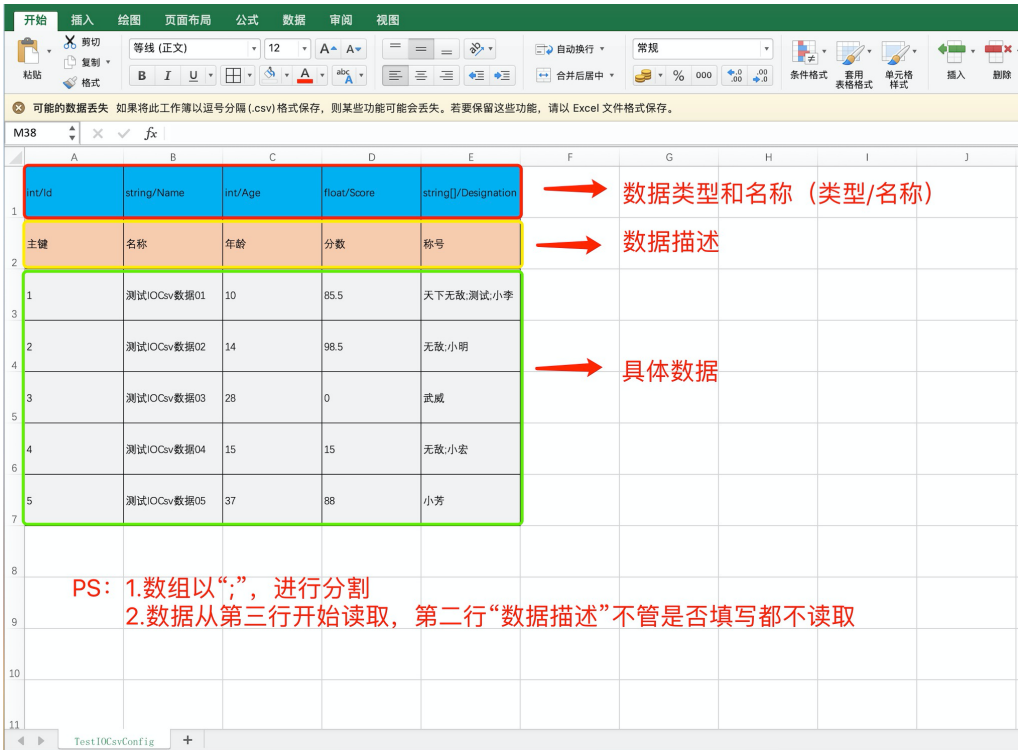
配置表管理工具（congig）

(MxFramework5.1)

目录

一、 介绍

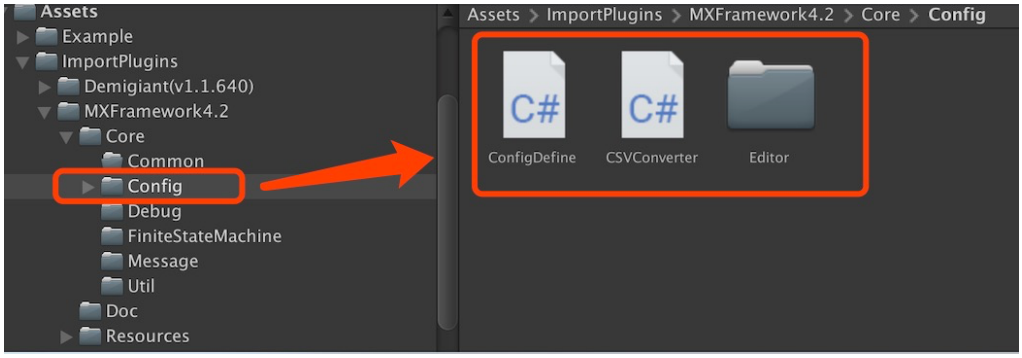
MxConfig 是一个自动化管理配置表工具。是 MxFramework 框架的一个子模块。常见的配置表格式有 Json 格式、Xml 格式、CSV 格式等，这些格式手动书写会存在 容易出错、不易扩展、工作量大、难以维护、等一系列问题。而 MxConfig 工具，就是为了解决这些问题而存在的，我们可以通过 Excel 来书写我们的配置表，不需要写任何脚本，一键完成配置表的解析和加载等工作，配置表最终会转换成加密格式，通过模板自定义解析脚本，将解析、加载和扩展 等一系列重复性高的工作交给工具自动生成，大量节省了我们的开发时间。



