Medoo中文文档-高效的轻量级PHP数据库框架[1.0.1]

Jewel_M

目　录

开始

为什么选择 Medoo

1. 非常的轻量 只有 20KB
2. 简单 非常的容易学习，快速上手
3. 强大 支持各种常见的SQL查询
4. 兼容
   支持各种数据：MySQL, MSSQL, SQLite, MariaDB, Oracle, 5. Sybase, Pos
5. 安全 防止SQL注入
6. 免费 MIT 协议，你可以进行任何修改

使用步骤

依赖安装

```
$ composer require catfan/Medoo
```

更新

```
$ composer update
```

开始

```
// 如果你使用php的依赖安装。可以使用以下方法自动载入
require 'vendor/autoload.php';

// 或者将你下载的medoo文件拷贝到你相应的目录，然后载入即可
require_once 'medoo.php';

// 初始化配置
$database = new medoo([
    'database_type' => 'mysql',
    'database_name' => 'name',
    'server' => 'localhost',
    'username' => 'your_username',
    'password' => 'your_password',
    'charset' => 'utf8'
]);

// 插入数据示例
$database->insert('account', [
    'user_name' => 'foo',
    'email' => 'foo@bar.com',
    'age' => 25,
    'lang' => ['en', 'fr', 'jp', 'cn']
```

```
]);
```

> 只有在 PHP 5.4+, 你可以使用[] 作为语法.

