ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN CUỐI KÌ

Validation Framework

Môn: Mẫu thiết kế hướng đối tượng

GVLT : **Nguyễn Minh Huy**

GVPT : **Mai Anh Tuấn**

NGƯỜI THỰC HIỆN : 1612078 - Nguyễn Đình Hoàng Đắc

1612173 – Đặng Anh Hào

1612543 – Phạm Anh Quốc

Tp. Hồ Chí Minh - Tháng 1/2020

ĐỒ ÁN CUỐI KÌ

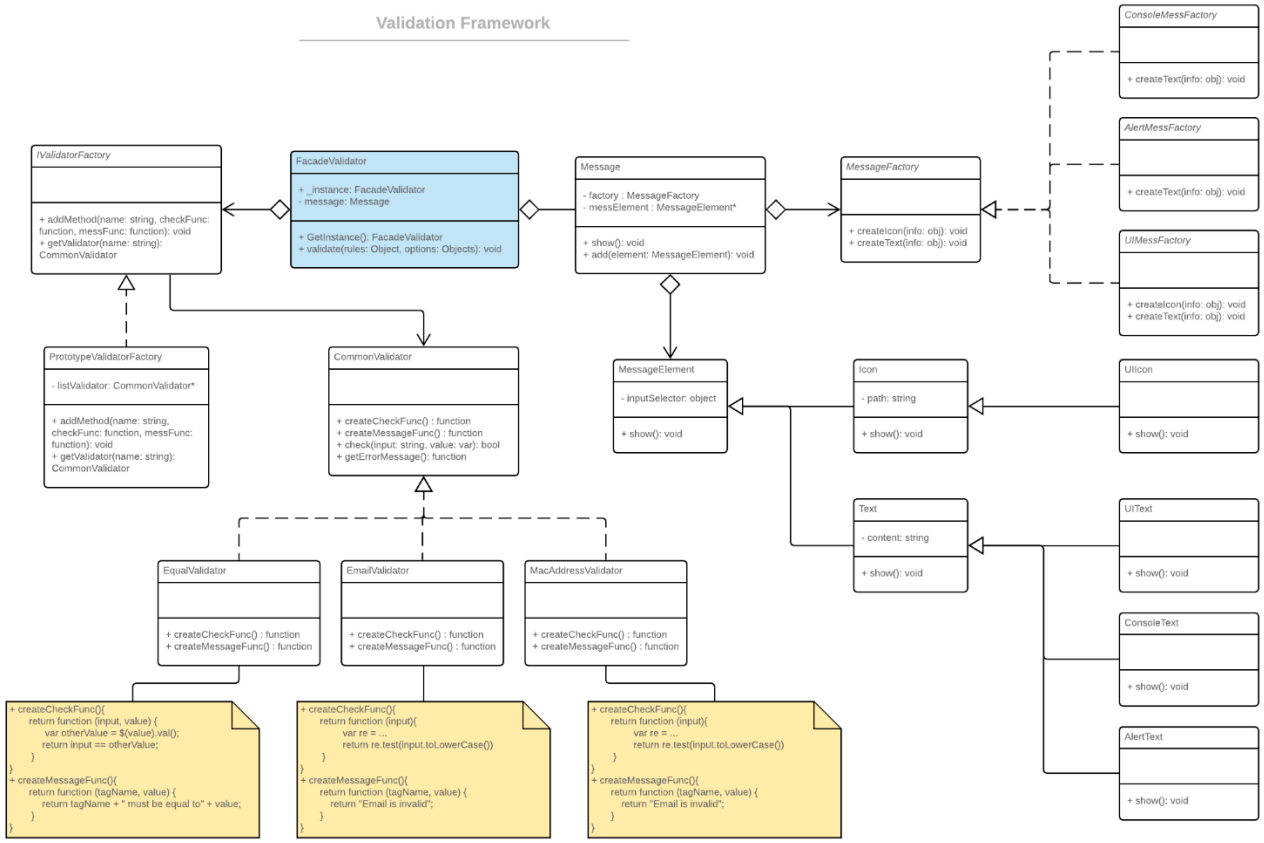
Validation Framework

Khoa Công nghệ Thông tin
Đại học Khoa học Tự nhiên, ĐHQG-HCM
Tháng 01/2020

MỤC LỤC

1 Cấu trúc framework

1.1 Sơ đồ lớp



1.2 Giải thích ý nghĩa

1. Lớp FacadeValidator

Lớp FacadeValidator là lớp bao bọc mã nguồn của framework, chứa và gọi các lớp con xác thực từng loại dữ liệu, lớp Message xử lý việc hiển thị thông báo lỗi. Lớp này đứng trung giữa giữa client và framework, nhận yêu cầu từ người dùng framework thông qua hàm validate, từ đó gọi các hàm xử lý cần thiết để validate những yêu cầu được truyền vào, sau đó gọi hiển thị thông báo lỗi dựa vào tùy chọn của client.

2. Lớp IvalidatorFactory

Interface mô tả cấu trúc của một ValidatorFactory, đảm bảo nguyên lý đảo ngược phụ thuộc, lớp FacadeFactory thay vì làm việc với nhiều lớp ValidatorFactory, mà làm việc với một interface factory chung. Các lớp ValidatorFactory sẽ implement lại interface này và cài đặt các hàm addMethod (thêm một custom validator) và getValidator (trả về một validator cụ thể tùy thuộc vào từ khóa truyền vào).

3. Lớp PrototypeValidatorFactory