二、 项目结构

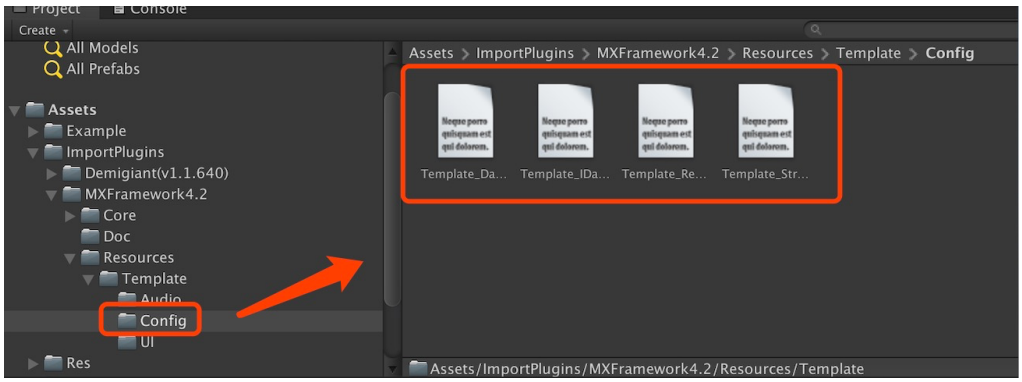
1. 源码存放路径

ImportPlugins/MXFramework**/Core/Config



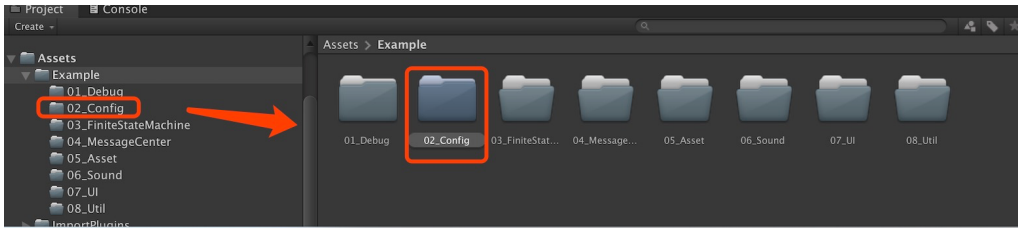
2. 模板存放路径

ImportPlugins/MXFramework**/Resources/Template/Config



3. 示例工程存放路径

Example/02_Config



4. 配置表原始文件存放路径

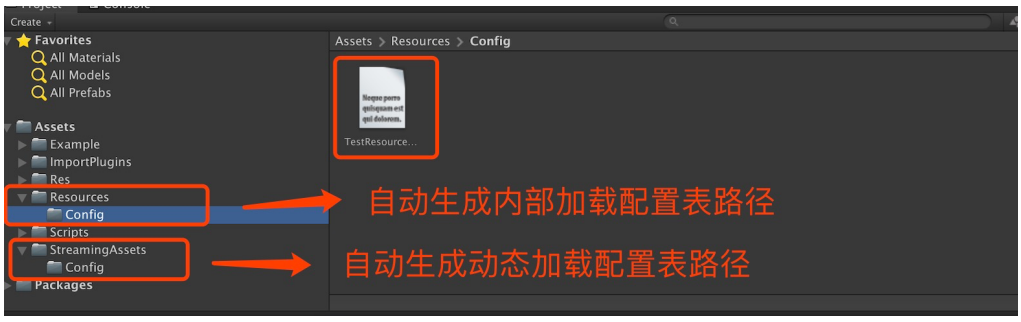
Res/Csv/IOCsv(动态加载方式的配置表)

