# Архитектура REST

Отдел R&D

Маркетинговая группа Текарт

15 июня 2009 г.

# Что такое REST

## **RE**presentational **S**tate **T**ransfer

- Не протокол, а паттерн проектирования;
- Принципы сформулированы в 2000 году Роем Филдингом;
- В диссертации Филдинга систематизированы принципы, реализованные ранее в виде протокола HTTP.
- WWW изначально построен на принципах REST.

# Основные принципы

Сформулируем базовые принципы архитектуры REST

# Все является ресурсом

- Приложение представляет собой набор ресурсов (источников информации);
- Каждый ресурс однозначно идентифицируется своим URI:

```
http://music.site/users/max
http://music.site/albums/8
http://music.site/max/playlist/20
```

- Коллекция ресурсов также является ресурсом:

```
http://music.site/users/max/playlist/
```

# Один ресурс – разные представления

- Каждому представлению соответствует свой URL, несколько представлений – несколько URL;
- Форматы представлений: стандартные MIME-типы, различные схемы XML, микроформаты.

```
/users/max/playlist/20.html - text/html
/users/max/playlist/index.xml - xml/atom
/users/max/playlist/index.json - application/json
```

- Использование стандартных форматов упрощает использование сервиса клиентом.

# Множество ресурсов – минимум операций

В идеале – всего 4 действия, поддерживаемые в HTTP

| GET | POST | PUT | DELETE |
|---|---|---|---|
| поиск | создание | изменение | удаление |
| SELECT | INSERT | UPDATE | DELETE |

HTTP – не транспортный протокол, а протокол уровня приложения.

# URL ресурсов и действия

- URI указывает на ресурс;
- URL указывают на представление ресурса;
- Действия не кодируются в URL!

| | **не REST** | | **REST** |
|---|---|---|---|
| POST | /albums/create | POST | /albums/ |
| GET | /albums/show/2 | GET | /albums/2 |
| POST | /albums/update/2 | PUT | /albums/2 |
| POST | /albums/destroy/2 | DELETE | /albums/2 |

# GET

Представление ресурса в определенном формате по URL.

Выбор представления:

- по расширению:

  `/albums/234.html` `/albums/234.xml` `/albums/234.json`

- по HTTP-заголовку Accept;
- в зависимости от параметров клиента.

Часто имеет смысл назначить формат по умолчанию.

# GET: модификаторы

Иногда необходимы разные представления в одном формате.

Используем модификаторы:

| | | |
|---|---|---|
| просмотр информации об альбоме | GET | /albums/23.html |
| форма редактирования альбома | GET | /albums/23/edit.html |
| форма нового альбома | GET | /albums/new.html |

Более общее решение – behaviour-ресурс (см. дальше).

Университет Текарт

# POST

Изменение части ресурса, которая недоступна как отдельный ресурс.

- Создание нового ресурса в коллекции ресурсов;

```
POST /albums/
HTTP 201 CREATED
Location: /albums/315.html - новый альбом
```

- Аналог привычного RPC: удаленный вызов метода.

```
POST /albums/206/publish.html
```

Нарушение правил, альтернатива – behaviour-ресурс.

# PUT и DELETE

Атомарное изменение состояния ресурса. **Применяется к ресурсу в целом**, а не к отдельным частям.

```
PUT /albums/23.html
HTTP 200 OK
Location: /albums/

DELETE /albums/23.html
HTTP 200 OK
Location: /albums/
```

PUT и DELETE плохо поддерживаются браузерами $\Rightarrow$ эмуляция через POST.

# Stateless протокол

- Состояние клиента хранится только на клиенте, нет серверных сессий;
- Вся информация, необходимая для обработки запроса, содержится в самом запросе.

Плюсы:

- Решение легко масштабируется;
- Кеширование, проксирование и т.д.
- Удобно для неинтерактивных сервисов.

Минусы:

- Достижимо на rich-клиенте (AJAX, FLEX ...)