源码

```
<?php
/**
 * Medoo database framework
 * http://medoo.in
 * Version 1.0.1
 *
 * Copyright 2016, Angel Lai
 * Released under the MIT license
 *
 * By Jewel_M 2016-2-22 Update
 */

class Medoo {
    // General
    protected $database_type;
    protected $charset;
    protected $database_name;
    // For MySQL, MariaDB, MSSQL, Sybase, PostgreSQL, Oracle
    protected $server;
    protected $username;
    protected $password;
    // For SQLite
    protected $database_file;
    // For MySQL or MariaDB with unix_socket
    protected $socket;
    // Optional
    protected $port;
    protected $prefix;
    protected $option = array();
    // Variable
    protected $logs = array();
    public $debug_mode = false;

    /**
     * Medoo constructor.
     * @param null $options
     * @throws Exception
     */
    public function __construct($options = null) {
        try {
            $commands = array();
            $dsn = '';
            if (is_array($options)) {
                foreach ($options as $option => $value) {
                    $this->{$option} = $value;
                }
            } else {
                return false;
            }
            if (isset($this->port) && is_int($this->port * 1)) {
```

```
                $port = $this->port;
            }
            $type = strtolower($this->database_type);
            $is_port = isset($port);
            if (isset($options['prefix'])) {
                $this->prefix = $options['prefix'];
            }
            switch ($type) {
                case 'mariadb':
                    $type = 'mysql';
                case 'mysql':
                    if ($this->socket) {
                        $dsn = $type . ':unix_socket=' . $this->socket .
';dbname=' . $this->database_name;
                    } else {
                        $dsn = $type . ':host=' . $this->server . ($is_po
rt ? ';port=' . $port : '') . ';dbname=' . $this->database_name;
                    }
                    // Make MySQL using standard quoted identifier
                    $commands[] = 'SET SQL_MODE=ANSI_QUOTES';
                    break;
                case 'pgsql':
                    $dsn = $type . ':host=' . $this->server . ($is_port ?
 ';port=' . $port : '') . ';dbname=' . $this->database_name;
                    break;
                case 'sybase':
                    $dsn = 'dblib:host=' . $this->server . ($is_port ? ':
' . $port : '') . ';dbname=' . $this->database_name;
                    break;
                case 'oracle':
                    $dbname = $this->server ? '//' . $this->server . ($is
_port ? ':' . $port : ':1521') . '/' . $this->database_name : $this->data
base_name;
                    $dsn = 'oci:dbname=' . $dbname . ($this->charset ? ';
charset=' . $this->charset : '');
                    break;
                case 'mssql':
                    $dsn = strstr(PHP_OS, 'WIN') ? 'sqlsrv:server=' . $th
is->server . ($is_port ? ',' . $port : '') . ';database=' . $this->databa
se_name : 'dblib:host=' . $this->server . ($is_port ? ':' . $port : '') .
 ';dbname=' . $this->database_name;
                    // Keep MSSQL QUOTED_IDENTIFIER is ON for standard qu
oting
                    $commands[] = 'SET QUOTED_IDENTIFIER ON';
                    break;
                case 'sqlite':
                    $dsn = $type . ':' . $this->database_file;
                    $this->username = null;
                    $this->password = null;
                    break;
            }
            if (in_array($type, array('mariadb', 'mysql', 'pgsql', 'sybas
e', 'mssql')) && $this->charset) {
                $commands[] = 'SET NAMES \'' . $this->charset . '\'';
            }
            $this->pdo = new PDO($dsn, $this->username, $this->password,
```

```
 $this->option);
            foreach ($commands as $value) {
                $this->pdo->exec($value);
            }
        } catch (PDOException $e) {
            throw new Exception($e->getMessage());
        }
    }

    /**
     * 执行带返回结果集的SQL语句
     * @param $query
     * @return bool|PDOStatement
     */
    public function query($query) {
        if ($this->debug_mode) {
            echo $query;
            $this->debug_mode = false;
            return false;
        }
        array_push($this->logs, $query);
        return $this->pdo->query($query);
    }

    /**
     * 执行无返回结果集的SQL语句
     * @param $query
     * @return bool|int
     */
    public function exec($query) {
        if ($this->debug_mode) {
            echo $query;
            $this->debug_mode = false;
            return false;
        }
        array_push($this->logs, $query);
        return $this->pdo->exec($query);
    }

    /**
     * 给字符串添加引号
     * @param $string
     * @return string
     */
    public function quote($string) {
        return $this->pdo->quote($string);
    }

    /**
     * 格式化数据
     * @param $string
     * @return string
     */
    protected function column_quote($string) {
        return '"' . str_replace('.', '"."', preg_replace('/(^#|\\(JSON\\
)\\s*)/', '', $string)) . '"';
```

```
    }

    /**
     * 格式化返回字段
     * @param $columns
     * @return array|string
     */
    protected function column_push($columns) {
        if ($columns == '*') {
            return $columns;
        }
        if (is_string($columns)) {
            $columns = array($columns);
        }
        $stack = array();
        foreach ($columns as $key => $value) {
            preg_match('/([a-zA-Z0-9_\\-\\.]*)\\s*\\(([a-zA-Z0-9_\\-]*)\\
)/i', $value, $match);
            if (isset($match[1], $match[2])) {
                array_push($stack, $this->column_quote($match[1]) . ' AS
' . $this->column_quote($match[2]));
            } else {
                array_push($stack, $this->column_quote($value));
            }
        }
        return implode($stack, ',');
    }

    /**
     * @param $array
     * @return string
     */
    protected function array_quote($array) {
        $temp = array();
        foreach ($array as $value) {
            $temp[] = is_int($value) ? $value : $this->pdo->quote($value)
;
        }
        return implode($temp, ',');
    }

    /**
     * @param $data
     * @param $conjunctor
     * @param $outer_conjunctor
     * @return string
     */
    protected function inner_conjunct($data, $conjunctor, $outer_conjunct
or) {
        $haystack = array();
        foreach ($data as $value) {
            $haystack[] = '(' . $this->data_implode($value, $conjunctor)
. ')';
        }
        return implode($outer_conjunctor . ' ', $haystack);
    }
```

```
    /**
     * @param $column
     * @param $string
     * @return string
     */
    protected function fn_quote($column, $string) {
        return strpos($column, '#') === 0 && preg_match('/^[A-Z0-9\\_]*\\
([^)]*\\)$/', $string) ? $string : $this->quote($string);
    }

    /**
     * @param $data
     * @param $conjunctor
     * @param null $outer_conjunctor
     * @return string
     */
    protected function data_implode($data, $conjunctor, $outer_conjunctor
 = null) {
        $wheres = array();
        foreach ($data as $key => $value) {
            $type = gettype($value);
            if (preg_match('/^(AND|OR)(\\s+#.*)?$/i', $key, $relation_mat
ch) && $type == 'array') {
                $wheres[] = 0 !== count(array_diff_key($value, array_keys
(array_keys($value)))) ? '(' . $this->data_implode($value, ' ' . $relatio
n_match[1]) . ')' : '(' . $this->inner_conjunct($value, ' ' . $relation_m
atch[1], $conjunctor) . ')';
            } else {
                preg_match('/(#?)([\\w\\.\\-]+)(\\[(\\>|\\>\\=|\\<|\\<\\=