Res/Csv/ResourcesCsv（内部加载方式的配置表）

5. 自动生成配置表存放路径

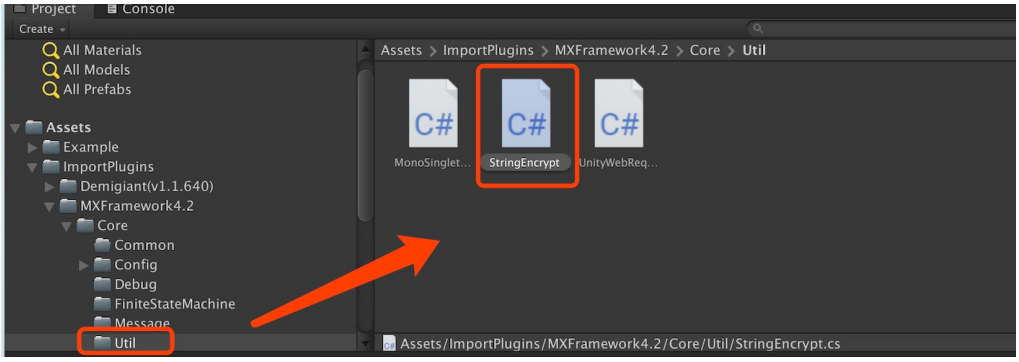
StreamingAssets/Config/配置表名称（自动生成动态加载方式的配置表数据)

Resources /Config/配置表名称（自动生成动内部加载方式的配置表数据）



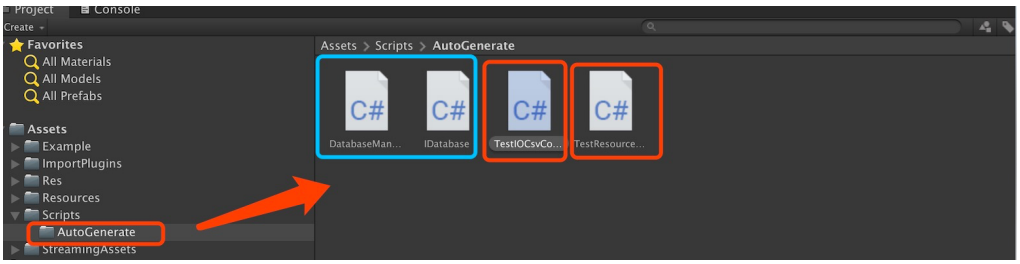
6. 加密算法存放路径

ImportPlugins/MXFramework**/Core/Util/StringEncrypt.cs



7. 自动生成的解析和加载代码路径

Scripts /AutoGenerate/配置表名称.cs



三、 加载方式

1. 动态加载规则

所有存放在 Res/Csv/IOCsv/配置表名称.csv 路径下的文件都是需要

动态加载，一般放在这个目录下的配置表是用来做热更新的，自动生成的数据文件存在 StreamingAssets/Config/配置表名称.txt 路径下。（通过以下代码进行加载）

```
StreamReader streamReader = null;
if (File.Exists(DataPath())) streamReader = File.OpenText(DataPath());
```

StreamingAssets 目录在 Android 设备上无法进行 io 操作，所以我们需要将数据拷贝到 persistentDataPath 目录下（详情请参考 Example/02_Config/TestIOCsvConfig.cs）

2. 内部加载规则

所有存放在 Res/Csv/ResourcesCsv/配置表名称.csv 路径下的文件都是内部加载，一般是存放不需要变更的配置文件，自动生成的数据文件存在 Resources/Config/配置表名称.txt 路径下。（如下）

```
TextAsset textAsset = Resources.Load<TextAsset>(DataPath());
```