Lớp concrete của interface ValidatorFactory. Lớp này lưu sẵn một danh sách các validator dưới dạng {key, value}, sau đó dựa vào từ khóa truyền vào, trả về một validator cụ thể.

4. CommonValidator

Tương tự như IValidatorFactory, CommonValidator là lớp cha quy ước các hàm cần thiết của một Validator, đồng thời định nghĩa template khởi động cho các lớp validator kế thừa nó phải tuân theo.

5. Các lớp Validator (EmailValidator, StringLengthValidator,...)

Các lớp Validator thể hiện cách cài đặt cụ thể của lớp CommonValidator, mỗi lớp Validator sẽ có cách cài đặt riêng phù hợp với chức năng của nó. Ví dụ: EmailValidator sẽ kiểm tra một địa chỉ email có hợp lệ hay không, StringLengthValidator sẽ kiểm tra độ dài của một chuỗi với điều kiện ràng buộc, tương tự với các lớp Validator khác.

6. Message

Lớp Message có vai trò hiển thị thông báo của các validator theo nhiều cách khác nhau. Message bao gồm 1 MessageFactory và mảng MessageElement. Message hiển thị bằng cách hiển thị toàn bộ MessageElement hiện có theo cách MessageFactory quy định.

7. MessageFactory

Lớp MessageFactory có vai trò quy định cách hiển thị thông báo của các MessageElement mà nó tạo ra.

8. Các lớp Factory (UIMessFactory, ConsoleMessFactory, AlertMessFactory)

Các lớp này kế thừa MessageFactory trong đó mỗi lớp quy định một cách hiển thị khác nhau cho các MessageElement tương tự giữa các factory.

9. MessageElement

Lớp MessageElement là lớp đóng vai trò là thành phần của Message như text, icon, image,... , cấu tạo nên Message hoàn chỉnh.

