[1. MVC 2](#_Toc462845308)

[1-1. MVC 仅仅是一种理念，而不是一种技术。 2](#_Toc462845309)

[1-2. MVC的目的 2](#_Toc462845310)

[1-3. 优缺点 2](#_Toc462845311)

[1-3-1. 优点 2](#_Toc462845312)

[1-3-2. 缺点 2](#_Toc462845313)

[1-4. 在DataGrid 中显示数据的代码应该在View还是Controller 2](#_Toc462845314)

[1-5. MVC 只是一个表示层的架构 3](#_Toc462845315)

[1-5-1. 业务逻辑是封装在业务逻辑层的BO 还是Model 里面 3](#_Toc462845316)

[2. MVP 3](#_Toc462845317)

[2-1. MVP和MVC的差别。 3](#_Toc462845318)

[3. MVVM 4](#_Toc462845319)

[4. MVC->MVP->MVVM演进过程 4](#_Toc462845320)

[4-1. 相同点 4](#_Toc462845321)

[4-2. 不同点 4](#_Toc462845322)

# MVC

MVC是相对于之前代码的大杂烩状态提出的。在代码大杂烩时，展示（View），流程控制(Controller) 和 业务逻辑(Model) 是混杂在一起的。

考虑早前的web page 或者 所有东西写在一起的win form 程序。

带来巨大的维护成本。程序往往也晦涩难懂。当有功能改动或者加新功能时非常的痛苦。可测试性也很差，大多依赖手工测试，比较难提供自动化测试脚本。

## MVC 仅仅是一种理念，而不是一种技术。

几乎没有两个MVC的框架是完全一样的

## MVC的目的

基本理念是分离，各司其职，技术术语是解耦。

耦合度低又带来几个好处：

* 封装/隔离变化：将变化封装，隔离在各自的组件内部
* 代码复用：
  + MVC中 View 是很难被复用的。用于Web page展示的View和用于桌面展示的View很难有什么共同的Code。
  + 而Controller 和 Model 则是存在复用可能的。对于Controller 来说，如果针对View的Interface 来操作，那么Controller 可以被不同的View 所复用。
  + 而Model 则天然就可以被不同的Controller 所复用。而且由于Controller需要访问Model的属性，而不仅仅是方法，所以Model 大多不定义为Interface
* 可测试性高： 可以针对每一部分单独写测试用例。对于不太容易测试的View 部分，则可以用人工测试覆盖
* 为分工带来可能： UI 工程师可以专心设计UI。 数据库专家可以涉及SQL和DB，软件工程师可以开发中间的程序控制和业务逻辑

## 优缺点

### 优点

### 缺点

The MVC pattern introduces new levels of indirection and thereof increases the complexity of the solution 架构的清晰以增加程序的复杂度为代价

## 在DataGrid 中显示数据的代码应该在View还是Controller

如果考虑Controller的复用。同一个Controller 可能会被Web Page 和 WinForm 的View 所复用。而显然Web page 中的DataGrid 和WinForm中的DataGrid是不同的。

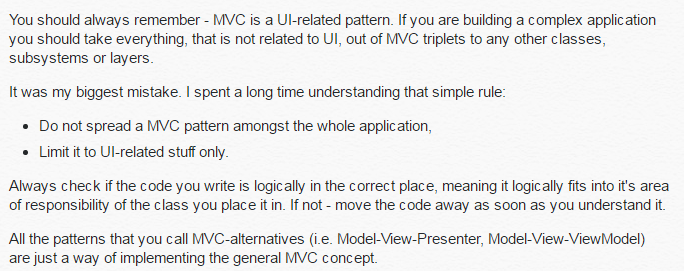
Controller 如果针对接口来做的话，可复用度高的做法时仅仅调用接口的一个方法，然后把变化后的数据传递给View. View 负责去做显示。

不过，这只是从可复用度高这个维度来说的。如果一个程序完全不用考虑Web Page 和Win Form的不同View，而只会用其一种，那么这部分放在Controller 也未尝不可。

## MVC 只是一个表示层的架构

从分层的架构来说， 三层的架构分为表示层 PresentationLayer， 业务逻辑层 Business Layer 和 数据访问层 Data Access Layer.

MVC只是表示层的一个架构，其M，V, C 只是在表示层内部实现 展示代码和业务逻辑代码分离的架构。这个所谓的业务逻辑 和 分层架构中中间层的业务逻辑层 不是一个概念。 这里的业务逻辑往往只是中间层的业务逻辑层的访问入口。



### 业务逻辑是封装在业务逻辑层的BO 还是Model 里面

比如要获取来自数据的数据，那么Controller是直接访问Model 对象的方法（Model 内部去访问BO对象），还是直接访问BO对象的方法来获取Model对象。

前一种Model 显然既有数据又有方法，后一种Model 更类似于纯粹的Bean。

感觉MVC 模式中的Model 更接近于前者，虽然对于后者来说，当BO返回了Model对象给Controller之后，还是Controller 和Model 之间交互，也仍然算是符合MVC

# MVP

## MVP和MVC的差别。

差别并不大。MVP引入了View 的一个Interface， Presenter直接和Interface 交互而不是View.

View与Model并不直接交互，而是通过与Presenter交互来与Model间接交互。而在MVC中View可以与Model直接交互。View 监听来自于Model的事件

# MVVM

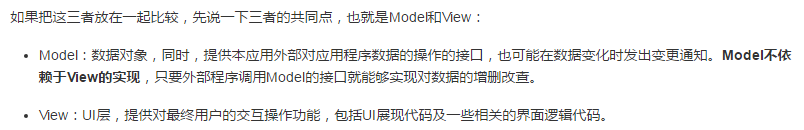
MVVM可以算是MVP的升级版，ViewModel和View之间的交互通过Data Binding完成，而Data Binding可以实现双向的交互

# MVC->MVP->MVVM演进过程

MVC -> MVP -> MVVM 这几个软件设计模式是一步步演化发展的，MVVM 是从 MVP 的进一步发展与规范，MVP 隔离了MVC中的 M 与 V 的直接联系后，靠 Presenter 来中转，所以使用 MVP 时 P 是直接调用 View 的接口来实现对视图的操作的，这个 View 接口的东西一般来说是 showData、showLoading等等。M 与 V已经隔离了，方便测试了，但代码还不够优雅简洁，所以 MVVM 就弥补了这些缺陷。在 MVVM 中就出现的 Data Binding 这个概念，意思就是 View 接口的 showData 这些实现方法可以不写了，通过 Binding 来实现。

<MVC,MVP,MVVM与架构经验谈.pdf>

## 相同点



## 不同点