|\\!|\\<\\>|\\>\\<|\\!?~)\\])?/i', $key, $match);
                $column = $this->column_quote($match[2]);
                if (isset($match[4])) {
                    $operator = $match[4];
                    if ($operator == '!') {
                        switch ($type) {
                            case 'NULL':
                                $wheres[] = $column . ' IS NOT NULL';
                                break;
                            case 'array':
                                $wheres[] = $column . ' NOT IN (' . $this
->array_quote($value) . ')';
                                break;
                            case 'integer':
                            case 'double':
                                $wheres[] = $column . ' != ' . $value;
                                break;
                            case 'boolean':
                                $wheres[] = $column . ' != ' . ($value ?
'1' : '0');
                                break;
                            case 'string':
                                $wheres[] = $column . ' != ' . $this->fn_
quote($key, $value);
                                break;
                        }
```

```
                    }
                    if ($operator == '<>' || $operator == '><') {
                        if ($type == 'array') {
                            if ($operator == '><') {
                                $column .= ' NOT';
                            }
                            if (is_numeric($value[0]) && is_numeric($valu
e[1])) {
                                $wheres[] = '(' . $column . ' BETWEEN ' .
 $value[0] . ' AND ' . $value[1] . ')';
                            } else {
                                $wheres[] = '(' . $column . ' BETWEEN ' .
 $this->quote($value[0]) . ' AND ' . $this->quote($value[1]) . ')';
                            }
                        }
                    }
                    if ($operator == '~' || $operator == '!~') {
                        if ($type != 'array') {
                            $value = array($value);
                        }
                        $like_clauses = array();
                        foreach ($value as $item) {
                            $item = strval($item);
                            $suffix = mb_substr($item, -1, 1);
                            if ($suffix === '_') {
                                $item = substr_replace($item, '%', -1);
                            } elseif ($suffix === '%') {
                                $item = '%' . substr_replace($item, '', -
1, 1);
                            } elseif (preg_match('/^(?!%).+(?<!%)$/', $it
em)) {
                                $item = '%' . $item . '%';
                            }
                            $like_clauses[] = $column . ($operator === '!
~' ? ' NOT' : '') . ' LIKE ' . $this->fn_quote($key, $item);
                        }
                        $wheres[] = implode(' OR ', $like_clauses);
                    }
                    if (in_array($operator, array('>', '>=', '<', '<=')))
 {
                        if (is_numeric($value)) {
                            $wheres[] = $column . ' ' . $operator . ' ' .
 $value;
                        } elseif (strpos($key, '#') === 0) {
                            $wheres[] = $column . ' ' . $operator . ' ' .
 $this->fn_quote($key, $value);
                        } else {
                            $wheres[] = $column . ' ' . $operator . ' ' .
 $this->quote($value);
                        }
                    }
                } else {
                    switch ($type) {
                        case 'NULL':
                            $wheres[] = $column . ' IS NULL';
                            break;
```

```
                        case 'array':
                            $wheres[] = $column . ' IN (' . $this->array_
quote($value) . ')';
                            break;
                        case 'integer':
                        case 'double':
                            $wheres[] = $column . ' = ' . $value;
                            break;
                        case 'boolean':
                            $wheres[] = $column . ' = ' . ($value ? '1' :
 '0');
                            break;
                        case 'string':
                            $wheres[] = $column . ' = ' . $this->fn_quote
($key, $value);
                            break;
                    }
                }
            }
        }
        return implode($conjunctor . ' ', $wheres);
    }

    /**
     * @param $where
     * @return string
     */
    protected function where_clause($where) {
        $where_clause = '';
        if (is_array($where)) {
            $where_keys = array_keys($where);
            $where_AND = preg_grep('/^AND\\s*#?$/i', $where_keys);
            $where_OR = preg_grep('/^OR\\s*#?$/i', $where_keys);
            $single_condition = array_diff_key($where, array_flip(explode
(' ', 'AND OR GROUP ORDER HAVING LIMIT LIKE MATCH')));
            if ($single_condition != array()) {
                $condition = $this->data_implode($single_condition, '');
                if ($condition != '') {
                    $where_clause = ' WHERE ' . $condition;
                }
            }
            if (!empty($where_AND)) {
                $value = array_values($where_AND);
                $where_clause = ' WHERE ' . $this->data_implode($where[$v
alue[0]], ' AND');
            }
            if (!empty($where_OR)) {
                $value = array_values($where_OR);
                $where_clause = ' WHERE ' . $this->data_implode($where[$v
alue[0]], ' OR');
            }
            if (isset($where['MATCH'])) {
                $MATCH = $where['MATCH'];
                if (is_array($MATCH) && isset($MATCH['columns'], $MATCH['
keyword'])) {
                    $where_clause .= ($where_clause != '' ? ' AND ' : ' W
```

```
HERE ') . ' MATCH ("' . str_replace('.', '"."', implode($MATCH['columns']
, '", "')) . '") AGAINST (' . $this->quote($MATCH['keyword']) . ')';
                }
            }
            if (isset($where['GROUP'])) {
                $where_clause .= ' GROUP BY ' . $this->column_quote($wher
e['GROUP']);
                if (isset($where['HAVING'])) {
                    $where_clause .= ' HAVING ' . $this->data_implode($wh
ere['HAVING'], ' AND');
                }
            }
            if (isset($where['ORDER'])) {
                $rsort = '/(^[a-zA-Z0-9_\\-\\.]*)(\\s*(DESC|ASC))?/';
                $ORDER = $where['ORDER'];
                if (is_array($ORDER)) {
                    if (isset($ORDER[1]) && is_array($ORDER[1])) {
                        $where_clause .= ' ORDER BY FIELD(' . $this->colu
mn_quote($ORDER[0]) . ', ' . $this->array_quote($ORDER[1]) . ')';
                    } else {
                        $stack = array();
                        foreach ($ORDER as $column) {
                            preg_match($rsort, $column, $order_match);
                            array_push($stack, '"' . str_replace('.', '".
"', $order_match[1]) . '"' . (isset($order_match[3]) ? ' ' . $order_match
[3] : ''));
                        }
                        $where_clause .= ' ORDER BY ' . implode($stack, '
,');
                    }
                } else {
                    preg_match($rsort, $ORDER, $order_match);
                    $where_clause .= ' ORDER BY "' . str_replace('.', '".
"', $order_match[1]) . '"' . (isset($order_match[3]) ? ' ' . $order_match
[3] : '');
                }
