# JSon

### 特点

1. JSON（JavaScript Object Notation）JS对象标记，一种轻量级数据交换格式
2. 完全独立于编程语言的文本格式存储和表示数据
3. 层次结构清晰，简单，易读，有序，可追溯
4. 使用者必须遵循W3C制定的JS规范
5. 易于解析，能够有效提高网络传输效率

### JSON结构

1. “名称/值”对的集合（A collection of name/value pairs）。不同的语言中，它被理解为对象（object），纪录（record），结构（struct），字典（dictionary），哈希表（hash table），有键列表（keyed list），或者关联数组 （associative array）
2. 值的有序列表（An ordered list of values）。在大部分语言中，它被理解为数组（array）

### 例子

名称/值

1. {
2. "二狗": {
3. "gender": "male",
4. "age": "18"
5. },
6. "冬凤": {
7. "gender": "female",
8. "age": "32"
9. }
10. }

有序列表

1. {
2. "persons": [
3. {
4. "name": "二狗",
5. "gender": "male",
6. "age": "18"
7. },
8. {
9. "name": "冬凤",
10. "gender": "female",
11. "age": "32"
12. }
13. ]
14. }

### 书写要点

1. KV结构必须遵循K：V外观，K后跟冒号，结尾处需要用逗号分割，
2. 有序列表（数组），起始以K：[ ]开始，在其内部只能填充对象级节点 ｛｝
3. 每一个对象级节点下｛｝只能填充KV模式
4. V结构可以是有序列表（数组）[]，也可以是实际数据，或是对象级节点｛｝

### 序列化/反序列化

序列化 (Serialization)将对象的状态信息转换为可以存储或传输的形式的过程。

反序列化，通过从存储区中读取或内存数据中，重新创建对象

### UE中操作JSON

UE中操作JSON需要引入Json模块，在模块类中加入JSON模块

### 操作注意事项

在UE中操作JSon必须要知道读取数据的结构，才可以进行读取操作，写出时也需要注意写出数据格式，JSon是有序数据列，必须要保证数据操作过程中顺序的有效性，否则读取将会失败

TJsonReaderFactory 工厂类用来加载解析Json数据

TJsonWriterFactory 工厂类用来写数据

TJsonReader 解析成功后用来存储JSON数据

FJsonValue 数据对象条目

操作流程

·TJsonReaderFactory进行解析

·TJsonReader接受解析数据

·FJsonValue进行数据读取

·FJsonObject操作数据对象

·操作结束必须关闭数据流

### 读取操作

步骤：

1. 获取Json字符串（注意操作编码问题）
2. 使用字符串，构建TJsonReader，借助TJsonReaderFactory工厂方法进行构建，注意使用共享引用进行创建
3. 借助FJsonSerializer解析器将TJsonReader内的数据序列化为对象FJsonObject
4. 使用键值进行读取操作

###### 名称：值读取

FString data = "{\"name\":\"hehehe\"}";

TSharedRef<TJsonReader<TCHAR>> reader = TJsonReaderFactory<TCHAR>::Create(data);

TSharedPtr<FJsonObject> JsonParse;

if (FJsonSerializer::Deserialize(reader, JsonParse))

{

FString name = JsonParse->GetStringField("name");

UE\_LOG(LogTemp, Log, TEXT("%s"), \*name);

}

注意，对于结构操作，如果名称下是对象，读取时使用GetObjectField进行读取，返回TSharePtr<FJsonObject>，如果名称下是数组，读取使用GetArrayField，返回数据是TArray<TSharePtr< FJsonValue > >

###### 有序列表

有序列表操作和名称值相同，在解析操作时需要构建的解析数据为TArray<TSharePtr< FJsonValue > >类型

//FString data = "{\"name\":\"中文\"}";

FString data2 = L"[{\"name\":\"中文\"}]";

TSharedRef<TJsonReader<TCHAR>> reader = TJsonReaderFactory<TCHAR>::Create(data2);

//TSharedPtr<FJsonObject> JsonParse;

TArray<TSharedPtr<FJsonValue>> JsonParse;

if (FJsonSerializer::Deserialize(reader, JsonParse))

{

FString name = JsonParse[0]->AsObject()->GetStringField("name");

UE\_LOG(LogTemp, Log, TEXT("的%s"), \*name);

}

**中文输出切记使用宏TEXT构建，否则系统将按照编辑器编码字符进行处理**

### 输出操作

步骤：

1. 构建对象基础节点TSharePtr<FJsonObject>对象
2. 向节点中写入数据，可以是基础数据，也可以是新的TSharePtr<FJsonObject>对象
3. 支持向节点中输入数组，TArray<TSharePtr<FJsonValue>>
4. 构建TJsonWriter对象，为进行输入操作准备
5. 借助FJsonSerializer，序列化数据到字符串操作

###### 名称值输出

TSharedPtr<FJsonObject> RootJson(new FJsonObject());

RootJson->SetStringField(TEXT("name"), TEXT("的"));

TArray<TSharedPtr<FJsonValue>> jarray;

TSharedPtr<FJsonObject> m\_data(new FJsonObject());

m\_data->SetBoolField(TEXT("bis"), true);

TSharedPtr<FJsonValueObject> j\_object(new FJsonValueObject(m\_data));

jarray.Add(j\_object);

RootJson->SetArrayField(TEXT("data"), jarray);

FString data;

TSharedRef<TJsonWriter<TCHAR>> writer = TJsonWriterFactory<TCHAR>::Create(&data);

if (FJsonSerializer::Serialize(RootJson.ToSharedRef(), writer))

{

UE\_LOG(LogTemp, Log, TEXT("%s"), \*data);

}

###### 有序列表

TArray<TSharedPtr<FJsonValue>> Jarray;

TSharedPtr<FJsonObject> obj(new FJsonObject());

obj->SetStringField(TEXT("name"), TEXT("OK"));

TSharedPtr<FJsonValueObject> data1(new FJsonValueObject(obj));

Jarray.Add(data1);

FString data;

TSharedRef<TJsonWriter<TCHAR>> Writer = TJsonWriterFactory<TCHAR>::Create(&data);

if (FJsonSerializer::Serialize(Jarray, Writer))

{

UE\_LOG(LogTemp, Log, TEXT("%s"), \*data);

}

**注意：构建Json数组对象时，顺序如下**

·TArray<TSharedPtr<FJsonValue>>

·TSharedPtr<FJsonObject>

·TSharedPtr<FJsonValueObject> 使用第二步的Object对象进行构建

·将FJsonValueObject加入到数组

### 从文件中读取操作

需要将内容读入内存，转化为文本方式

FString f\_path = FPaths::GameContentDir() + "/1.json";

if (FPaths::FileExists(f\_path))

{

FString data;

FFileHelper::LoadFileToString(data, \*f\_path);

UE\_LOG(LogTemp, Log, TEXT("%s"), \*data);

}

解析和上面的解析方式相同

### 写出到文件中

FString f\_path = FPaths::GameContentDir() + "/2.json";

FString data2 = L"[{\"name\":\"中文\"}]";

FFileHelper::SaveStringToFile(data2, \*f\_path);