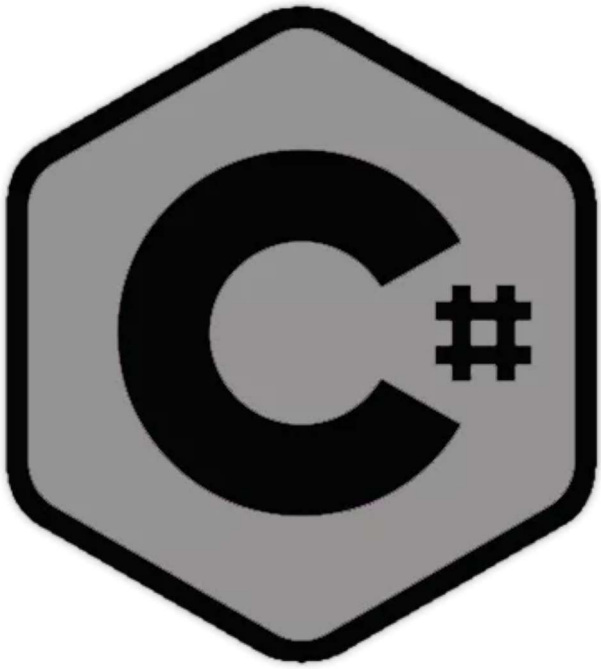
Сериализация-десериализация JSON в C#

Метки: api c# class JSON



Краткий экскурс

Фактически каждый программист рано (или поздно) сталкивается с сериализацией.

Сериализация — это процесс перевода структуры данных в последовательность битов, или же в другую структуру данных, которую удобно хранить, передавать.

Десериализация — это обратный процесс. Процесс преобразования сериализованных данных в структуру данных.

Подобные вещи часто встречаются при обмене данными между клиентом и сервером, между двумя клиентами (P2P). В общем много где встречается. Моё первое знакомство было при работе с API одного из российских сервисов.

API — это **A**pplication **P**rogramming **I**nterface. То есть прослойка между функциональной частью какого-либо программного обеспечения и вашего программного обеспечения.

Например — Вам хочется написать программу, которая будет отображать погоду в Москве. Вам не хочется устанавливать свои метеодатчики и другое оборудование.

Допустим, что есть сервис «Погода в России», который имеет свой **API**. И данный сервис на бесплатно (или же платно) дает доступ к нему вашим приложениям. Конечно — всегда можно написать парсер нужной страницы и выводить информацию. Но любые изменения в содержимом страницы могут привести к тому, что парсер придется тоже обновлять.

Глупо говорить о преимуществах API по сравнению с парсингом. Это скорость, удобство работы, а также, очень часто, более расширенный функционал.

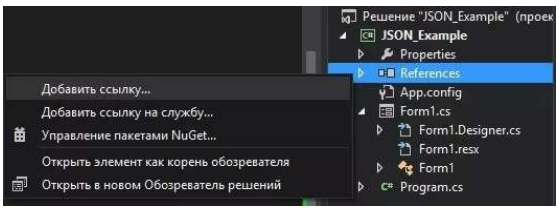
Так что если сервис предлагает API — глупо отказываться.

Сериализация

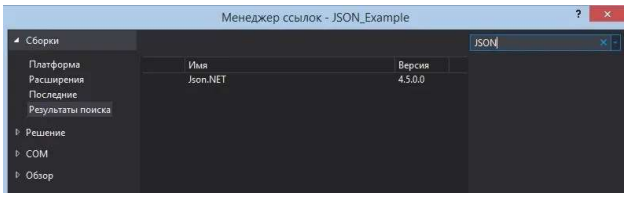
В данном примере мы рассмотрим библиотеку «Newtonsoft.Json» для работы с JSON. Почему её? Потому что она удобная.

Для начала её нужно добавить в references.

Для этого нужно нажать ПКМ на References и выбрать «Добавить ссылку…».



Тут можно найти его в поиске (если уже использовали), скачать его через NuGet, или же просто добавить как обычную библиотеку через «Обзор».



На всякий случай оставлю библиотеку тут: **скачать** [Newtonsoft.Json.dll].

Теперь можно приступить к коду.