# Stateless протокол: классическое web-приложение

Хранение состояния на клиенте затруднено:

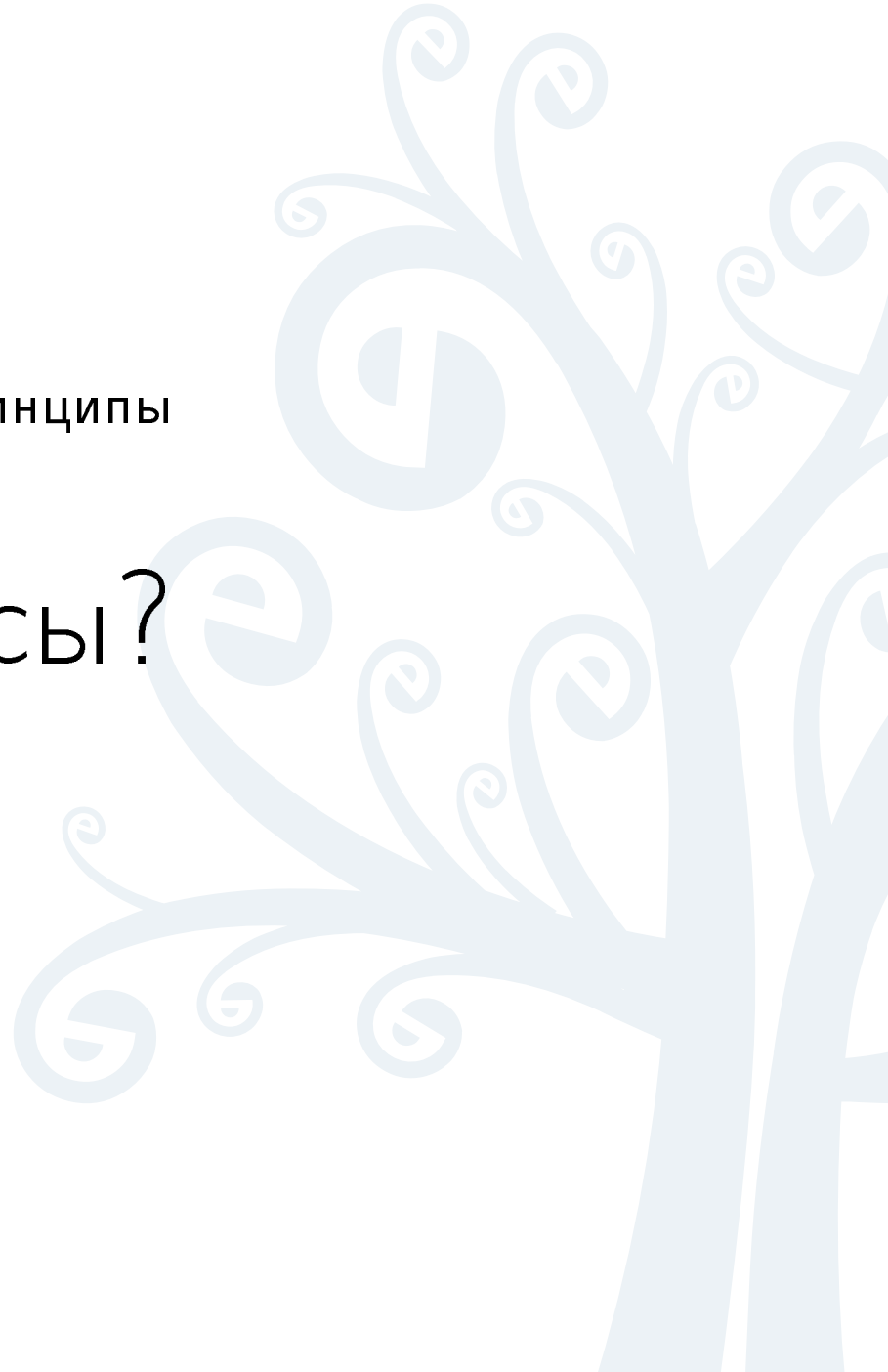
- Хранение состояния в виде постоянно передаваемого набора параметров;
- Cookies.

Если добавим сессии, разработка упростится,но это уже не REST. Тем не менее, используя остальные принципы, все равно выигрываем:

- Простая и предсказуемая структура приложения;
- Использование альтернативных представлений;
- Можно выделить набор stateless-сервисов для других видов клиентов.

Основные принципы

# Вопросы?



# Паттерны REST

Выделим основные элементы структуры REST-приложений.

# 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 |

# Контейнер

- Entity Pattern + 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 |

# Поведение

- 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 |

Паттерны REST

# Вопросы?

# Проектирование web-приложений

Внимание, проект-менеджеры!

