Catalyst and Rose::DB

сборка

Rose::DB

Описание работы с ORM смотри

здесь

Rose::DB (+)

- Значительно быстрее, чем DBIx::Class
- Проще в формировании запросов
- Не так «страшен», как DBIx::Class, для переходящих с DBI
- И … пожалуй всё.

Rose::DB (-)

- Пересоздание классов таблиц затрёт все добавленные программистом методы (в отличие от DBIx::Class). Что актуально при изменении структуры базы.
- get_имяТаблицы_iterator может упасть по Out of memory, если возвращает действительно много строк.
- При переходе по отношению между таблицами (например один-ко-многим), вы не можете конкретизировать запрос. Вам вернутся все строки. Имхо, DBIx::Class в этом смысле менее избыточен.

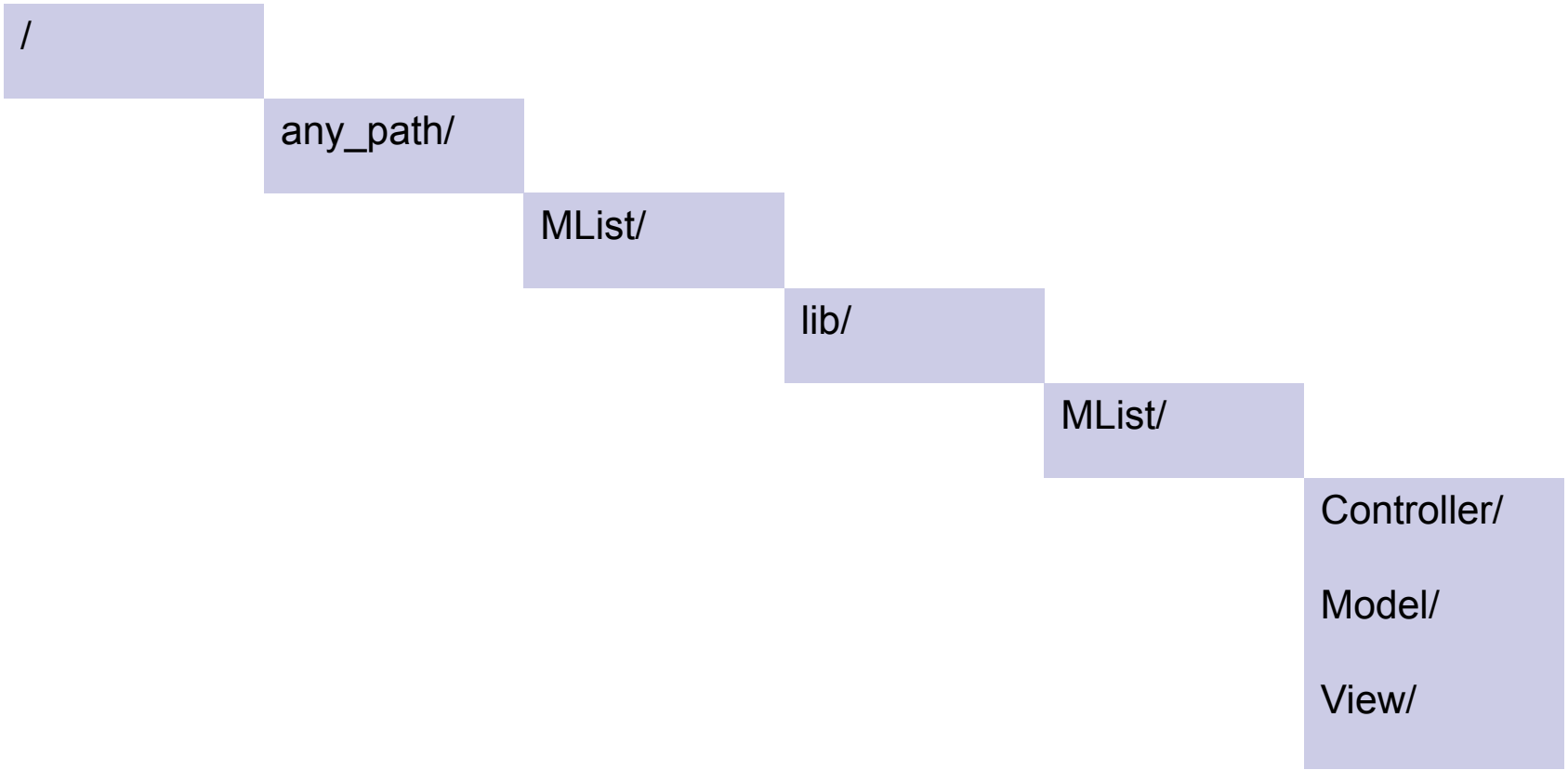
Прикручиваем к Catalyst

Допустим, вы правильно раскурили мануалы по Rose::DB, и всё ещё хотите прикрутить именно этот ORM к вашему приложению. Ну что ж… тогда мы идём к Вам.

