Иерархические мапперы или как построить уровень модели на базе одной единственной концепции

Сергей Свистунов

Маркетинговая группа Текарт

Что мы хотим от слоя модели

- сущности предметной области в виде объектов, потому что это удобно;
- построение выборок в терминах предметной области, потому что это логично;
- возможность использования SQL при необходимости, потому что это необходимо;
- поменьше магии, потому что избыток магии в слое модели – причина последующей борьбы с ней;

Две стороны объекта домена

[ ]	→
скалярные значения атрибутов	объектное представление свойств
`row_get_{property}` `row_set_{property}`	`get_{property}` `set_{property}`

- набор атрибутов доступен как свойство `attrs`;
- набор атрибутов не фиксирован ⇒ частично загруженные сущности или дополнительные атрибуты.

Базовая концепция: маппер

- наследуется от базовых абстрактных классов;
- содержит ссылку на родительский маппер;
- наследует значения свойств от родительского маппера;
- дочерние мапперы уточняют условия выборки родительских;
- маппер можно передать в качестве параметра и модифицировать дополнительно.

```
$db->news->stories->
  where('pub_date < :date', $date)->
  order_by('pub_date DESC');

$db->news->stories->
  for_category($category)->most_popular(20);
```

Минимальный набор операций:

- spawn порождение дочернего маппера своего класса;
- pmap отображение свойства в дочерний маппер;
- cmap отображение вызова в дочерний маппер.

Отображение свойств и методов – путем реализации методов с префиксами `pmap_`, `cmap_` и `map_`.

Наследование свойств: например, все мапперы в цепочке могут использовать подключение к базе и информацию о текущем пользователе из корневого маппера.

DB.ORM.MapperSet: группирующие мапперы

DevConf

- логически группирует дочерние мапперы;
- корневой маппер: группировка + объект подключения и общие свойства;
- pmap – возвращает дочерний маппер, кешируя экземпляр;
- cmap – возвращает новый экземпляр дочернего маппера.

```
class App_DB_NewsMapperSet extends DB_ORM_MapperSet {
  protected function map_stories() {
    return App_DB::StoriesMapper($this);
  }
}

$stories = $db->news->stories;
$another_stories = $db->news->stories();
```

DB.ORM.SQLMapper: SQL мапперы

- абстрактный класс
- содержит набор опций генерации SQL-запросов;
- наследует опции генерации родительских мапперов;
- работает как итератор;
- настройка опций – наследование + метод setup().

```
$all = $db->news->categories->select();
foreach ($db->news->categories as $c) echo $c->title."\n";
$category = $db->news->categories->find($id);
$category->title = 'New title';
$db->news->categories->update($category);
$db->news->categories->insert($new_category);
$db->news->categories->delete($category);
$num_of_categories = $db->news->categories->stat();
```

SQL мапперы: изменение параметров запроса

- выполняем spawn();
- дочерний маппер содержит модифицированные опции, остальное – из родительского маппера;
- реализация не содержит циклических ссылок.

```
$db->news->stories->spawn()->
  where('pub_date < :date', $date)->
  order_by('pub_date DESC')->
    select()
```

---

spawn() может выполняться неявно, например для закешированного в MapperSet маппера.

Выборки в терминах предметной области

DevConf

Переопределяем вызов или свойство:

```
class StoriesMapper extends DB_ORM_SQLMapper {
  protected function map_for_category(Category $category) {
    return $this->where('category_id = :id', $category);
  }
}
```

`$db->news->stories->for_category($c)`:

1. неявно выполняется spawn();
2. map-метод вызывается для дочернего маппера;
3. для дальнейшего изменения параметров явный `spawn()` уже не нужен.

$$\texttt{\$mapper[\$id]} = \texttt{\$mapper->find(\$id)} + \textbf{кеширование}$$

Позволяет удобно реализовать выборку связанного объекта:

```
class Story {
  protected function get_category() {
    return App::db()->news->categories[$this['category_id']];
  }
  protected function set_category($category) {
    if ($category->id) $this['category_id'] = $category->id;
  }
}
```

Кеш может быть загружен заранее: `$mapper->preload()`.

Отношение 1:N

Через дочерний маппер:

```
$stories = $db->news->stories->for_category($category);
```

Или через метод объекта домена:

```
class Category {
  protected function get_stories() {
    return self::$db->news->stories->
      for_category($this);
  }
```

Отношение N:M

DevConf

Через отдельный маппер для связующей таблицы:

```
class StoriesTagsMapper {
  protected function setup() {
    return $this->table('news_tag_refs')->
                  columns('story_id', 'tag_id', 'ord')->
                  key('story_id', 'tag_id');
  }
  public function associate($tag, $story, $ord) {
    return $this->insert(
      array($tag->id, $story->id, $ord), 'ignore');
  }
  public function dissociate($tag, $story) {
    return $this->delete(array($tag->id, $story->id));
  }
}
```

И другие возможности

- валидация перед вставкой и изменением;
- вызов обработчиков событий в классе сущности, если таковые определены;
- возможность настройки хинтов для индексов MySQL;
- интеграция со Sphinx.

Подведем итог

- Несложно реализовать легковесную альтернативу тяжелым ORM-библиотекам;
- Даже простое решение может помочь писать компактный и выразительный код;
- Часто проще непосредственно реализовать действие в коде, чем конфигурировать сложный автоматический механизм;
- Возможность отложить непосредственное обращение к базе до последнего момента – удобно для кеширования в шаблонах;
- Иерархическая структура мапперов – хорошая основа для API приложения в стиле REST.

Вопросы?

http://www.techart.ru/

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