УДК 004

Выполнение веб-запросов к удалённому серверу в приложении, созданном в Unity3D

Фатеенков Данила Витальевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В статье описываются методы, реализующие в приложении, созданном в Unity3D, получение и отправку данных на сервер. Приводится описание работы двух основных классов, предоставляющие доступ к веб-ресурсам: классы WWW и UnityWebRequest. Описываются основные методы и функции классов, а также описаны сопрограммы, отправляющие и получающие данные с сервера.
Ключевые слова: Unity3D, UnityWebRequest, C#, клиент-серверная архитектура

Making web requests to a remote server in an application created in Unity3D

Fateenkov Danila Vialievich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article describes the methods that implement receiving and sending data to the server in the application created in Unity3D. It describes how the two main classes that provide access to web resources work: the WWW and UnityWebRequest classes. The main methods and functions of the classes are described, and the coroutines that send and receive data from the server are described.
Keywords: Unity3D, UnityWebRequest, C#, client-server architecture

1 Введение

1.1 Актуальность

В сфере информационных систем значительную роль занимают развлекательные технологии. К таким технологиям также относятся компьютерные игры, которые на протяжении многих лет развиваются и предлагают пользователю всё больше возможностей. Приоритетным направлением в развитии компьютерных игр остаются игры, реализующие сетевое взаимодействие между игроками.

В настоящее время средств разработки компьютерных игр на рынке программного обеспечения много и развивающиеся программы уже имеют функционал для работы с серверной стороной приложения. Например,

средство разработки компьютерных игр Unity3Dсодержит функционал для реализации клиент-серверной архитектуры.

1.2 Обзор исследований

Иванцов В.В., Зиганшин А.Т., Катышева Т.М., Хаустов П.А. выделили главные особенности разработки многопользовательских клиент-серверных приложений на Unity3D [1]. Первун О.Е., Халилов Э.Р. описали процесс создания серверной части для игры, разработанной на Unity3D, визуально представив описание необходимых для корректной работоспособности классов [2]. Петров А.А., Федотов Е.А. рассмотрели средства для реализации сетевых игр, а также основные архитектуры сетевых приложений [3]. Гиричев А.А. провёл анализ и описал процесс оптимизации серверной части приложения, разработанного на Unity3D [4].

1.3 Цель исследования

Цель – реализовать клиент-серверное приложение в Unity3D. Приложение способно отправлять данные на сервер и получать их по запросу со стороны клиента.

2 Материалы и методы

Для реализации применяется язык программирования C# и средство разработки игровых приложений Unity3D.

3 Результаты и обсуждения

API Unity в настоящее время поддерживает 2 способа передачи информации на сервер (и получение данных соответственно): использование класса WWW, который предоставляет доступ к веб-ресурсам или применение модуля Networking из API Unity. Первый способ считается устаревшим, но функционал класса не отключен. На смену классу WWW пришёл UnityWebRequest, который расширяет возможности работы с серверной стороной приложения. UnityWebRequest поддерживает отправку таких запросов, как GET, POST, PUT, DELETE и другие, а класс WWW поддерживает только отправку данных на сервер POST-методом, а также получение данных с сервера.

Класс WWW реализуется через создание формы, в которую будут помещены все необходимые для отправки на сервер данные. Форма является объектом и создаётся вызовом класса WWWForm:

```
WWWForm form = new WWWForm();
```

После создания объекта WWW Form, формируются данные для отправки. Данные необходимо отправлять, заранее задавая им определённый ключ. Это необходимо для дальнейшей работы: данные будут отправлены в виде массива и на сервере обработка происходит с использованием PHP-сценариев. Ключи данным в форме задаются, чтобы на сервере была

возможность ссылаться на конкретный элемент массива по предоставленному ключу.

Для добавления данных в форму используется функция AddField:

```
form.AddField(*ключ*, *данные*);
```

После заполнения формы данными, создаётся объект класса WWW, который принимает 2 параметра: ссылку на сервер (ссылаться необходимо на PHP-сценарий, в котором происходит обработка данных) и форма с данными. Отправленный массив обрабатывается на сервере и возвращает клиенту определённое значение (в зависимости от сценария, в котором происходит обработка). Результат, полученный с сервера, представлен как объект класса WWW, текст из которого можно получить методом text. В случае возникновения ошибок необходимо прервать выполнение функции. Чтобы определить наличие ошибки, используется метод error.

