JSON-Schema-PHP 2.0 用户手册

moxie(system128@gmail.com)

出版日期 2010-10-10~2012-10-03

目录

项目介绍

这是一个JSON 结构验证 PHP 实现。JSON Schema 用于描述JSON数据的结构和类型。如同DTD与XML的关系。本实现用于使用 PHP 调用 JSON Schema 对 JSON 数据进行验证。 包括 PHP 和 PHP扩展 ，建议PHP版本 PHP 5 >= 5.2.0、PECL json >= 1.2.0。

警告

JSON Schema 标准尚未定稿，实际开发生产请酌情施用。

下载

https://github.com/downloads/zoeey/json-schema-php/json-schema-php_latest.zip

版本库

```
git clone git://github.com/zoeey/json-schema-php.git
```

主页

https://github.com/zoeey/json-schema-php

第 1 章 快速入门

简单示例

```
$value = 'teststring';
$schema = array (
  'type' => 'string',  /* 数据类型为字符串 */
  'minLength' => 3,   /* 最小长度 */
  'maxLength' => 300, /* 最大长度 */
);

$jsonSchema = new JsonSchema();
echo assert($jsonSchema->validate($schema, $value));
```

例 1.1. 简单示例

生成 JSON Schema

由JSON生成一个全格式的Schema，方便编辑修改（勿随便直接使用在实践中）。

```
$value = new stdClass();
$value->name = 'a name';
$value->age = 23;
$value->height = 183.5;
$jsonSchema = new JsonSchema();
var_dump($jsonSchema->getSchema($value));
```

例 1.2. 生成 JSON Schema

```
array(
    'type' => 'object',
    'properties' =>
    array(
        'name' =>
        array(
            'type' => 'string',
            'format' => 'regex',
            'pattern' => '/^[a-z0-9]+$/i',
            'minLength' => 0,
            'maxLength' => 2147483647,
        ),
        'age' =>
        array(
            'type' => 'integer',
            'default' => 0,
            'minimum' => 0,
            'maximum' => 2147483647,
            'exclusiveMinimum' => 0,
            'exclusiveMaximum' => 2147483647,
        ),
        'height' =>
        array(
            'type' => 'number',
            'default' => 0,
            'minimum' => 0,
            'maximum' => 2147483647,
            'exclusiveMinimum' => 0,
            'exclusiveMaximum' => 2147483647,
        )
    )
)
```

例 1.3. 结果

使用 JSON Schema 验证 JSON

验证一个由 User(account,email) 对象组成的数组。

```
$value = new stdClass();
$value->name = 'a name';
$value->age = 23;
$value->height = 183.5;

$schema = array(
    'type' => 'object',
    'properties' =>
    array(
        'name' =>
        array(
            'type' => 'string',
            'format' => 'regex',
            'pattern' => '/^[a-z0-9 ]+$/i',
            'minLength' => 0,
            'maxLength' => 2147483647,
        ),
        'age' =>
        array(
            'type' => 'integer',
            'default' => 0,
            'minimum' => 0,
            'maximum' => 2147483647,
            'exclusiveMinimum' => 0,
            'exclusiveMaximum' => 2147483647,
        ),
        'height' =>
        array(
            'type' => 'number',
            'default' => 0,
            'minimum' => 0,
            'maximum' => 2147483647,
            'exclusiveMinimum' => 0,
            'exclusiveMaximum' => 2147483647,
        )
    )
);

$jsonSchema = new JsonSchema();
var_dump($jsonSchema->validate($schema, $value)), PHP_EOL;
```

例 1.4. 验证 JSON

第 2 章 Schema 生成

2.1. 各类型生成

string

```
$value = 'test string';

$jsonSchema = new JsonSchema();
var_export($jsonSchema->getSchema($value));

----RESULT---

array(
    'type' => 'string',
    'format' => 'regex',
    'pattern' => '/^[a-z0-9]+$/i',
    'minLength' => 0,
    'maxLength' => 2147483647
)
```

例 2.1. 生成代码与结果

number

```
$value = 123.321;

$jsonSchema = new JsonSchema();
var_export($jsonSchema->getSchema($value));

----RESULT---

array(
    'type' => 'number',
    'default' => 0,
    'minimum' => 0,
    'maximum' => 2147483647,
    'exclusiveMinimum' => 0,
    'exclusiveMaximum' => 2147483647,
)
```