            }
            if (isset($where['LIMIT'])) {
                $LIMIT = $where['LIMIT'];
                if (is_numeric($LIMIT)) {
                    $where_clause .= ' LIMIT ' . $LIMIT;
                }
                if (is_array($LIMIT) && is_numeric($LIMIT[0]) && is_numer
ic($LIMIT[1])) {
                    if ($this->database_type === 'pgsql') {
                        $where_clause .= ' OFFSET ' . $LIMIT[0] . ' LIMIT
 ' . $LIMIT[1];
                    } else {
                        $where_clause .= ' LIMIT ' . $LIMIT[0] . ',' . $L
IMIT[1];
                    }
                }
            }
        } else {
            if ($where != null) {
                $where_clause .= ' ' . $where;
```

```
            }
        }
        return $where_clause;
    }

    /**
     * 格式化SQL语句
     * @param $table
     * @param $join
     * @param null $columns
     * @param null $where
     * @param null $column_fn
     * @return string
     */
    protected function select_context($table, $join, &$columns = null, $w
here = null, $column_fn = null)
    {
        $table = '"' . $this->prefix . $table . '"';
        $join_key = is_array($join) ? array_keys($join) : null;
        if (isset($join_key[0]) && strpos($join_key[0], '[') === 0) {
            $table_join = array();
            $join_array = array('>' => 'LEFT', '<' => 'RIGHT', '<>' => 'F
ULL', '><' => 'INNER');
            foreach ($join as $sub_table => $relation) {
                preg_match('/(\\[(\\<|\\>|\\>\\<|\\<\\>)\\])?([a-zA-Z0-9_
\\-]*)\\s?(\\(([a-zA-Z0-9_\\-]*)\\))?/', $sub_table, $match);
                if ($match[2] != '' && $match[3] != '') {
                    if (is_string($relation)) {
                        $relation = 'USING ("' . $relation . '")';
                    }
                    if (is_array($relation)) {
                        // For ['column1', 'column2']
                        if (isset($relation[0])) {
                            $relation = 'USING ("' . implode($relation, '
", "') . '")';
                        } else {
                            $joins = array();
                            foreach ($relation as $key => $value) {
                                $joins[] = (
                                    strpos($key, '.') > 0
                                        ?
                                        // For ['tableB.column' => 'colum
n']
                                        '"' . str_replace('.', '"."', $ke
y) . '"'
                                        :
                                        // For ['column1' => 'column2']
                                        $table . '."' . $key . '"') . ' =
 ' . '"' . (isset($match[ 5 ]
                                    )
                                        ?
                                        $match[ 5 ]
                                        :
                                        $this->prefix . $match[ 3 ]
                                    ) . '."' . $value . '"';
                            }
```

```
                            $relation = 'ON ' . implode($joins, ' AND ');
                        }
                    }
                    $table_join[] = $join_array[$match[2]] . ' JOIN "' .
$this->prefix . $match[3] . '" ' . (isset($match[5]) ? 'AS "' . $match[5]
 . '" ' : '') . $relation;
                }
            }
            $table .= ' ' . implode($table_join, ' ');
        } else {
            if (is_null($columns)) {
                if (is_null($where)) {
                    if (is_array($join) && isset($column_fn)) {
                        $where = $join;
                        $columns = null;
                    } else {
                        $where = null;
                        $columns = $join;
                    }
                } else {
                    $where = $join;
                    $columns = null;
                }
            } else {
                $where = $columns;
                $columns = $join;
            }
        }
        if (isset($column_fn)) {
            if ($column_fn == 1) {
                $column = '1';
                if (is_null($where)) {
                    $where = $columns;
                }
            } else {
                if (empty($columns)) {
                    $columns = '*';
                    $where = $join;
                }
                $column = $column_fn . '(' . $this->column_push($columns)
 . ')';
            }
        } else {
            $column = $this->column_push($columns);
        }
        return 'SELECT ' . $column . ' FROM ' . $table . $this->where_cla
use($where);
    }

    /**
     * 数据库查询
     * @param $table
     * @param $join
     * @param null $columns
     * @param null $where
     * @return array|bool
```

```
     */
    public function select($table, $join, $columns = null, $where = null)
 {
        $query = $this->query($this->select_context($table, $join, $colum
ns, $where));
        return $query ? $query->fetchAll(is_string($columns) && $columns
!= '*' ? PDO::FETCH_COLUMN : PDO::FETCH_ASSOC) : false;
    }

    /**
     * 插入新数据[可插入多条] 返回插入后的ID
     * @param $table
     * @param $datas
     * @return array
     */
    public function insert($table, $datas) {
        $lastId = array();
        // Check indexed or associative array
        if (!isset($datas[0])) {
            $datas = array($datas);
        }
        foreach ($datas as $data) {
            $values = array();
            $columns = array();
            foreach ($data as $key => $value) {
                array_push($columns, $this->column_quote($key));
                switch (gettype($value)) {
                    case 'NULL':
                        $values[] = 'NULL';
                        break;
                    case 'array':
                        preg_match('/\\(JSON\\)\\s*([\\w]+)/i', $key, $co
lumn_match);
                        $values[] = isset($column_match[0]) ? $this->quot
e(json_encode($value)) : $this->quote(serialize($value));
                        break;
                    case 'boolean':
                        $values[] = $value ? '1' : '0';
                        break;
                    case 'integer':
                    case 'double':
                    case 'string':
                        $values[] = $this->fn_quote($key, $value);
                        break;
                }
            }
            $this->exec('INSERT INTO "' . $this->prefix . $table . '" ('
. implode(', ', $columns) . ') VALUES (' . implode($values, ', ') . ')');
            $lastId[] = $this->pdo->lastInsertId();
        }
        return count($lastId) > 1 ? $lastId : $lastId[0];
    }

    /**
     * 更新指定条件的数据
     * @param $table
```

```
     * @param $data
     * @param null $where
     * @return bool|int
     */
    public function update($table, $data, $where = null)
    {
        $fields = array();
        foreach ($data as $key => $value) {
            preg_match('/([\\w]+)(\\[(\\+|\\-|\\*|\\/)\\])?/i', $key, $ma
tch);
            if (isset($match[3])) {
                if (is_numeric($value)) {
                    $fields[] = $this->column_quote($match[1]) . ' = ' .
$this->column_quote($match[1]) . ' ' . $match[3] . ' ' . $value;
                }
            } else {
                $column = $this->column_quote($key);
                switch (gettype($value)) {
                    case 'NULL':
                        $fields[] = $column . ' = NULL';
                        break;
                    case 'array':
                        preg_match('/\\(JSON\\)\\s*([\\w]+)/i', $key, $co