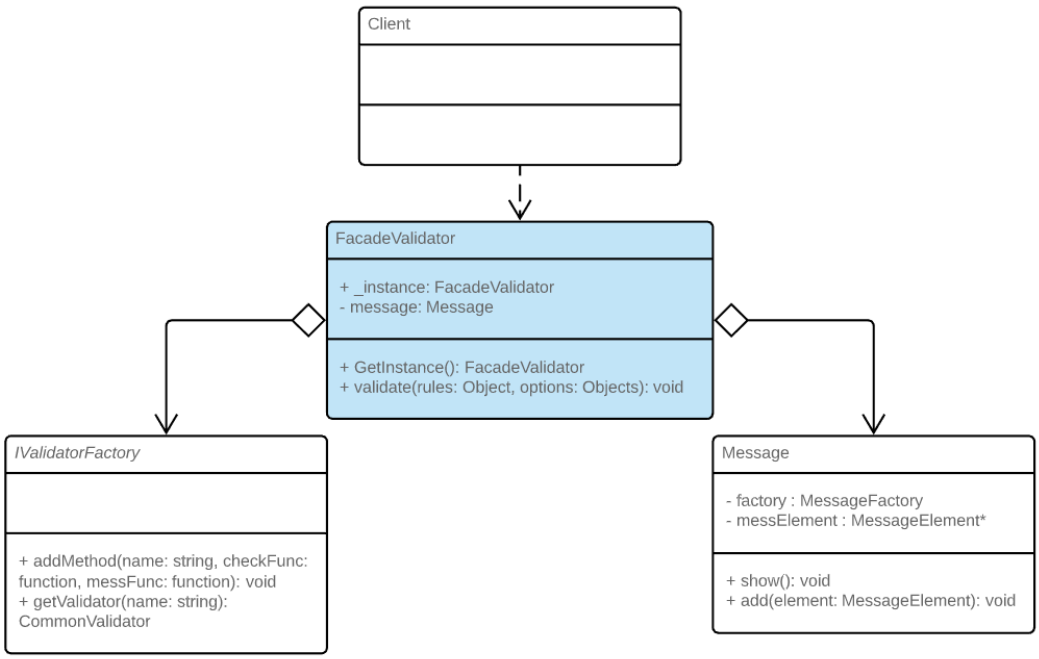
10. Các lớp MessageElement(Icon, Text)

Các lớp này là thành phần như icon, text,... của Message với các lớp con như UIText, UIIcon, ConsoleText,... cho các MessageFactory khác nhau.

2 Mẫu thiết kế được sử dụng

2.1 Façade

1. Sơ đồ lớp



2. Đoạn code sử dụng

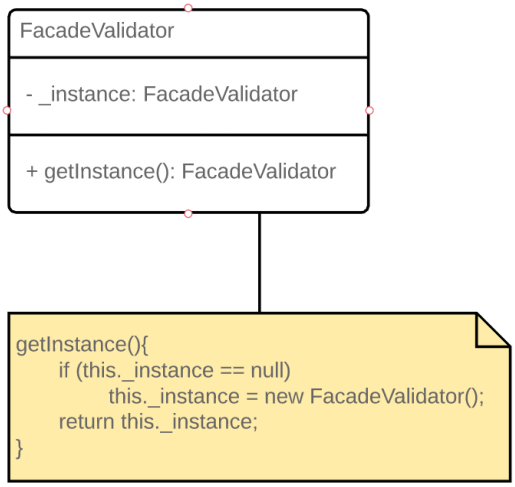
```
	class FacadeValidator {
	constructor() {
		this.messFactory = new UIMessFactory();
		this.message = new Message(this.messFactory);
		this.validatorFactory = new PrototypeValidatorFactory();
	}

	validate(rules, options) {
		this.getMessFactory(options.errorDisplay);
		this.arrInvalid = [];
		$.each(rules, function (tagName, value) {
			var result = this.validateInputWithRule(tagName, value);
			if (result != null) {
				this.arrInvalid.push(result);
				this.addMessage(result);
			}
		}.bind(this));
		this.showMessage();
	}
}
```

3. Ý nghĩa

Framework hỗ trợ nhiều validator khác nhau, do đó việc truy cập vào từng phương thức validate của các validator là điều không cần thiết, dẫn đến việc sử dụng code không hiệu quả, dễ gây ra lỗi. Vì thế, việc áp dụng mẫu Facade vào thiết kế framework là hợp lý, người dùng chỉ cần getInstance() của lớp FacadeValidator, sau đó khai báo các rules cần thiết thì hệ thống sẽ tự validate các element đã được khai báo.

