



# C# 物件導向程式開發

## 泛型 委派 Lambda Expressions

2017/02/16

網技二課

# 大綱

---

- 泛型
- 委派
- Func
- Lambda Expressions

泛型

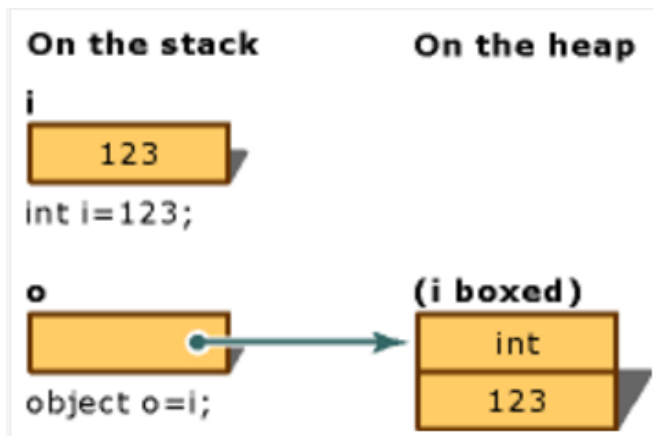
# 泛型概觀

.Net Framework 2.0 後才出現泛型

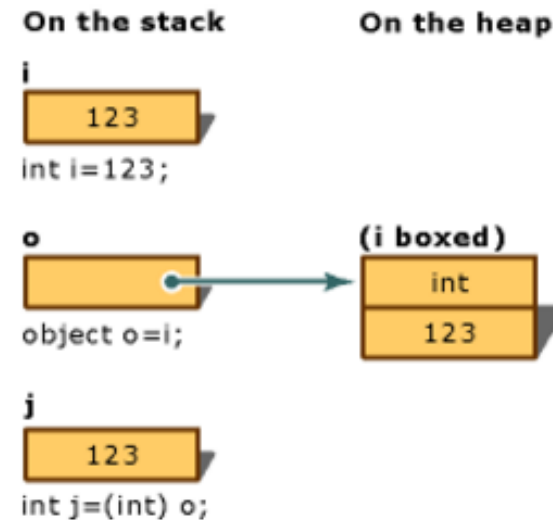
泛型是強型別的概念

避免容器操作的

## Boxing



## Unboxing



# 泛型

泛型類別 (C# 程式設計手冊)

```
class BaseNode { }  
class BaseNodeGeneric<T> { }
```

泛型介面 (C# 程式設計手冊)

```
interface IBaseInterface1<T> { }  
interface IBaseInterface2<T, U> { }
```

泛型方法 (C# 程式設計手冊)

```
void DoWork() { }  
void DoWork<T>() { }
```

泛型和陣列 (C# 程式設計手冊)

```
static void ProcessItems<T>(IList<T> coll)
```

泛型委派 (C# 程式設計手冊)

```
public delegate void Del<T>(T item);
```

**T** 是任意型別的意思 並**不是泛型**

**< >** 才是**泛型**

# 相似嗎

```
Product product;  
try  
{  
    product = JsonConvert.DeserializeObject<Product>(json);  
}  
catch (Exception)  
{  
    product = new Product();  
}
```

```
Order order;  
try  
{  
    order = JsonConvert.DeserializeObject<Order>(json);  
}  
catch (Exception)  
{  
    order = new Order();  
}
```

# 泛型方法

```
public static T JsonObject<T>(string json)
    where T:new()
{
    try
    {
        return JsonConvert.DeserializeObject<T>(json);
    }
    catch (Exception)
    {
        return new T();
    }
}
```



# 泛型約束

條件約束	描述
where T: struct	型別引數必須是實值型別。您可以指定 <code>Nullable</code> 以外的任何實值型別。如需詳細資訊，請參閱 <a href="#">用可為 Null 的類型</a> 。
where T: class	型別引數必須是參考型別，這是指任何類別、介面、委派或陣列型別。
where T: new()	型別引數必須擁有公用的無參數建構函式。將 <code>new()</code> 條件約束與其他條件約束一起使用時，一定要將其指定為最後一個。
where T: <base class name>	型別引數必須本身是指定的基底類別，或衍生自該類別。
where T: <interface name>	型別引數必須本身是指定的介面，或實作該介面。您可以同時指定多個介面條件約束。限制的介面也可以是泛型的。
where T: U	提供給 T 的型別引數必須是 (或衍生自) 提供給 U 的引數。