例 2.2. 生成代码与结果

integer

```
$value = 123;

$jsonSchema = new JsonSchema();
var_export($jsonSchema->getSchema($value));

----RESULT---

array(
    'type' => 'integer',
    'default' => 0,
    'minimum' => 0,
    'maximum' => 2147483647,
    'exclusiveMinimum' => 0,
    'exclusiveMaximum' => 2147483647
)
```

例 2.3. 生成代码与结果

boolean

```
$value = true;

$jsonSchema = new JsonSchema();
var_export($jsonSchema->getSchema($value));

----RESULT---

array(
    'type' => 'boolean'
    , 'default' => false
)
```

例 2.4. 生成代码与结果

object

```
$value = new stdClass();
$value->name = 'a name';
$value->age = 23;
$value->height = 183.5;

$jsonSchema = new JsonSchema();
var_export($jsonSchema->getSchema($value));

----RESULT---

array(
    'type' => 'object',
    'properties' =>
    array(
        'name' =>
        array(
            'type' => 'string',
            'format' => 'regex',
            'pattern' => '/^[a-z0-9]+$/i',
            'minLength' => 0,
            'maxLength' => 2147483647,
        ),
        'age' =>
        array(
            'type' => 'integer',
            'default' => 0,
            'minimum' => 0,
            'maximum' => 2147483647,
            'exclusiveMinimum' => 0,
            'exclusiveMaximum' => 2147483647,
        ),
        'height' =>
        array(
            'type' => 'number',
            'default' => 0,
            'minimum' => 0,
            'maximum' => 2147483647,
            'exclusiveMinimum' => 0,
            'exclusiveMaximum' => 2147483647,
        )
    )
)
```

例 2.5. 生成代码与结果

array（map）

```
$value = array();
$value['name'] = 'a name';
$value['age'] = 23;
$value['height'] = 183.5;

$jsonSchema = new JsonSchema();
var_export($jsonSchema->getSchema((object)$value));

----RESULT---

array(
    'type' => 'object',
    'properties' =>
    array(
        'name' =>
        array(
            'type' => 'string',
            'format' => 'regex',
            'pattern' => '/^[a-z0-9 ]+$/i',
            'minLength' => 0,
            'maxLength' => 2147483647
        ),array(
    'type' => 'object',
    'properties' =>
    array(
        'name' =>
        array(
            'type' => 'string',
            'format' => 'regex',
            'pattern' => '/^[a-z0-9]+$/i',
            'minLength' => 0,
            'maxLength' => 2147483647
        ),
        'age' =>
        array(
            'type' => 'integer',
            'default' => 0,
            'minimum' => 0,
            'maximum' => 2147483647,
            'exclusiveMinimum' => 0,
            'exclusiveMaximum' => 2147483647
        ),
        'height' =>
        array(
            'type' => 'number',
            'default' => 0,
            'minimum' => 0,
            'maximum' => 2147483647,
            'exclusiveMinimum' => 0,
            'exclusiveMaximum' => 2147483647
        )
    )
)
```

例 2.6. 生成代码与结果

array (list)

```
$value = array();
$value[] = 'a name';
$value[] = 23;
$value[] = 183.5;

$jsonSchema = new JsonSchema();
var_export($jsonSchema->getSchema($value));

----RESULT---

array(
    'type' => 'array',
    'minItems' => 0,
    'maxItems' => 20,
    'items' =>
    array(
        'type' => 'string',
        'format' => 'regex',
        'pattern' => '/^[a-z0-9]+$/i',
        'minLength' => 0,
        'maxLength' => 2147483647,
    )
)
```