lumn_match);
                        $fields[] = $column . ' = ' . $this->quote(isset(
$column_match[0]) ? json_encode($value) : serialize($value));
                        break;
                    case 'boolean':
                        $fields[] = $column . ' = ' . ($value ? '1' : '0'
);
                        break;
                    case 'integer':
                    case 'double':
                    case 'string':
                        $fields[] = $column . ' = ' . $this->fn_quote($ke
y, $value);
                        break;
                }
            }
        }
        return $this->exec('UPDATE "' . $this->prefix . $table . '" SET '
 . implode(', ', $fields) . $this->where_clause($where));
    }

    /**
     * 删除指定条件的数据
     * @param $table
     * @param $where
     * @return bool|int
     */
    public function delete($table, $where) {
        return $this->exec('DELETE FROM "' . $this->prefix . $table . '"'
 . $this->where_clause($where));
    }

    /**
```

```
     * 将新的数据替换旧的数据
     * @param $table
     * @param $columns
     * @param null $search
     * @param null $replace
     * @param null $where
     * @return bool|int
     */
    public function replace($table, $columns, $search = null, $replace =
null, $where = null) {
        if (is_array($columns)) {
            $replace_query = array();
            foreach ($columns as $column => $replacements) {
                foreach ($replacements as $replace_search => $replace_rep
lacement) {
                    $replace_query[] = $column . ' = REPLACE(' . $this->c
olumn_quote($column) . ', ' . $this->quote($replace_search) . ', ' . $thi
s->quote($replace_replacement) . ')';
                }
            }
            $replace_query = implode(', ', $replace_query);
            $where = $search;
        } else {
            if (is_array($search)) {
                $replace_query = array();
                foreach ($search as $replace_search => $replace_replaceme
nt) {
                    $replace_query[] = $columns . ' = REPLACE(' . $this->
column_quote($columns) . ', ' . $this->quote($replace_search) . ', ' . $t
his->quote($replace_replacement) . ')';
                }
                $replace_query = implode(', ', $replace_query);
                $where = $replace;
            } else {
                $replace_query = $columns . ' = REPLACE(' . $this->column
_quote($columns) . ', ' . $this->quote($search) . ', ' . $this->quote($re
place) . ')';
            }
        }
        return $this->exec('UPDATE "' . $this->prefix . $table . '" SET '
 . $replace_query . $this->where_clause($where));
    }

    /**
     * 从表中返回一行数据
     * @param $table
     * @param null $join
     * @param null $column
     * @param null $where
     * @return bool
     */
    public function get($table, $join = null, $column = null, $where = nu
ll) {
        $query = $this->query($this->select_context($table, $join, $colum
n, $where) . ' LIMIT 1');
        if ($query) {
```

```
            $data = $query->fetchAll(PDO::FETCH_ASSOC);
            if (isset($data[0])) {
                $column = $where == null ? $join : $column;
                if (is_string($column) && $column != '*') {
                    return $data[0][$column];
                }
                return $data[0];
            } else {
                return false;
            }
        } else {
            return false;
        }
    }

    /**
     * 验证数据是否存在
     * @param $table
     * @param $join
     * @param null $where
     * @return bool
     */
    public function has($table, $join, $where = null) {
        $column = null;
        $query = $this->query('SELECT EXISTS(' . $this->select_context($t
able, $join, $column, $where, 1) . ')');
        if ($query) {
            return $query->fetchColumn() === '1';
        } else {
            require false;
        }
    }

    /**
     * 统计符合条件的数据行数
     * @param $table
     * @param null $join
     * @param null $column
     * @param null $where
     * @return bool|int
     */
    public function count($table, $join = null, $column = null, $where =
null) {
        $query = $this->query($this->select_context($table, $join, $colum
n, $where, 'COUNT'));
        return $query ? 0 + $query->fetchColumn() : false;
    }

    /**
     * 获取表中值最大的数据
     * @param $table
     * @param $join
     * @param null $column
     * @param null $where
     * @return bool|int|string
     */
```

```
    public function max($table, $join, $column = null, $where = null) {
        $query = $this->query($this->select_context($table, $join, $colum
n, $where, 'MAX'));
        if ($query) {
            $max = $query->fetchColumn();
            return is_numeric($max) ? $max + 0 : $max;
        } else {
            return false;
        }
    }

    /**
     * 获取表中值最小的数据
     * @param $table
     * @param $join
     * @param null $column
     * @param null $where
     * @return bool|int|string
     */
    public function min($table, $join, $column = null, $where = null) {
        $query = $this->query($this->select_context($table, $join, $colum
n, $where, 'MIN'));
        if ($query) {
            $min = $query->fetchColumn();
            return is_numeric($min) ? $min + 0 : $min;
        } else {
            return false;
        }
    }

    /**
     * 获得某个列字段的平均值
     * @param $table
     * @param $join
     * @param null $column
     * @param null $where
     * @return bool|int
     */
    public function avg($table, $join, $column = null, $where = null) {
        $query = $this->query($this->select_context($table, $join, $colum
n, $where, 'AVG'));
        return $query ? 0 + $query->fetchColumn() : false;
    }

    /**
     * 获得某个列字段的和
     * @param $table
     * @param $join
     * @param null $column
     * @param null $where
     * @return bool|int
     */
    public function sum($table, $join, $column = null, $where = null) {
        $query = $this->query($this->select_context($table, $join, $colum
n, $where, 'SUM'));
        return $query ? 0 + $query->fetchColumn() : false;
```

```
    }

    /**
     * 启动一个事务
     * @param $actions 事务内执行的方法
     * @return bool
     */
    public function action($actions) {
        if (is_callable($actions)) {
            $this->pdo->beginTransaction();
            $result = $actions($this);
            if ($result === false) {
                $this->pdo->rollBack();
            } else {
                $this->pdo->commit();
            }
        } else {
            return false;
        }
    }

    /**
     * 开启调式模式,只输出SQL不执行
     * 如:$medoo->debug()->select(...)
     * @return $this
     */
    public function debug() {
        $this->debug_mode = true;
        return $this;
    }

    /**
     * 获得最后一个执行的错误.
     * @return array
     */
    public function error() {
        return $this->pdo->errorInfo();
    }

    /**
     * 返回最后一条执行的SQL语句
     * @return mixed
     */
    public function last_query() {
        return end($this->logs);
    }

    /**
     * 返回当前页面执行的所有查询SQL
     * @return array
     */
    public function log() {
        return $this->logs;
    }

    /**
```

```
     * 获得当前所连接数据库的信息
     * @return array
     */
    public function info() {
        $output = array('server' => 'SERVER_INFO', 'driver' => 'DRIVER_NA
ME', 'client' => 'CLIENT_VERSION', 'version' => 'SERVER_VERSION', 'connec
tion' => 'CONNECTION_STATUS');
        foreach ($output as $key => $value) {
            $output[$key] = $this->pdo->getAttribute(constant('PDO::ATTR_
' . $value));
        }
        return $output;
    }
}
```

