unity对象池

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class shaderPool : MonoBehaviour
{
public  static shaderPool instance;


public int poolCount;//对象池的大小
public GameObject shaderPrefab;
private Queue<GameObject> availableObject = new Queue<GameObject>();//对象池


private void Awake()
{
    instance = this;
    //初始化对象池
    FillPool();
}
public void FillPool()//初始化，填充对象池
{
    for (int i = 0; i < poolCount; i++)
    {
        var newShader = Instantiate(shaderPrefab);
        newShader.transform.SetParent(transform);//初始化的对象跟随父物体

        // 生成之后要立马取消使用，所以返回对象池
        ReturnPool(newShader);
    }
}
public void ReturnPool(GameObject gameObject)//返回对象池
{
    gameObject.SetActive(false);
    availableObject.Enqueue(gameObject);//把不用的对象返回到队列中，等待再次使用。
}
public GameObject GetFeomPool()
{
    if (availableObject.Count == 0)
    {
        FillPool();//如果运行中对象池的对象不够用，则再次填充对象
    }
```

```
    var outshader = availableObject.Dequeue();//出队。
    outshader.SetActive(true);
    return outshader;
}
```

}

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ObjectPool : MonoBehaviour
{
    public static ObjectPool instance;
    //通过字典可以快速的根据名称查找到相应对象池，使用队列可以方便的删减元素(无需指定元素)
    private Dictionary<string, Queue<GameObject>> objectPool = new Dictionary<string, Queue<GameObject>>();

    private GameObject pool;//作为所有对象池的父物体，便于管理。

    private void Awake()
    {
        instance = this;
    }

    public GameObject GetObjectPool(GameObject perfab)
    {
        GameObject gameObject;
        /*现在已经知道 了所需的游戏对象，我们要从对象池中获得这个对象。现在分为三种情况:
         * @相应对象池中有剩余对象
         * ②相应对象池中没有剩余对象
         * ③没有相应对象池
         * 对应的处理办法:
         * @取出相应对象池的对象，激活，并返回
         * @实例化一个对象，并将其放入对象池中，执行
         * ③实例化一个对象，(创建新的对象池) 并将其放入对象池中，执行
         * 使用if区分不同的处理方法，将共同的处理方法放在外面*/
        if (!objectPool.ContainsKey(perfab.name) || objectPool[perfab.name].Count == 0)//③，②
        {
            gameObject = GameObject.Instantiate(perfab);//实例化预制体
```

```
33
34          if (pool == null)
35          {
36              pool = new GameObject("objectPool");//创建一个物体管理所有对象池
37          }
38          //寻找并绑定相应对象池物体，如果对象物体不存在则创建一个对象池 物体
39          GameObject childPool = GameObject.Find(perfab.name + "Pool");
40          if (!childPool)
41          {
42              childPool = new GameObject(perfab.name + "Pool");
43          }
44          childPool.transform.SetParent(pool.transform);
45          gameObject.transform.SetParent(childPool.transform);
46
47          PushObjectPool(gameObject);//把预制体放到对象池
48      }
49      gameObject = objectPool[perfab.name].Dequeue();
50      gameObject.SetActive(true);
51      return gameObject;
52  }
53  //进入对象池
54  public void PushObjectPool(GameObject prefab)
55  {
56      /*我们得到了 需要放入对象池的对象，现存在两种情况
57      @存在对应对象池
58      @不存在对应对象池
59      对应的处理办法
60      @将对象放入相应对象池中并使其失效
61      @向字典中添加新的对象池元素，执*/
62      string name = prefab.name.Replace("(Clone)", string.Empty);//获得对象的名称
63      if (!objectPool.ContainsKey(name))//判断对象池中是否有这个对象
64      {
65          objectPool.Add(name, new Queue<GameObject>());//创建对象池
66      }
67      objectPool[name].Enqueue(prefab);//加入对象池中
68      prefab.SetActive(false);
69  }
70
71 }
72
```

}