PHP+REST: универсальная архитектура контроллера приложения

Максим Тимохин

Маркетинговая группа Текарт

Слой контроллера

DevConf

Что мы хотим от контроллера?

- удобная абстракция HTTP-запроса и отклика;
- модульность: комбинация стандартных обработчиков запросов, приложение как набор сервисов;
- простая система правил диспетчеризации запросов;
- поддержка REST: методы HTTP, content negotiation, различные форматы представления, вложенные ресурсы;
- максимальная независимость отдельных компонент;
- работа в условиях среднестатистического хостинга без нестандартных расширений.

Абстракция HTTP-запроса и отклика

DevConf

Net.HTTP.Request: параметры запроса через индексный доступ, объектные uploads, upload в массивах полей.

```
$id   = $request['id'];
$auth = $request->headers['Authorization'];
foreach ($request['upload'] as $line) $this->parse($line);
$stored =  $request['upload']->copy_to($path);
```

Net.HTTP.Response: заголовки отклика, body как строка, итератор, файловый объект:

```
$file = IO_FS::File($path);
$response = Net_HTTP::Response()->
  status(Net_HTTP::OK)->
  content_type($file->mime_type)->
  body($file);
```

Объект класса, реализующего стандартный интерфейс, run() возвращает Net.HTTP.Response.

```
interface WS_ServiceInterface {
  public function run(WS_Environment $env);
}
```

WS.Environment:

- обмен информацией между различными частями приложения;
- можно порождать дочерние объекты, переопределяющие отдельные параметры;
- по умолчанию содержит `request` типа `Net.HTTP.Request`.

```
abstract class WS_MiddlewareService
  implements WS_ServiceInterface {

  protected $app;

  public function __construct(WS_ServiceInterface $app) {
    $this->app = $app;
  }
}
```

- запоминает следующий обработчик и вызывает его;
- может модифицировать `environment`, `request`, `response`.

Стандартные компоненты: конфигурация, подключение к БД, кеширование, сессии, авторизация, роутинг адресов.

Приложение – набор слабо связанных сервисов

Минимизируем объем подгружаемого кода и конфигурационной информации для обработки запроса:

- разделяем приложение на минимально связанные сервисы;
- все сервисы используют общее midideware;
- максимально простая диспетчеризация – по имени субдомена или подкаталогу первого уровня;
- генерация адресов – отдельным специализированным классом;
- алгоритм генерации адресов – максимально простой.

DSL описания структуры приложения

DevConf

```
static public function main() {
  WS_Runner()->run(
    WS_DSL::application()->
      status(array(404 => 'http/not-found', 500))->
      environment(array('urls' => App_URL::mapper()))->
      config('../etc/app.php')->
      orm(App::db())->
      session()->
      auth_session()->
```

```
      dispatcher(
        array(
          './'   => 'App.Front.Application',
          'blog' => 'App.Blog.Application'),
        'App.WS.Pages.Application');
}
```

Все и так знают, но на всякий случай:

- приложение = набор ресурсов;
- множество ресурсов – минимум операций: GET, POST, PUT, DELETE;
- один ресурс – много представлений;
- выбор представления по URL или HTTP-заголовкам;
- stateless протокол (не так важно в нашем случае);

Типовые паттерны: сущность

- чтение через `GET`, изменение через `PUT` и `DELETE`
- отдельные свойства ресурса могут быть представлены как вложенные ресурсы.

GET	/resource	выборка всего ресурса, параметры запроса – выборка по частям
PUT	/resource	замена всего ресурса и вложенных ресурсов
DELETE	/resource	удаление всего ресурса и вложенных ресурсов
GET	/resource/nested	выборка вложенного ресурса, параметры – выборка по частям
PUT	/resource/nested	замена всего вложенного ресурса и его вложенных ресурсов
DELETE	/resource/nested	удаление всего вложенного ресурса и его подресурсов
*	/resource/nested	вложенный ресурс может поддерживать дополнительные действия

- сущность + возможность динамически создавать и удалять вложенные ресурсы.
- после создания нового ресурса должно быть возвращено его местоположение.

GET	/resource	выборка всего ресурса, параметры запроса – выборка по частям
PUT	/resource	замена всего ресурса и вложенных ресурсов
DELETE	/resource	удаление всего ресурса и вложенных ресурсов
POST	/resource	создание вложенного ресурса
GET	/resource/nested	выборка вложенного ресурса, параметры – выборка по частям
PUT	/resource/nested	замена всего вложенного ресурса и его вложенных ресурсов
DELETE	/resource/nested	удаление всего вложенного ресурса и его подресурсов
*	/resource/nested	вложенный ресурс может поддерживать дополнительные действия

- Дополнительные действия могут быть выражены в виде отдельных вложенных ресурсов, поддерживающих операцию POST.
- Ресурсы, представляющие поведение, не могут содержать вложенных ресурсов.

GET	/resource	показ формы для ввода параметров выполняемого действия (опционально)
POST	/resource	выполнение действия

JAX-RS – основа реализации

Многие фреймворки копируют REST-контроллер RoR, мы выбрали JAX-RS. Почему?

