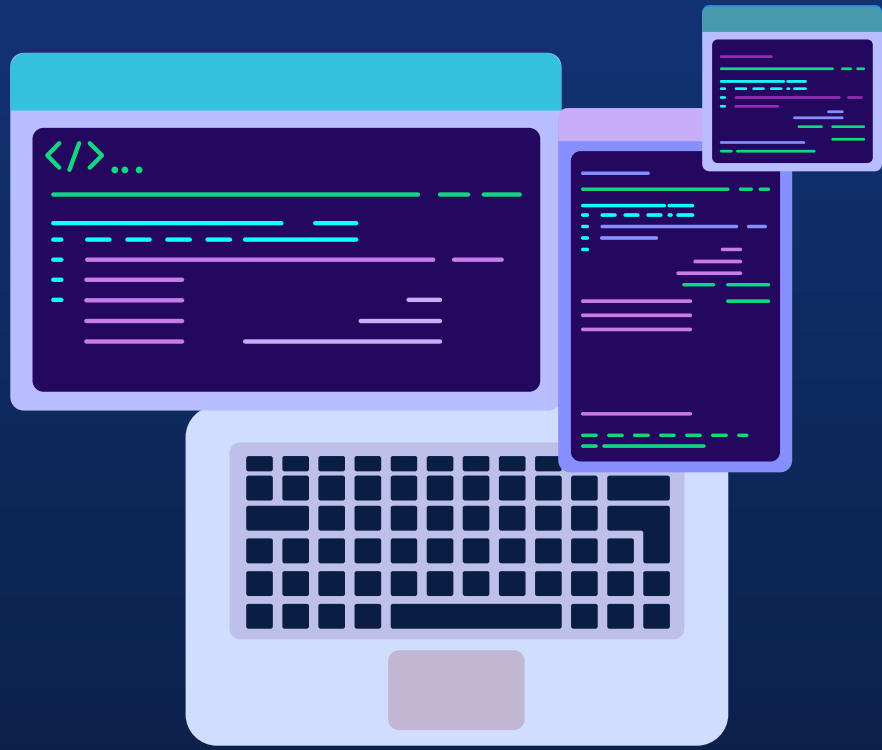
Clean Architecture and Command Pattern

Harry Yan



本次 分享/讨论 目的

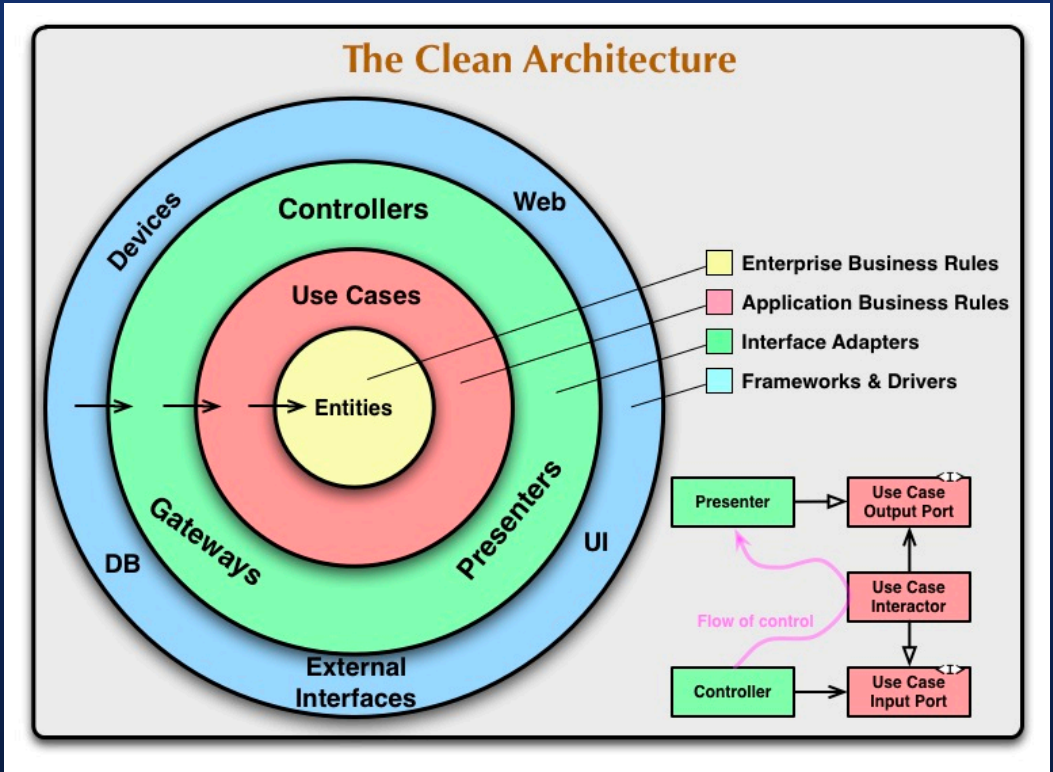
我们至少会收获:

- 了解Clean Architecture，命令模式的一些应用
- 这不是个单纯的分享架构和设计模式的session，而是触发思考: what, why and how
- 系统设计面试如何体现平时的积累？如何应用？哪些问题会涉及到架构和设计模式？
- 抛砖引玉

01

Clean Architecture

一图胜千言


The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers
Presenter
Use Case
Output Port
Use Case
Interactor
Flow of control
Controller
Use Case
Input Port

优点

独立于外部框架

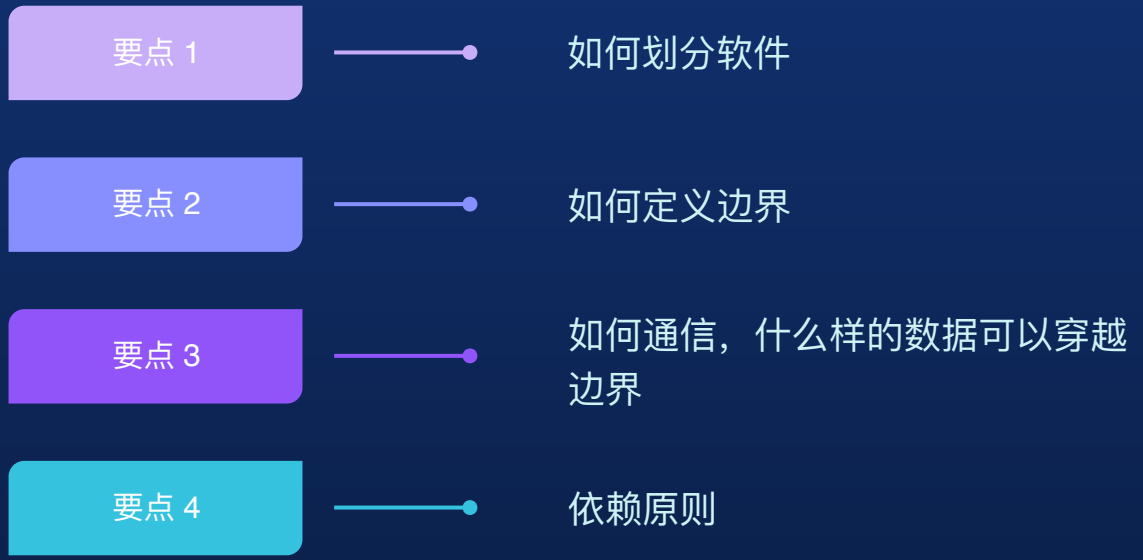
独立于任何UI

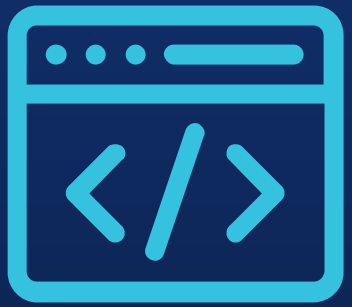
可测试

独立于任何外部媒介

独立于数据库

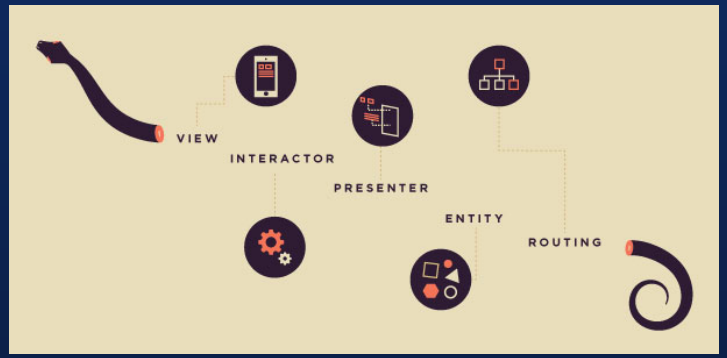
要点





案例

MEGA 面试题
VIPER



为什么 - 解决问题

01

易于维护

代码量上升

02

快速迭代

功能增多

03

有效合作

团队扩张

04

性能保障

复杂度上升

05

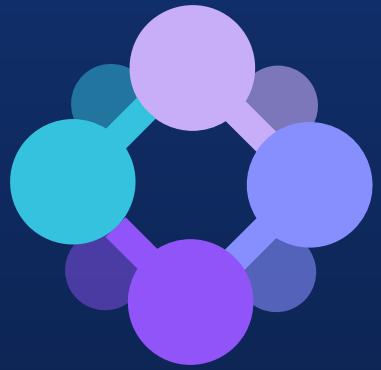
快速接入

新兴技术

06

质量保障

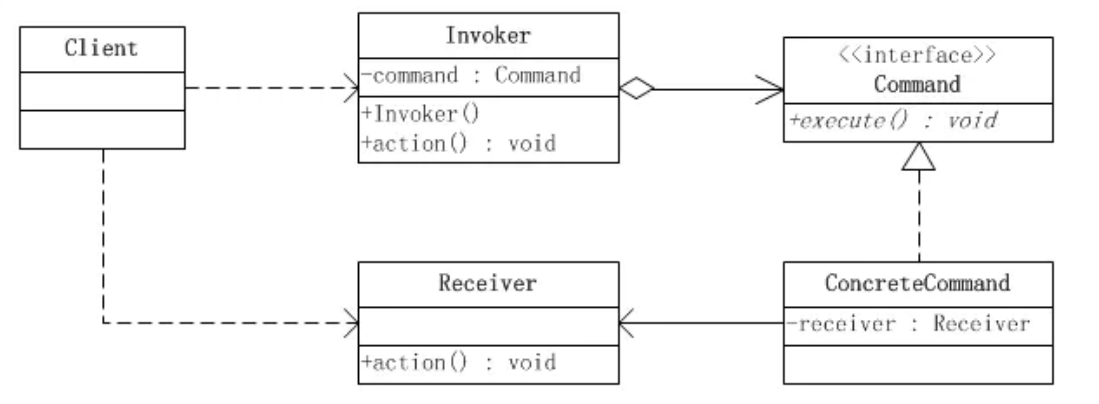
代码约束



02

命令模式

命令模式

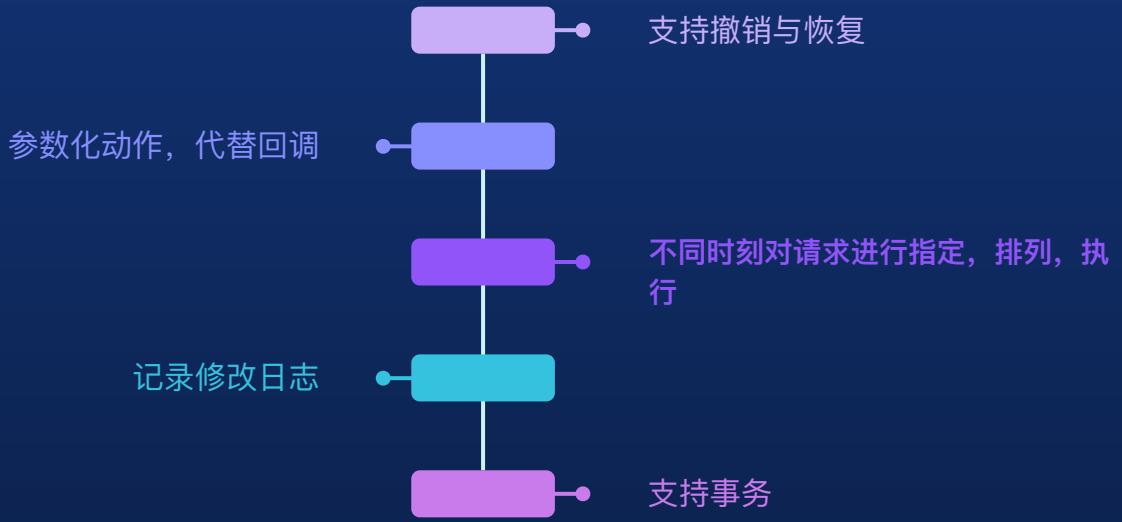


- 命令对象
- 调用者
- 接收者

命令模式

```java
public class Receiver {
    /**
     * 真正执行命令相应的操作
     */
    public void action(){
        System.out.println("执行操作");
    }
}

public interface Command {
    /**
     * 执行方法
     */
    void execute();
}

public class ConcreteCommand implements Command {
    //持有相应的接收者对象
    private Receiver receiver = null;
    /**
     * 构造方法
     */
    public ConcreteCommand(Receiver receiver){
        this.receiver = receiver;
    }
    @Override
    public void execute() {
        //通常会转调接收者对象的相应方法，让接收者来真正执行功能
        receiver.action();
    }

}

public class Invoker {
    /**
     * 持有命令对象
     */
    private Command command = null;
