RESTful API 언리얼에서 사용하기

Created	@January 3, 2022 6:59 PM
Tags	

- 본 문서는 REST API에 대해 알고 있다고 전제합니다.
- 본 문서는 Json에 대해 알고 있다고 전제합니다.

로컬 서버 세팅

- 로컬에서 테스트하려는 경우, 로컬에 서버를 세팅합니다.
  1. Docker 설치(https://www.docker.com/get-started)
     - Docker DeskTop을 설치합니다.
  2. CodeStrory Github에서 도커 인증서버를 다운받고, windows\start.bat을 실행합니다.
- 브라우저에서 테스트하고 싶은 경우에는 서버 개발자분에게 포스트맨 초대장을 보내달라고 하신 후, CodeStory Workspace의 코드스토리 인증서버 api에 들어가셔서 테스트해보시면 됩니다.

프로젝트 세팅

- 프로젝트의 프로젝트명.Build.cs를 열고 PublicDependencyModuleNames를 추가합니다.

```
PublicDependencyModuleNames.AddRange(new string[]
    {
      "Core", "CoreUObject", "Engine", "InputCore",

      // Json, JsonUtilities, Http를 추가합니다.
      "Json", "JsonUtilities", "Http"
    });
```

Json 포맷의 스트링의 생성과 파싱

- HttpRequest에 포함될 Content, 즉 Body에 Json 포맷의 스트링을 넣습니다.
- POST, PUT에 필수적으로 Content가 포함됩니다.

Json 포맷 스트링 생성

- FJsonObject
  - #include "Dom/JsonObject.h”
  - FJsonObject는 스트링을 Key값으로, TSharedPtr<TJsonValue>를 Value로 가집니다.
    - #include "Dom/JsonValue.h”
    - TJsonValue는 array, boolean, object, number, string으로 확장됨. 여기서 object는 FJsonObject를 뜻함. 다른 FJsonValue 자료형을 참고하고 싶으면 링크를 참조하십시오.
    - TJsonValue는 생성자는 protected에 있기 때문에 사용 불가능합니다.
    - 사용할 때는 공유포인터로 사용해야 함.
    - TJsonValue 사용 예시

```
// 사용할 때는 공유포인터로 사용하기로 약속되어 있습니다.
// FJsonValue는 확장된 한가지 형태로 생성해야 합니다.
// FJsonValue의 생성자가 protected에 숨겨져 있기 때문에 생성불가능.

TSharedPtr<FJsonValue> Value = MakeShareable(new FJsonValueString("Test String"));

FString ValueAsString = Value->AsString();   // ValueAsString = Test String

// FJsonValue는 실체가 Array, Boolean, Object, Number, String일 수도 있기 때문에
// 확인하고 사용하고 싶으면 TryGet~()을 사용하면 됩니다.

FString ValueAsString2;

if (Value->TryGetString(ValueAsString2))
{
  UE_LOG(LogTemp, Warning, TEXT("Value is String!"));

  // ValueAsString2 = Test String
  // Value is String이 로그창에 출력.
```

```
}

int32 ValueAsInt32;

if (Value->TryGetNumber(ValueAsInt32))
{
  UE_LOG(LogTemp, Warning, TEXT("Value is Int32!"));

  // Value is Int32는 로그창에 출력되지 않음.
}
```

- FJsonObject는 특정 필드의 값을 설정할 수 있습니다.
- 특정 필드의 값을 설정하는 방법은 필드의 값을 TSharedPtr<FJsonValue>로 설정하는 방법1이 있고, 필드의 값을 호출하는 함수의 종류에 따라 값의 자료형이 달라지는 방법2가 있습니다.
- 예시

```
TSharedRef<FJsonObject> JsonObject = MakeShareable(new FJsonObject);

// 방법 1
TSharedPtr<FJsonValue> EmailValue = MakeShareable(new FJsonValueString("test@gmail.com"));
JsonObject->SetField("email", EmailValue);
// JsonObject->SetStringField("email", "test@gmail.com")인 방법2로도 할 수 있습니다.

// 방법 2
TArray<TSharedPtr<FJsonValue>> ValueArr;
Arr.Emplace(new FJsonValueString("ArrTest1"));
Arr.Emplace(new FJsonValueString("ArrTest2"));
// Arr.Emplace(new FJsonValueNumber(3)); 이런 이상한 짓은 하짓 말길 바랍니다.
// 배열의 자료형을 하나로 하지 않으면 정의되지 않은 오류가 발생할 수 있습니다.

JsonObject->SetArrayField("TestStrings", ValueArr);
```

- FJsonObject를 이용하여 Json 포맷 스트링을 만들 TJsonWriter 공유 레퍼런스를 생성합니다.
  - TJsonWriter
    - #include "Serialization/JsonWriter.h”