new medoo()

开始
使用Medoo是非常简单的事!

要求
- PHP 5.1+, 推荐 PHP 5.4+ ， PDO 支持.
- 支持 MySQL, MSSQL, SQLite 等数据库.
- 如果使用 php_pdo_xxx (xxx = 数据库类型) 你需要在 php.ini中启用相关扩展.
- 需要懂一些SQL语法.

Tips
在 PHP 5.4+ 中你可以使用 [] 作为参数，否则只能使用 array().

```
// On PHP 5.1
$data = array("foo", "bar");

// On PHP 5.4+
$data = ["foo", "bar"];
```

Php_pdo　扩展列表

> - MySQL, MariaDB -> php_pdo_mysql
> - MSSQL (Windows) -> php_pdo_sqlsrv
> - MSSQL (Liunx/UNIX) -> php_pdo_dblib
> - Oracle -> php_pdo_oci
> - SQLite -> php_pdo_sqlite
> - PostgreSQL -> php_pdo_pgsql
> - Sybase -> php_pdo_dblib

PHP PDO安装
medoo需要PHP支持PDO扩展，请在安装相关扩展后继续以下操作

```
// 打开php.ini找到你想要的相应扩展,去掉前面的;号即可
// 将
;extension=php_pdo_mysql.dll
// 修改成
```

```
extension=php_pdo_mysql.dll
// 保存，重启你的PHP或者服务器
//如果PDO安装成功，你可以通过phpinfo()查看到它.
```