- Простой стандарт
- Прозрачно переносит модель REST на уровень кода;
- Удобная работа с вложенными ресурсами;

Наше решение на принципах JAX RS:

- собственный DSL описания ресурсов вместо аннотаций;
- отказ от произвольного порядка определения ресурсов для упрощения алгоритма;

Ресурс – объект произвольного класса.

```
class App_Blog_Index {

  protected $env;

  public function __construct(WS_Environment $env) { $this->env = $env; }
```

HTTP-методы: обрабатывают запросы, формируют отклик.

```
  public function index() {
    return $this->
      html('index')->
        with('entries', $this->db->blog->entries->range(20));
  }
```

Сублокаторы: порождают объекты вложенных ресурсов.

```
  public function entry($id) {
    return App_Blog::Entry($this->db->blog->entries->find($id));
  }
}
```

name – имя ресурса;

classname – имя класса, реализующего ресурс;

path – путь к ресурсу в виде шаблона URL.

- не требуется наследование или реализация интерфейса;
- внутри пользовательского приложения имеет смысл наследовать чтобы избежать дублирования кода;
- аргументы конструктора подставляются автоматически.

Ресурсы: описание и реализация

DevConf

```
$service = WS_REST_DSL::Application()->
  begin_resource('blog', 'App.WS.Blog',
                 'blogs/{nickname:[a-z][a-z0-9-]+}')->
  // ... описание методов ...
  end->
end;
```

```
class App_WS_Resource {
  protected $env;
  public function __construct(WS_Environment $env) { $this->env = $env; }
}

class App_WS_Blog extends App_WS_Resource {
  $protected $blog;

  public function __construct(WS_Environment $env, $nickname) {
    parent::__construct($env);
    $this->blog = $env->db->blogs->lookup($nickname);
  }
}
```

Шаблоны URL и параметры методов

```
entries/{id:\d+}
archive/{year:\d\d\d\d}/{month:\d\d?}/{day:\d\d?}
```

- Произвольные аргументы методов, подстановка по имени;
- Если имя совпало с именем параметра шаблона, подставляется значение параметра;
- Если имя входит в базовый набор параметров, подставляется соответствующее значение.
- Неизвестное имя – подставляется null.

Базовый набор параметров:

environment текущее окружение, `Net.HTTP.Environment`;

request обрабатываемый запрос, `Net.HTTP.Request`;

format требуемый формат представления.

name – имя метода

http_mask – набор обрабатываемых http-методов;

path – шаблон URL метода;

formats – список форматов представлений.

- один метод – различные виды запросов (по `request`);
- один метод – несколько форматов представления (по `$format`);
- можно использовать стандартные имена: `index()`, `create()`, `update()`, `delete()`;
- PUT и DELETE эмулируются через POST (пока).

HTTP-методы: описание и реализация

DevConf

```
begin_resource('blog', 'App.WS.Blog', 'blogs/{nickname:[a-z][a-z0-9-]+}')-
  for_format('html')->
    get_for('{page_no:\d+}', 'index')->
    post()->
    index()->
  end->
  for_format('rss')->
    get('index_rss')->
  end->
end;
```

```
public function index($page_no = 1) {
  $entries = $this->db->entries->for_blog($this->blog);

  $pager = Data_Pagination::Pager($entries->count(), 20);

  return $this->html('index')->
    with('entries', $entries->paginate_with($pager));
}
```

Форматы представлений

DevConf

- поддерживаемые форматы указываются при описании приложения;
- если для запрошенного документа указано расширение – определяется по нему;
- если не указано – определяется по заголовкам запроса или используется формат по умолчанию;
- обработку различных форматов можно выносить в отдельные методы или объединять в один.
- форматы можно указывать для отдельных методов и целых ресурсов.

Объект контроллера для вложенного ресурса можно создавать динамически при обработке запроса.

- пустая http-маска, сублокатор не формирует отклик.
- создает экземпляр класса ресурса по данным запроса;
- для этого экземпляра поиск метода выполняется заново;
- простая работа с вложенными ресурсами для любого числа уровней вложенности.

Killer feature:

- тривиальная работа с вложенными ресурсами любого уровня вложенности;
- отсутствие привязки к жестко определенной схеме.

Сублокаторы: описание и реализация

DevConf

```
begin_resource('blog', 'App.WS.Blog', 'blogs/{nickname:[a-z][a-z0-9-]+}')->
  sublocator('entry', '{id:\d+}')->
end->
begin_resource('entry', 'App.WS.Entry', null)->
  index()->
end;
```

```
class App_WS_Blog extends App_WS_Resource {
  public function entry($id) {
    if ($entry = $this->db->entries[$id]) return App_WS::Entry($entry);
  }
}

class App_WS_Entry extends App_WS_Resource {
  protected $entry;

  public function __construct(App_DB_Entry $entry) { $this->entry = $entry; }

  public function index() {
    return $this->html('index')->with('entry', $this->entry);
  }
}
```