2.2 Singleton

1. Sơ đồ lớp



2. Đoạn code sử dụng

```
FacadeValidator._instance = null;
FacadeValidator.getInstance = function () {
    if (FacadeValidator._instance == null)
        FacadeValidator._instance = new FacadeValidator();
    return FacadeValidator._instance;
}
```

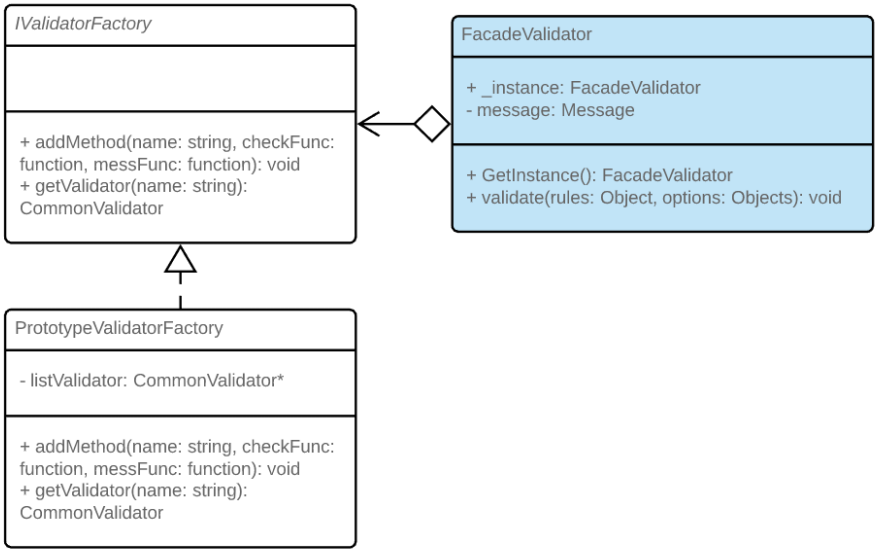
3. Ý nghĩa khi sử dụng

Vì có sử dụng mẫu Façade, nên client chỉ cần duy nhất 1 instance của đối tượng FacadeValidator để validate dữ liệu, không cần phải tạo nhiều instance. Hơn nữa instance này có thể được gọi ở nhiều nơi, và không biết nơi nào khởi tạo trước. Chính vì vậy dùng mẫu Singleton cho lớp FacadeValidator là phù hợp.