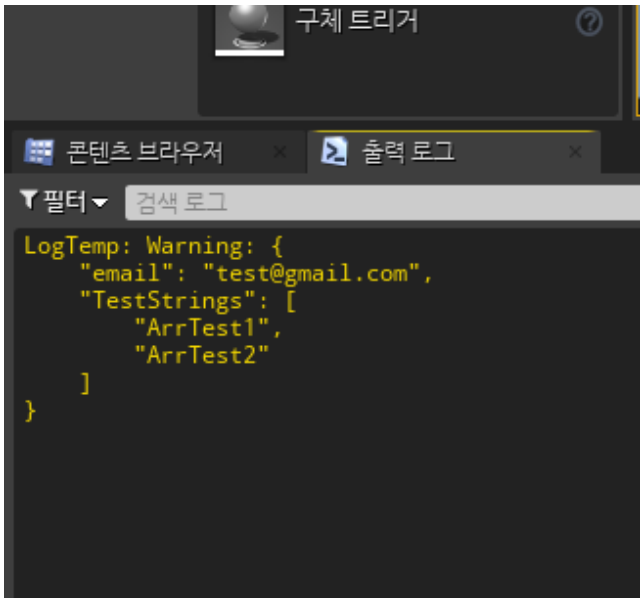
        - 템플릿 클래스입니다.
        - TJsonStringWriter를 TJsonWriter<class CharType, class PrintPolicy>가 상속합니다.
        - <class CharType = TCHAR, class PrintPolicy = TPrettyJsonPrintPolicy<CharType>>으로 기본 템플릿 인수가 설정되어 있습니다.
    - TJsonWriterFactory
        - #include "Serialization/JsonWriter.h”
        - 템플릿 클래스입니다.
        - TJsonWriter와 같은 템플릿 인수 목록과 기본 템플릿 인수가 설정되어 있습니다.

```
// 최종 Json 포맷의 스트링을 선언하고 이 스트링을 쓰는 TSharedRef<TJsonWriter<>>를 선언

FString ResultJsonStr;

TSharedRef<TJsonWriter<>> Writer = TJsonWriterFactory<>::Create(&ResultJsonStr);
FJsonSerializer::Serialize(JsonObject, Writer);
// JsonObject는 Key, Value가 Set되어 있다고 가정합니다.

// ResultJsonStr의 결과는 다음과 같습니다.
```



- 위 설명 코드의 전체 코드를 첨부합니다.

```
TSharedRef<FJsonObject> JsonObject = MakeShareable(new FJsonObject);

TSharedPtr<FJsonValue> EmailValue = MakeShareable(new FJsonValueString("test@gmail.com"));
JsonObject->SetField("email", EmailValue);

TArray<TSharedPtr<FJsonValue>> ValueArr;
ValueArr.Emplace(new FJsonValueString("ArrTest1"));
ValueArr.Emplace(new FJsonValueString("ArrTest2"));

JsonObject->SetArrayField("TestStrings", ValueArr);

FString ResultJsonStr;

TSharedRef<TJsonWriter<>> Writer = TJsonWriterFactory<>::Create(&ResultJsonStr);
if (FJsonSerializer::Serialize(JsonObject, Writer))
{
    UE_LOG(LogTemp, Warning, TEXT("\r\n%s"), *ResultJsonStr);
}
// FJsonSerializer::Serialize 함수가 JsonObject를 Json 포맷 스트링으로 직렬화합니다.
```

HttpRequest 선언 및 초기화, 그리고 요청

- `TSharedRef<IHttpRequest, ESPMode::ThreadSafe>` 자료형을 선언 및 할당
  - #include "HttpModule.h”
  - FHttpModule은 싱글턴입니다.
  - CreateRequest를 하면 HttpRequest를 생성, 반환합니다.

```
TSharedRef<IHttpRequest, ESPMode::ThreadSafe> Request = FHttpModule::Get().CreateRequest();
```

- HttpRequest를 초기화합니다.

```
// 요청의 URL를 설정합니다.
Request->SetURL("http://localhost:8080/api/v1/authentication/user");

// 요청의 행위를 설정합니다.
Request->SetVerb("POST");

// 요청의 헤더를 설정합니다.
```

```
Request->SetHeader("User-Agent", "X-UnrealEngine-Agent");
Request->SetHeader("Content-Type", "application/json");

// 요청의 콘텐트(바디)를 설정합니다. Body는 Json 포맷의 스트링입니다.
// 위 Json 포맷 스트링 생성 예시에서의 ResultJsonStr입니다.
Request->SetContentAsString(Body);

// 요청을 마치게 된 후 호출될 함수를 Bind해줍니다.
// OnHttpRequestComplete()에 대해선 추후 설명하겠습니다.
Request->OnProcessRequestComplete().BindUObject(this, &ThisClass::OnHttpRequestComplete);
```

- 초기화를 마쳤으면 요청을 진행합니다.

