浅析MVP设计模式

汇报人：

2024-01-12

目录

01

引言

目的和背景

01

提高软件质量

通过分离视图层和业务逻辑，使代码更加清晰、可维护，从而提高软件质量。

02

适应需求变化

MVP设计模式能够灵活应对需求变更，降低维护成本。

03

提升开发效率

合理的架构设计和代码组织可以提高开发效率，减少不必要的返工。

MVP设计模式的定义

01

Model（模型）

负责处理数据的存取和逻辑计算，是应用程序的核心部分。模型不依赖于视图和表示器，可独立进行测试和重用。

02

View（视图）

负责显示数据给用户，并接收用户的交互操作。视图通常是用户界面，可以是Web页面、桌面应用界面或移动应用界面等。

03

Presenter（表示器）

作为模型和视图之间的桥梁，负责处理用户交互事件、调用模型处理数据，并更新视图。表示器解耦了模型和视图，使得它们可以独立进行开发和测试。

02

MVP设计模式的核心思想

分离关注点

逻辑与视图分离

在MVP中，Model负责数据处理，View负责界面展示，Presenter负责逻辑处理，三者各司其职，实现了逻辑与视图的分离。

提高代码可维护性

由于逻辑与视图分离，当界面发生变化时，只需要修改View层代码，而不需要改动业务逻辑代码，从而提高了代码的可维护性。

```
<!DOCTYPE html>
<html lang="en-us">
  <head>
    <title>pagename</title>
    <meta name="Author" content="author">
    <meta name="Description" content="description">
    <meta name="Keywords" content="keywords">
    <meta charset="utf-8">
    <link rel="icon" type="image/icon" href="favicon.ico">
    <link rel="stylesheet" type="text/css" href="style.css">
    <style>
      .reset                                { margin:0; padding:0; }
      .clear                                { clear:both; }
      .cleared:after                        { content:"."; display:block; height:0; clear:both; visibility:hidden; }
      .right                                { float:right; }
      .left                                 { float:left; }
      a img                                 { border:0; }
      img                                   { max-width:100%; }
      header, nav, section, article, aside, footer  { display:block; }
      body                                  { margin:0;padding:0; }
    </style>
    <script src="script.js"></script>
    <script>
      $(document).ready(function(){
      });
    </script>
  </head>
  <body>
    <header></header>
    <nav></nav>
    <section>
      <article></article>
    </section>
    <aside></aside>
    <footer></footer>
  </body>
</html>
```