2.3 Factory Method

1. ValidatorFactory

1.1 Sơ đồ lớp



1.2 Đoạn code sử dụng mẫu

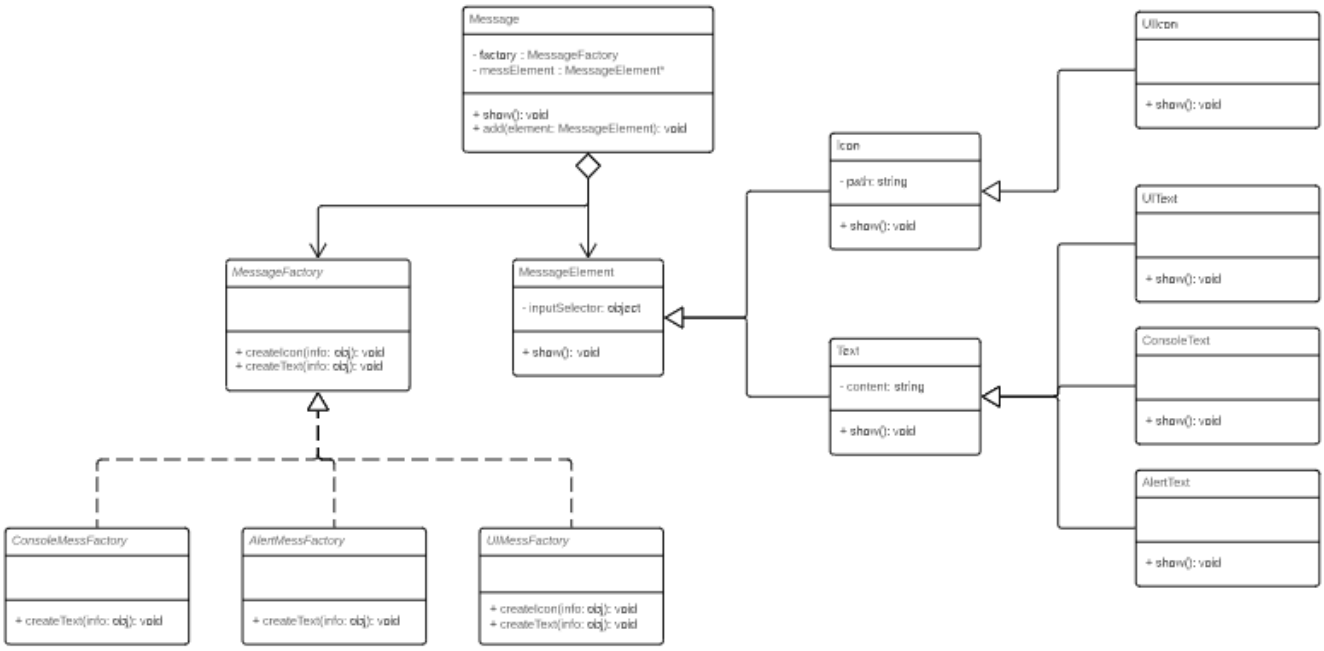
```
addMethod(name, func, errorMessage){
            this.validatorFactory.addMethod(name, func, errorMessage);
}

getValidatorByType(strType) {
      return this.validatorFactory.getValidator(strType);
}
```

1.3 Ý nghĩa khi sử dụng mẫu

Vì các validator có nhiều kiểu khởi tạo cũng như cách cài đặt khác nhau (hiện tại trong framework chỉ có một kiểu khởi tạo cũng như cài đặt), nên để dễ dàng mở rộng framework sau này, ta dùng mẫu Factory Method, giao việc tạo tập các validator cho một lớp riêng, cài đặt từ interface ValidatorFactory. Interface này phụ trách việc tạo lập validator nên cần có 2 hàm getValidator (trả về validator dự vào type truyền vào) và addMethod (thêm validator mới hoặc custom validator). Đồng thời, cũng tạo một lớp ảo CommonValidator để các lớp Validator kế thừa, các lớp validator này cho dù được cài đặt như thế nào cũng cần đảm bảo một mẫu chung là có 2 hàm check và getErrorMessage.

2. MessageFactory

2.1 Sơ đồ lớp



2.2 Đoạn code sử dụng mẫu

```
class Message{

    constructor(factory) {
        this.factory = factory;
        this.messElement = [];
    }

    show(){
            this.messElement.forEach(function (element) {
                element.show();
            })
    }

    add(element)
    {
        this.messElement.push();
    }
}
```