例 2.7. 生成代码与结果

array list<object>

例 2.8. 生成代码与结果

```
$value = new stdClass();
$value->users = array();

$user = new stdClass();
$user->id = 1;
$user->account = 'userA';
$value->users[] = $user;

$user = new stdClass();
$user->id = 3;
$user->account = 'userB';
$value->users[] = $user;

$user = new stdClass();
$user->id = 5;
$user->account = 'userC';
$value->users[] = $user;

$jsonSchema = new JsonSchema();
var_export($jsonSchema->getSchema($value));

----RESULT---

array(
    'type' => 'object',
    'properties' =>
    array(
        'users' =>
        array(
            'type' => 'array',
            'minItems' => 0,
            'maxItems' => 20,
            'items' =>
            array(
                'type' => 'object',
                'properties' =>
                array(
                    'id' =>
                    array(
                        'type' => 'integer',
                        'default' => 0,
                        'minimum' => 0,
                        'maximum' => 2147483647,
                        'exclusiveMinimum' => 0,
                        'exclusiveMaximum' => 2147483647,
                    ),
                    'account' =>
                    array(
                        'type' => 'string',
                        'format' => 'regex',
                        'pattern' => '/^[a-z0-9]+$/i',
                        'minLength' => 0,
                        'maxLength' => 2147483647,
                    )
                )
            )
        )
    )
)
```

第 3 章 JSON 验证

3.1. type 类型

string
: 值必须为字符串。

number
: 值必须为数字，包括 整型和浮点型。

integer
: 值必须为整型数字。

boolean
: 值必须为布尔型。

object
: 值必须为对象。

array
: 值必须为数组。关联数组应使用 object 进行描述。

null
: 值必须为 null。标准中规定，null仅存在于多类型集合时可为null。本实现 null 可用于描述任意位置。

any
: 任意类型，包括 null。

$ref
: 自定义复合类型。

3.2. string 类型

minLength
: 字符串最小长度。

maxLength
: 字符串最小长度。

format

特殊格式信息

1. date-time ： ISO 8601 格式 YYYY-MM-DDThh:mm:ssZ UTC 时间。
2. date ： YYYY-MM-DD 日期。
3. time ： hh:mm:ss 时间。
4. utc-millisec ： UNIX 时间戳，可为整型和浮点型。
5. regex ： 正则匹配，符合 ECMA 262/Perl 5，需要与 pattern 属性配合。
6. color ： 颜色。符合 CSS 2.1 规范 [W3C.CR-CSS21-20070719 [www.w3.org/TR/2007/CR-CSS21-20070719/]][Color [www.w3.org/TR/2007/CR-CSS21-20070719/syndata.html#value-def-color]]。格式示例：#rrggbb、#rgb、rgb(255,255,255).rgb(100%,100%,100%)。
7. style ： CSS 样式格式（尚未实现）。
8. phone ： 电话号码，现基于 E.123 [http://en.wikipedia.org/wiki/E.123]。
9. uri ： URI。现仅验证为 URL rfc2396 [http://www.faqs.org/rfcs/rfc2396]。
10. email ：邮箱地址。
11. ip-address ： IPv4地址。
12. ipv6 ： IPv6地址。
13. host / host-name ： [Host rfc810 [http://tools.ietf.org/html/rfc810]]

pattern
正则表达式。

```
array (
    'type' => 'string',
    'format' => 'regex',
    'pattern' => '/^[a-z0-9]+$/i',
    'minLength' => 0,
    'maxLength' => 2147483647,
)
```

例 3.1. 正则验证

```
array (
    'type' => 'string',
    'format' => 'email',
)
```

例 3.2. 邮箱地址验证

3.3. number 类型

minimum
最小值。

maximum
    最大值。

exclusiveMinimum
    最小值，包含临界值。

exclusiveMaximum
    最大值，包含临界值。

```
array (
    'type' => 'number',
    'minimum' => 1,
    'maximum' => 100,
)
```

例 3.3. 1～100 之间的数字(不包括1和100)

```
array (
    'type' => 'number',
    'exclusiveMinimum' => 1,
    'exclusiveMaximum' => 100,
)
```

例 3.4. 1～100 之间的数字（包括1和100）

3.4. integer 类型

验证为整型，属性继承 number类型。

3.5. boolean 类型

无格式限制属性，验证为布尔型。

3.6. object 类型

properties
    指定对象所含属性和属性的类型。

required
    用于描述属性是否为必有。

```
array(
    'name' =>
    array(
        'type' => 'string',
        'format' => 'regex',
        'pattern' => '/^[a-z0-9 ]+$/i',
        'minLength' => 0,
        'maxLength' => 5000,
    ),
    'age' =>
    array(
        'type' => 'integer',
        'exclusiveMinimum' => 1,
        'exclusiveMaximum' => 200,
        'required' => false,
    ),
    'height' =>
    array(
        'type' => 'number',
        'minimum' => 110,
        'maximum' => 230,
    )
)
```