類型參數的條件約束

<https://msdn.microsoft.com/zh-tw/library/d5x73970.aspx>



# 設計一個類別相容下面 Json

```
{          {  
  "Version": "V1",    "Version": "V1",  
  "Data": {          "Data": "發生例外",  
    "ID": 87,        "ErrorCode": 3.1  
    "Name": "產品"  }  
  },  
  "ErrorCode": 0  
}
```

Data 跟 ErrorCode 物件跟型別不一樣  
如何克服



# 泛型介面

# 泛型類別

```
public interface IAPIData<T,U>
    where T:class
    where U:struct
{
    3 個參考| 0/1 通過
    string Version { get; set; }

    3 個參考| 0/1 通過
    T Data { get; set; }

    3 個參考| 0/1 通過
    U ErrorCode { get; set; }
}
```

```
public class APIData<T,U> : IAPIData<T,U>
    where T:class
    where U:struct
{
    3 個參考| 0/1 通過
    public string Version { get; set; }

    3 個參考| 0/1 通過
    public T Data { get; set; }

    3 個參考| 0/1 通過
    public U ErrorCode { get; set; }
}
```

# 設計一個類別相容下面 Json

```
{  
  "Version": "V1",  
  "Data": {  
    "ID": 87,  
    "Name": "產品"  
  },  
  "ErrorCode": 0  
}
```

```
public class APIData<T,U> : IAPIData<T,U>  
    where T:class  
    where U:struct  
{  
    3 個參考| 0/1 通過  
    public string Version { get; set; }  
  
    3 個參考| 0/1 通過  
    public T Data { get; set; }  
  
    3 個參考| 0/1 通過  
    public U ErrorCode { get; set; }  
}
```

```
return Helper.JsonToObject<APIData<Product, int>>(json);
```

# 設計一個類別相容下面 Json

```
{  
  "Version": "V1",  
  "Data": "發生例外",  
  "ErrorCode": 3.1  
}
```

```
public class APIData<T,U> : IAPIData<T,U>  
    where T:class  
    where U:struct  
{  
    3 個參考 | 0/1 通過  
    public string Version { get; set; }  
  
    3 個參考 | 0/1 通過  
    public T Data { get; set; }  
  
    3 個參考 | 0/1 通過  
    public U ErrorCode { get; set; }  
}
```

```
return Helper.JsonToObject<APIData<string, float>>(json);
```



# 泛型類別 可以包泛型類別

```
return Helper.JsonToObject<APIData<string, float>>(json);
```

JsonToObject<T> 泛型方法

T 是 APIData<string, float>

APIData<T, U> 泛型類別

T 是 string

U 是 float

# 請實作泛型方法

---

```
int id = 9;  
var typeName1 = id.GetType().Name; //Int32  
  
Product p = new Product();  
var typeName2 = p.GetType().Name; //Product
```

Helper.GetTypeName(id)  
Helper.GetTypeName(p)

# 請實作泛型方法

---

```
int id = 9;  
var typeName1 = id.GetType().Name; //Int32  
  
Product p = new Product();  
var typeName2 = p.GetType().Name; //Product
```

Helper.GetTypeName(id)  
Helper.GetTypeName(p)

# 請實作泛型約束

讓 `GetTypeName` 只能傳參考型別

`Helper.GetTypeName(id)` - Error

`Helper.GetTypeName(p)` - OK

`where T: struct`

`where T: class`

`where T: new()`

`where T: <base class  
name>`

`where T: <interface  
name>`

`where T: U`





# 實務運用 - 發現重複程式碼

```
// Mapping to SearchParameterDto
Mapper.CreateMap<HFIDsParameter, HFIDsParameterDto>();
HFIDsParameterDto parameterModel = Mapper.Map<HFIDsParameter, HFIDsParameterDto>(parameter);

var parameterJson = JsonConvert.SerializeObject(parameterModel);
var request = actionContext.Request;
request.Properties.Add("HFIDsParameterDto", parameterJson);
```

```
// 取得驗證後的查詢參數
var parameterJson =
    ControllerContext.Request.Properties["HFIDsParameterDto"].ToString();

var parameterDto =
    JsonConvert.DeserializeObject<HFIDsParameterDto>(parameterJson);
```

