System.Text.Json Шпаргалка с комментариями

Основы сериализации и десериализации

```
// Простая сериализация объекта в JSON-строку
// myObject - это экземпляр любого класса с публичными свойствами
string json = JsonSerializer.Serialize(myObject);

// Десериализация JSON-строки в объект указанного типа
// Генерик-параметр <MyClass> указывает, в какой тип преобразовать JSON
MyClass obj = JsonSerializer.Deserialize<MyClass>(json);

// Работа с JSON без привязки к конкретному классу C#
// Создаем JsonDocument из строки JSON для последующего анализа
JsonDocument doc = JsonDocument.Parse(json);
// Получаем корневой элемент документа
JsonElement root = doc.RootElement;
// Получаем значения отдельных свойств, указывая их имена и тип
string name = root.GetProperty("name").GetString(); // Получаем строковое свойство "name"
int age = root.GetProperty("age").GetInt32(); // Получаем числовое свойство "age"
```

JsonSerializerOptions

```
// Создаем и настраиваем параметры сериализации/десериализации
var options = new JsonSerializerOptions
{
    WriteIndented = true,                     // Добавляет отступы и переносы строк для читаемости
    PropertyNamingPolicy = JsonNamingPolicy.CamelCase, // Преобразует имена свойств в camelCase (первая буква в нижнем регистре)
    PropertyNameCaseInsensitive = true,       // При десериализации игнорирует регистр имен свойств
    DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull, // Не включает свойства со значением null в JSON
    Encoder = JavaScriptEncoder.UnsafeRelaxedJsonEscaping, // Разрешает больше символов без экранирования (например, кириллицу)
    NumberHandling = JsonNumberHandling.AllowReadingFromString, // Позволяет десериализовать числа из строковых значений
    ReferenceHandler = ReferenceHandler.Preserve, // Сохраняет ссылки для предотвращения бесконечных циклов
    MaxDepth = 32                             // Максимальная глубина вложенности JSON-объектов
};

// Сериализация с применением настроенных параметров
string jsonFormatted = JsonSerializer.Serialize(myObject, options);
```

Атрибуты для управления сериализацией

```
public class Person
{
    // Изменяет имя свойства в JSON с "Name" на "full_name"
    [JsonPropertyName("full_name")]
    public string Name { get; set; }

    // Полностью исключает свойство из JSON при сериализации и десериализации
    [JsonIgnore]
    public int InternalId { get; set; }

    // Включает приватное поле в процесс сериализации/десериализации
    [JsonInclude]
    private DateTime _birthDate;

    // Применяет пользовательский конвертер для преобразования типа
    [JsonConverter(typeof(CustomConverter))]
    public Address Address { get; set; }

    // Задает особый способ обработки чисел - записывает число как строковое значение
    [JsonNumberHandling(JsonNumberHandling.WriteAsString)]
    public decimal Price { get; set; }
}
```

Пользовательские конвертеры

```
// Пользовательский конвертер для управления форматом DateTime
public class DateTimeConverter : JsonConverter<DateTime>
{
    // Метод вызывается при десериализации из JSON в DateTime
    public override DateTime Read(ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)
    {
        // Преобразуем строковое значение из JSON в DateTime
        return DateTime.Parse(reader.GetString());
    }

    // Метод вызывается при сериализации DateTime в JSON
    public override void Write(Utf8JsonWriter writer, DateTime value, JsonSerializerOptions options)
    {
        // Записываем дату в формате "yyyy-MM-dd" (например, 2023-03-15)
        writer.WriteStringValue(value.ToString("yyyy-MM-dd"));
    }
}

// Пример регистрации пользовательского конвертера
var options = new JsonSerializerOptions();
options.Converters.Add(new DateTimeConverter()); // Добавляем конвертер в список конвертеров
```

