-приложение — это реализация бизнес-логики, которая также отвечает за представление.
  - Пример (perl):

    #!/usr/bin/perl -w
  - if (length ($ENV{'QUERY_STRING'}) > 0) {
  - $buffer = $ENV{'QUERY_STRING'};
  - @pairs = split(/&/, $buffer);
  - foreach $pair (@pairs){
  - ($name, $value) = split(/=/, $pair);
  - $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
  - $query{$name} = $value; }}
  - print "Content-Type: text/html;charset=UTF-8\r\n\r\n";
  - print "<!doctype html>\n";
  - print "<html>\n";
  - print "<body>\n";
  - print "<h1>Здравствуйте</h1>\n";

Подходы к разработке веб-приложений

- Шаблонный подход
  - веб-приложение — это шаблон представления с примитивными средствами описания бизнес-логики.
  - Пример (Server Side Includes):
  -

```
<!DOCTYPE html>
<html>
    <head>
        <title>Server Side Includes Example</title>
    </head>
    <body>
        <!--#include file="menu.html"-->
        <p>
            Some content
        </p>
    </body>
</html>
```

Подходы к разработке веб-приложений

- Гибридный подход
  - примитивные средства описания бизнес-логики шаблонного подхода заменяются мощными средствами программно-ориентированного подхода.
  - Пример (php):
  - <!doctype html>
  - <html>
  - <head><meta charset="utf-8"></head>
  - <body>
  - <h1>Здравствуйте</h1>
  - Ваш IP-адрес
  - <?php
  - echo $_SERVER{'REMOTE_ADDR'};
  - ?>
  -

Фрэймворки

- Программный каркас (фрэймворк) — готовое программное решение общего назначения, на основе которого, путем расширения существующих компонентов, могут быть разработаны новые программные продукты под конкретную задачу.

Фрэймворки

- Ключевые особенности:
  - инверсия управления (inversion of control, IoC): фреймворк управляет кодом программиста, а не программист управляет фреймворком
  - поведение по умолчанию;
  - расширяемость;
  - код программного каркаса не модифицируется.

Фрэймворки для разработки веб-приложений

- Основная задача — упростить разработку за счет предоставления готовых решений рутинных и однотипных задач:
  - разбор запросов,
  - управление сессиями,
  - доступ к БД,
  - шаблонизация, и т.д.

Архитектурные подходы

- MVC:
  - Model (данные)
  - View (представление)
  - Controller (логика)
- Three-tier architecture (Трёхуровневая архитектура):
  - Presentation tier (слой интерфейса)
  - Business logic tier (слой бизнес логики)
  - Data tier (слой данных)
- XML-based (древовидная)
- ...

Типы фрэймворков

- общего назначения;
- системы управления содержанием (CMS);
- системы управления обучением (LMS);
- системы создания Интернет-магазинов;
- системы организации форумов;
- Wiki-системы;
- прочие узкоспециализированные решения.

Возможности веб-фрэймворков

- Скаффолдинг: генерация кода по спецификации
- Обработка и маршрутизация URL.
- Абстракция работы с БД, ORM, миграция БД.
- Генерация представления из шаблона.
- Средства для выполнения тестирования.
- Валидация форм.
- Средства для выполнения аутентификации и авторизации.
- Кеширование.
- AJAX.

Разнообразие фрэймворков

- Ruby: Ruby On Rails
- PHP: Yii, Kohana, Zend
- Java: Struts, Spring
- Python: Flask, Django, Pylons
- .Net: ASP.NET MVC
- Scala: Lift
- Smalltalk: Seaside
- Whatever: …

Паттерны и принципы

- MVC: отделение логики, данных и представлений
- ActiveRecord: абстракция данных
- Convention over configuration: умолчания
- DRY: «Don't Repeat Yourself» (не повторяйся)
- KISS: «Keep it short and simple» (Не усложняй код без веской причины)
- Инкапсуляция: уменьшение связности
- Принцип подстановки Барбары Лисков: Поведение класса-наследника не должно противоречить поведению, заданному классом-родителем.
- Принцип единственной ответственности: Класс должен делать что-то одно

Паттерны и принципы

- Принцип разделения интерфейса: «Клиенты не должны зависеть от методов, которые они не используют»
- Принцип инверсии зависимостей: «… Модули верхних уровней не должны зависеть от модулей нижних уровней. Оба типа модулей должны зависеть от абстракций. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций...»
- Принцип открытости/закрытости: "… Программные сущности (классы, модули, функции и т. п.) должны быть открыты для расширения, но закрыты для изменения…"
- YAGNI: «You aren't gonna need it», «Вам это не понадобится»