# Преимущества REST

Что мы выигрываем, переходя к использованию принципов REST, пусть даже и частичному:

- Унифицированная структура приложений;
- Единый интерфейс – больше возможностей повторного использования кода;
- Простые адреса – больше возможностей расширения;
- Разрабатываемые приложения легче адаптируются под JavaScript-интерфейсы на стороне клиента;
- На крупных сайтах можно реализовывать API для использования извне.

# Что плохо в проектировании сейчас

Мы давно используем паттерн MVC, но стереотип "сайт – набор страниц"еще жив.

Как следствие:

- Проектирование сайта сводится прежде всего к проектированию навигации – это неправильно;
- Структура URL часто привязывается к структуре навигации – это плохо.

Правильно:

- Веб-приложение – набор функциональных сервисов, навигация вторична.

# Проектирование в терминах сервисов и ресурсов

1. Пытаемся представить приложение как набор приложений-сервисов, допускающих раздельное определение ресурсов;
2. Отдельные сервисы выносятся на отдельные поддомены или в подкаталоги первого уровня.
3. Для каждого сервиса проектируется структура приложения их ресурсов и сервисов. Адреса ресурсов определяются отношениями между ними, а не навигацией.
4. Полученная структура полностью описывает все приложение, и может быть разработана менеджером проекта в качестве техзадания для реализации.

Можно разработать стандартную нотацию для описания ресурсов, из нее можно генерировать визуальные схемы и код описания ресурсов на внутреннем DSL.

Проектирование web-приложений

# Вопросы?

# Реализация

Переходим к самому интересному

# Реализации REST

Множество реализаций, различные пути написания кода REST-приложений.

- Компоненты универсальных фреймворков: Ruby On Rails, Django, CherryPy и т.д.
- Отдельные фреймворки: Restlets, Jersey.

Компания SUN предложила стандарт для Java-приложений:

## JAX RS (JSR 311)

Мы решили взять его за основу для нашей реализации.

# JAX RS (JSR 311)

Почему мы выбрали JAX RS:

- Очень простой стандарт;
- Прозрачно переносит модель REST на уровень кода;
- Удобно реализована работа с вложенными ресурсами;
- Не накладывает ограничений на классы ресурсов.

Наше решение на принципах JAX RS:

- Собственный DSL описания ресурсов вместо аннотаций;
- Отказ от произвольного порядка определения ресурсов для упрощения алгоритма;
- Поддерживаются не все возможности – реализуем позже.

# Реализация: ресурсы

- Ресурс – просто PHP объект;
- Не обязательно наследоваться от родительского класса или реализовывать интерфейс.

```
class App_Stories_Index {
  public function index() { ... }
  public function draft() { ... }
  public function create() { ... }
  public function story($id) { return App_Stories::Story($this->db->stories->find($id));
}
```

- На практике имеет смысл выносить общую функциональность в родительский класс, но это не требование фреймворка.

# Методы класса ресурса

Класс ресурса реализует набор методов:

- Вспомогательные методы для внутреннего использования;
- HTTP-методы: методы обрабатывающие различные виды запросов;
- Сублокаторы: методы, порождающие экземпляры классов вложенных ресурсов.
- На параметры методов не накладывается никаких ограничений.
- Значения параметров подставляются автоматически исходя из имени параметра.

# Язык описания ресурсов

- Для принятия решения о вызове метода ресурса необходимо иметь полное описание доступных ресурсов и их методов;
- Реально выполняется код одного или нескольких ресурсов, нет необходимости загружать все ресурсы;
- Описание ресурсов, составляющих приложение, должно быть отделено от кода ресурсов;
- Создаем свой DSL для описания структуры приложения.
- Часто приложение можно разбить на отдельные независимые приложения – минимизация объема описания для отдельного запроса.

PHP – не самый удобный язык для написания DSL, но кое-что сделать можно.

# Описание ресурсов приложения

Приложение показывает новостные статьи и RSS-ленты для новостей, принадлежащих к различным рубрикам.

```
WS_REST_DSL::Application($this)->
  media_type('rss', 'application/xhtml+xml')->
  begin_resource('category', 'P2.WS.News.Category', '{name:[a-zA-Z-]+/[a-zA-Z]+}')->
    get_for('{page_no:\d+}', 'index')->
    get_for('', 'index_rss', 'rss')->
    get_for('top', 'top_rss', 'rss')->
    get_for('most-popular', 'most_popular_rss', 'rss')->
    index()->
  end->
  begin_resource('story', 'P2.WS.News.Story', 'stories/{id:\d+}')->
    get_for('{page_no:\d+}', 'index')->
    index()->
  end->
  begin_resource('index', 'P2.WS.News.Index')->
    get_for('most-popular', 'most_popular_rss', 'rss')->
    get_for('', 'index_rss', 'rss')->
    index('html')->
  end->
end;
```