Работа с JsonDocument

```
// Исходная JSON-строка для разбора
string jsonString = @"{ ""name"": ""John"", ""age"": 30, ""address"": { ""city"": ""New York"" } }";

// Используем конструкцию using для автоматического освобождения ресурсов
using (JsonDocument document = JsonDocument.Parse(jsonString))
{
    // Получаем корневой элемент документа
    JsonElement root = document.RootElement;

    // Получаем значения разных свойств
    string name = root.GetProperty("name").GetString(); // Извлекаем строковое свойство
    int age = root.GetProperty("age").GetInt32(); // Извлекаем числовое свойство

    // Безопасное получение свойства с проверкой его существования
    bool hasAddress = root.TryGetProperty("address", out JsonElement address);

    // Перебор элементов массива (если он существует и действительно является массивом)
    if (root.TryGetProperty("items", out JsonElement items) && items.ValueKind == JsonValueKind.Array)
    {
        foreach (JsonElement item in items.EnumerateArray())
        {
            Console.WriteLine(item.GetString());
        }
    }

    // Определение типа JSON-значения (число, строка, объект и т.д.)
    JsonValueKind kind = root.GetProperty("age").ValueKind; // Вернет JsonValueKind.Number
}
```

Сериализация/десериализация потоков

```
// Сериализация объекта непосредственно в поток (например, файловый или сетевой)
using (var stream = new MemoryStream()) // MemoryStream - поток в памяти
{
    // Асинхронно записываем сериализованный объект в поток
    await JsonSerializer.SerializeAsync(stream, myObject, options);
    stream.Position = 0; // Сбрасываем позицию в потоке для последующего чтения
    // Далее можно работать с полученным потоком...
}

// Десериализация из потока (например, при чтении JSON из файла)
using (var stream = new FileStream("data.json", FileMode.Open)) // Открываем файл
{
    // Асинхронно читаем и десериализуем объект из потока
    var result = await JsonSerializer.DeserializeAsync<MyClass>(stream, options);
    // Теперь в result находится десериализованный объект
}
```

Работа с JSON в .NET 6+

```
// Создание JSON-структуры программно через JsonNode (доступно с .NET 6)
JsonObject person = new JsonObject
{
    ["name"] = "John", // Добавляем строковое свойство
    ["age"] = 30, // Добавляем числовое свойство
    ["address"] = new JsonObject // Вложенный объект
    {
        ["city"] = "New York",
        ["zipCode"] = "10001"
    },
    ["hobbies"] = new JsonArray { "reading", "sports", "music" } // Массив строк
};

// Доступ и изменение значений по ключу
person["age"] = 31; // Изменяем возраст
string city = person["address"]["city"].GetValue<string>(); // Получаем город
person["address"]["city"] = "Boston"; // Изменяем город

// Работа с JSON-массивом
JsonArray hobbies = (JsonArray)person["hobbies"]; // Приводим к типу JsonArray
hobbies.Add("programming"); // Добавляем новое хобби в массив

// Преобразование обратно в строку
string json = person.ToJsonString(); // Получаем JSON-строку из построенной структуры
```

Обработка ошибок

```
try
{
    // Пытаемся десериализовать JSON
    var result = JsonSerializer.Deserialize<MyClass>(json);
}
catch (JsonException ex)
{
    // Обрабатываем специфические ошибки JSON
    Console.WriteLine($"Ошибка при десериализации JSON: {ex.Message}");
}
```

```
// Настройка параметров для более гибкой обработки некорректного JSON
var options = new JsonSerializerOptions
{
    AllowTrailingCommas = true,           // Разрешает запятые после последнего элемента (например, [1,2,3,])
    ReadCommentHandling = JsonCommentHandling.Skip // Пропускает комментарии в JSON, которые обычно не допускаются
};
```

Утилиты для работы с JSON-путями

```
using System.Text.Json.Nodes;

// Создаем JSON-структуру из строки
var jsonObj = JsonNode.Parse(@"{
  ""store"": {
    ""books"": [
      { ""title"": ""Book 1"", ""price"": 10.99 },
      { ""title"": ""Book 2"", ""price"": 15.99 }
    ]
  }
}");

// Получаем значение по пути, используя индексаторы
// Переходим к store -> books -> второй элемент (индекс 1) -> price
decimal price = jsonObj["store"]["books"][1]["price"].GetValue<decimal>();

// Изменяем значение по пути
jsonObj["store"]["books"][0]["price"] = 12.99; // Устанавливаем новую цену для первой книги

// Добавляем новый элемент в массив books
((JsonArray)jsonObj["store"]["books"]).Add(new JsonObject
{
    ["title"] = "Book 3", // Название новой книги
    ["price"] = 20.99 // Цена новой книги
});
```

