02

OPEN ORIENTED

凹 凸 实 验 室

# 探索表单验证的最佳实践

# 传统面条式代码



```
//检测数据是否为空,true——检验通过, false——检验失败
checkDate(daily) {
    // 任务名, 项目名不能为空
    if (!daily.name) {
        this.setData({
            proEmpt: true
        })
    } else {
        this.setData({
            proEmpt: false
        })
    }
    if (!daily.taskName) {
        this.setData({
            taskEmpt: true
        })
    } else {
        this.setData({
            taskEmpt: false
        })
    }
    if (this.data.taskEmpt || this.data.proEmpt) {
        return false;
    } else {
        return true;
    }
```

优点

- 可读性高

缺点

- 扩展性差【如果我们改变了变量名，需要深入该函数修改】

- 复用性差【如果在程序中另外的地方用到，我们要把这个复制到其他地方，并且还要修改对应的属性】

# 能不能像写配置一样写表单验证?

# 理想的验证方式



```
//创建表单检验实例
let validator = new Helper.Validator();
//编写检验配置
validator.add(teamName, [{
    strategy: 'isNonEmpty',
    errorMsg: '内容不能为空'
}]);
validator.add(teamSlogan, [{
    strategy: 'isNonEmpty',
    errorMsg: '内容不能为空'
}]);
//开始验证，接收返回的错误信息
let errorMsg = validator.start();
```

**优点：**

- **易于拓展**【假如想新增检验手机号码，邮箱等功能，只需要修改对应的 strategy 】
- **复用性好**【可以避免许多重复复制粘贴的工作】

# 策略模式

策略，顾名思义，就是做事情的方法，比如，表单验证里面，判断是否为空，判断最小长度，判断是否为手机号码，这些就是策略。我们把这些“算法”放入策略类。

定义：定义一系列的算法，把它们一个个封装起来，并且使他们可以相互替换

策略模式的目的：将算法的使用和算法的实现分离开

策略模式的组成：策略类，环境角色

# 编写策略类

# 编写环境角色

# 编写策略类



## 将检验逻辑封装成策略类

```
const strategies = {
    isNonEmpty(value, errorMsg) {
        return value == '' ? errorMsg : void 0;
    },
    isEmail(value, errorMsg) {
        return !/^\w+([+-.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*$/.test(value) ?
            errorMsg : void 0
    }
}
```

# 编写环境角色



```
class Validator {
    constructor() {
        this.cache = []; //缓存校验规则
    }
    add(value, rules) {
        rules.map((rule) => {
            let strategyAry = rule.strategy.split(':');
            let errorMsg = rule.errorMsg;

            this.cache.push(()=> {
                let strategy = strategyAry.shift();
                strategyAry.unshift(value);
                strategyAry.push(errorMsg);
                return strategies[strategy].apply(value, strategyAry)
            })
        })
    }
    start() {
        for (let validatorFunc of this.cache) {
            let errorMsg = validatorFunc()//开始校验，并取得校验后的返回信息

            if (errorMsg) {
                return errorMsg
            }
        }
    }
}
```

存储对应的 校验请求
然后再转发让策略类进行处理

# 调用方法



```
//创建表单检验实例
let validator = new Helper.Validator();
//编写检验配置
validator.add(teamName, [{
    strategy: 'isNonEmpty',
    errorMsg: '内容不能为空'
}]);
validator.add(teamSlogan, [{
    strategy: 'isNonEmpty',
    errorMsg: '内容不能为空'
}]);
//开始验证，接收返回的错误信息
let errorMsg = validator.start();
```

# THANKS

FOR YOUR WATCHING

02

OPEN ORIENTED

凹 凸 实 验 室