Для начала нужно добавить NameSpace библиотеки в код. Для этого добавим в using следующую строку:

```
using System;
// Далее ваши библиотеки
using Newtonsoft.Json;
```

Теперь нужно описать класс, который будем сериализовать и десериализовать:

```
class MyMusic
        {
            public Track[] Tracks { get; set; }
        }

        class Track
        {
            public string Artist { get; set; }
            public string Album { get; set; }
            public string Title { get; set; }
            public string Year { get; set; }

        }
```

Будем считать, что пишем программу для ведения учета домашней музыкальной коллекции. Базовый класс — MyMusic, вспомогательный класс — Track.

Теперь добавим несколько треков:

```
MyMusic myCollection = new MyMusic();
            myCollection.Tracks = new Track[3];

            myCollection.Tracks[0] = new Track()
            {
                Artist="Artist1",
                Album="Album1",
                Title="Title1",
                Year="2015"
            };
            myCollection.Tracks[1] = new Track()
            {
                Artist = "Artist2",
                Album = "Album2",
                Title = "Title2",
```

```
            Year = "2015"
        };
        myCollection.Tracks[2] = new Track()
        {
            Artist = "Artist3",
            Album = "Album3",
            Title = "Title3",
            Year = "2015"
        };
```

Теперь сериализуем этот класс и выведем на RichTextBox:

```
string serialized = JsonConvert.SerializeObject(myCollection);
richTextBox.Text = serialized;
```

В результате чего мы должны получить такую строку на выходе:

```
{"Tracks":[{"Artist":"Artist1","Album":"Album1","Title":"Title1","Year":"2015"},{"Artist":"A
```

Десериализация

Теперь нужно эту строку десериализовать обратно в класс. Для этого, естественно, должен быть объявлен соответствующий класс, иначе программа не сможет ничего десериализовать.

Сам код десериализации следующий:

```
string json = richTextBox.Text;
MyMusic newMusic = JsonConvert.DeserializeObject<MyMusic>(json);
```

Теперь выведем на экран содержимое десериализованного JSON'а:

```
richTextBox.Text = "Всего треков добавлено: "+newMusic.Tracks.Length+Environment.NewLine;
        foreach(var track in newMusic.Tracks)
        {
            richTextBox.Text += track.Artist + " (" + track.Album + "-" + track.Year + "
        }
```

На экран должно вывестись следующий текст:

```
Всего треков добавлено: 3
Artist1 (Album1-2015) - Title1
Artist2 (Album2-2015) - Title2
Artist3 (Album3-2015) - Title3
```

Как видите — с JSON очень удобно работать. Но есть одна маленькая хитрость, которая сильно упростит вашу жизнь при десериализации. Чтобы вам вручную не создавать огромные классы анализируя пример JSON'а, который вам возвращает API — можно его сгенерировать автоматически. В этом вам поможет сайт Json2CSharp. Просто вставьте в него строку с JSON, которую вы получили из API и он сгенерирует вам необходимые классы. Основной класс будет всегда RootObject (в примере был MyMusic), но никто не мешает вам его переименовать в нужный.

Скачать весь проект в 7Z-архиве.

Пароль к архиву: котодомик.рф

Update 08.07.2019

Важный момент.

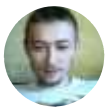
Класс, который Вы хотите сериализовать/десериализовать, должен соответствовать некоторым требованиям, чтобы не было проблем при работе с ним:

1. У класса должен быть обязательно объявлен конструктор без параметров, или не объявлено конструкторов вовсе;
2. Лучше использовать свойства, а не поля (с {get; и set;} как в примерах выше;
3. Если у свойства отсутствует get'тер, или set'тер — то будьте готовы к тому, что часть функционала JSON-сериализатора будет работать не верно. Например:
    1. У вас отсутствует публичный GET'тер, но есть SET'тер. Тогда вы сможете десериализовать (JSON в объект) свойство объекта, но сериализовать (объект в JSON) его вы не сможете;
    2. У вас отсутствует публичный SET'тер, но есть GET'тер. Тогда вы не сможете десериализовать (JSON в объект) свойство объекта, но сможете его сериализовать (объект в JSON);

ПРЕДЫДУЩАЯ
Unity3D. Particle Systems путь по траектории

СЛЕДУЮЩАЯ
C# Работа с файлами и кодировки

58 комментариев для "Сериализация-десериализация JSON в C#"



БУЛАТ
09.10.2015 в 14:23

Спасибо! Три дня ковырял. Очень сильно помогли…

Ответить

ADMINISTRATOR
14.10.2015 в 12:31