例 3.5. 一个Person对象，拥有 name,age,height 三个属性，其中age可选

3.7. array 类型

items
: 指定数组元素的验证描述。

minItems
: 数组单元数目最小值。

maxItems
: 数组单元数目最大值。

3.8. null 类型

```
array(
    'type' =>
    array(
        'boolean',
        'integer',
        'null'
    ),
    'minimum' => 110,
    'maximum' => 230
)
```

例 3.6. null 的使用示例

3.9. any 类型

无格式限制属性，任意格式可通过验证。

3.10. $ref 类型

复合类型可以通过addType方法载入到验证过程内。

```
$value = array();
$value['users'][] = array('account' => 'userA', 'email' => 'userA@example.com');
$value['users'][] = array('account' => 'userB', 'email' => 'userB@example.com');
$value['users'][] = array('account' => 'userC', 'email' => 'userC@example.com');

$user_schema = array(
    'id' => 'user',
    'description' => 'user info',
    'type' => 'object',
    'optional' => true,
    'properties' =>
    array(
        'account' =>
        array(
            'type' => 'string',
        ),
        'email' =>
        array(
            'type' => 'string',
            'optional' => true,
        ),
    ),
);
$schema = array(
    'type' => 'object',
    'properties' =>
    array(
        'users' =>
        array(
            'type' => 'array',
            'items' =>
            array(
                '$ref' => 'user',
            )
        )
    )
);
$jsonSchema = new JsonSchema();
$jsonSchema->addType($user_schema);
echo assert($jsonSchema->validate($schema, $value)), PHP_EOL;
```

例 3.7. 复合类型的使用

第 4 章 class JsonSchema

4.1. 类摘要

```
        JsonSchema
       {
  __construct();

  array getSchema(
            mixed

            $value
          );

  void addType(
            array

            $typeSchema
          );

  boolean validate(
            array

            $schema
          ,

            mixed

            $value
          );

  array getErrors();
}
```

例 4.1. 导航

JsonSchema::__construct
    JsonSchema对象初始化。

JsonSchema::addType
    载入复合类型。

JsonSchema::getErrors
    验证失败时获取错误信息。

JsonSchema::getSchema
    生成JSON Schema，生成结果切勿直接使用。

JsonSchema::validate
    使用JsonSchema对JSON进行验证。

4.2. __construct

JsonSchema对象初始化。

描述

```
  __construct();
```

创建JsonSchema对象，载入需要进行操作的JSON数据。

范例

```
/* 需要进行验证的数据 */
$value = 'teststring';

/* json schema */
$schema = array(
    'type' => 'string',
    'format' => 'regex',
    'pattern' => '/^[a-z]+$/i',
    'minLength' => 0,
    'maxLength' => 2147483647,
);

/* 验证 */
$jsonSchema = new JsonSchema();
echo assert($jsonSchema->validate($schema, $value));
```

例 4.2. 简单验证

4.3. getSchema

生成JSON Schema。

描述

```
  array getSchema(
          mixed

          $value
        );
```

生成JSON Schema。生成结果切勿直接使用。请依照具体需求进行修改后使用。如去掉重复或冲突的描述，提取复合类型等。

参数

value
    数据示例。

返回值

返回一个与JSON结构基本相符的JSON Schema字符串。

范例

```
$value = 123.321;

$jsonSchema = new JsonSchema();
var_export($jsonSchema->getSchema($value));

----RESULT---

array(
    'type' => 'number',
    'default' => 0,
    'minimum' => 0,
    'maximum' => 2147483647,
    'exclusiveMinimum' => 0,
    'exclusiveMaximum' => 2147483647
);
```

例 4.3. 生成一个简单的JSON Schema

注意

上例很显然的可以看出， minimum 和 exclusiveMinimum 具有一定的重复性，在实际使用过程中应依据具体情况予以删减。

4.4. addType

载入复合类型。

描述

```
  void JsonSchema::addType(
          string

          $typeSchema
        );
```