Функция, реализующая получение данных с сервера, выглядит следующим образом:

```
private IEnumerator GetData() {
     WWW www = new WWW
("http://gameserver.com/data/getdata.php");
     yield return www;
     if (www.error != null) {
          Debug.Log("Error: " + www.error);
          yield break;
          }
     PlayerPrefs.SetString("DataMain", www.text);
}
```

Объект класса WWW может не получать данные перед отправкой, что позволит отправить GET-запрос на сервер и получить необходимые данные из PHP-сценария “getdata.php” и записать их в реестр ОС, используя метод text.

Для отправки данных на сервер нужно создать форму и добавить её как параметр в объект WWW:

```
private IEnumerator Get1() {
     WWWForm form = new WWWForm();
     form.AddField("id", 1);
     form.AddField("name", "Someone");
     form.AddField("exp", 154);
     form.AddField("played_minutes", 11);
     WWW www = new WWW
("http://gameserver.com/data/writedata.php ", form);
     yield return www;
     if (www.error != null) {
          yield break;
     }
     Debug.Log("Data saved!");
}
```

Данные будут отправлены на сервер и PHP-сценарий, если не возникнет проблем с интернет-соединением, сможет обработать полученную информацию (например, создать новую запись в базе данных).

Описанный выше способ не актуален в новых версиях, поэтому рекомендуется использовать UnityWebRequest. Данный класс предоставляет более широкий функционал для отправки и обработки данных и основными функциями класса являются:

1. UnityWebRequest.Get()
2. UnityWebRequest.Post()
3. UnityWebRequest.Put()
4. UnityWebRequest.Delete()

Все перечисленные функции формируют HTTP-запросы. GET-запрос принимает в качестве параметра только ссылку на сервер. Сформированный GET-запрос посылается на сервер функцией SendWebRequest:

```
     UnityWebRequest www =
UnityWebRequest.Get("http://gameserver.com/data/getdata.php ");
     yield return www.SendWebRequest();
```

Для проверки наличия ошибок при отправке данных также можно использовать метод error, но в библиотеке Networking также есть 2 метода: isNetworkError и isHttpError. Оба метода предназначены для определения ошибок при работе с сервером и возвращают “true” или “false”.

В отличие от класса WWW, в UnityWebRequest необходимо вызвать метод downloadHandler, предназначенный для получения данных с сервера. downloadHandler является методом объекта DownloadHandler, который нужен для получения и обработки данных с сервера. Таким образом, функция для формирования GET-запроса через UnityWebRequest описывается на C# как сопрограмма, которую можно вызвать при запуске скрипта:

```
IEnumerator GetData() {
     UnityWebRequest www = UnityWebRequest.Get(getDataURL);
     yield return www.SendWebRequest();
     if (www.isHttpError || www.isNetworkError) {
          Debug.Log("Something went wrong!");
          yield break;
     }

     Debug.Log(www.downloadHandler.text);
}
```

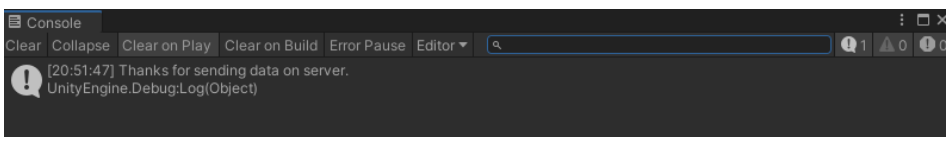

Рисунок 1. Ответ, отправленный со стороны сервера при выполнении GET-запроса

Для POST-запроса также нужно сформировать массив отправляемых данных перед выполнением запроса. Сопрограмма на C# реализуется следующим образом:

```
IEnumerator WriteData() {
     WWWForm form = new WWW Form();
     form.AddField("id", 1);
     form.AddField("name", "Player1");
     form.AddField("exp", 15);
     form.AddField("played_minutes", 20);
     UnityWebRequest www = UnityWebRequest.Post(postDataURL,
     form);
     yield return www.SendWebRequest();
     if (www.isHttpError || www.isNetworkError) {
          Debug.Log("Something went wrong!");
          yield break;
     }
     Debug.Log(www.downloadHandler.text);
}
```