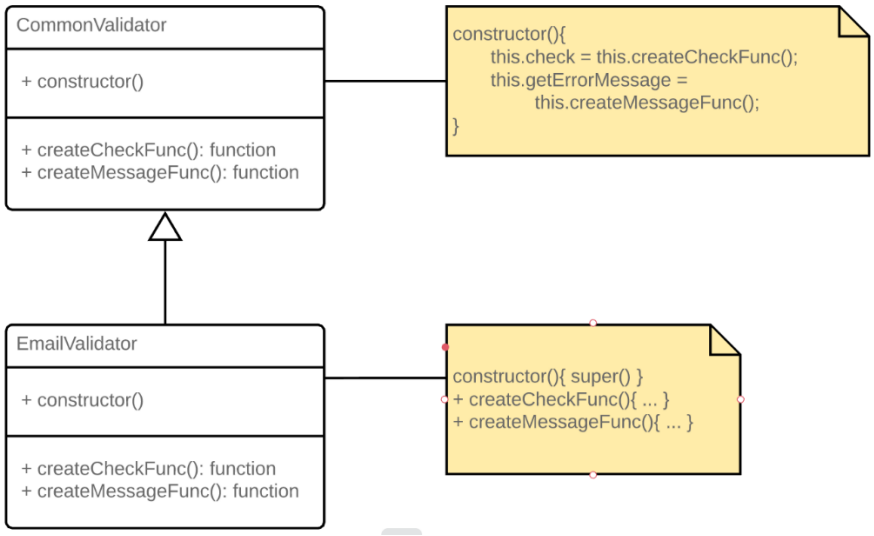
```
class MessageFactory{
    createIcon(info)
    {
        return new Icon(info);
    }

    createText(info)
    {
        return new Text(info);
    }
}
```

2.3 Ý nghĩa khi sử dụng mẫu

Message và MessageFactory đóng vai trò hiển thị các error message của các validator theo nhiều cách hiển thị khác nhau và có thể thêm cách hiển thị mới một cách dễ dàng dựa theo mẫu Factory. Hiện tại trong framework có sẵn các MessageFactory với các cách hiển thị như UI, console hay alert. MessageFactory có vai trò tạo ra các lớp gọi là MessageElement với input là info do các validator quy định, một MessageElement là một thành phần tạo nên Message, từ đó Message sẽ hiển thị toàn bộ các MesageElement của mình để tạo ra một thông báo hoàn chỉnh. Hiện tại đã có hai MessageElement là Icon và Text.

2.4 Template Method

1. Sơ đồ lớp



2. Đoạn code sử dụng mẫu

```
class CommonValidator{
    constructor(checkFunc, messageFunc){
        if (!checkFunc)
            checkFunc = this.createCheckFunc();
        if (!messageFunc)
            messageFunc = this.createMessageFunc();
        this.check = checkFunc;
        this.getErrorMessage = messageFunc;
    }

    createCheckFunc(){
        throw new Error('Derived class must override method createCheckFunc!');
    }

    createMessageFunc(){
        throw new Error('Derived class must override method createMessageFunc!');
    }
}

class EmailValidator extends CommonValidator {
    constructor(){
        super();
    }

    createCheckFunc(){
        return function (str) {
            var regex = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@
((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-
Z]{2,}))$/;
            return regex.test(str.toLowerCase())
        };
    }

    createMessageFunc(){
        return function (tagName, value) {
            return "Email is invalid";
        }
    }
}
```

3. Ý nghĩa khi sử dụng mẫu

Mỗi lớp Validator có hàm "kiểm tra dữ liệu" và hàm "trả về chuỗi thông báo lỗi" khác nhau giữa các lớp, nên mỗi lớp cần định nghĩa 2 hàm này khi khởi tạo. Để đảm bảo 2 hàm này được cài đặt ở lớp con, và 2 hàm này được gọi, ta sử dụng mẫu Template Method. Cụ thể trong lớp cha (CommonValidator), ta định nghĩa 2 hàm ảo là createCheckFunc và createMessageFunc để đảm bảo được lớp con override lại. Đồng thời gọi 2 hàm này trong phương thức khởi tạo, tạo thành khuôn (template) để các lớp con tuân theo.

Vì trong javascript không có hàm ảo, nên ở 2 hàm này của lớp cha, ta throw 1 lỗi, để khi lớp con không override lại 2 hàm này, chương trình sẽ dừng lại, hiển thị lỗi tương tự như dùng từ khóa abstract trong các ngôn ngữ khác.