    /**
     * 构造方法
     */
    public Invoker(Command command){
        this.command = command;
    }
    /**
     * 行动方法
     */
    public void action(){

        command.execute();
    }
}

public class Client {

    public static void main(String[] args) {
        //创建接收者
        Receiver receiver = new Receiver();
        //创建命令对象，设定它的接收者
        Command command = new ConcreteCommand(receiver);
        //创建请求者，把命令对象设置进去
        Invoker invoker = new Invoker(command);
        //执行方法
        invoker.action();
    }

}
```

命令模式



命令模式

```
enum Command: CommandType, Equatable {
        case startLoading
        case finishLoading
        case showRegion(String, callingCode: String)
        case sendCodeToPhoneNumberError(message: String)
}

var invokeCommand: ((Command) -> Void)?

// 调用
invokeCommand?(.startLoading)

// 接收者
protocol ViewType {
    associatedtype Command: CommandType

    func executeCommand(_ command: Command)
}


@IBAction private func didTapLogoutButton() {
    setEditing(false, animated: true)
    viewModel.dispatch(.logout)
}

func executeCommand(_ command: SMSVerificationViewModel.Command) {
    switch command {
    case .startLoading:
        SVProgressHUD.show()
    case .finishLoading:
        SVProgressHUD.dismiss()

       ...
}

// unit test
test(viewModel: sut,
     action: SMSVerificationAction.loadRegionCodes,
     expectedCommands: [.startLoading, .finishLoading])
```