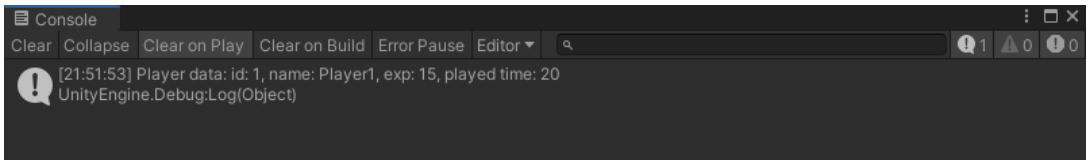

Рисунок 2. Ответ со стороны сервера, полученный после отправки данных POST-запросом

Также классы WWWи UnityWebRequest поддерживают работу с медиа-данными – можно скачивать изображения и аудио-файлы для дальнейшего использования в проекте. Это полезно при разработке мобильных приложений: компилировать все файлы в apk пакет не всегда рационально, так как приложение на выходе может иметь большой размер, что затруднит скачивание. Для решения такой проблемы, файлы загружаются на удалённый сервер и при первом запуске приложения скачиваются на устройство пользователя.

Примером применения такого варианта скачивания файла служит загрузка текстур для игры. Например, текстуры для материала, который используется для отображения неба на уровне игры. Шейдеров для работы с небом и его материалом на игровом пространстве в Unity есть 4:

1. Процедурно-генерируемые – не используют текстуры, а генерируют материал исходя из заданных параметров.

2. 6 Sided – шейдер, делящий материал на 6 текстур, которые при свёртке образуют куб. Необходимо учитывать, что текстуры должны дополнять друг друга, чтобы не возникало стыков или визуальных ошибок.

3. Cubemap – шейдер, создающий небо из объекта Cubemap. Cubemap – объект, являющийся набором также из 6 текстур, но которые также представляют отражения в окружении (отличие от шейдера 6 Sided).

4. Panoramic – шейдер, использующий сферу для генерации неба (текстура используется в шейдере одна).

Для реализации загрузки файлов с сервера используется шейдер 6 Sided. В первую очередь создаются 2 массива: параметры шейдера для работы с текстурами и названия изображений, которые будут скачиваться с сервера:

```
public string[] skyboxTextureNames;
public string[] textureImageNames;
```

Для заполнения массива с параметрами шейдера нужно открыть настройки шейдера и найти необходимые названия:

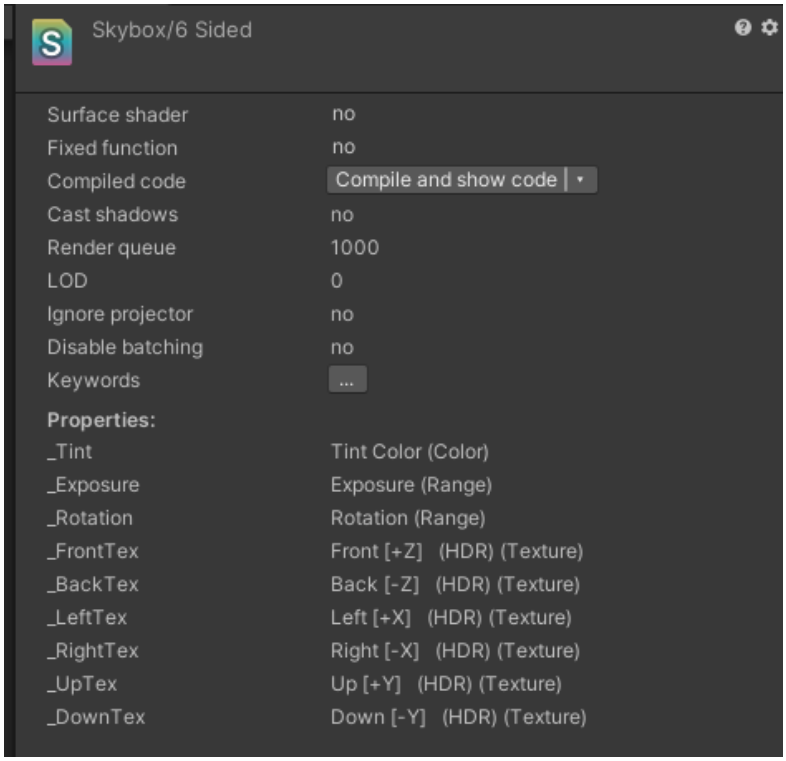

Рисунок 3. Параметры шейдера 6 Sided

Параметры, отвечающие за текстуры материала: _FrontTex, _BackTex, _LeftTex и другие. Всего элементов 6. Заполненные массивы могут выглядеть следующим образом:

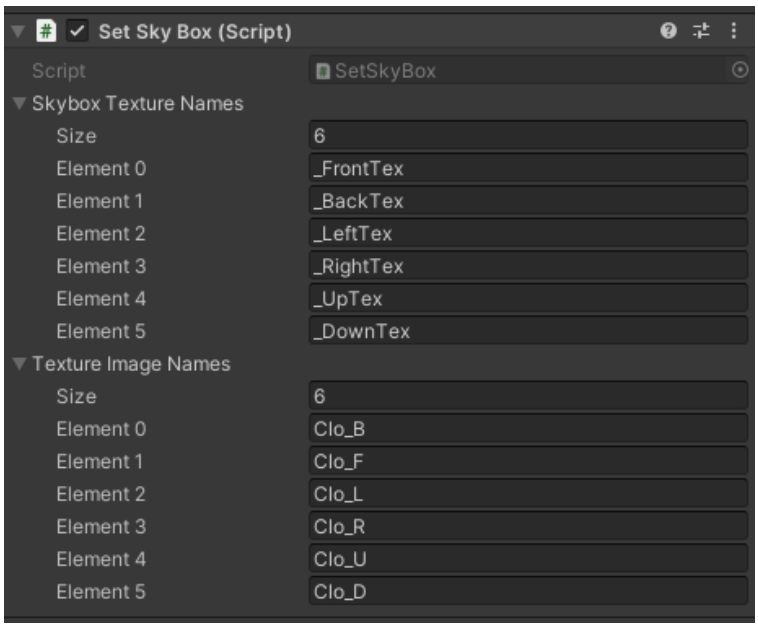

Рисунок 4. Массивы, заполненные названиями необходимых элементов

Для генерации ссылок для загрузки изображений с сервера создаётся цикл, в котором образуется ссылка на изображения. После создания ссылки запускается сопрограмма для загрузки файлов, параметрами которой являются ссылка на файл и индекс массива:

```
void CreateSkyBox(){
     for (int i = 0; i < 6; i++) {
          string getUrl =
     "http://gameserver.com/images/skybox/";
          getUrl += textureImageNames[i] + ".png";
          StartCoroutine(GetImages(getUrl, i));
     }
}
```

Сопрограмма для загрузки изображений получает текстуру по ссылке getUrl и устанавливает её в материал шейдера по индексу массива. Для доступа к материалу, который установлен для отображения неба, используется ссылка на skybox из объекта настроек отображения (RenderSettings). Для изменения текстуры необходимо применить метод SetTexture. Также при скачивании изображений используется обработчик DownloadHadlerTexture, а запрос на сервер GetTexture:

```
IEnumerator GetImages(string getUrl, int i){
     UnityWebRequest www =
UnityWebRequestTexture.GetTexture(getUrl);
     yield return www.SendWebRequest();

     if (www.isHttpError || www.isNetworkError){
          Debug.Log("Something went wrong!");
          Debug.Log(www.responseCode);
          yield break;
     }
```

```
        Texture skyBoxImage
    =((DownloadHandlerTexture)www.downloadHandler).texture;
        var mat = RenderSettings.skybox;
        mat.SetTexture(skyboxTextureNames[i], skyBoxImage);
    }
```

