02. 单例的模板

上一篇文章中说到的 manager of managers,其中每个 manager 都是单例的实现,当然也可以使用静态类实现,但是相比于静态类的实现,单例的实现更为通用,可以适用大多数情况。

如何设计这个单例的模板?

先分析下需求,当设计一个 manager 时候,我们希望整个程序只有一个该 manager 对象实例,一般马上能想到的实现是这样的:

```
public class XXXManager
{
    private static XXXManager instance = null;

    private XXXManager
     {
        // to do ...
    }

    public static XXXManager()
 {
        if (instance == null)
        {
            instance = new XXXManager();
        }
        return instance;
    }
}
```

如果一个游戏需要10个各种各样的manager,那么以上这些代码要复制粘贴好多遍。重复的代码太多!!!想要把重复的代码抽离出来,怎么办?答案是引入泛型。实现如下:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Reflection;

namespace QFramework
{
    public abstract class QSingleton<T> where T : QSingleton<T>
    {
        protected static T instance = null;

        protected QSingleton()
        {
        }

        public static T Instance()
        {
            if (instance == null)
            {
                // 如何new 一个T???
            }

            return instance;
        }
    }
}
```

为了可以被继承,静态实例和构造方法都使用protect修饰符。以上的问题很显而易见,那就是不能new一个泛型(3月9日补充:并不是不能new一个泛型,参考:new一个泛型的实例，编译失败了，为什么?-CSDN论坛-CSDN.NET-中国最大的IT技术社区),(4月5日补充:有同学说可以new一个泛型的实例,不过要求改泛型提供了public的构造函数,好吧,这里不用new的原因是,无法显示调用private的构造函数)。因为泛型本身不是一个类型,那该怎么办呢?答案是使用反射。实现如下:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Reflection;

/// <summary>
/// 1.泛型
/// 2.反射
/// 3.抽象类
/// 4.命名空间
/// </summary>
namespace QFramework
{
    public abstract class QSingleton<T> where T : QSingleton<T>
    {
        protected static T instance = null;

        protected QSingleton()
        {
        }

        public static T Instance()
        {
            if (instance == null)
            {
                // 先获取所有非public的构造方法
                ConstructorInfo[] ctors =
typeof(T).GetConstructors(BindingFlags.Instance |
BindingFlags.NonPublic);
                // 从ctors中获取无参的构造方法
                ConstructorInfo ctor = Array.Find(ctors, c =>
c.GetParameters().Length == 0);
                if (ctor == null)
                    throw new Exception("Non-public ctor() not
found!");
                // 调用构造方法
                instance = ctor.Invoke(null) as T;
            }

            return instance;
        }
    }
```

```
}
```

以上就是最终实现了。这个实现是在任何C#程序中都是通用的。其测试用例如下所示:

```
using QFramework;
// 1.需要继承QSingleton。
// 2.需要实现非public的构造方法。
public class XXXManager : QSingleton<XXXManager>
{
    private XXXManager()
    {
        // to do ...
    }
}


public static void main(string[] args)
{
    XXXManager.Instance().xxxyyyzzz();
}
```

总结:

这个单例的模板是平时用得比较顺手的工具了,其实现是在其他的框架中发现的,拿来直接用了。反射的部分可能会耗一些性能,但是只会执行一次。在 Unity 中可能会需要继承 MonoBehaviour 的单例,因为很多游戏可能会只创建一个GameObject,用来获取 MonoBehaviour 的生命周期,这些内容会再下一讲中介绍:)。

转载请注明地址：凉鞋的笔记：liangxiegame.com

更多内容

- QFramework 地址：https://github.com/liangxiegame/QFramework

- QQ 交流群：623597263
- **Unity 进阶小班**：
  - 主要训练内容：
    - 框架搭建训练（第一年）
    - 跟着案例学 Shader（第一年）
    - 副业的孵化（第二年、第三年）
  - 权益、授课形式等具体详情请查看《小班产品手册》：https://liangxiegame.com/master/intro
- 关注公众号：liangxiegame 获取第一时间更新通知及更多的免费内容。