Пример десериализации ответа API

```
// Пример классов для десериализации данных о погоде
public class WeatherResponse
{
    // Свойство с измененным именем для соответствия JSON от API
    [JsonPropertyName("current_weather")]
    public CurrentWeather CurrentWeather { get; set; }

    // Свойство с прямым соответствием имени
    public Location Location { get; set; }
}

public class CurrentWeather
{
    public double Temperature { get; set; }

    [JsonPropertyName("wind_speed")]
    public double WindSpeed { get; set; }

    [JsonPropertyName("wind_direction")]
    public int WindDirection { get; set; }

    // Конвертируем числовой код погоды в перечисление
    [JsonConverter(typeof(WeatherCodeConverter))]
    [JsonPropertyName("weather_code")]
    public WeatherCondition WeatherCondition { get; set; }
}

public class Location
{
    public string City { get; set; }
    public string Country { get; set; }
    public double Latitude { get; set; }
    public double Longitude { get; set; }
}

// Перечисление для типов погоды
public enum WeatherCondition
{
    Clear,
    PartlyCloudy,
    Cloudy,
    Rain,
    Snow,
    Thunderstorm
}

// Конвертер для преобразования числового кода погоды в перечисление
public class WeatherCodeConverter : JsonConverter<WeatherCondition>
{
    public override WeatherCondition Read(ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)
    {
        int code = reader.GetInt32();
        return code switch
        {
```

```
            0 => WeatherCondition.Clear,
            1 or 2 => WeatherCondition.PartlyCloudy,
            3 => WeatherCondition.Cloudy,
            >= 51 and <= 65 => WeatherCondition.Rain,
            >= 71 and <= 77 => WeatherCondition.Snow,
            >= 95 and <= 99 => WeatherCondition.Thunderstorm,
            _ => WeatherCondition.Clear // Значение по умолчанию
        };
    }

    public override void Write(Utf8JsonWriter writer, WeatherCondition value, JsonSerializerOptions options)
    {
        int code = value switch
        {
            WeatherCondition.Clear => 0,
            WeatherCondition.PartlyCloudy => 2,
            WeatherCondition.Cloudy => 3,
            WeatherCondition.Rain => 61,
            WeatherCondition.Snow => 71,
            WeatherCondition.Thunderstorm => 95,
            _ => 0
        };
        writer.WriteNumberValue(code);
    }
}

// Полный пример получения и десериализации данных API
public async Task<WeatherResponse> GetWeatherDataAsync(string city)
{
    // Создаем HttpClient для отправки запросов
    using (var client = new HttpClient())
    {
        try
        {
            // Формируем URL запроса (пример)
            string url = $"https://api.weatherservice.com/v1/current?city={city}";

            // Выполняем GET-запрос
            HttpResponseMessage response = await client.GetAsync(url);

            // Проверяем успешность запроса
            response.EnsureSuccessStatusCode();

            // Читаем содержимое ответа как строку
            string jsonResponse = await response.Content.ReadAsStringAsync();

            // Настраиваем параметры десериализации
            var options = new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true, // Игнорировать регистр имен свойств
                DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull // Игнорировать null-свойства при сериализации
            };

            // Десериализуем JSON в объект WeatherResponse
            WeatherResponse weatherData = JsonSerializer.Deserialize<WeatherResponse>(jsonResponse, options);

            return weatherData;
        }
        catch (HttpRequestException e)
        {
            Console.WriteLine($"Ошибка при запросе к API: {e.Message}");
            throw;
        }
        catch (JsonException e)
        {
            Console.WriteLine($"Ошибка при разборе JSON: {e.Message}");
            throw;
        }
    }
}

// Пример использования
public async Task DisplayWeatherAsync()
{
    try
    {
        WeatherResponse weather = await GetWeatherDataAsync("Moscow");

        Console.WriteLine($"Погода в городе {weather.Location.City}, {weather.Location.Country}");
        Console.WriteLine($"Температура: {weather.CurrentWeather.Temperature}°C");
        Console.WriteLine($"Условия: {weather.CurrentWeather.WeatherCondition}");
        Console.WriteLine($"Скорость ветра: {weather.CurrentWeather.WindSpeed} м/с");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Произошла ошибка: {ex.Message}");
    }
}
```