可测试性

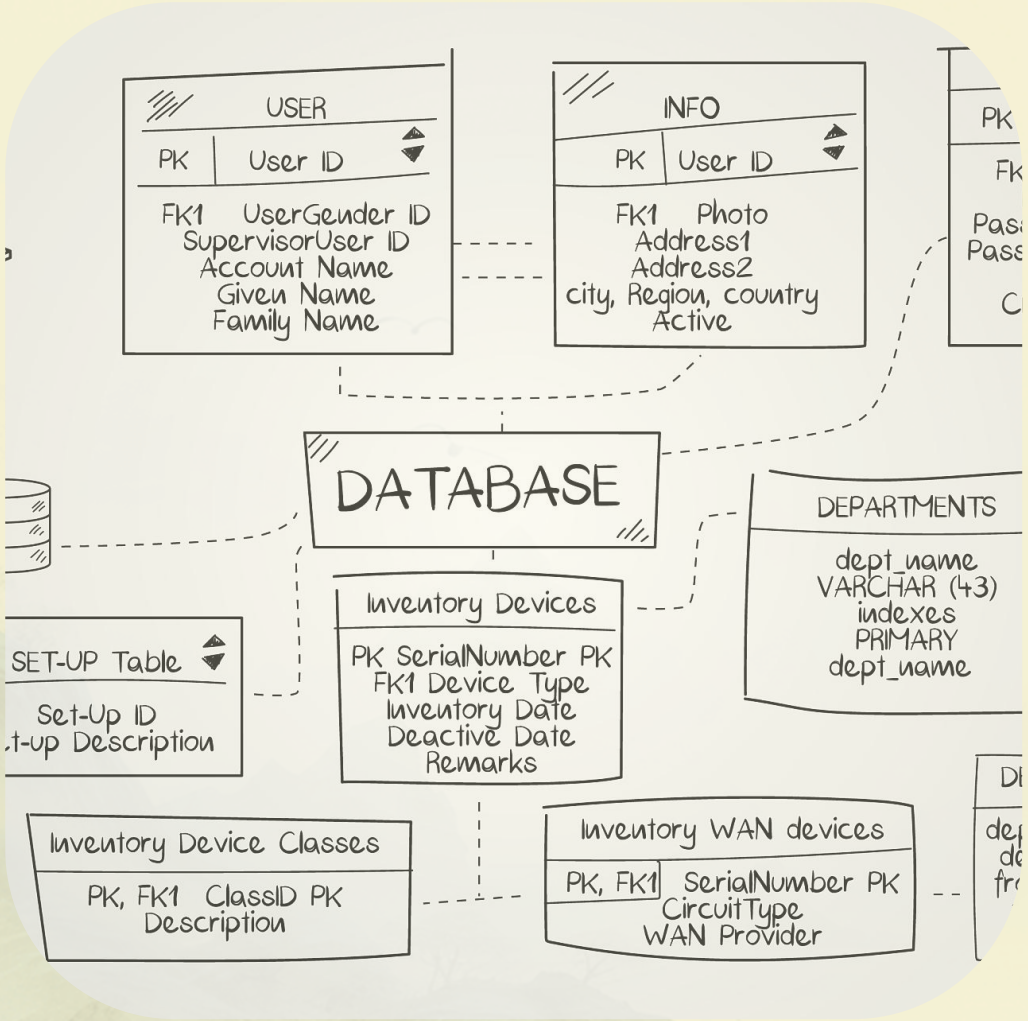
单元测试的便利性

MVP模式使得业务逻辑集中在Presenter中，这样就可以针对Presenter编写单元测试，验证业务逻辑的正确性。

模拟视图层

在测试过程中，可以通过模拟View层来测试Presenter的逻辑，而不需要依赖实际的UI界面。

松耦合



降低模块间依赖

MVP模式通过接口定义View和Model的交互方式，使得Presenter与具体的View和Model实现解耦，降低了模块间的依赖。

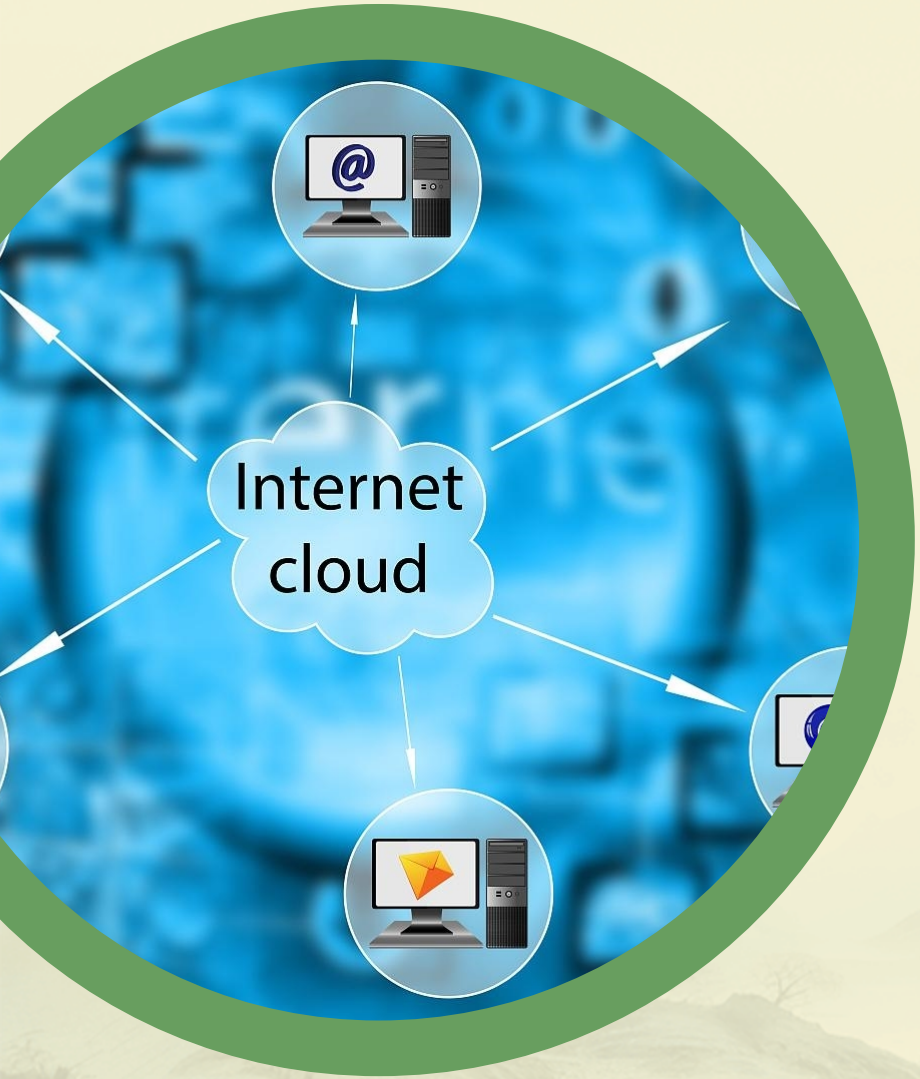
提高代码复用性

由于模块间依赖降低，MVP模式使得代码更容易在不同的项目或模块中复用。同时，也有利于代码的模块化和组件化开发。

03

MVP设计模式的组成

Model（模型）



01

数据处理

Model负责处理和存储应用的数据，包括数据的获取、存储和管理。

02

业务逻辑

Model还包含了应用的业务逻辑，用于处理各种数据操作和业务规则。

03

数据一致性

Model确保数据的一致性和完整性，提供数据验证和错误处理机制。

View（视图）

用户界面

View是应用的用户界面，负责展示数据和与用户交互。

视图更新