（详情请参考 Example/02_Config/TestResourcesCsvConfig.cs）

四、 加密算法

1．异或加密算法

异或是对两个运算元的一种逻辑分析类型，符号为 XOR 或 EOR。与一般的逻辑或 OR 不同，当两两数值相同为否，而数值不同时为真。异或密码（simple XOR cipher）是密码学中一种简单的加密算法，是指对信息进行异或操作来达到加密和解密目的。按这种逻辑，文本串行的每个字符可以通过与给定的密钥进行按位异或运算来加密。如果要解密，只需要将加密后的结果与密钥再次进行按位异或运算即可。

2．加密

public static string EncryptDES(string encryptString, string key = "45131929"){}

Key 个数是 8 位默认是 “45131929”

3．解密

public static string DecryptDES(string decryptString, string key = "45131929"){}

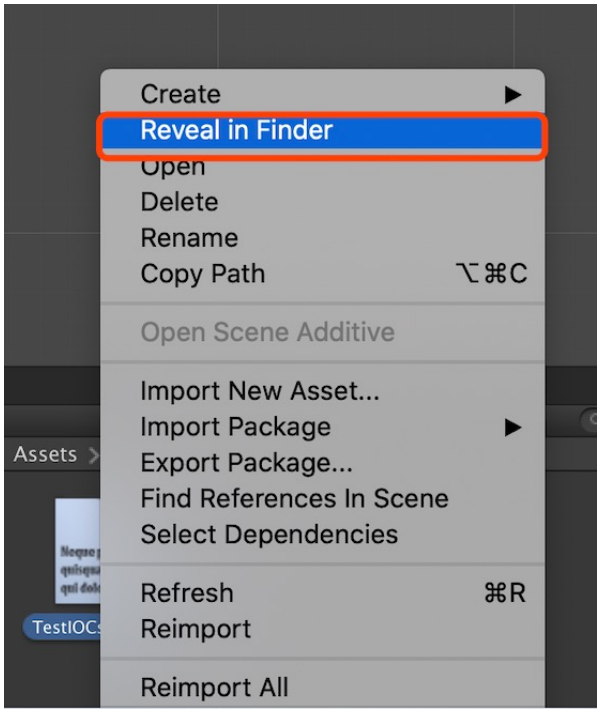
Key 个数是 8 位默认是 “45131929”

五、 如何使用

1. 配置表格式（以 TestIOCsvConfig.csv 为例）【修改】

2. 书写配置表（以 TestIOCsvConfig.csv 为例）

首先我们复制 Res/Csv/IOCsv/ 目录下的模板文件，命名为 “TestIOCsvConfig.csv”，然后右键 Reveal in Finder



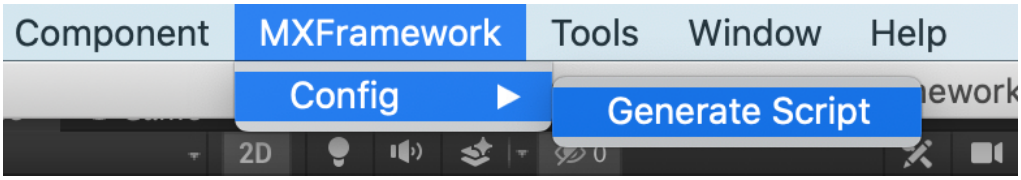
双击 “TestIOCsvConfig.csv” 文件，以 Excel 软件打开文件，第一行书写的是 数据类型/名称，第二行以后的是具体数据（数组以分号间隔）书写完成 Control+S 保存文件，

A	B	C	D	E
int/Id	string/Name	int/Age	float/Score	string[]/Designation
主键	名称	年龄	分数	称号
1	测试ResourcesCsv数据01	10	85.5	天下无敌;测试;小李
2	测试ResourcesCsv数据02	14	98.5	无敌;小明
3	测试ResourcesCsv数据03	28	0	武威
4	测试ResourcesCsv数据04	15	15	无敌;小宏
5	测试ResourcesCsv数据05	37	88	小芳

