
Superstruct
Javascript 运行时接口验证工具

设计原则

- 可组合的
- 可自定义
- 有用的运行时错误
- 语法自然

为什么需要运行时验证

众所周知，JS是一门对隐式类型转换容忍度极高的语言 一不注意就会写出意想不到的代码，比如

```
const a = '1';
const b = 1;

function add(x, y) {
  return (x + y)/2;
}

// some code ...
add(a,b)
```

我们这里期待函数应当是`(x:number, y:number) ⇒ number`但是实参是字符串和数字，并且数字和字符串可以相互隐式转换，从而导致意料之外的结果，而这种错误不像某些语言，是不会通过抛出的类型错误暴露给用户的

Superstruct

因此，我们需要一个工具来帮助我们验证运行时的接口,Superstruct就是为了应对这种情况而产生的工具

基本类型

最简单的情况就是基本类型的验证

```
import { string } from 'superstruct'

const Struct = string()

assert('a string', Struct) // passes
assert(42, Struct) // throws!
```

这里`assert`将会抛出一个运行时错误

```
error:
```

```
const User = object({
  id: tt(),
  email: string(),
  name: string(),
})

// passes
assert(
  {
    id: 1,
    email: 'jane@example.com',
    name: 'Jane',
  },
  User
)
```

```
// also throws! (email is missing)
assert(
  {
    id: 1,
    name: 'Jane',
  },
  User
)
```

可以将基本的类型进行组合，来构造更复杂的类型

可选

可以通过optional函数来指定某个属性可选

```
import { optional } from "superstruct"
const User = object({
  id: number(),
  name: string(),
  email: optional(string()), //可选属性
})
```

自定义验证

只进行类型验证是远远不够的， superstruct还可以添加自定义的值验证

```
import { define } from 'superstruct'
import isEmail from 'is-email'

const email = () => define('email', (value) => isEmail(value))
```

默认值

Typescript支持