# **回顾游戏中的设计模式-策略模式vs抽象工厂**

作者：[Mitty](http://gad.qq.com/user/index?id=363714)

链接：<http://gad.qq.com/article/detail/287679>

前言：

最近有时间看看设计模式方面的书，好多东西时间久了不看就变得生疏起来， 在这里会抽出时间把自己在工作中使用过的设计模式做下总结。

刚才在看到别人说，简单工厂模式【通过提供单独的一个类，来实现创建实例对象的过程】，可以使用反射来替换掉程序中的switch/if..else，嗯，这是没问题的，但实际应用中，在移动端还要尽量少的去频繁使用反射Reflection，严重依赖字符串的功能都会存在效率的问题。

今天在看到策略模式（Strategy Pattern)的时候，突然间意识到自己在16年的一个项目中，有一处应用不够合理。

当时在存储游戏数据部分，我是通过直接将对象图转化成字节流的形式，即序列化。

那么序列化有多种方式，你可以序列化成XML，也可以是二进制，或者是SOAP（类似于XML，不建议使用，只是为了多提供一个策略演示)

那么这时候，采用哪种设计模式？

当时使用的是抽象工厂，但实际上，他更符合策略模式。即我们将多个“策略”抽象成接口的形式来解耦合。

比如说我要回大连，我可以坐火车，飞机，客车，或是自驾。我最终的目的是相同的，只是到达目的的方式不同。

然而在代码上，策略模式和抽象工厂区别不是很大，后来我搜索了一下关于两个设计模式之间的区别，找到了一个非常好的解释 。

**策略模式vsr抽象工厂的区别**

https://bbs.csdn.net/topics/320073328

2009年的一个帖子。

解释如下：

**抽象工厂属于创建型的，而策略模式属于行为型。**

**抽象工厂往往是抽取的同一基类(Class)的不同子类。**

**策略模式往往抽取的是同一接口的不同实现。**

那么显然，我在序列化的时候，我序列化为哪种格式，只是行为不同（纯行为上），结果相同，而且也符合同一接口的不同实现。

那么说到这里了，就把之间的代码给修改一下，修改为策略模式。

定义序列化接口：

**ISerializable.cs**

public interface ISerializable ｛

void Serialize<T> (string filePath, T data);

T Deserialize<T> (string filePath);

｝

声明了两个方法，序列化和反序列化。

下面实现具体的序列化类，分别是Binary,XML,SOAP

**BinarySerialized.cs**

using System.IO;

using System;

using System.Xml;

using System.Runtime.Serialization.Formatters.Binary;

public class BinarySerialized : ISerializable ｛

public void Serialize<T> (string filePath, T data)

｛

using (FileStream fs = new FileStream(filePath, FileMode.Create))

｛

BinaryFormatter bf = new BinaryFormatter();

bf.Serialize(fs, data);

｝

｝

public T Deserialize<T> (string filePath)

｛

if (!File.Exists(filePath))

｛

return default(T);

｝

// return base.Deserialize<T>();

using (FileStream fs = new FileStream(filePath, FileMode.Open))

｛

BinaryFormatter bf = new BinaryFormatter();

T data = (T)bf.Deserialize(fs);

if (data != null)

｛

return data;

｝

｝

return default(T);

｝

｝

**XMLSerialized .cs**

using System.IO;

using System.Xml;

using System.Xml.Serialization;

public class XMLSerialized : ISerializable ｛

public void Serialize<T> (string filePath, T data)

｛

using (XmlTextWriter xWrite = new XmlTextWriter(filePath, null))

｛

XmlSerializer sl = new XmlSerializer(data.GetType());

sl.Serialize(xWrite, data);

｝

｝

public T Deserialize<T> (string filePath)

｛

if (!File.Exists(filePath))

｛

return default(T);

｝

using (XmlReader xRead = new XmlTextReader(filePath))

｛

XmlSerializer sl = new XmlSerializer(typeof(T));

T data = (T)sl.Deserialize(xRead);

if (data != null)

｛

return data;

｝

｝

return default(T);

｝

｝

**SOAPSerialized.cs(SOAP要引入dll[System.Runtime.Serialization.Formatters.Soap.dll])**

using System.IO;

using System.Runtime.Serialization.Formatters.Soap;

public class SOAPSerialized : ISerializable ｛

public void Serialize<T> (string filePath, T data)

｛

using (FileStream fs = new FileStream(filePath, FileMode.Create))

｛

SoapFormatter bf = new SoapFormatter();

bf.Serialize(fs, data);