或者另存为 CSV UTF-8（逗号分隔）（.CSV）

3. 自动生成配置表和解析代码

点击 MXFramework/Config/Generate Script 按钮



（生成的配置表路径和代码路径请参考 二、项目结构）

六、 自动生成的代码 API

1. 数据类型（以 TestIOCsvConfig.csv 为例）

```
[Serializable]
public class TestIOCsvConfigData
{
    public int Id;
    public string Name;
    public int Age;
    public float Score;
    public string[] Designation;
}
```

2. 解析脚本（以 TestIOCsvConfig.csv 为例）

public uint TypeID()//获取类型 ID

public string DataPath()//获取数存放路径

public void Load()//加载数据

public TestIOCsvConfigData GetDataByKey(string key)//通过主键获取数据

public string GetJsonStringBykey(string key)// 通过主键获取 Json 类型数据【新增】

public string GetAllJsonString()//获取所有 Json 数据【新增】

public int GetCount()//获取数据数量

public List <TestIOCsvConfigData> GetAllData()//获取所有的数据

public TestIOCsvConfigData[] GetAllDataArray()//获取所有的数据

【新增】

七、 加载和解析配置表

1. 加载配置表

private TestIOCsvConfigDatabase m_Database;

m_Database = new TestIOCsvConfigDatabase();

m_Database.Load();

2. 解析配置表（以 TestIOCsvConfig.csv 为例）

```
for (int i = 0; i < m_Database.GetAllData().Count; i++)
{
    TestIOCsvConfigData data = m_Database.GetAllData()[i];

    GameObject item = Instantiate(m_Prefab, m_Parent);
    item.SetActive(true);
    item.name = data.Id.ToString();

    item.transform.Find("Id").GetComponent<Text>().text = data.Id.ToString();
    item.transform.Find("Name").GetComponent<Text>().text = data.Name;
    item.transform.Find("Age").GetComponent<Text>().text = data.Age.ToString();
    item.transform.Find("Score").GetComponent<Text>().text = data.Score.ToString();

    Text Designation = item.transform.Find("Designation").GetComponent<Text>();
    Designation.text = null;
    for (int j = 0; j < data.Designation.Length; j++)
    {
        Designation.text += data.Designation[j] + ";";
    }

}
```

八、 扩展

1. 模块扩展

自动生成的代码是通过模板作为样式生成(请产考“二、项目结构”),我们可以更改模板后重新生成代码进行扩展(模块扩展方式是对所有配置表 API 都进行扩展)

```
public List <$DataClassName> GetAllDataList()
{
    return listData;
}

public $DataClassName[] GetAllDataArray()
{
    return listData.ToArray();
}

//我是后面扩展的API
public void PrintCount()
{
  print(listData.Count);
}


}
```

2. 代码扩展

有些时候配置表的 API 不够用，想针对特定的配置表进行扩展，这样修改模板的方式就行不通了,直接修改特定配置表 API 代码是不行的,因为下次自动生成代码会将其覆盖掉,所以自动生成的代码都是部分类,我只有新建一个名称一样的部分类，进行扩展就可以了。(以 TestIOCsvConfig.csv 为例)

```
using UnityEngine;

//记得命名空间保持一致
namespace Mx.Config
{
    /// <summary>我是新建的部分类</summary>
    public partial class TestIOCsvConfigDatabase
    {
        //我是扩展的API
        public void PrintCount()
        {
            Debug.Log(listData.Count);
        }
    }
}
```

九、 源码下载

MxFramework 源码: https://github.com/yongliangchen/MXFramework

博客文章：https://blog.csdn.net/a451319296/article/details/109038323

QQ 交流群:1079467433