RespectValidation

Respect/Validation 是 PHP 中功能强大的数据验证库，支持链式调用、多规则组合、自定义验证规则等特性，适用于表单验证、API 参数校验等场景。以下是其核心使用方法：

一、安装

通过 Composer 安装：

```
composer require respect/validation
```

二、基础使用

1. 简单验证（单个规则）

```
use Respect\Validation\Validator as v;
// 验证邮箱格式
$email = 'test@example.com';
v::email()->validate($email); // 返回 true/false
// 验证失败时抛出异常
try {
    v::email()->assert($email); // 验证通过无返回，失败则抛 ValidationException
} catch (Respect\Validation\Exceptions\ValidationException $e) {
    echo $e->getMessage(); // 输出错误信息
}
```

2. 链式验证（多规则组合）

```
// 验证密码：长度6-20位，包含字母和数字
$password = 'Pass1234';
v::length(6, 20)
    ->alnum()
    ->containsLetter()
```

```
    ->containsDigit()
    ->assert($password);
```

三、常用验证规则

1. 数据类型验证

```
v::stringType(); // 字符串类型
v::intVal();    // 整数类型
v::floatVal();  // 浮点类型
v::boolVal();   // 布尔类型
v::arrayVal();  // 数组类型
```

2. 字符串验证

```
v::email();        // 邮箱格式
v::url();          // URL格式
v::regex('/^[a-z]+$/'); // 正则匹配
v::length(3, 10);   // 长度3-10位
v::notEmpty();      // 非空
v::lowercase();     // 全小写
v::uppercase();     // 全大写
```

3. 数值验证

```
v::numericVal();    // 数值类型
v::between(18, 60); // 数值在18-60之间
v::positive();      // 正数
v::negative();      // 负数
v::equals(100);     // 等于100
```

4. 数组 / 集合验证

```
```

```
v::arrayVal()->length(2); // 数组长度为2
v::contains('apple');    // 包含元素'apple'
v::key('username', v::stringType()); // 数组包含'username'键且值为字符串
```

5. 日期验证

```
v::date();         // 有效日期（如'2024-01-01'）
v::date('Y-m-d');  // 指定日期格式
v::between('2020-01-01', '2024-12-31'); // 日期范围
```

四、复杂验证场景

1. 表单验证（多字段）

```
// 验证用户注册表单
$userData = [
    'username' => 'test_user',
    'email'    => 'test@example.com',
    'password' => 'Pass1234',
    'age'      => 25
];
v::key('username', v::stringType()->length(3, 20))
    ->key('email', v::email())
    ->key('password', v::length(6, 20)->alnum())
    ->key('age', v::intVal()->between(18, 60))
    ->assert($userData);
```

2. 条件验证（`when` 规则）

```
// 如果是学生，验证学号；否则验证工号
$person = [
    'type' => 'student',
    'id'   => 'S12345'
];
v::key('type', v::equals('student'))
```

```
    ->when(
        v::key('type', v::equals('student')),
        v::key('id', v::regex('/^S\d+$/')), // 学生学号规则
        v::key('id', v::regex('/^W\d+$/'))  // 员工工号规则
    )
    ->assert($person);
```

3. 自定义验证规则

```
// 自定义规则：验证是否为偶数
v::with('MyValidationRules'); // 注册自定义规则命名空间
class MyValidationRules
{
    public static function even($input)
    {
        return $input % 2 === 0;
    }
}
v::even()->assert(4); // 验证通过
```

五、错误信息处理

1. 获取详细错误信息

```
try {
    v::email()->assert('invalid-email');
} catch (Respect\Validation\Exceptions\ValidationException $e) {
    // 获取所有错误信息（数组）
    print_r($e->getMessages());

    // 获取格式化错误信息
    echo $e->getFullMessage();
}
```

2. 自定义错误消息

```
v::email()->setName('邮箱')->assert('invalid-email');
// 错误信息："邮箱必须是有效的电子邮件地址"
```

六、结合 MVC 控制器使用

```
// 控制器中验证表单提交
public function register(Request $request)
{
    $data = $request->post();

    try {
        // 验证规则
        v::key('username', v::stringType()->notEmpty()->length(3, 20))
            ->key('email', v::email())
            ->key('password', v::length(6, 20)->alnum())
            ->assert($data);

        // 验证通过，执行注册逻辑
        $this->userModel->create($data);
        $this->redirect('/login');
    } catch (ValidationException $e) {
        // 验证失败，返回错误信息到视图
        $this->view('register', [
            'errors' => $e->getMessages()
        ]);
    }
}
```

核心优势

1. **链式调用**：规则组合简洁直观；
2. **丰富的内置规则**：覆盖大部分验证场景；
3. **灵活的错误处理**：支持自定义消息和多语言；
4. **可扩展性**：轻松实现自定义验证规则。

Respect/Validation 能显著简化 MVC 项目中的数据验证逻辑，提升代码可读性和健壮性。

> （注：文档部分内容可能由 AI 生成）