
Sedge

Sedge 数据验证类(Checker)

说明文档

开源框架：作业提交平台的数据验证类(Checker)说明文档

1. 概述

本文档对数据验证类（Checker）的代码、功能、接口进行说明，便于开发者在上层开发中正确调用。

此外，**对数据验证的所有操作，必须通过 Checker 类进行**，不得通过其它方式验证。

2. 包含文件

要进行数据验证，首先包含“checker.php”文件，使用 require_once()函数确保只包含一次。

例如_login.php 要进行数据验证，在文件的开始处使用：

```
require_once('lib/checker.php');
 //假设 checker.php 位于 lib 文件夹下
```

3. 创建实例

Checker 是一个类，要进行数据验证，首先要创建实例。

```
$check = new Checker;
```

(注：“checker”是该类的保留字，禁止再次声明)

4. 类的概述

Checker 类的功能非常强大，连作者本人都不得不佩服自己，唯一不足的是 PHP 写起来并不优美，使代码变得复杂和冗余，如果能用 lisp 写的话，想必一定会非常完美。

那 Checker 到底是怎么个好法呢？首先，它内置了**丰富的正则表达式库**，能够轻松应对数值(例如整数、正整数、非负整数、负整数、非正整数、浮点数……)、日期、邮箱、用户名、IP 地址等各种数据的验证；此外，它封装了 Sedge 数据库中的所有字段名，因此能够对传入的数据**自动进行验证**，还能对 IP 和日期进行**自动转换**；最后，它提供了**多种验证接口**，用于不同情况的数据验证。

在介绍接口前，有必要了解一下 Checker 的正则表达方法。（这些都是 private 方法，不能直接调用）

	方法名	备注	功能（判断是否为）
1	is_pint	positive integer	正整数
2	is_nnint	non-negative integer	非负整数
3	is_nint	negative integer	负整数
4	is_npint	non-positive integer	非正整数
5	is_int	integer	整数
6	is_num	number	仅包含数字 0-9
7	is_pfloat	positive float	正浮点数
8	is_nnfloat	non-negative float	非负浮点数
9	is_nfloat	negative float	负浮点数
10	is_npfloat	non-positive float	非正浮点数
11	is_float	float	浮点数
12	is_alpha	alphabet	a-z、A-Z、下划线
13	is_alphanum	alphabet and numeric	alphabet、0-9
14	is_cn	chinese	汉字、下划线
15	is_cn_alphanum		表格（13)、(14）行
16	is_text		ASCII（除部分特殊字符）
17	is_cn_text		表格（14)、(16）行
18	no_html	html tag	不包含 HTML 标签
19	is_email		邮箱地址
20	is_qq		QQ 号
21	is_ip	ipv4 address	IP 地址（ipv4）
22	is_id		1-11 位的数字
23	is_date	YYYY-MM-DD hh:mm:ss	日期时间，判断日期合法性（包括闰年） hh:mm:ss 可省略 :ss 可省略

以下两个是转换方法

	方法名	备注	功能
24	ip2int	ip to int	将 ip 转成整形
25	date_to_stamp		将日期转成时间戳

还有两个判断方法

	方法名	参数	功能
26	check_size	$data, $low, $high	判断数值是否∈[low,high]
27	date_to_stamp	$data, $low, $high	判断长度是否∈[low,high]

好了， private 方法基本介绍完了，后面的方法就全靠它们了。

还要提到的是，**数据库字段名**被封装在 checker 类中（注意这是另一个类，验证类叫 Checker）
比如：

```
$checker -> is_ip -> func = array('is_ip','ip2int');
```

表示对数据库的 ip 字段，依次用 is_ip、ip2int 方法进行验证。

```
$checker -> is_user_id -> len = array(5,50);
```

表示数据库的 user_id 字段，长度应介于 5 到 15（闭区间）。

4. 类的接口

总的来说，Checker 类提供了 3 种进行数据验证的方法。我们将依次来介绍。

（1） check 方法

首先要介绍的就是最最强大的 check 方法，它主要用于数据库字段的**自动验证**。

它的参数列表是：

```
& $data , [$func , $len_low , $len_high]
```

可以看出它有 4 个参数，后 3 个参数是可省略的。

a. & $data

第 1 个参数，当然是要验证的数据了。

注意到它是一个引用，意味着数据可能会被修改（比如调用了转换方法）；还有它必须是变量，不能是常量。

$data 可以是数组，也可以是字符串，如果是字符串，那么必须提供第 2 个参数。

1） $data 是数组

如果它是一个关联数组，即键名就是字段名，那么就能进行自动验证。

程序会自动匹配键名、验证它的长度、调用正则方法验证它的格式。如果验证失败则立即停止，并返回 false，用户通过调用 info()方法查看错误信息。

如果键名不是字段名，它可以是一个方法名（之前的 private 方法），程序会自动调用那个方法进行验证。

如果键名既不是字段名，也不是方法名，此时就需要第 2 个参数$func 来指定方法。否则会引发错误。

2） $data 是字符串

如果它是字符串，相当于它没有键名，那么同样需要第 2 个参数$func 来指定

方法。否则会引发错误。

b. $func

这个参数必须是一个数组（即使它只有一个方法，也应该写成 array(‘method_name’);