# 實務運用 - 泛型方法 Set

## 原程式碼

```
// Mapping to SearchParameterDto
Mapper.CreateMap<HFIDsParameter, HFIDsParameterDto>();
HFIDsParameterDto parameterModel = Mapper.Map<HFIDsParameter, HFIDsParameterDto>(parameter);

var parameterJson = JsonConvert.SerializeObject(parameterModel);
var request = actionContext.Request;
request.Properties.Add("HFIDsParameterDto", parameterJson);
```

## 改泛型方法

```
protected void SetParameterDto<T>(T parameterModel, HttpRequestMessage request) where T : IParameterDTO
{
    var parameterJson = JsonConvert.SerializeObject(parameterModel);
    request.Properties.Add(parameterModel.GetType().Name, parameterJson);
}
```

## 使用泛型方法

```
var parameterModel = Mapper.Map<RecommendParameterDto>(parameter);
SetParameterDto(parameterModel, actionContext.Request);
```

# 實務運用 - 泛型方法 Get

## 原程式碼

```
// 取得驗證後的查詢參數
var parameterJson =
    ControllerContext.Request.Properties["HFIDsParameterDto"].ToString();

var parameterDto =
    JsonConvert.DeserializeObject<HFIDsParameterDto>(parameterJson);
```

## 改泛型方法

```
protected T GetParameterDto<T>() where T : IParameterDTO
{
    var parameterJson = ControllerContext.Request.Properties[typeof(T).Name].ToString();
    var parameterDto = JsonConvert.DeserializeObject<T>(parameterJson);
    return parameterDto;
}
```

## 使用泛型方法

```
var recommendParameterDto = GetParameterDto<RecommendParameterDto>();
recommendParameterDto.MobileID = requestParameterDto.MobileID;
```

委派



# 參考型別 vs 實質型別

```
public void Demo1()
{
    var i = 5;
    var o = new Order()
    {
        ID = 9,
        Name = 'a'
    };

    Method(o, i);
    Console.WriteLine(i);
    Console.WriteLine(o.ID);
    Console.WriteLine(o.Name);
}
```

1 個參考 | 0 項變更 | 0 位作者, 0 項變更

```
private void Method(Order no, int ni)
{
    ni = 6;
    no.ID = 10;
    no.Name = 'b';
}
```



```
public class Order
{
    3 個參考 | 0 項變更 | 0 位作者, 0 項變更
    public int ID { get; set; }

    3 個參考 | 0 項變更 | 0 位作者, 0 項變更
    public char Name { get; set; }
}
```

# 參考型別 vs 實質型別


```
public void Demo1()
{
    var i = 5;
    var o = new Order()
    {
        ID = 9,
        Name = 'a'
    };

    Method(o, i);
    Console.WriteLine(i);
    Console.WriteLine(o.ID);
    Console.WriteLine(o.Name);
}
```

1 個參考 | 0 項變更 | 0 位作者, 0 項變更

```
private void Method(Order no,
{
    ni = 6;
    no.ID = 10;
    no.Name = 'b';
}
```

Stack			Heap	
i	06ffe8dc	5	02EDF258	ID = 10 Name = b
o	079eebc8	02EDF258		
ni	079eebb8	6		
no	079eeb70	02EDF258		



[C#] 基礎 - Value Type , Reference Type 用看記憶體內容 來測試

[https://dotblogs.com.tw/initials/2017/01/28/a00\\_basis](https://dotblogs.com.tw/initials/2017/01/28/a00_basis)

# 委派

---

委派是一種方法簽章的型別

委派可以用來將方法當做參數傳

C# 中的委派是重疊的（鏈式委派）

謝謝大家