**Singleton**

**设计初衷**

本类用于构建不依赖 MonoBehaviour 生命周期的全局管理类，适合数据管理、配置读取、逻辑控制等 Unity 场景之外的单例对象。

**文件路径**

文件路径：Frame/Scripts/Singleton/

示例：Frame/Test/Singleton/

**使用方法**

继承该类并实现 Reset()方法：

*public class GameData : Singleton<GameData>*

*{*

*public string PlayerName;*

*public override void Reset()*

*{*

*// 清空资源*

*PlayerName = string.Empty;*

*}*

*}*

如果需要初始化，则可以在构造函数中进行，子类构造函数需声明为 protected 或 private。  
**公共方法  
 -**Instance: 获取单例实例；

-IsInstantiated: 该单例是否已经被实例化？若是则返回true，若还未被实例化返回false；

-Reset: 重置该单例类，如过场景时清空当前数据。它不会销毁实例；

**描述**

1. 泛型单例模式

通过 where T : Singleton<T> 的泛型约束，确保子类能够正确返回自身类型的单例实例。

1. 线程安全的懒加载

使用双重检查锁（Double-Check Locking）保证多线程环境中只会创建一个实例，避免竞态条件。

1. 构造函数支持初始化

子类可以在构造函数中编写初始化逻辑，无需依赖 Unity 的 Awake 或 Start 生命周期函数。

1. 实例化状态判断

提供 IsInstantiated 属性，用于判断单例是否已被创建，便于在某些逻辑中做安全检查。

1. 生命周期接口支持

实现了 IResetable，可用于在场景切换、模块热更、资源回收等场景中重置。

**优化方向**

1. 没有提供销毁方法，单例一旦创建就会伴随整个程序的生命周期，未来可创建单例管理器，统一管理单例的生命周期，明确Reset和Destroy时机；
2. 没有对Reset()调用进行限制，需要开发者明确知道Reset调用时机。为了防止误用，可引入权限、密钥或Internal接口等方法控制Reset()调用权限。