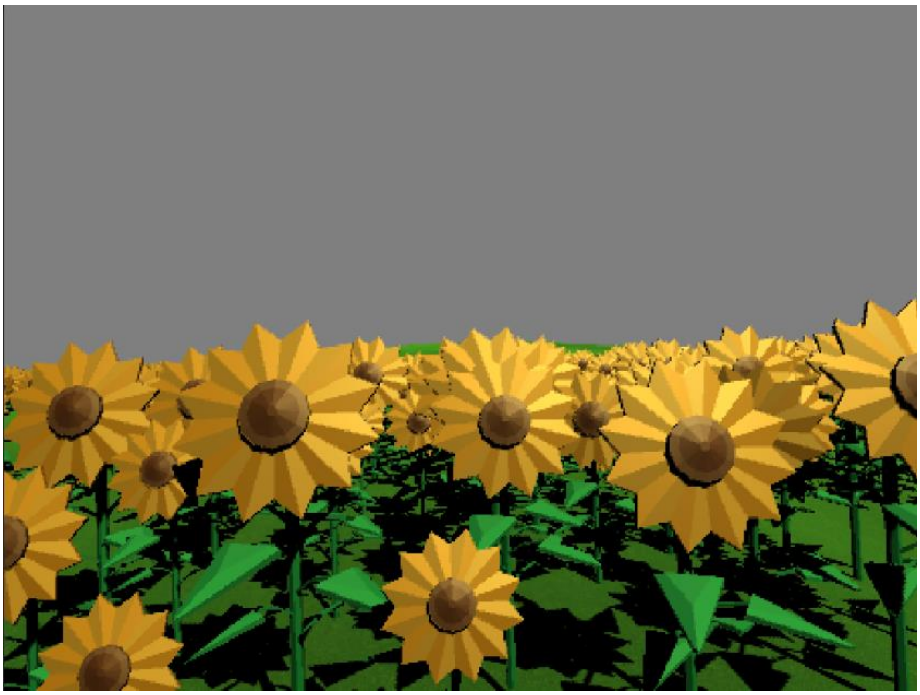

Рисунок 5. Игровое пространство до запуска сопрограммы GetImages

После запуска скрипта, текстуры начнут скачиваться (при отсутствии проблем с соединением) и устанавливаться на необходимый материал:

Рисунок 6. Игровое пространство после запуска GetImages

Были рассмотрены 2 класса для реализации клиент-серверной архитектуры в приложении, созданном на Unity: WWW и UnityWebRequest. WWW предоставляет базовый функционал (GET и POST запросы) и на данный момент считается устаревшим, что является значительным недостатком. UnityWebRequest расширяет возможности WWW и предлагает разработчику больше возможностей для реализации соединения в созданном приложении. В классе UnityWebRequest содержится больше функций для контроля соединения (такие как “isHttpError” или “isNetworkError”).

Для удобства работы с отправляемыми данными, в Unity существует класс WWWForm, который формирует массив данных с ключом, задаваемым пользователем.

В статье были рассмотрены реализации классов WWW и UnityWebRequest, а также описаны их особенности и недостатки.

Библиографический список

1. Иванцов В.В., Зиганшин А.Т., Катышева Т.М., Хаустов П.А. Особенности разработки многопользовательского клиент-серверного приложения на графическом движке Unity 2D // Молодёжь и современные информационные технологии. Томск: Национальный исследовательский Томский политехнический университет, 2014. С. 134-135.
2. Первун О.Е., Халилов Э.Р. Разработка серверной части игрового приложения для платформы Android с использованием Unity // Информационно-компьютерные технологии в экономике, образовании и социальной сфере. Крым: Крымский инженерно-педагогический университет, 2018. С. 26-33.
3. Петров А.А., Федотов Е.А. Разработка простых сетевых игр с использованием игрового движка Unity // XII международный молодёжный форум "Образование. Наука. Производство". Белгород: Белгородский государственный технологический университет им. В.Г. Шухова, 2020. – С. 1961-1968.
4. Гиричев А.А. Об оптимизации серверной части в Unity // Новая наука: проблемы и перспективы. Белгород: Белгородский государственный национальный исследовательский университет, 2020. С. 114-118.
5. Unity – Manual: Unity User Manual URL: https://docs.unity3d.com