Как все это работает

DevConf

1. Определяем формат (расширение или заголовок запроса);
2. Просматриваем описания ресурсов и сопоставляем шаблон каждого с началом URL;
3. Если не нашли такого, что URL соответствует и формат поддерживается – ошибка;
4. Ресурс нашли, ищем метод. Убираем из URL совпавшую с путем ресурса часть;
5. Просматриваем описания методов ресурса, сопоставляя шаблон с началом остатка URL;
6. Нет http-метода или сублокатора, подходящих по шаблону, формату и т.д. – ошибка.
7. Найден http-метод с соответствующим шаблоном, форматом и маской – создаем объект ресурса и вызываем метод, получаем отклик, обработка завершена.
8. Найден сублокатор с соответствующим шаблоном URL – вызываем сублокатор.
9. Результат выполнения сублокатора – новый ресурс. Ищем в списке ресурсов описание ресурса соответствующего класса.
10. Если такого ресурса нет – ошибка;
11. Удаляем совпавшую строку из начала URL и возвращаемся в пункт 4, и так до тех пор, пока не найдется http-метод.

Количество проходов ограничено, /resource/ и /resource/index.html – эквивалентны.

DB.ORM, простое ORM-решение:

```
$blog = $db->blog->blogs[$blog_id];
$entries = $blog->entries->
            published->
            later_than($date)->
            range(20);
$entry = $entries[$id];
$entries->update($entry);
```

Маппер:

- отображение записей в таблицах и объектов домена;
- порождает дочерние мапперы с дополнительными условиями, группирует дочерние мапперы;
- производный маппер имеет доступ к родительскому;
- корневой маппер: подключение к базе и общая информация для всего дерева.

Единая точка доступа к модели

- `$env->db` – единый доступ к любому объекту модели;
- всего две сущности: маппер и объект домена;
- объект домена может вернуть маппер (отношение 1 к N);
- маппер умеет select(), insert(), update() и delete();
- порожденные мапперы можно формировать в терминах предметной области.
- ограничения доступа отрабатываются на уровне модели;
- информация об авторизованном пользователе доступна в корневом маппере, а значит и во всем дереве.

Универсальный API: REST-интерфейс

DevConf

```
$db->blog->entries->most_popular->select()
```
GET /api/blog/entries/most_popular/

```
$db->blog->entries->insert($entry);
```
POST /api/blog/entries/

```
$entry = $db->blog->entries[15];
$db->blog->entries->update($entry);
$db->blog->entries->delete($entry);
```

PUT /api/blog/entris/15/
DELETE /api/blog/entries/15/

```
$db->blog->blogs[5]->entries;
```
GET /api/blog/blogs/5/entries/

Универсальный API: описание приложения

DevConf

```
$api = WS_REST_DSL::Application()->
  for_format('json')->
    begin_resource('mapper', 'App.API.Mapper', '')->
      sublocator('spawn', '{path:[^/]*}')->
      index()->
      post()->
    end->
    begin_resource('entity', 'App.API.Entity', null)->
      sublocator('spawn', '{path:[^/]*}')->
      index()->
      put()->
      delete()->
    end->
  end->
end;
```

Универсальный API: реализация*

```
class App_API_Mapper extends App_API_Resource {
  protected $mapper;

  public function __construct(WS_Environment $env, DB_ORM_Mapper $mapper = null) {};

  public function spawn($path) {
    return (isset($this->mapper->$path)) ?
      new App_API_Mapper($this->mapper->$path) :
      new App_API_Entity($this->mapper[$path]);
  }
  public function index()  {}
  public function create() {}
}

class App_API_Entity extends App_API_Resource {
  protected $entity;

  public function __construct(WS_Environment $env, DB_ORM_Entity $entity) {
    parent::_construct($env);
    $this->entity = $entity;
  }

  public function spawn($path) {
    if (isset($this->entity->$path)) {
      $value = $this->entity->$path;
      switch (true) {
        case $value instanceof DB_ORM_Mapper:
          return new App_API_Mapper($this->env, $value);
        case $value instanceof DB_ORM_Entity:
          return new App_API_Entity($this->env, $value);
      }
    }
  }
  public function index()   {}
  public function update()  {}
  public function delete()  {}
}
```

* отсутствуют необходимые проверки и т.д., реальный код сложнее.

Универсальный API: стандартный сервис

DevConf

- реализация не зависит от конкретной предметной области;
- подключается к приложению простой привязкой к URL в диспетчере;
- можно использовать для Rich UI: тривиально интегрируется с ExtJS, достаточно легко – с Dojo;
- можно использовать как веб-сервис;

Что выигрываем от использования REST

DevConf

- универсальность: классические веб-приложения, веб-сервисы, бэкенд AJAX-приложений;
- однородная структура кода классов контроллеров;
- простая работа с вложенными ресурсами: страничный контент, каталоги товаров и т.д.;
- возможность использования отдельных классов ресурсов в виде библиотечных компонент;
- расширяемость.

Вопросы?

http://www.techart.ru/

http://github.com/techart/tao-base/