# Описание приложения

```
resource($name, $classname, $path)
```

---

| | |
|---|---|
| `$name` | имя ресурса; |
| `$classname` | имя класса, реализующего ресурс; |
| `$path` | путь к ресурсу в виде шаблона URL. |

---

```
begin_resource('category',
               'P2.WS.News.Category',
               '{name:[a-zA-Z-]+/[a-zA-Z]+}');
```

# Шаблоны URL

- Регулярные выражения с именованными параметрами:

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

- Значения параметров автоматически подставляются при вызове методов.

```
public function story($id);

public function archive_daily($year, $month, $day);
```

# Описание методов ресурса

Метод ресурса выполняет обработку запросов различного вида и формирует отклик.

```
method($name, $http_mask, $path, $formats = 'html');
```

---

| | |
|---|---|
| `$name` | имя метода; |
| `$http_mask` | `GET \| POST \| PUT \| DELETE`; |
| `$path` | URL-шаблон для метода; |
| `$formats` | список форматов, поддерживаемых методом. |

---

# Описание методов ресурса: DSL

```
m = method($name, $http_mask, $path, $formats = 'html');
```

| | |
|---|---|
| `index($f= 'html')` | `m('index', GET, 'index', $f)` |
| `get($n, $f= 'html')` | `m($n, GET, $n, $f)` |
| `get_for($p, $n = '', $f = 'html')` | `m($n, GET, $p, $f)` |
| `post($n = '', $f = 'html')` | `m($n ? $n : 'create', POST, $n ? $n : 'index', $f)` |
| `post_for($p, $n = '', $f = 'html')` | `m($n, POST, $p, $f)` |
| `put($n = '', $f = 'html')` | `m($n, $n ? $n : 'update', PUT, $n ? $n : 'index', $f)` |
| `put_for($p, $n = '', $f = 'html')` | `m($n, PUT, $p, $f)` |
| `delete($n = '', $f = 'html')` | `m($n ? $n : 'delete', DELETE, $n ? $n : 'index', $f)` |
| `delete_for($p, $n = '', $f = 'html')` | `m($n, DELETE, $p, $f)` |

Можно более подробно:

```
$resource->
    begin_method($name)->http($http)->path($path)->produces($fmt1, $fmt2')->end;
```

Имена по умолчанию: `index`, `create`, `update`, `delete`.

# Параметры вызова методов

- Метод может иметь любой набор параметров;
- Если имя параметра совпадает с именем параметра шаблона – подставляется значение параметра;
- Если имя параметра входит в набор предопределенных имен – подставляется соответствующее значение.
- В противном случае подставляется null.

Предопределенные значения:

| | |
|---|---|
| env | окружение (запрос + данные, созданные предыдущими обработчиками); |
| request | http-запрос; |
| format | формат, для которого вызван метод. |

# Форматы представлений

- Для каждого метода можно указать список форматов представлений.
- На данный момент реализовано определение формата по расширению запрашиваемого документа.
- В процессе реализации – определение по HTTP-заголовкам.
- На каждый формат можно предусмотреть отдельный метод или обрабатывать все в одном методе, используя параметр $format.
- Можно ограничивать форматы для целых ресурсов.

---

```
get_for('most-popular', 'most_popular', 'html,rss');
```

# Сублокаторы

Создание класса, соответствующего вложенному ресурсу, может быть выполнено динамически в процессе обработки.

- Сублокатор не обрабатывает запрос;
- Вместо этого он принимает решение о создании экземпляра класса вложенного ресурса;
- Для полученного ресурса заново выполняется процесс поиска метода обработки запроса.

---

```
sublocator($name, $path)
```

name имя метода-сублокатора;

path шабон URL.

