C# FileStream文件流

流是面向对象的抽象概念 是二进制字节序列

在计算机中 文件是保存在磁盘中的进制字节 使用固定的格式存储的信息

使用者可以对文件进行读取 写入等操作 所以文件是静态的

当一个文件被打开对其读写操作 这个文件就成为了流(Stream)

它是一种动态的特殊的数据结构

文件流 主要用于读取和写入磁盘中的文件

用来存储数据或者读取配置文件

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;//输入输出流的命名空间
using System.Text;//文本命名空间

public class Stream : MonoBehaviour {

    void OnGUI()
    {
        if (GUILayout.Button("写入数据"))
        {
            //打开文件流  可创建可追加 初始化文件流
            FileStream fileStream = File.Open(Application.dataPath + "/myText.txt",FileMode.Append);
            //把字符串转换为byte数组
            byte[] array = Encoding.UTF8.GetBytes("你好中国");
            //写入文本
            fileStream.Write(array, 0, array.Length);
            //关闭文件流
            fileStream.Close();

        }
        if (GUILayout.Button("读取数据"))
        {
```

```
            //打开文件流  只读方式初始化文件流
            FileStream fileStream = File.Open(Application.dataPath +
"/myText.txt", FileMode.Open);

            byte[] array = new byte[fileStream.Length];

            fileStream.Read(array, 0, array.Length);

            string str = Encoding.UTF8.GetString(array);

            Debug.Log(str);

            fileStream.Close();
        }
    }
}

```

StreamReader/StreamWriter 读写流

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;//输入输出流的命名空间
using System.Text;//文本命名空间

public class Stream : MonoBehaviour {

    string path;

    void Start()
    {
        path = Application.dataPath + "/MyText.txt";
    }
    void OnGUI()
    {
        if (GUILayout.Button("写入数据"))
        {
            //写入流  路径 是否追加 编码格式
            StreamWriter writer = new StreamWriter(path, true,
```

```
Encoding.UTF8);

            //逐行写入
            writer.WriteLine("Good 再见");
            writer.WriteLine("Good 再见");
            writer.WriteLine("Good 再见");
            writer.Close();
        }
        if (GUILayout.Button("读取数据"))
        {
            //读取流  路径 编码格式
            StreamReader reader = new StreamReader(path,
Encoding.UTF8);

            //没有读到结尾就一直循环
            while(!reader.EndOfStream)
            {
               string str=reader.ReadLine();//读取一行
               Debug.Log(str);
            }
            reader.Close();

        }
    }
}
```

数据持久化

即 游戏开发中对数据，内容等信息的存储方式（存档）

数据持久化的存储类型：

1 以文件的方式来存储

包括 二进制文件 自定义文件 XML文件 Json

自定义文件：一般是.bat和.dat文件

bat文件是windows的批处理文件 实际上本身就是txt文本文件

bat和dat都可以直接转化为文本文件

dat文件：一种是VCD的媒体文件 另外一种是数据文件

使用读写流生成和读取自定义文件 一般来说保密性不够

提高保密性的方法 是文件序列化数据 再通过反序列化读取

（把对象的数据转化为二进制方式存储 反序列化把二进制文件转为类的结构数据）

2 数据库(DataBase)

单机数据库：SQLite(安卓内置数据库 开源的无加密的)

网络数据库：Oracle 一般使用MySQL SQLServer

3 Unity存储

用户自定义数据 如果是简单数据 使用Unity自带的PlayerPrefs类进行存储

如果是大量数据 还是交由服务器存储在数据库中

序列化和反序列化

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;//系统
using System.IO;//输入输出流
using System.Text;//文本命名空间
using System.Runtime.Serialization.Formatters.Binary;//二进制命名空间

[Serializable]//子类要被序列化 父类也必须添加Serializable特性
public class Data
{
    public string name;//姓名
    public int id;//编号

    public Data(string name,int id)
    {
        this.name = name;
        this.id = id;
    }
}

```

```
[Serializable]//说明当前类型 是可以被序列化的
public class PlayerData:Data
{
    private int level;//等级

