Основная функциональность по работе с JSON сосредоточена в пространстве имен **System.Text.Json**.

Ключевым типом является класс **JsonSerializer**, который и позволяет сериализовать объект в json и, наоборот, десериализовать код json в объект C#.

Для сохранения объекта в json в классе JsonSerializer определен статический метод **Serialize()**, который имеет ряд перегруженных версий. Некоторые из них:

* string Serialize(Object obj, Type type, JsonSerializerOptions options): сериализует объект obj типа type и возвращает код json в виде строки. Последний необязательный параметр options позволяет задать дополнительные опции сериализации
* string Serialize<T>(T obj, JsonSerializerOptions options): типизированная версия сериализует объект obj типа T и возвращает код json в виде строки.
* Task SerializeAsync(Object obj, Type type, JsonSerializerOptions options): сериализует объект obj типа type и возвращает код json в виде строки. Последний необязательный параметр options позволяет задать дополнительные опции сериализации
* Task SerializeAsync<T>(T obj, JsonSerializerOptions options): типизированная версия сериализует объект obj типа T и возвращает код json в виде строки.
* object Deserialize(string json, Type type, JsonSerializerOptions options): десериализует строку json в объект типа type и возвращает десериализованный объект. Последний необязательный параметр options позволяет задать дополнительные опции десериализации
* T Deserialize<T>(string json, JsonSerializerOptions options): десериализует строку json в объект типа T и возвращает его.
* ValueTask<object> DeserializeAsync(Stream utf8Json, Type type, JsonSerializerOptions options, CancellationToken token): десериализует текст UTF-8, который представляет объект JSON, в объект типа type. Последние два параметра необязательны: options позволяет задать дополнительные опции десериализации, а token устанавливает CancellationToken для отмены задачи. Возвращается десериализованный объект, обернутый в ValueTask
* ValueTask<T> DeserializeAsync<T>(Stream utf8Json, JsonSerializerOptions options, CancellationToken token): десериализует текст UTF-8, который представляет объект JSON, в объект типа T. Возвращается десериализованный объект, обернутый в ValueTask

Рассмотрим применение класса на простом примере. Сериализуем и десериализуем простейший объект:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | using System;  using System.Text.Json;    namespace HelloApp  {      class Person      {          public string Name { get; set; }          public int Age { get; set; }      }      class Program      {          static void Main(string[] args)          {              Person tom = new Person { Name = "Tom", Age = 35 };              string json = JsonSerializer.Serialize<Person>(tom);              Console.WriteLine(json);              Person restoredPerson = JsonSerializer.Deserialize<Person>(json);              Console.WriteLine(restoredPerson.Name);          }      }  } |

Здесь вначале сериализуем с помощью метода JsonSerializer.Serialize() объект типа Person в стоку с кодом json. Затем обратно получаем из этой строки объект Person посредством метода JsonSerializer.Deserialize().

Консольный вывод:

{"Name":"Tom","Age": 35}

Tom

Хотя в примере выше сериализовался/десериализовался объект класса, но подобным способом мы также можем сериализовать/десериализовать структуры.

**Некоторые замечания по сериализации/десериализации**

Объект, который подвергается десериализации, должен иметь конструктор без параметров. Например, в примере выше этот конструктор по умолчанию. Но можно также явным образом определить подобный конструктор в классе.

Сериализации подлежат только публичные свойства объекта (с модификатором public).

**Запись и чтение файла json**

Поскольку методы SerializeAsyc/DeserializeAsync могут принимать поток типа Stream, то соответственно мы можем использовать файловый поток для сохранения и последующего извлечения данных:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | using System;  using System.IO;  using System.Text.Json;  using System.Threading.Tasks;    namespace HelloApp  {      class Person      {          public string Name { get; set; }          public int Age { get; set; }      }      class Program      {          static async Task Main(string[] args)          {              // сохранение данных              using (FileStream fs = new FileStream("user.json", FileMode.  OpenOrCreate))              {                  Person tom = new Person() { Name = "Tom", Age = 35 };                  await JsonSerializer.SerializeAsync<Person>(fs, tom);                  Console.WriteLine("Data has been saved to file");              }                // чтение данных              using (FileStream fs = new FileStream("user.json", FileMode.  OpenOrCreate))              {                  Person restoredPerson = await JsonSerializer.DeserializeAsync  <Person>(fs);                  Console.WriteLine($"Name: {restoredPerson.Name}  Age: {restoredPerson.Age}");              }          }      }  } |
|  |  |

В данном случае вначале данные сохраняются в файл user.json и затем считываются из него.

**Настройка сериализации с помощью JsonSerializerOptions**

По умолчанию JsonSerializer сериализует объекты в минимифицированный код. С помощью дополнительного параметра типа **JsonSerializerOptions** можно настроить механизм сериализации/десериализации, используя свойства JsonSerializerOptions. Некоторые из его свойств:

* **AllowTrailingCommas**: устанавливает, надо ли добавлять после последнего элемента в json запятую. Если равно true, запятая добавляется
* **IgnoreNullValues**: устанавливает, будут ли сериализоваться/десериализоваться в json объекты и их свойства со значением null
* **IgnoreReadOnlyProperties**: аналогично устанавливает, будут ли сериализоваться свойства, предназначенные только для чтения
* **WriteIndented**: устанавливает, будут ли добавляться в json пробелы (условно говоря, для красоты). Если равно true устанавливаются дополнительные пробелы

Применение:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | var options = new JsonSerializerOptions  {      WriteIndented = true  };    Person tom = new Person { Name = "Tom", Age = 35 };  string json = JsonSerializer.Serialize<Person>(tom, options);  Console.WriteLine(json);  Person restoredPerson = JsonSerializer.Deserialize<Person>(json);  Console.WriteLine(restoredPerson.Name); |

Консольный вывод:

{

"Name": "Tom",

"Age": 35

}

Tom

**Настройка сериализации с помощью атрибутов**

По умолчанию сериализации подлежат все публичные свойства. Кроме того, в выходном объекте json все названия свойств соответствуют названиям свойств объекта C#. Однако с помощью атрибутов **JsonIgnore** и **JsonPropertyName**.

Атрибут **JsonIgnore** позволяет исключить из сериализации определенное свойство. А **JsonPropertyName** позволяет замещать оригинальное название свойства. Пример использования:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | using System;  using System.Text.Json;  using System.Text.Json.Serialization;    namespace HelloApp  {      class Person      {          [JsonPropertyName("firstname")]          public string Name { get; set; }          [JsonIgnore]          public int Age { get; set; }      }      class Program      {          static void Main(string[] args)          {              Person tom = new Person() { Name = "Tom", Age = 35 };              string json = JsonSerializer.Serialize<Person>(tom);              Console.WriteLine(json);              Person restoredPerson = JsonSerializer.Deserialize<Person>(json);              Console.WriteLine($"Name: {restoredPerson.Name}  Age: {restoredPerson.Age}");          }      }  } |

В данном случае свойство Age будет игнорироваться, а для свойства Name будет использоваться псевдоним "firstname". Консольный вывод:

{"firstname":"Tom"}

Name: Tom Age: 0

Обратите внимание, что, поскольку свойство Age не было сериализовано, то при десериализации для него используется значение по умолчанию.