载入复合类型，以id进行识别，使用$ref属性引用。

参数

typeSchema
    复合类型描述Schema。

范例

```
$value = array();
$value['users'][] = array('account' => 'userA', 'email' => 'userA@example.com');
$value['users'][] = array('account' => 'userB', 'email' => 'userB@example.com');
$value['users'][] = array('account' => 'userC', 'email' => 'userC@example.com');

$user_schema = array(
    'id' => 'user',
    'description' => 'user info',
    'type' => 'object',
    'optional' => true,
    'properties' =>
    array(
        'account' =>
        array(
            'type' => 'string',
        ),
        'email' =>
        array(
            'type' => 'string',
            'optional' => true,
        ),
    ),
);
$schema = array(
    'type' => 'object',
    'properties' =>
    array(
        'users' =>
        array(
            'type' => 'array',
            'items' =>
            array(
                '$ref' => 'user',
            )
        )
    )
);
$jsonSchema = new JsonSchema();
$jsonSchema->addType($user_schema);
echo assert($jsonSchema->validate($schema, $value)), PHP_EOL;
```

例 4.4. 使用一个复合类型对JSON数据进行验证

4.5. validate

对JSON进行验证。

描述

```
boolean validate(
        array
```

```
        $schema
    ,
        mixed
        $value
    );
```

依据 $schema 的描述对JSON进行验证。

参数

schema
    JSON Schema

value
    需要验证的数据。

返回值

验证成功返回 true ， 失败则为 false。验证失败时可通过 getErrors 方法获取具体描述信息。

范例

```
$value = new stdClass();
$value->users = array();

$user = new stdClass();
$user->id = 1;
$user->account = 'userA';
$value->users[] = $user;

$user = new stdClass();
$user->id = 3;
$user->account = 'userB';
$value->users[] = $user;

$user = new stdClass();
$user->id = 5;
$user->account = 'userC';
$value->users[] = $user;

$jsonSchema = new JsonSchema();
$schema = array(
    'type' => 'object',
    'properties' =>
    array(
        'users' =>
        array(
            'type' => 'array',
            'items' =>
            array(
                'type' => 'object',
                'properties' =>
                array(
                    'id' =>
                    array(
                        'type' => 'integer',
                        'default' => 0,
                        'minimum' => 0,
                        'maximum' => 2147483647,
                        'exclusiveMinimum' => 0,
                        'exclusiveMaximum' => 2147483647,
                    ),
                    'account' =>
                    array(
                        'type' => 'string',
                        'minLength' => 0,
                        'maxLength' => 2147483647,
                    )
                )
            )
        )
    )
);
echo assert($jsonSchema->validate($schema, $value));
$value = array();
$value['users'][] = array('id' => 1, 'account' => 'userA');
$value['users'][] = array('id' => 3, 'account' => 'userB');
$value['users'][] = array('id' => 5, 'account' => 'userC');
echo ',', assert($jsonSchema->validate($schema, $value));
```

例4.5. 复杂的数组验证

4.6. getErrors

获取错误信息。

描述

```
array JsonSchema::getErrors();
```

验证失败时获取错误描述列表。

返回值

当无错误时返回一个空数组，有错误时返回一个包含各错误描述的数组。

范例

```
$value = 'test s p a c e s string';
$schema = array(
    'type' => 'string',
    'format' => 'regex',
    'pattern' => '/^[a-z.]+$/i',
    'minLength' => 0,
    'maxLength' => 2147483647,
);
$jsonSchema = new JsonSchema();
echo assert(!$jsonSchema->validate($schema, $value));
var_export($jsonSchema->getErrors());
```

例 4.6. 验证失败时打印错误信息

术语表

术语表

JSON	JSON（Javascript Object Notation）是一种轻量级的数据交换语言，以文字为基础，且易于让人阅读。尽管JSON是在Javascript的一个子集，但JSON是独立于语言的文本格式，并且采用了类似于C语言家族的一些习惯。 来源：http://zh.wikipedia.org/wiki/JSON
JSON Schema	JSON (JavaScript Object Notation) Schema defines the media type "application/schema+json", a JSON based format for defining the structure of JSON data. JSON Schema provides a contract for what JSON data is required for a given application and how to interact with it. JSON Schema is intended to define validation, documentation, hyperlink navigation, and interaction control of JSON data. 来源：http://tools.ietf.org/html/draft-zyp-json-schema-03