    [NonSerialized]//当前成员不会被序列化
    private bool b;

    public PlayerData(string name,int id,int level,bool b):base(name,id)
    {
        this.level = level;
        this.b = b;
    }
}

public class DataManager : MonoBehaviour {

    List<PlayerData> players = new List<PlayerData>();

    string fileName;//文件的存储路径
    void Start () {
        fileName = Application.dataPath + "/Players.dat";
        players.Add(new PlayerData("张三", 101, 10, true));
        players.Add(new PlayerData("李四", 102, 30, true));
        players.Add(new PlayerData("王五", 103, 120, false));
        players.Add(new PlayerData("赵六", 113, 20, false));
    }


    void OnGUI () {
        if (GUILayout.Button("序列化数据"))
        {
            FileStream fStream = new FileStream(fileName, FileMode.OpenOrCreate);
            BinaryFormatter bFormat = new BinaryFormatter();//初始化二进制序列化器
            bFormat.Serialize(fStream, players);//序列化数据
            fStream.Close();
        }
        if (GUILayout.Button("反序列化数据"))
        {
```

```
            players.Clear();

            FileStream fStream = new FileStream(fileName, FileMode.Open);

            BinaryFormatter bFormat = new BinaryFormatter();//初始化二进制序列化器

            players=bFormat.Deserialize(fStream)as List<PlayerData>;//反序列化数据

            foreach (var it in players)
            {
                Debug.Log(it.name);
            }

            fStream.Close();
        }
    }
}
```

PlayerPrefs 玩家偏好

Unity用于本地持久化保存与读取的类

以键值的形式将数据存储在文件中 然后根据键取值获取存储数据

PlayerPrefs类支持的数据类型：int float string

需要存储对象 先把对象的信息存为固定的字符串形式 然后存入PlayerPrefs

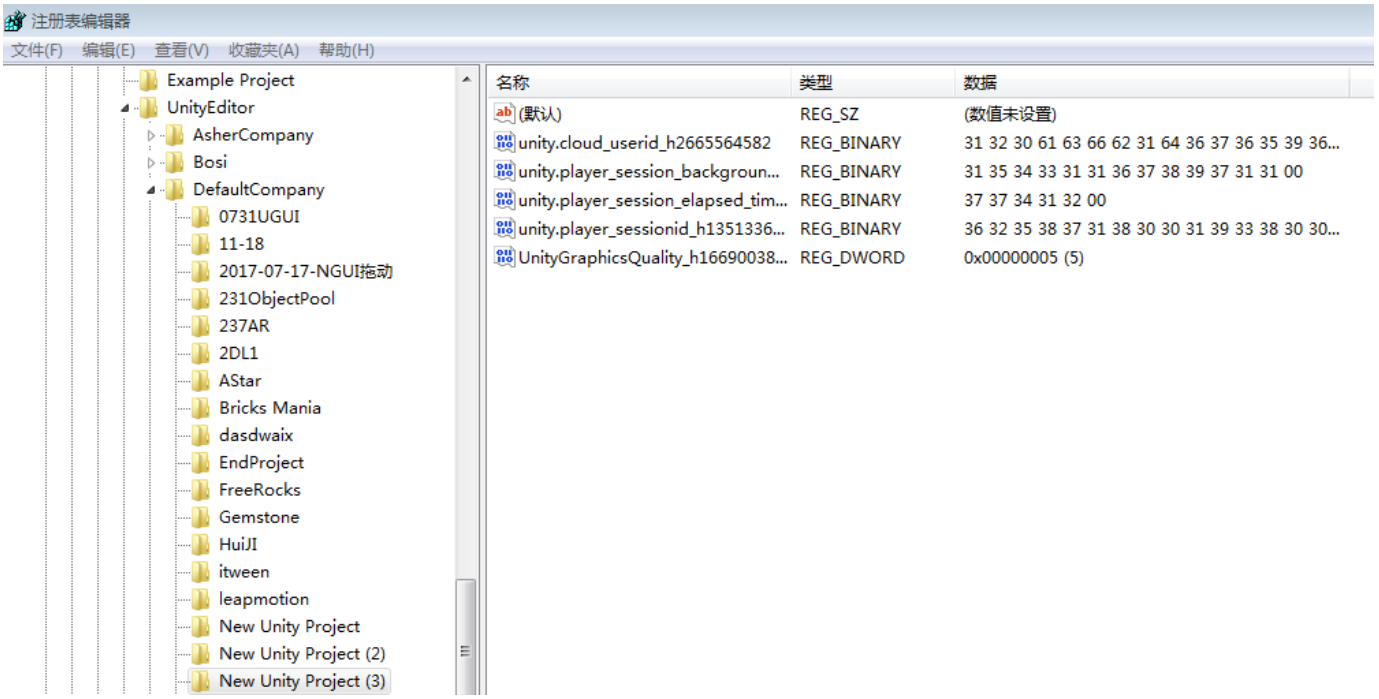
一般网游开发 如果涉及到客户端本地保存玩家设置的情况 可以使用PlayerPrefs实现