它用于上述那种键名既不是字段名也不是方法名的情况。此时程序会利用 $func 参数中提供的方法进行验证。

c. $len_low , $len_high

这两个参数必须一起出现，它用于指定**数据长度**的下限和上限。

它适用于**键名不是字段名**的情况（因为字段名的验证包含了长度和格式），也就是说，对于键名是方法名、键名既不是字段名也不是方法名、没有键名的情况，程序会根据这两个参数验证它们的长度。

这个方法讲完了，貌似不太好理解。举个例子吧。

假设有以下数据需要验证：

```
$data = array(
    'user_login' => 'weitao201@163.com',
    'user_pass' => '123456',
    'user_id' => '1091',
    'is_qq' => '44300',
    'hahaha' => '这个家伙很懒...',
)
```

这是一个数组，其中前 3 个键名都是数据库中的字段，我们可以确定它们会被自动验证；倒数第二个是方法名（之前的表格第 20 行），同样可以确定它会被 is_qq 方法验证；而最后一个既不是字段名也不是方法名，需要我们指定验证方法，它是一个中文字符串，我们就用 is_cn_text 方法来验证。

因此，调用方法就是

```
$check = new Checker;
$check -> check($data, array('is_cn_text'));
```

另外，如果我们指定了$len_low , $len_high， 那么数组的后两个元素都会被验证。

（2） check_array 方法

可以看出 check 方法主要是对字段名进行自动验证，对于自定义验证的支持并不够强大。所以有了 check_array 方法，从它的名字就可以猜出，它是对数组元素进行验证。（不支持字段名自动验证）

它的参数列表是：

& $data , $pattern_array , [$len_array]

它有 3 个参数，最后 1 个参数是可省略的。

它的 3 个参数全都是数组，并且是一一对应的。（3 个参数的第一个元素相对应，第二个元素相对应…）

a. & $data

没什么好说的，数组，存放要验证的数据。

b. $pattern_array

数组，描述了对应数据的验证方法。例如第 0 个元素值的是’is_ip’，表示$data 的第 0 个元素用 is_ip 方法进行验证。

元素值也可以是数组，比如第 2 个元素值是 array(‘is_text’,’no_html’)，表示$data 的第 2 个元素用 is_text 和 no_html 方法进行验证。

如果第 n 个数据不想被验证，将它的元素值赋为 null 或空数组 array()。

c. $len_array

数组，描述了对应数据的长度范围,元素值必须是数组。例如第 0 个元素值是 array(5,6)，表示$data 的第 0 个元素的长度是 5 到 6。

如果第 n 个数据不想被验证，将它的元素值赋为 null 或 array(0,0)。

介绍完毕，这个方法比较好懂，就不举例了。

（3） check_one 方法

以上两个方法功能虽然强大，但过于复杂，并且没有充分发挥面向对象的优点。那有没有“轻量级”的方法呢？

利用 check_one 方法，能够进行简单的验证，还能构造“验证器”实例。

它只有一个参数 & $data ， 就是要验证的数据，数据必须是字符串。

在验证之前，必须调用以下公共方法进行设置：

	方法名	参数	功能
1	set_size	$low, $high	设置数值范围
2	set_len	$low, $high	设置长度范围
3	set_pattern	$preg	传入一个字符串或数组，设置验证方法
4	clear		清除所有设置

不必全部调用，调用 1、2、3 的任一（两）个也可以。

设置好后，check_one 就能工作啦，一次验证一个数据。

举个例子，比如我们只想验证数值大小在 100 到 1000 之间。(注意，只有 check_one 方法支持数值大小验证)

```
$size_check = new Checker;
$size_check -> set_size(100,1000);
$size_check -> check($data);
```

好了，至此 3 个方法都讲完了，还有最后一个 public 方法要再提一下

	方法名	参数	功能
1	info		返回错误信息

对于之前的 3 个验证方法（check、check_array、check_one），如果验证出错都会返回 false 并写错误信息。

此时调用 info 方法就可以查看到错误信息。

虽然还没对代码进行过完整的测试（orz），相信 Checker 类一定能胜任数据验证的工作！

5. 版本与声明

5.1 版本

本文档名称：《Sedge 数据验证类(Checker)说明文档》

版本：2014.2.26

5.2 声明

本文档以**最新版**为准，最新版发布在 github.com/ng1091/Sedge 及 Sedge 开发 QQ 群，群号 185288022。