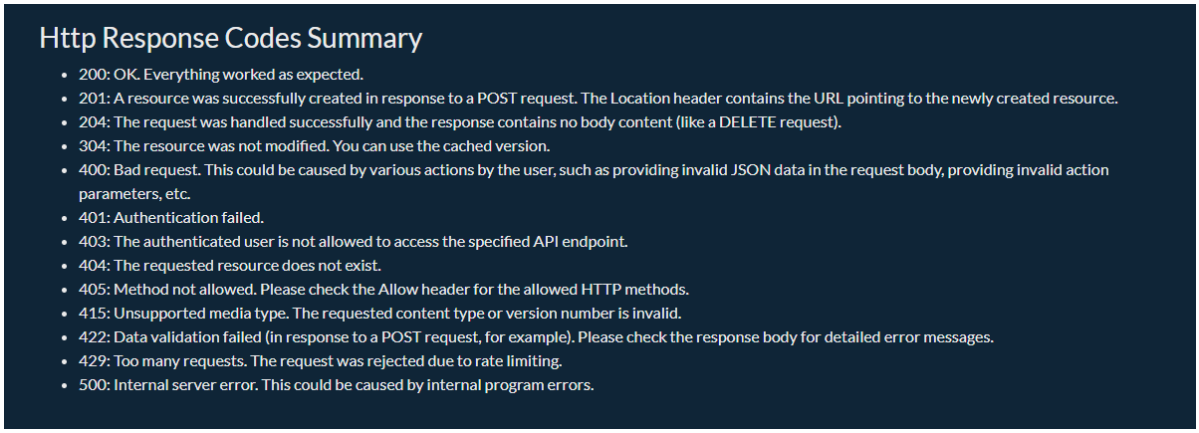
```
Request->ProcessRequest();
```

요청에 대한 반응으로 받은 Json 포맷 스트링 파싱

- 위 예시에서의 HttpRequest가 끝나게 되면 OnProcessRequestComplete()에 Bind 해두었던 함수가 호출됩니다. 위의 예시에선 OnHttpRequestComplete()
- 바인드해줄 함수는 다음과 같은 시그니처를 지닙니다.
  - `void FunctionName(FHttpRequestPtr, FHttpResponsePtr, bool)`
  - FHttpRequestPtr, FHttpResponsePtr은 IHttpRequest.h에 typedef 되어 있습니다.
  - FHttpRequestPtr은 우리가 위에서 생성해두었던 HttpRequest로, 요청시 정보가 담겨 있습니다.
  - FHttpResponsePtr은 요청에 따른 서버의 DTO(Data Transfer Object)로 ResponseCode, Content외 정보가 있습니다.
  - FHttpResponsePtr::GetResponseCode()로 반응코드를 읽을 수 있습니다.

```
void AHttpTest::OnHttpRequestComplete(FHttpRequestPtr Request, FHttpResponsePtr Response,
                                      bool bConnectedSuccessfully)
{
  if (Response->GetResponseCode() == 200)
  {
    // 보낸 요청이 성공했을 시 들어옵니다...
  }
}
```

- 다양한 ResponseCode가 있는데 다음과 같습니다.



출처 : gorest.co.in

- FHttpResponsePtr이 가지고 있는 Content에는 Json 포맷의 스트링이 있을 수 있는데 이를 파싱해야 합니다.
- TJsonReader
  - 템플릿 클래스로 기본 템플릿 인수가 <class CharType = TCHAR>입니다.
- TJsonReaderFactory
  - 템플릿 클래스로 템플릿 인수 목록과 기본 템플릿 인수가 TJsonReader와 같습니다.

```
// 요청에 대해 올바른 반응으로 보낸 Json 스트링과 같은 Json 스트링을 받는다고 가정합니다.

const FString ResponseBody = Response->GetContentAsString();

TSharedRef<TJsonReader<>> Reader = TJsonReaderFactory<>::Create(ResponseBody);
// TJsonReaderFactory를 통해 TJsonReader를 생성합니다.
// 생성하는 방식은 정해져 있고, Json 파일 스트림을 이용해서 생성하고 싶다면
// Create 함수안에 해당 스트림을 넣어주면 됩니다. Writer도 동일.

TSharedPtr<FJsonObject> JsonObject; // Json 스트링을 생성때와는 달리 TSharedPtr로 생성

FString Email;
TArray<FString> TestStrings;

if (FJsonSerializer::Deserialize(Reader, JsonObject))
{
  Email = JsonObject->GetStringField("email");
```

```
  TArray<TSharedPtr<FJsonValue>> ValueArr = JsonObject->GetArrayField("TestStrings");
  TestStrings[0] = ValueArr[0]->AsString();
  TsetStrings[1] = ValueArr[1]->AsString();
}

// 결과
// Email = test@gmail.com
// TestStrings[0] = ArrString1, TestStrings[1] = ArrString2
```

주의 사항

- 배열은 보통 반복문을 사용하기 때문에 같은 형만을 넣습니다. 다른 자료형 섞어넣는 이상한 짓 금지.