```
public class DataManager : MonoBehaviour {
    string data = "";
    string key = "gameData";

    void Start () {
        //PlayerPrefs.SetFloat("a", 1.0f);//a=1.0f
```

```
        //PlayerPrefs.GetFloat("a");//a=1.0f

        //是否包含gameData key
        if (PlayerPrefs.HasKey(key))
        {
            //根据key取value
            data = PlayerPrefs.GetString(key);

            //PlayerPrefs.DeleteAll();//删除所有注册表中存储的键值信息
            //PlayerPrefs.DeleteKey(key);//删除某一个键值
        }
    }

    void OnGUI () {
        data = GUI.TextArea(new Rect(100, 100, 100, 50), data);
        if (GUI.Button(new Rect(200, 100, 100, 50), "Save"))
        {
            //键值对   一个键对应一个值 键不能重复
            PlayerPrefs.SetString(key, data);
        }
    }
}
```

存储位置：

运行regedit->HKEY-CURRENT-USER->SoftWare->Unity->UnityEditor->DefaultCompany->ProjectName



练习：

使用OnGUI制作两个输入框 用来输入用户名和密码

制作两个按钮 登录和注册

当用户没有注册时 点登录 提示 "请先注册"

点击注册时 提示"注册成功" 并存储用户的账号和密码

退出Unity重新运行

用户输入账号密码

判断用户输入的是否正确 提示是否登录成功

```
public class DataManager : MonoBehaviour {

    string username = "请输入用户名";
    string password = "请输入密码";

    void OnGUI () {
        username = GUI.TextArea(new Rect(100, 100, 200, 20), username);
        password = GUI.TextArea(new Rect(100, 130, 200, 20), password);
        if (GUI.Button(new Rect(100, 160, 50, 40), "登录"))
        {
            if (!PlayerPrefs.HasKey("username"))
            {
                Debug.Log("请先去注册");
            }
            else
            {
                if (PlayerPrefs.GetString("username").Equals(username) &&
 PlayerPrefs.GetString("password").Equals(password))
                {
                    Debug.Log("登录成功");
                }
                else
                    Debug.Log("用户名或密码输入错误");
```

```
            }
        }
        if (GUI.Button(new Rect(170, 160, 50, 40), "注册"))
        {
            PlayerPrefs.SetString("username", username);
            PlayerPrefs.SetString("password", password);
            Debug.Log("注册成功");
        }
    }
}
```

Json

轻量级的数据交换格式

以文本格式来存储和表示数据 简介清晰 易于在网络中进行数据传递

Json整体使用{}表示

在Json字符串中 表示string 使用两个双引号 "" ""

前后数据的对应关系使用冒号 :

数据和数据之间使用逗号区分

表示集合使用[]表示

表示集合中的元素 每一个元素 使用{}表示 中间用逗号隔开

```
{
    ""ClassName"":""程序班"",
    ""ClassID"":244,
    ""openingTime"":""2018-10-10"",
    ""Students"":
    [
        {
        ""StuName"":""张三"",
        ""url"":""http://zhagnsan.com"",
        ""StuHeight"":180
        },
        {
```

```
        ""StuName"":""张三"",
        ""url"":""http://zhagnsan.com"",
        ""StuHeight"":180
        },
        {
        ""StuName"":""张三"",
        ""url"":""http://zhagnsan.com"",
        ""StuHeight"":180
        }
    ]
}
```

在Unity中构建和解析Json需要使用第三方类库 LitJson

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using LitJson;//LitJson命名空间
using System.Text;

public class DataManager : MonoBehaviour {


    void OnGUI () {
        if (GUILayout.Button("解析Json"))
        {
            ResolveJson();
        }
        if (GUILayout.Button("构建Json"))
        {
            MergerJson();
        }
    }

    void ResolveJson()
    {
        string str = @"{
        ""ClassName"":""程序班"",
        ""ClassID"":244,
```

```
        ""openingTime"":""2018-10-10"",
        ""Students"":
            [
            {
            ""StuName"":""张三"",
            ""url"":""http://zhagnsan.com"",
            ""StuHeight"":180
            },
            {
            ""StuName"":""张三"",
            ""url"":""http://zhagnsan.com"",
            ""StuHeight"":180
            },
            {
            ""StuName"":""张三"",
            ""url"":""http://zhagnsan.com"",
            ""StuHeight"":180
            }
            ]
         }";

        //把json字符串 直接转化为JsonData类型的对象
        JsonData jd = JsonMapper.ToObject(str);

        Debug.Log((string)jd["ClassName"]);
        Debug.Log((int)jd["ClassID"]);
        Debug.Log((string)jd["openingTime"]);

        //获取Json字符串中 Students集合
        JsonData jdItems = jd["Students"];

        for (int i = 0; i < jdItems.Count; i++)
        {
            Debug.Log("学生:" + jdItems[i]["StuName"]);
            Debug.Log("学生:" + jdItems[i]["url"]);
            Debug.Log("学生:" + (int)jdItems[i]["StuHeight"]);
        }

    }

    void MergerJson()
    {
```

