PC-Base 使用者介面開發實務

2023.5.17

洽富科技有限公司

鍾湫浤

Generics (泛型)

- Generic Class
  - Type parameter : **T**
  - **T could be**
    - **Reference Types or Value Types**

- **Remove the need for casting**
- **Improve type safety**
- **Reduce the amount of boxing required**

```
class Class_Name<T>
{
    private List<T> items;

    public Class_Name
    {
        items = new List<T>();
    }

    public T Method1()
    {
        …
        return T;
    }

    public void Method2(T item)
    {
        items.Add(item);
    }
}
```

Generics type with an interface class

- **Create an interface:**
  - Define an interface that declares **the method(s)** you want to implement.

- **Create a class with inheritance from a generic class:**
  - Create a class that inherits from a generic class, where the generic type parameter is constrained to the interface you created in step 1. This ensures that any type used **with the generic class must implement the shared interface.**

- **Implement the generic class:**
  - Implement the generic class, specifying the desired behavior and functionality using the generic type parameter. This **allows the class to work with different types** that adhere to the shared interface.

Object Casting & Unboxing

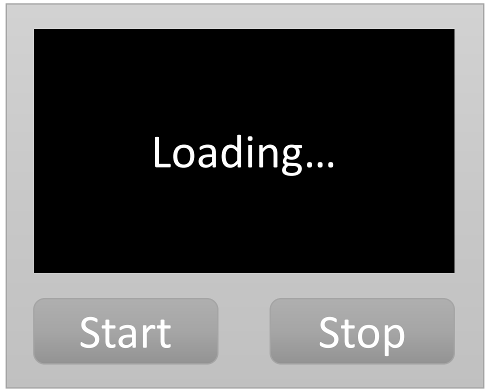
- **Use the object type to refer to an instance of any class**
  - Object obj = new MyClass();
  - Unboxing :
    - var unboxObj = obj **as** MyClass; // use **as** operator
    - var unboxObj = **(MyClass)**obj;   // force casting
- **Use the object type to store a value of any type**
  - int myIntValue;
    Object obj = myIntValue;
- **Define parameters by using the object type**
  - void MyMethod(Object obj){… }

MVVM (Model, View, ViewModel)

- 目的: 去耦合(decouple) ➔ 程式易擴展、修改、再利用
- Reusable、Flexible and Easy to Maintain

  ➔ Class/Method: 功能單一且明確

$$\begin{tikzcd}\text{View} \arrow[r, leftrightarrow] & \text{ViewModel} \arrow[r, leftrightarrow] & \text{Model}\end{tikzcd}$$



```
Public class MyViewModel
{
  public User GetData()
  {
     User user = New User();
     …
     return user;
  }
  …
}
```

```
Public class User
{
   public int Id {get; set;}
   public string Name {get; set;}
   public string Email {get; set;}
   …
}
```

MVC v.s. MVVM

- MVC is more widely adopted and supported in various programming languages and frameworks
  - Model
  - View
  - Controller
- MVVM is commonly associated with frameworks that provide **data-binding capabilities.**
  - Model
  - View
  - ViewModel

- **Model:** The model represents **the data** and the **business logic** of the application. It is responsible for
  - **Managing the data**
  - **Performing computations**
  - **Enforcing business rules**

- **View:** The view is responsible for **presenting the user interface to the user.** It **displays the data from the model** and **interacts with the user for input**. In the MVC pattern, the view is passive and does not contain any business logic.