Будем считать, что у Вас уже развёрнуто cat-приложение и оно называется MList.

Прикручиваем к Catalyst

Приблизительная структура папок Вашего проекта:



Прикручиваем к Catalyst

Нам потребуются 2 модуля. В одном из них (RDB.pm) будет описан коннект, во втором (RDBBase.pm) – подключение.

Итак, создаём RDB.pm по пути /any_path/MList/lib/MList/Model :

```
package MList::Model::RDB;

use warnings;
use strict;

use base qw(Rose::DB);

__PACKAGE__->use_private_registry;

__PACKAGE__->register_db(
    driver   => 'mysql',
    type   => 'main',
    database => ‘basename',
    host     => ‘db_host',
    username => ‘user',
    password => ‘password',
    connect_options =>
    {
      AutoCommit => 1,
      RaiseError => 1,
    }
);

1;
```

Прикручиваем к Catalyst

Создаём модель RDBBase

```
$ cd /any_path/MList
$ script/mlist_create.pl model RDBBase
```

*создание модуля через script/mlist_create.pl нужно, чтобы у вас автоматически создались тесты для Модели в папке t/.

И отредактируем полученную модель до вида:

```
package MList::Model::RDBBase;

use strict;

use MList::Model::RDB;
use Rose::DB::Object;
our @ISA = qw(Rose::DB::Object);

use parent 'Catalyst::Model';

# вот так корректно подключается helper в вашем cat-приложении для Rose::DB
use Rose::DB::Object::Helpers qw(load_or_insert load_speculative);

sub init_db {MList::Model::RDB->new(type => 'main')}

# метод для доступа к классу Manager вашей таблицы
sub custom_rdb_manager {$_[0].'::Manager'}

1;
```

Прикручиваем к Catalyst

Итого у вас получилось следующее:

- /
  - any_path/
    - MList/
      - lib/
        - MList/
          - Controller/
          - Model/ **RDB.pm** **RDBBase.pm**
          - View/

Прикручиваем к Catalyst

Теперь создаём скрипт, который выльет нам структуру базы в классы Rose::DB::Object, используя созданные RDB.pm и RDBBase.pm.

```
#!/usr/bin/perl

use warnings;
use strict;

use Rose::DB::Object::Loader;
use lib '/any_path/MList/lib';
use MList::Model::RDB;
use MList::Model::RDBBase;

my $loader = Rose::DB::Object::Loader->new(
        db => MList::Model::RDB->new(type => 'main'),
        class_prefix => 'MList::Model::RDB',
        base_classes => 'MList::Model::RDBBase',
        with_foreign_keys => 1,
        with_relationships => 1
);

$loader->make_modules(
        module_dir=>'/any_path/MList/lib',
        exclude_tables=>'Tmp'
);

exit;
```

Прикручиваем к Catalyst

use lib – позволяет подгрузить созданные модули.

make_modules – этот метод выльет структуру таблиц, при этом, параметр exclude_tables => ‘Tmp’ исключит из обработки таблицы. Соотвествующие regexp /^Tmp/

Для ознакомления с другими параметрами. Вы можете ознакомиться с Rose::DB::Object::Loader.

Итак, после запуска скрипта, по пути /any_path/MList/lib/MList/Model/RDB вы можете увидеть все созданные классы.

Всегда помните, что это – не DBIx::Class, и если у вас уже были созданые по этому пути классы таблиц, то Rose::DB::Object::Loader просто перезапишет их.

Использование

Допустим у вас в базе была таблица ml_persons. Тогда для неё будут созданы 2 модуля:

- /any_path/MList/lib/MList/Model/RDB/MlPerson.pm

Этот модуль отвечает за выборку строк из таблицы. Обращение к нему:

$c->model('RDB::MlPerson')->any_method(…)

, где any_method – это методы Rose::DB::Object

- /any_path/MList/lib/MList/Model/RDB/MlPerson/Manager.pm

Этот модуль отвечает за обработку набора строк из таблицы. Обращение к нему:

$c->model('RDB::MlPerson::Manager')->any_method(…)

, где any_method – это методы Rose::DB::Object::Manager

Использование

Итого у вас получилось следующее:

```
/
    any_path/
        MList/
            lib/
                MList/
                    Controller/
                    Model/      RDB.pm
                                RDBBase.pm
                                RDB/        MIPerson.pm
                                MIPerson/   Manager.pm
                                …
                    View/
```

Использование

И, напоследок, вспомним про необязательный метод:

```
sub custom_rdb_manager {$_[0].'::Manager'}
```

из модуля RDBBase.pm.

Он позволяет вам делать вот такие обращения:

```
$c->model('RDB::MlPerson')->custom_rdb_manager->any_method(…)
```

, где any_method – это методы Rose::DB::Object::Manager

Здесь важно, чтобы название метода (которое вы дадите) просто не совпадало с названием ни одного поля в базе.

Но, данный метод опционален и, вы всегда можете обращаться к классам ::Manager так, как было показано на предыдущем слайде.