俺とModelとAPI

!SLIDE

自己紹介

Perl日誌というblogを書いています。

http://d.hatena.ne.jp/okamuuu

最近Perlと関係ない仕事ばかりやっています。

!SLIDE

俺が考えるModelとAPI

俺はModelをこう考える

* DB操作、WebAPIへのリクエストなどプログラム内部の操作を抽象化
* 使い方を誤ると例外が発生
* アプリケーションに必要な機能を実装する

APIをこう考える

* アプリケーションの要件を満たすインターフェースを実装する
* 具体的な機能はModelに実装されているのでそれらを組み合わせる
* どんな使い方をしても例外は起きないように実装する
* 例外の代わりにstatus messages, error messagesとして保持する
* Web::Controller、CLIなどがこれを実行する

!SLIDE

こんな感じで使いたい

Modelはこんな感じ。

```
    my $user = App::Model::User->find('okamuuu');

    eval { $user->buy_item('どくばり') };
    Carp::croak $@ if $@; # not enough money!!
```

APIはこんな感じ

```
    sub dispatch_hoge {
```

```
    my $api = App::Api::User->new;
    $api->user_buy_item( user => $user, item => 'どくばり', item_count => 1 );

    if ( $api->has_error_msgs ) {
        $c->stash->error_msgs($api->error_msgs); # 所持金が不足しています
    }
    else if {
        $c->flash->status_msgs($api->status_msgs); # どくばりを購入しました
    }
}
```

!SLIDE

実装例

App::Api::User

```
package App::Api::User;
use strict;
use warnings;
use parent 'App::Api::Base';

sub buy_item { ... }
```

App::Api::Base

```
package App::Api::Base;
use strict;
use warnings;
use App::Api::Msg;
use App::Api::DebugMsg;
use Class::Accessor::Lite 0.05 (
    ro => [qw/log_path/],
    rw => [qw/error_msg status_msg debug_msg/] );

sub new {
    my ($class, %args) = @_;

    bless {
        log_path   => $args{log_path},
        status_msg => Sid::Api::Msg->new,
        error_msg  => Sid::Api::Msg->new,
        debug_msg  => Sid::Api::DebugMsg->new,
    }, $class;
}

sub set_status_msgs {
    my ( $self, @new_msgs ) = @_;
```

```
    $self->status_msg->set_msgs(@new_msgs);
    $self->debug_msg->set_msgs(@new_msgs);
}

sub set_error_msgs {
    my ( $self, @new_msgs ) = @_;
    $self->error_msg->set_msgs(@new_msgs);
    $self->debug_msg->set_msgs(@new_msgs);
}

sub set_debug_msgs {
    my ( $self, @new_msgs ) = @_;
    $self->debug_msg->set_msgs(@new_msgs);
}

sub DESTROY {
    my $self = shift;

    if ( $self->log_path and $self->debug_msg->has_msgs ) {

        my $fh = IO::File->new( $self->log_path, 'a');

        $fh->print( '-' x 10, "\n" );
        $fh->print($_) for map { "$_\n" } $self->debug_msg->get_msgs();
        $fh->print("\n");
        $fh->close;
    }
    elsif ( $self->debug_msg->has_msgs ) {
        warn $_ for $self->debug_msg->get_msgs();
    }
    else {
        return;
    }
}

1;
```

App::Api::Msg

```
package App::Api::Msg;
use strict;
use warnings;

sub new { return bless { _msgs => [] }, $_[0]; }

sub get_msgs { @{ $_[0]->{_msgs} }  }

sub has_msgs { scalar $_[0]->get_msgs }
```

```
    sub set_msgs {
        my ( $self, @new_msgs ) = @_;
        $self->set_msg($_) for @new_msgs;
    }

    sub set_msg {
        my ( $self, $new_msg ) = @_;
        $self->{_msgs} = [ $self->get_msgs, $self->_edit_preset_msg($new_msg) ];
    }

    sub _edit_preset_msg { return $_[1] } # hook

    1;

App::Api::DebugMsg

    package App::Api::DebugMsg;
    use strict;
    use warnings;
    use parent 'App::Api::Msg';
    use Time::HiRes ();

    sub new {
        my $class = shift;
        my $self = $class->SUPER::new;
        $self->{_start} = [Time::HiRes::gettimeofday];
        return $self;
    }

    sub start { $_[0]->{_start} }

    sub _edit_preset_msg {
        my ($self, $msg) = @_;

        my $now = _now();
        my $elapsed = Time::HiRes::tv_interval($self->start, [Time::HiRes::gettimeofday]);

        return "[$now] $msg ( $elapsed sec )";
    }

    ### copied from Log::Minimal
    sub _now {
        my ( $sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdst ) =
          localtime(time);

        my $time = sprintf(
            "%04d-%02d-%02dT%02d:%02d:%02d",
            $year + 1900,
            $mon + 1, $mday, $hour, $min, $sec
```

```
    );
}

1;
```

