

ModelHelper

对laravel Eloquent OMR 的一层带缓存的封装

参考了<https://github.com/angejia/pea.git> 的缓存层实现

封装的目的

在laravel的使用过程中，发现很多对model层缓存的透明化封装，而这些封装主要是对所有的sql语句进行了缓存，如果有数据的更新或者删除，则对数据进行了全部删除，做的好一些的，就是对表级数据，进行了删除。

在实际使用过程中，特别是web类，面向用户的操作，更多的只是简单的select操作，如果我们将这些简单的select单条查询存入缓存，讲连表操作改成 select一张表中的数据list，然后foreach 该list，循环取另一张表的info类型，我们可以避免80%以上的连表操作。

但是，这种做法，对缓存控制要求就很高了，对于缓存脏数据的清理，我们希望更精准，谁脏了，就干掉谁，而不是批量处理的做法，封装了此扩展。

用法

1.安装

```
composer require hbclare/model-helper:dev-master
修改config/app.php
'Eloquent' => 'Illuminate\Database\Eloquent\Model',
改为
'Eloquent' => 'Hbclare\ModelHelper\Model',
```

假设我们有一个user表

字段	说明
id	主键id
passport	登陆账号
password	登陆密码
nickname	昵称
...	...

2.在model中，通过简单的设置，则可以实现透明的原子化缓存

```
<?php
namespace App\Models\Desktop;
use Hbclare\ModelHelper\Model;

class User extends Model
{
    ...
    public function __construct(array $attributes = [])
    {
        parent::__construct($attributes);
        $this->startAutoEachCache();//开启自动原子缓存
    }

    public function getUserInfo($id){
        return $this->findOne($id);
    }

    public function changePWD($id, $pwd){
        $return $this->saveInfo(['id'=>$id, 'password'=>$pwd]);
    }
    ...
}
```

以上代码，在执行getUserInfo的时候，就会自动生成auto_table_{table}id{id} 的缓存
执行 changePWD方法的时候，就会删除auto_table_{table}id{id} 这个缓存

什么是原子化缓存？

对应于原子化操作的定义，对一条不可在细分数据的缓存就是原子化缓存，通俗点说，就是对表的行数据的缓存

2.可以对较复杂的Model层操作，快速的进行缓存，同时对于该缓存key的设计，可以自动的进行缓存更新操作

key 支持多种通配处理：

*如 user_info_{id},括号中的id,对应的就是表中的字段，如果 where 条件中有 id=12 这样的条件，就会将key变成 user_info_12

*一个key里面可以包含多个{},如: user_info_type_{type}id{id}

*使用的时候要注意，如果通配符字段id=2(一定是等于),不在where条件中,升值的这条缓存key会被抛弃

在reids的驱动下，可以支持*,?等通配符的支持

注意，*,?和{},不能组合使用

```
<?php
namespace App\Models\Desktop;
use Hbclare\ModelHelper\Model;

class User extends Model
{
    ...
    public function __construct(array $attributes = [])
    {
        parent::__construct($attributes);
        $this->stopAutoEachCache(); // 关闭自动原子缓存
        $this->setAfterUpdateFlushKey(
            ['userinfo_{id}']
        ); // 设置update后，需要清空的缓存
        $this->setAfterDeleteFlushKey([]); // 设置Delete后，需要清空的缓存
        $this->setAfterInsertFlushKey([]); // 设置Insert后，需要清空的缓存
    }

    public function getUserInfo($id){
        $this->setCacheKeys('userinfo_.'.$id); // 设置findOne后的key
        return $this->findOne($id);
    }

    public function changePWD($id, $pwd){
        $this->setFlushKeys('getUserList*'); // 设置findOne后的key
        $return $this->saveInfo(['id'=>$id, 'password'=>$pwd]);
    }
    ...
}
```