> 如果你通过终端(linux)命令行安装，系统会自动安装配置相应扩展

```
$ sudo apt-get install php5-mysql
```

PHP依赖安装

> 如果你通过php自带的依赖扩展安装它，可以使用下面的命令，或者你根据自己的需要修改即可。

```
$ composer require catfan/Medoo
```

> 升级方法

```
$ composer update
```

安装源文件安装

> 这是最简单的方法，下载medoo源文件，放到你的PHP开发目录里，载入即可

```
require  'medoo.php';
```

配置

有3种方法来配置你的数据库连接.

```
$database = new medoo([
    // 必须配置项
    'database_type' => 'mysql',
    'database_name' => 'name',
    'server' => 'localhost',
    'username' => 'your_username',
    'password' => 'your_password',
    'charset' => 'utf8',

    // 可选参数
```

```
    'port' => 3306,

    // 可选，定义表的前缀
    'prefix' => 'PREFIX_',

    // 连接参数扩展，更多参考 http://www.php.net/manual/en/pdo.setattribute.
php
    'option' => [
        PDO::ATTR_CASE => PDO::CASE_NATURAL
    ]
]);

$database->insert("account", [
    "user_name" => "foo",
    "email" => "foo@bar.com"
]);
```

For MSSQL

> 如果你要使用Medoo连接你的MSSQL数据库，你需要安装相关扩展：Windows安装pdo_sqlsrv、Linux/UNIX安装pdo_dblib. pdo_mssql 扩展已被PHP废弃，不建议使用.

For SQLite

```
$database = new medoo([
    'database_type' => 'sqlite',
    'database_file' => 'my/database/path/database.db'
]);

$database->insert("account", [
    "user_name" => "foo",
    "email" => "foo@bar.com"
]);
```

where

WHERE 语句

SQL中使用where可能会有一些不安全的动态参数传入或者一些复杂的SQL语句,但是Medoo提供非常简介和安全的方法来实现这些 .

基础使用

在基础使用中. 你可以使用一些符号对参数进行过滤

```
$database->select("account", "user_name", [
    "email" => "foo@bar.com"
]);
// WHERE email = 'foo@bar.com'

$database->select("account", "user_name", [
    "user_id" => 200
]);
// WHERE user_id = 200

$database->select("account", "user_name", [
    "user_id[>]" => 200
]);
// WHERE user_id > 200

$database->select("account", "user_name", [
    "user_id[>=]" => 200
]);
// WHERE user_id >= 200

$database->select("account", "user_name", [
    "user_id[!]" => 200
]);
// WHERE user_id != 200

$database->select("account", "user_name", [
    "age[<>]" => [200, 500]
]);
// WHERE age BETWEEN 200 AND 500

$database->select("account", "user_name", [
    "age[><]" => [200, 500]
]);
// WHERE age NOT BETWEEN 200 AND 500

// [><] 和 [<>] 可以用于 datetime
$database->select("account", "user_name", [
    "birthday[><]" => [date("Y-m-d", mktime(0, 0, 0, 1, 1, 2015)), date("
Y-m-d")]
```

```
]);
//WHERE "create_date" BETWEEN '2015-01-01' AND '2015-05-01' (now)

// 你不仅可以使用字符串和数字，还可以使用数组
$database->select("account", "user_name", [
    "OR" => [
        "user_id" => [2, 123, 234, 54],
        "email" => ["foo@bar.com", "cat@dog.com", "admin@medoo.in"]
    ]
]);
// WHERE
// user_id IN (2,123,234,54) OR
// email IN ('foo@bar.com','cat@dog.com','admin@medoo.in')

// 多条件查询
$database->select("account", "user_name", [
    "AND" => [
        "user_name[!]" => "foo",
        "user_id[!]" => 1024,
        "email[!]" => ["foo@bar.com", "cat@dog.com", "admin@medoo.in"],
        "city[!]" => null,
        "promoted[!]" => true
    ]
]);
// WHERE
// `user_name` != 'foo' AND
// `user_id` != 1024 AND
// `email` NOT IN ('foo@bar.com','cat@dog.com','admin@medoo.in') AND
// `city` IS NOT NULL
// `promoted` != 1

// 或者嵌套 select() ak  get() 方法
$database->select("account", "user_name", [
    "user_id" => $database->select("post", "user_id", ["comments[>]" => 4
0])
]);
// WHERE user_id IN (2, 51, 321, 3431)
```