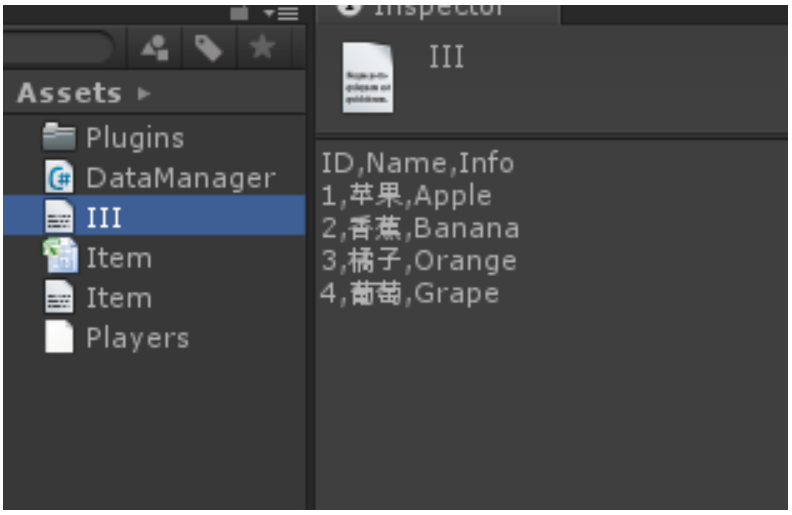
```
        StringBuilder sb = new StringBuilder();
        JsonWriter writer = new JsonWriter(sb);

        //开始写入对象
        writer.WriteObjectStart();

        //写入属性名字
        writer.WritePropertyName("Name");
        //写入属性的值
        writer.Write("学生");
        writer.WritePropertyName("Age");
        writer.Write(26);


        writer.WritePropertyName("Boys");
        //开始写入集合
        writer.WriteArrayStart();

        // 写入集合中的第一个对象
        writer.WriteObjectStart();
        writer.WritePropertyName("name");
        writer.Write("张三");
        writer.WritePropertyName("age");
        writer.Write("22");
        //结束写入第一个对象
        writer.WriteObjectEnd();

        // 写入集合中的第二个对象
        writer.WriteObjectStart();
        writer.WritePropertyName("name");
        writer.Write("李四");
        writer.WritePropertyName("age");
        writer.Write("26");
        //结束写入第二个对象
        writer.WriteObjectEnd();

        //结束写入集合
        writer.WriteArrayEnd();

        //结束写入对象
        writer.WriteObjectEnd();

```

```
110         Debug.Log(sb.ToString());
111
112         JsonData jd = JsonMapper.ToObject(sb.ToString());
113         Debug.Log("name" + jd["Name"]);
114         Debug.Log("name" + jd["Age"]);
115
116         JsonData items = jd["Boys"];
117
118         for (int i = 0; i < items.Count; i++)
119         {
120             Debug.Log(items[i]["name"]);
121         }
122     }
123 }
124
```

CSV逗号分隔文件的读取

Banana

	A	B	C	D
1	ID	Name	Info	
2	1	鑻规灘	Apple	
3	2	棣欑晧	Banana	
4	3	姗樺璟	Orange	
5	4	鑸ㄖ悇	Grape	
6				
7				
8				
9				




