

设计模式大作业

实现了三个组件，帮助快速开发程序。Applicatoin 下 ConsoleApplication(控制台应用)、EventAggragator(事件分发器)、IOC 容器。

一、IOC 容器

仿照 Microsoft.Extensions.DependencyInjection 包，编写一个 IOC 容器组件。两个核心类：ServiceCollection,ServiceProvider

1、ServiceCollection

用于服务的配置注册，在注册服务时会生成对应服务的描述符，存入服务描述符数组。ServiceCollection 在 BuildServiceProvider()后会将服务描述符数组传入 ServiceProvider，由 ServiceProvider 实现服务对象的创建和管理。

```
IServiceCollection builder = new ServiceCollection();
builder.RegisterTransient(IMyService.class,MyService.class);
builder.RegisterSingleton(MySingletonService.class);

IServiceProvider services = builder.BuildServiceProvider();
```

如上代码中：

1. 在 IOC 容器中注册了 MyService 类,作为瞬态服务，使用 IMyService 接口作为 MyService 的索引，只能用接口拿取对应的服务实例。
2. 在 IOC 容器中注册了 MySingletonService 类,作为单例服务，这里并没使用接口，可以直接用类拿取服务实例。

2、ServiceProvider

用于对 ServiceCollection 提供的注册服务进行实例创建、生命周期管理等功能。我们可以通过 GetService()函数后去服务实例。

```
// 继续上述代码
IMyService service1 = services.GetService(IMyService.class);
MySingletonService service2 = services.GetService(MySingletonService.class);
```

3、依赖注入的支持

(1)、构造函数注入

```
public class NumberCount
{
    private int number;
}

public class NumberService
{
    private NumberCount count;

    //在构造函数的参数列表中添加需要依赖的对象，IOC会自动注入
    public NumberService(NumberCount count)
    {
        this.count = count;
    }
}

public static Main()
{
    IServiceCollection builder = new ServiceCollection();
    builder.RegisterSingleton(NumberCount.class);
    builder.RegisterTransient(NumberService.class);

    IServiceProvider services = builder.BuildServiceProvider();
}
```

(2) 、注解注入

```
public class NumberCount
{
    private int number;
}

public class NumberService
{
    // 为字段添加注解，IOC会自动为这个注解添加实例
    @Dependence
    private NumberCount count;
}

public static Main()
{
    IServiceCollection builder = new ServiceCollection();
    builder.RegisterSingleton(NumberCount.class);
    builder.RegisterTransient(NumberService.class);

    IServiceProvider services = builder.BuildServiceProvider();
}
```

(3) 、IServiceProvider 获取

```
public class NumberCount
{
    private int number;
}

public class NumberService
{
    private NumberCount count;

    //在构造函数的参数列表中添加IServiceProvider, 通过IServiceProvider获取实例
    public NumberService(IServiceProvider provider)
    {
        count = provider.GetService(NumberCount.class);
    }
}

public static Main()
{
    IServiceCollection builder = new ServiceCollection();
    builder.RegisterSingleton(NumberCount.class);
    builder.RegisterTransient(NumberService.class);

    IServiceProvider services = builder.BuildServiceProvider();
}
```

二、EventAggregator (事件分发器)

观察者模式, 略

三、ConsoleApplication

1、内置功能

使用了 IOC 容器和 EventAggregator。

内置了三个控制器, ViewController(视图控制器:管理视图栈)、ConsoleController(控制台控制器:管理控制台运行逻辑)、IOController(输入输出控制器:管理控制台的输入输出)。

2、视图基类(ConsoleViewBase)

提供了视图基类, 项目中的视图需要继承 ConsoleViewBase。

(1)、基类实现的界面的导航功能

1. Translate("View1") 或者 Translate(View1.class),界面会转跳至 View1 界面。
2. Back() 退出本界面

(2)、添加视图功能

使用注解 `ConsoleCommand` 添加, `Name` 为命令的名称, `Index` 为命令排序。

```
@ConsoleCommand(Name = "View1", Index = 0)
public String View1() throws Exception {
    return Translate("View1");
}
```

注: `ConsoleApplication` 会自动将"Program/View"下的继承自 `ConsoleViewBase` 的类注入到 IOC 容器中, 不用手动注入。

3、如何使用

1. 创建一个 `Application` 类, 继承 `ConsoleApplication`, 作为项目的启动配置项。
2. 编写视图类, 继承 `ConsoleViewBase`, 实现抽象类的功能。要求放在"Program/View"文件夹下。

```
public class MainView extends ConsoleViewBase {
    public MainView(IViewController viewController) {
        super(viewController);
    }

    @Override
    protected String GetTitle() {
        return "Main View";
    }

    @ConsoleCommand(Name = "View1", Index = 0)
    public String View1() throws Exception {
        return Translate("View1");
    }
}

public class View1 extends ConsoleViewBase {
    public View1(IViewController viewController) {
    }

    @Override
    protected String GetTitle() {
        return "View1";
    }
}
```

这里我们创建了两个视图类 `MainView` 和 `View1`, `MainView` 可以通过执行"View1"命令转跳到 `View1` 界面。

3. 继续完善 `Application` 类

```
public class Application extends ConsoleApplication {

    public Application() throws Exception {
        super();
    }

    // 注册项目需要的服务
    @Override
    protected void RegisterService(IServiceCollection serviceCollection)
throws Exception {
    }

    // 设置初始界面（这里设为MainView）
    @Override
    protected Class<?> GetStartView() {
        return MainView.class;
    }
}
```

4. 运行即可。