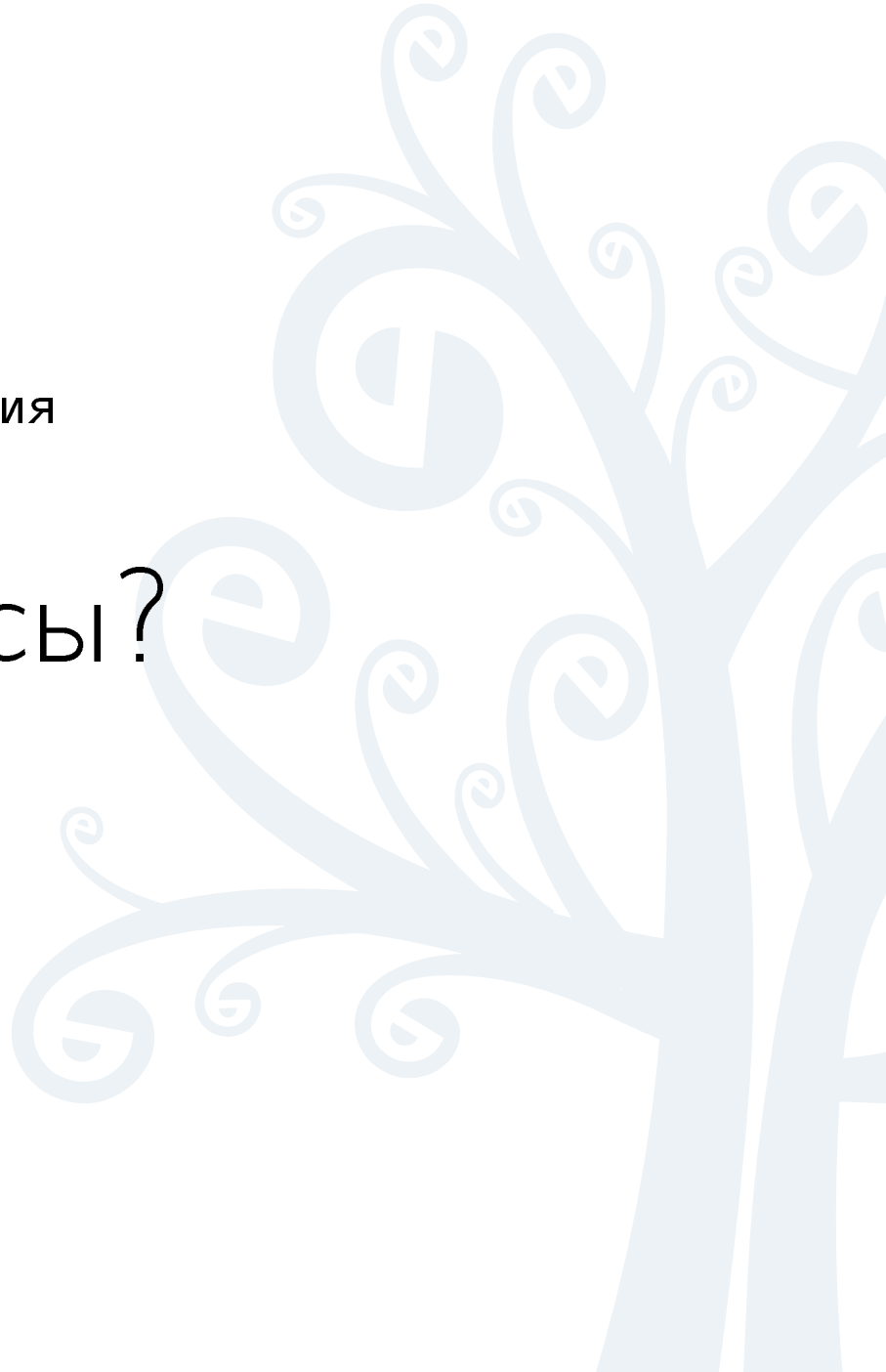
# Как все это работает

1. Определяем формат (расширение или заголовок);
2. Просматриваем описания ресурсов и сопоставляем путь 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. Просмотриваем все описания методов ресурса, сопоставляя шаблон с началом остатка URL;
6. Если нет ни http-метода, ни сублокатора, подходящих по шаблону URL, формату и т.д. – ошибка, метод не поддерживается.
7. Если найден http-метод с соответствующим шаблоном URL, форматом и маской HTTP-методов – создаем объекта класса ресурса, подставляем параметры и вызываем метод. Результат работы и есть результат выполнения. Работа завершена.
8. Если найден сублокатор с соответствующим шаблоном URL – подставляем параметры и вызываем метод.
9. Результат выполненя метода – 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.

Реализация

# Вопросы?



# Что читать

- RESTful Web Services
- Representational State Transfer
- JAX RS
- Discovering a World of Resources on Rails
- REST Wiki
- Restlet
- REST и WS
- REST search engine

# Спасибо за внимание