!SLIDE

私は開発中にこんな口グを見ています。

```
[2011-04-20T20:07:04] start create document ( 0.000108 sec ) at lib/Sid/Api/Base.pm line 51.
[2011-04-20T20:07:04] end create document ( 0.087972 sec ) at lib/Sid/Api/Base.pm line 51.
[2011-04-20T20:07:04] start write_html ( 0.088269 sec ) at lib/Sid/Api/Base.pm line 51.
[2011-04-20T20:07:04] start write category readme ( 0.088326 sec ) at lib/Sid/Api/Base.pm line 51.
[2011-04-20T20:07:04] end write category readme ( 0.262469 sec ) at lib/Sid/Api/Base.pm line 51.
[2011-04-20T20:07:04] start write category 01 ( 0.262537 sec ) at lib/Sid/Api/Base.pm line 51.
[2011-04-20T20:07:04] end write category 01 ( 0.264307 sec ) at lib/Sid/Api/Base.pm line 51.
[2011-04-20T20:07:04] start write category 02 ( 0.264377 sec ) at lib/Sid/Api/Base.pm line 51.
[2011-04-20T20:07:04] end write category 02 ( 0.265846 sec ) at lib/Sid/Api/Base.pm line 51.
[2011-04-20T20:07:04] end write_html ( 0.265913 sec ) at lib/Sid/Api/Base.pm line 51.
```

もたもたしている処理を発見するのにもたもたしたくないです。

!SLIDE

APIというレイヤーを必要だと思った理由1

* 複数人で開発を進めるとModelというレイヤーが広くなる
* Model内に上位レイヤーと下位レイヤーができあがる
* 例外補足処理が共通化できる箇所が生まれる
* Modelに例外処理を埋め込む事になる
* Model内の上位レイヤーは例外を補足する記述をする
* Model内に例外を起こすものとそうでないものが生まれる
* Modelよりもさらに上位のレイヤーが欲しい

!SLIDE

APIというレイヤーを必要だと思った理由2

* 開発中に処理時間を計測したい
* ユーザーがどのような操作をして、その処理は何秒かかるのか？
* 思いつくのはDecoratorパターン。どのように実装する？
* Modelという名前空間よりも上位となるレイヤーが欲しい
* 本番運営中でもAPIが終了するまでにx秒かかったら口グ出力させるとか

!SLIDE

APIというレイヤーを必要だと思った理由3

* 入力された値が適切かどうかをチェックするValidation
* Web::ControllerがValidationを呼び出しているのは妥当か？
* CLIから実行した場合、CLIにもValidationを記述する事になる。
* skinny Controller, fat Modelだよね？
* App::ValidationとApp::Modelをまとめるレイヤーがあるとskinny Controllerに近づく
* やっぱもう一個レイヤーが欲しい

!SLIDE

結果、APIが欲しくなった

* Web::ControllerとModelの間にもう一つレイヤーが欲しい
* APIという名前に固執はしないが、どう考えてもAPIという名前がしっくりする
* でもAPIとか言うと、「何だよAPIって？」とか言われるけれど
* 必要なのは例外処理、ロギング、トランザクションの処理をどこに実装するかというルール
* とりあえずAPIと呼ぶ事にする

!SLIDE

必ずAPIを通すのか？

* show, listingなどRead処理でいちいちApiを実装するのは面倒
* 例外処理や口ギング正直書かないからこの場合はWeb::ControllerからModelを直接呼べばいい
* CREATE, UPDATE, DELETEなどは重要な処理が多いのでAPIを通す
* ソーシャルアプリなど、ユーザーがアイテムを購入する処理などで記録が残ってうれしいかも

!SLIDE

課題

* ログの記述方法が独創的すぎる
* トランザクションをAPIで書くというルールでよいか？
* そもそもDB操作はModel内で抽象化すれば一番よい気がする。
* それ、version number使って楽観ロックすると抽象化できる、かも。
* トランザクションが入れ子とか、実装した事ないからわかりません。
* 自分の家でやってる俺々プロジェクトでしか実装したことありません。
* だから良い手法かどうかは不明です。

!SLIDE

最後に

* 巷では俺々WaFがたくさんある。
* でも、どういう風に実装しているのか、という話が少ない気がする。
* 他人がどうやって実装しているのか気になる、教えて欲しい。
* とりあえず俺はこうする。
* みんなはどうしてる?

おしまい