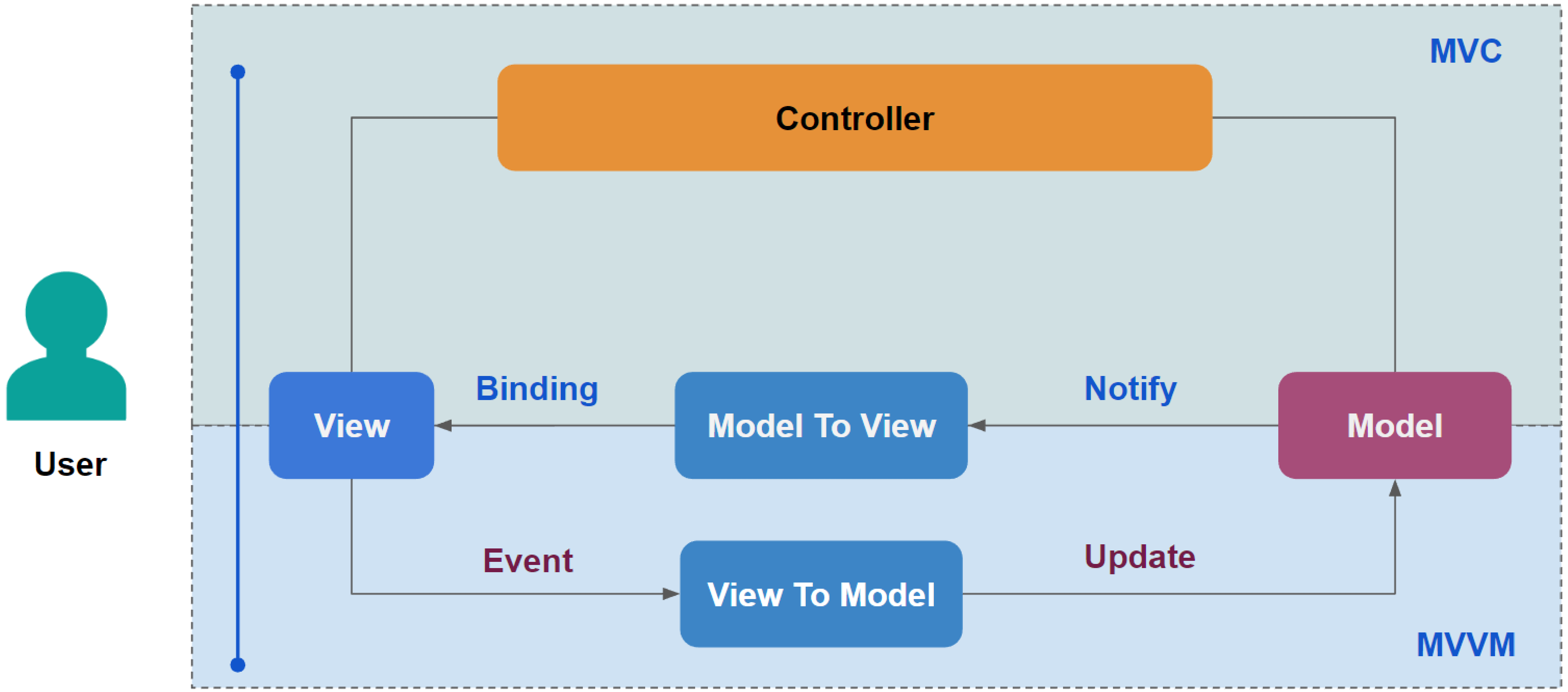
- **Controller:** The controller acts as an intermediary between the model and the view. It **receives input** from the user through the view and updates the model accordingly. It also listens to changes in the model and updates the view to reflect those changes. The controller is responsible for handling user interactions and coordinating the flow of data between the model and the view.

- **Model**: The model represents the data and the business logic, similar to the MVC pattern.

- **View:** The view is responsible for displaying the user interface, just like in MVC.

- **ViewModel:** The ViewModel acts as an intermediary between the view and the model. It exposes the data and commands from the model to the view, and it also contains the presentation logic for the view. The ViewModel does not have direct knowledge of the view, ensuring a decoupling between the two. It provides data-binding capabilities, allowing the view to automatically update when the data in the ViewModel changes.

MVC/MVVM


MVC
Controller
User
View
Binding
Model To View
Notify
Model
Event
View To Model
Update
MVVM