设计要点：

restful、自动模型、约定优于配置；

超轻量，只开发核心和重要的功能，其它一律放在Lib包中或交给用户开发。

不要制造过多的概念。本框架不是大众化的框架，没必要过多的封装，只封装最基本的功能即可。

数据表名不区分大小写。

模型名使用大驼峰法命名，大小写转换规则：如果全部为大写，则不转换，例如：BOM对于的数据表为cims\_BOM；只要有小写字母和大于一个的大写字母，则进行如下转换：CustmoerGroup->customer\_group

BOm对于的表为cims\_b\_om

框架只有一个模型类：Model类，项目的所有模型类都继承于此Model类（或者注册为Model类的观察者）。

要实现的功能：

1.关联查询：

一旦在模型类中定义了$view就自动调用视图模型，定义了$relation就自动使用关联模型。视图模型由join或view来实现。同时定义了就默认使用视图模型，也可以在Get()方法中指定：

Get(0)：普通模型

Get(1)：关联模型

视图模型是默认的，无需指定

当然视图模型只用来查询，而关联模型则功能更多一些。视图模型的性能好像比关联模型高一些。

1. 支持websocket

说明：

由于操作关系数据库、nosql时接口不一样，所以需要pdo.class.php、Couchdb.class.php、Couchdb.class.php等数据库操作类，这些是根据相应驱动的接口对具体数据库的操作方法的封装，均继承于Db这个抽象类。之所以使用抽象类而不使用接口的原因是，数据库操作类有很多公用的成员变量和成员方法，只有部分方法是需要中间类自己各自实现的，所以这里需要一个比接口更加严格的约束及更合适的组织方式。

Db类提供一个静态方法，用来获取数据库的唯一实例（在Model.class.php中获取）。数据库操作类类都主要有4个方法：Get()、Post()、Put()、Delete()，用来封装各自的数据库操作。还提供两个一个静态的魔术方法：**\_\_callStatic()** 调用具体数据库操作类的方法。也就是说，pbay的Db类相当于TP的Db类和Driver类的组合（针对TP4.0）

在Model类中，根据配置中的数据库类型来获取那个实例，不同数据库的实例。

Model类中也有与数据库类同名的方法，不过Model类中的这些方法只是调用对应数据库类的同名方法而已，并且利用数据库操作类的这4个方法来实现orm映射。

gobay框架的实现也是如此。

权限控制

使用访问权限控制表（ACL）这种ZF采用的方式来实现权限控制。

users表、role表、access表 共3个表

access表

|  |  |  |
| --- | --- | --- |
| 角色ID | 资源标识（URI） | 允许的方法(不区分大小写) |
| role\_id | uri | method |
| 1 | 1-0-2-12 | head,get,post,put |
| 1 | 1-1-2-12 | get,post, |
| 1 | 1-2-2-12 | head,get,post, |
| 1 | 1-0-2-12 | get,post,put |

access表没有主键，只有一个索引：role\_id

没有资源表，只要访问的URI在access表的当前角色的URI字段中存在就是有效的资源。

一个role可以对应多个用户，但一个用户只能对应一个角色。一个员工如果需要多个角色，那么就必须给这个员工开启多个用户。也就是说，一个员工可以对应多个用户。

users表的的role\_id字段就表示该用户所属的角色。

如果角色和用户是多对多的话就需要role\_user表，但这样不合理。一个用户就只该有一个角色。

框架用例图：

框架类图：