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

public class Fruit
{
    public int id;
    public string name;
    public string info;
}
public class DataManager : MonoBehaviour {
    public TextAsset textAsset;

    void Start () {
        Debug.Log(textAsset.text);
        LoadCSV();
    }

    void LoadCSV()
    {
        string str = textAsset.text;
        str=str.Replace("\r\n", "|");

        string[] datas = str.Split('|');

        for (int i = 1; i < datas.Length; i++)
        {
            string[] fruit=datas[i].Split(',');
            Fruit f = new Fruit();
```

```
            f.id = int.Parse(fruit[0]);
            f.name= fruit[1];
            f.info = fruit[2];
            Debug.Log(f.name + ":" + f.id + "---" + f.info);
        }

    }
}
```

XML

一种可扩展的标记语言

在程序开发中一般用于传输个存储数据 作为配置文件来使用

XML可以简单实现不同系统之间的数据交换

因为XML数据以文本格式存储 使得XML可以在不损失任何数据的情况下完成扩展和升级

XML文档形成树结构

XML文档必须包含根元素 根元素是文档中其他所有元素的父元素

XML文档中的元素形成一颗文档书 这个树从根部开始 并扩展到树的最底端

所有元素均可以拥有子元素 相同层级的子元素是兄弟元素

```
<?xml version="1.0" encoding="UTF-8"?>
<PlayerConfig>
    <Player name="张无忌" id="1">
        <Atk>1000</Atk>
        <Atk>800</Atk>
        <Atk>500</Atk>
    </Player>
    <Player name="赵敏" id="2">
        <Atk>500</Atk>
        <Atk>100</Atk>
        <Atk>200</Atk>
    </Player>
```

```
    <Player name="周芷若" id="3">
        <Atk>500</Atk>
        <Atk>100</Atk>
        <Atk>200</Atk>
    </Player>
</PlayerConfig>
```

PlayerConfig是当前XML文档的根元素

一共有3个Player元素 他们是PlayerConfig的子元素 所以他们是兄弟关系

每一个Player元素 都包含name 和 id两个属性 又包含多个Atk的子元素

XML写入

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Xml;//XML命名空间

public class XMLWriter : MonoBehaviour {

    void Start () {
        //创建一个XML文档
        XmlDocument xmldoc = new XmlDocument();

        XmlDeclaration
xmlDeclaration=xmldoc.CreateXmlDeclaration("1.0", "UTF-8", null);

        //把创建的声明段落 添加到文档中
        xmldoc.AppendChild(xmlDeclaration);

        //使用文档创建一个新元素
        XmlElement xmle1=xmldoc.CreateElement("StudentConfig");

        //把创建的根节点添加到文档中
        xmldoc.AppendChild(xmle1);

        for (int i = 0; i < 3; i++)
        {
```

```
            XmlNode node=xmldoc.SelectSingleNode("StudentConfig");
            XmlElement xmle2= xmldoc.CreateElement("Student");
            xmle2.SetAttribute("Name", "张三");
            xmle2.SetAttribute("Age", "26");

            XmlElement xmle3=xmldoc.CreateElement("Address");
            xmle3.InnerText = "中国上海";
            xmle2.AppendChild(xmle3);

            XmlElement xmle4 = xmldoc.CreateElement("Info");
            xmle4.InnerText = "一个小男孩";
            xmle2.AppendChild(xmle4);

            XmlElement xmle5 = xmldoc.CreateElement("Date");
            xmle5.InnerText = "2018-12-17";
            xmle2.AppendChild(xmle5);

            node.AppendChild(xmle2);

        }
        xmldoc.Save(Application.dataPath + "/StudentConfig.xml");
    }
}

```

得到的XML文档：

```
<?xml version="1.0" encoding="UTF-8"?>
<StudentConfig>
  <Student Name="张三" Age="26">
    <Address>中国上海</Address>
    <Info>一个小男孩</Info>
    <Date>2018-12-17</Date>
  </Student>
  <Student Name="张三" Age="26">
    <Address>中国上海</Address>
    <Info>一个小男孩</Info>
    <Date>2018-12-17</Date>
  </Student>
  <Student Name="张三" Age="26">
```

```
    <Address>中国上海</Address>
    <Info>一个小男孩</Info>
    <Date>2018-12-17</Date>
  </Student>
</StudentConfig>
```

练习：

写出以下XML文档

```
<Root url="http://baidu.com">
    <Title>百度首页</Title>
    <Body title="百科" />
    <Body title="新闻" />
    <Body title="搜索" />
    <End>关于百度</End>
</Root>
```

写出Windows的XML文档

xsl-mappings.xml

Client.xml