当Model中的数据发生变化时，View需要相应地更新界面以反映最新的数据状态。

事件处理

View处理用户的输入事件，如点击、滑动等，并将这些事件传递给Presenter处理。

Presenter（主持人）

逻辑处理

Presenter作为Model和View之间的协调者，负责处理用户交互逻辑和业务逻辑。

01

02

数据绑定

Presenter从Model中获取数据，并将其绑定到View中，确保数据与视图的同步。

03

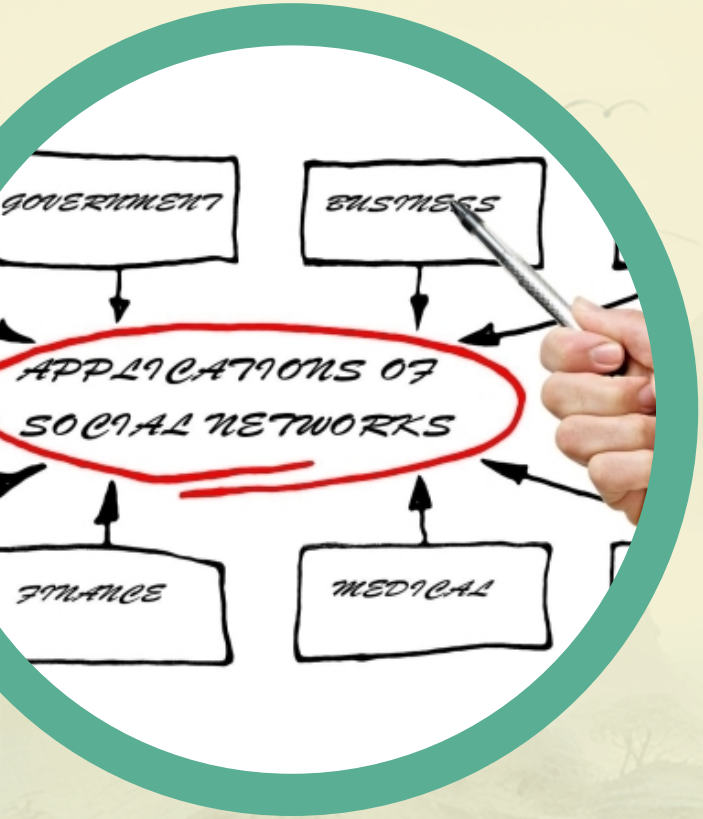
事件响应

Presenter响应由View传递的用户事件，调用Model中的相应方法处理事件，并更新View以反映处理结果。

04

MVP设计模式与MVC的对比

MVC设计模式的组成



Model（模型）

负责处理数据的存取和逻辑计算。

View（视图）

负责显示数据给用户，并接收用户的交互操作。

Controller（控制器）

负责接收用户的请求，并调用模型和视图进行处理。

MVP与MVC的相似之处

分离关注点

MVP和MVC都强调将数据处理、用户交互和界面展示分离，以提高代码的可维护性和可测试性。

模型与视图的解耦

在MVC和MVP中，模型与视图都是解耦的，它们之间通过接口或抽象类进行通信，降低了代码的耦合度。