｝

｝

public T Deserialize<T> (string filePath)

｛

if (!File.Exists(filePath))

｛

return default(T);

｝

using (FileStream fs = new FileStream(filePath, FileMode.Open))

｛

SoapFormatter bf = new SoapFormatter();

T data = (T)bf.Deserialize(fs);

if (data != null)

｛

return data;

｝

｝

return default(T);

｝

｝

**最后我们定义一个序列化的上下文，面向接口编程：**

**SerializableContext.cs**

public class SerializableContext｛

private ISerializable SerializableStrategy;

public SerializableContext(ISerializable strategy)

｛

this.SerializableStrategy = strategy;

｝

public void ExecuteSerialize<T> (string filePath, T data)

｛

SerializableStrategy.Serialize<T> (filePath, data);

｝

public T ExecuteDeserialize<T> (string filePath)｛

return SerializableStrategy.Deserialize<T> (filePath);

｝

｝

**演示代码：**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class SerializedDemo: MonoBehaviour ｛

public SerializableContext context;

private string FILE\_PATH;

DataManager data;

// Use this for initialization

void Start () ｛

FILE\_PATH = Application.persistentDataPath + "/test20181015";

data = new DataManager();

｝

// Update is called once per frame

void Update () ｛

｝

public void OnGUI()

｛

if (GUILayout.Button ("Serialized XML")) ｛

context =new SerializableContext (new XMLSerialized());

context.ExecuteSerialize (FILE\_PATH, data);

｝

if (GUILayout.Button ("Serialized Binary")) ｛

context =new SerializableContext (new BinarySerialized());

context.ExecuteSerialize (FILE\_PATH, data);

｝

if (GUILayout.Button ("Serialized SOAP")) ｛

context =new SerializableContext (new SOAPSerialized());

context.ExecuteSerialize (FILE\_PATH, data);

｝

if (GUILayout.Button ("Deserialized")) ｛

DataManager d = context.ExecuteDeserialize<DataManager>(FILE\_PATH);

if (d != null)

｛

d.debug();

｝

｝

｝

｝

[System.Serializable]

public class DataManager

｛

List<Person> personList;

List<PetData> petList;

public DataManager()

｛

personList = new List<Person>();

Person person1 = new Person();

person1.id = 1;

person1.name = "one";

Person person2 = new Person();

person2.id = 2;

person2.name = "two";

personList.Add(person1);

personList.Add(person2);

petList = new List<PetData>();

PetData pet1 = new PetData();

pet1.id = 10;

pet1.name = "john";

pet1.petQuality = 1;

PetData pet2= new PetData();

pet2.id = 20;

pet2.name = "lucy";

pet2.petQuality = 2;

PetData pet3 = new PetData();

pet3.id = 30;

pet3.name = "david";

pet3.petQuality = 3;

petList.Add(pet1);

petList.Add(pet2);

petList.Add(pet3);

｝

public void debug()

｛

foreach (Person p in personList)

｛

Debug.Log(p.ToString());

｝

foreach (PetData p in petList)

｛

Debug.Log(p.ToString());

｝

｝

｝

[System.Serializable]

class Person

｛

public int id;

public string name;

public string ToString()

｛

return "Person:id:" + id.ToString() + ",name:" + name;

｝

｝

[System.Serializable]

class PetData

｛

public int id;

public string name;

public int petQuality;

public string ToString()

｛

return "PetData:id:" + id.ToString() + ",name:" + name+",quality:"+petQuality.ToString();

｝

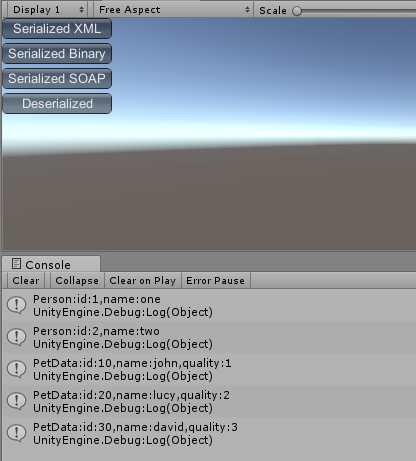
｝

**在这里定义了一个DataManager类，DataManager包含了**

List<Person> personList;

List<PetData> petList;

**两个字段，并进行初始化，然后对DataManager类，进行序列化和反序列化测试。**



**演示代码下载（微云）：**

**链接：https://share.weiyun.com/5Cg9MT4**

**感谢您的阅读，如文中有误，欢迎指正~**