条件搜索

你可以使用"AND" 或 "OR" 来拼接非常复杂的SQL语句

```
// 基础使用
$database->select("account", "user_name", [
    "AND" => [
        "user_id[>]" => 200,
        "age[<>]" => [18, 25],
        "gender" => "female"
    ]
]);
// WHERE user_id > 200 AND age BETWEEN 18 AND 25 AND gender = 'female'

$database->select("account", "user_name", [
    "OR" => [
```

```
        "user_id[>]" => 200,
        "age[<>]" => [18, 25],
        "gender" => "female"
    ]
]);
// WHERE user_id > 200 OR age BETWEEN 18 AND 25 OR gender = 'female'

// 复合条件
$database->has("account", [
    "AND" => [
        "OR" => [
            "user_name" => "foo",
            "email" => "foo@bar.com"
        ],
        "password" => "12345"
    ]
]);
// WHERE (user_name = 'foo' OR email = 'foo@bar.com') AND password = '123
45'

// 注意
// 因为medoo使用的是数组传参，所以下面这种用法是错误的。
$database->select("account", '*', [
    "AND" => [
        "OR" => [
            "user_name" => "foo",
            "email" => "foo@bar.com"
        ],
        "OR" => [
            "user_name" => "bar",
            "email" => "bar@foo.com"
        ]
    ]
]);
// [X] SELECT * FROM "account" WHERE ("user_name" = 'bar' OR "email" = 'b
ar@foo.com')

// 正确的方式是使用如下方式定义复合条件
$database->select("account", '*', [
    "AND #Actually, this comment feature can be used on every AND and OR
relativity condition" => [
        "OR #the first condition" => [
            "user_name" => "foo",
            "email" => "foo@bar.com"
        ],
        "OR #the second condition" => [
            "user_name" => "bar",
            "email" => "bar@foo.com"
        ]
    ]
]);
// SELECT * FROM "account"
// WHERE (
//  (
//      "user_name" = 'foo' OR "email" = 'foo@bar.com'
//  )
```

```
//  AND
//  (
//      "user_name" = 'bar' OR "email" = 'bar@foo.com'
//  )
// )
```

模糊匹配 Like

LIKE 使用语法 [~] .

```
// 默认情况下，使用%在前后包含关键词
$database->select("person", "id", [
    "city[~]" => "lon"
]);

WHERE "city" LIKE '%lon%'

// 数组形式，查询多个关键词
$database->select("person", "id", [
    "city[~]" => ["lon", "foo", "bar"]
]);

WHERE "city" LIKE '%lon%' OR "city" LIKE '%foo%' OR "city" LIKE '%bar%'

// 不包含 [!~]
$database->select("person", "id", [
    "city[!~]" => "lon"
]);

WHERE "city" NOT LIKE '%lon%'

// 使用SQL自带的一些通配符
// 你可以使用sql自带的一些通配符来完成较复杂的查询
$database->select("person", "id", [
    "city[~]" => "stan%" // Kazakhstan,  Uzbekistan, Türkmenistan
]);

$database->select("person", "id", [
    "city[~]" => "Londo_" // London, Londox, Londos...
]);

$database->select("person", "id", [
    "name[~]" => "[BCR]at" // Bat, Cat, Rat
]);

$database->select("person", "id", [
    "name[~]" => "[!BCR]at" // Eat, Fat, Hat...
]);
```

排序使用

```
$database->select("account", "user_id", [
```

```
    // "ORDER" => "age DESC"
    "ORDER" => "age",

]);
//  SELECT user_id FROM account
//  ORDER BY age

// 多个排序
$database->select("account", "user_id", [

    "ORDER" => ['user_name DESC', 'user_id ASC']

]);
//  SELECT user_id FROM account
//  ORDER BY "user_name" DESC, "user_id" ASC


// 根据字段自定义排序顺序
// "ORDER" => array("column_name", [array #ordered array])
$database->select("account", "user_id", [

    "user_id" => [1, 12, 43, 57, 98, 144],

    "ORDER" => ["user_id", [43, 12, 57, 98, 144, 1]]

]);
// SELECT "user_id"
// FROM "account"
// WHERE "user_id" IN (1,12,43,57,98,144)
// ORDER BY FIELD("user_id", 43,12,57,98,144,1)

// array(6) {
//  [0]=> string(2) "43"
//  [1]=> string(2) "12"
//  [2]=> string(2) "57"
//  [3]=> string(2) "98"
//  [4]=> string(3) "144"
//  [5]=> string(1) "1"
// }
```

全文检索

```
// [MATCH]
$database->select("post_table", "post_id", [
    "MATCH" => [
        "columns" => ["content", "title"],
        "keyword" => "foo"
    ]
]);
// WHERE MATCH (content, title) AGAINST ('foo')
```

使用SQL函数

在一些特殊的情况下，你可能需要使用SQL系统函数，只需要字段名前加上#号即可

```
$data = $database->select('account', [
    'user_id',
    'user_name'
], [
    '#datetime' => 'NOW()'
]);

// SELECT "user_id","user_name"
// FROM "account"
// WHERE "datetime" = NOW()

// [IMPORTANT] Keep in mind that, the value will not be quoted should be
matched as XXX() uppercase.
// The following sample will be failed.
$database->select('account', [
    'user_id',
    'user_name'
], [
    '#datetime2' => 'now()',

    'datetime3' => 'NOW()',

    '#datetime4' => 'NOW'
]);
```

附加条件

```
$database->select("account", "user_id", [
    "GROUP" => "type",

    // Must have to use it with GROUP together
    "HAVING" => [
        "user_id[>]" => 500
    ],

    // LIMIT => 20
    "LIMIT" => [20, 100]
]);
//  SELECT user_id FROM account
//  GROUP BY type
//  HAVING user_id > 500
//  LIMIT 20,100
```