ALLAN THRAEN | 1 years ago | PDF |

.NET Development C#

USING SYSTEM.TEXT.JSON TO DO POLYMORPHIC JSON CONVERSION IN .NET 6

When using System.Text.Json to serialize complex objects you sometimes need to go a bit beyond how the default serialization works. Here are a few helpful converters - for example for doing polymorphic conversion.

Polymorphic Json Conversion

Often, when I'm serializing complex object in .NET I find myself needing polymorphic serialization. A classic example is this:

I have a List<IMyInterface> property which contain various objects that all have in common that they implement IMyInterface. Normally these doesn't serialize well, cause how would a deserializer know which class to instantiate.

In .NET 7 there is a way to handle this but not in .NET 6 or before. So, inspired by this post I made a generic approach that can easily be added to the Json Converters:

var options = new JsonSerializerOptions();
options.Converters.Add(new PolymorphicJsonConverter<IMyInterface>());

And you should be good to go. The code is in the gist below.

IgnoredTypeInJsonConverter

Another useful converter is a simple generic converter that ignores elements of a given type when serializing to Json.
Sometimes when you serialize large complex types from existing packages you might not want to include everything - and if you are not in control of the types you can't add Ignore attributes to them. But luckily you can just ignore them as part of the serialization. See the converter below and attach it in the same way as above.

CustomStringConversionJsonConverter

The last useful converter snippet I will share here is this one. There might be times when you want to decide yourself how a given type is serialized to a string - and this will help you do it.

Assume for example that you have a very large object inside another serialized object, but all you really want to serialize is the ID property of it - then you can simply use it like this:

options.Converters.Add(newCustomStringConversionJsonConverter<MyVeryDetailedType>(mvdt => mvdt.ID.ToString(), s => new MyVeryDetailsType(){ID = Int.Parse(s)}));

I hope you find the above useful. Remember to leave a comment if you have any suggestions!

```
public class CustomStringConversionJsonConverter<G> : JsonConverter<G>
{
    private readonly Func<G, string> _convertToString;
    private readonly Func<string, G> _convertFromString;
    public CustomStringConversionJsonConverter(Func<G,string> ConvertToString, Func<string,G> ConvertFromString)
    {
        _convertFromString = ConvertFromString;
        _convertToString = ConvertToString;
    }

    public override bool CanConvert(Type typeToConvert)
    {
        return typeof(G).IsAssignableFrom(typeToConvert);
    }

    public override G? Read(ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)
    {
        if (reader.TokenType != JsonTokenType.String)
        {
            throw new JsonException();
        }
        var s = reader.GetString();
        return _convertFromString(s);
    }

    public override void Write(Utf8JsonWriter writer, G value, JsonSerializerOptions options)
    {
```

```
            writer.WriteStringValue(__convertToString(value));
        }
    }
```

CustomStringConversionJsonConverter.cs hosted with ♥ by GitHub view raw

```
    public class IgnoreTypeInJsonConverter<G> : JsonConverter<G> where G: class
    {
        public override bool CanConvert(Type typeToConvert)
        {
            return typeof(G) == typeToConvert;//.IsAssignableFrom(typeToConvert);
        }

        public override G? Read(ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)
        {
            if (reader.TokenType == JsonTokenType.StartObject)
            {
                while (reader.Read())
                {
                    if (reader.TokenType == JsonTokenType.EndObject) return default(G);
                }
            }
            else reader.Read();
            return default(G);
        }

        public override void Write(Utf8JsonWriter writer, G value, JsonSerializerOptions options)
        {
            //Do nothing
            writer.WriteNullValue();
        }
    }
```

IgnoreTypeInJsonConverter.cs hosted with ♥ by GitHub view raw

```
    public class PolymorphicJsonConverter<G> : JsonConverter<G>
    {
        //Inspired by https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/converters-how-to?pivots=dotnet-6-0#support-polymorp

        public override bool CanConvert(Type typeToConvert)
        {
            return typeof(G) == typeToConvert;//.IsAssignableFrom(typeToConvert);
        }

        public override G Read(ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)
        {
            if (reader.TokenType != JsonTokenType.StartObject)
            {
                throw new JsonException();
            }

            reader.Read();
            if (reader.TokenType != JsonTokenType.PropertyName)
            {
                throw new JsonException();
            }

            string? propertyName = reader.GetString();

            if (propertyName != "__type")
            {
                throw new JsonException();
            }

            reader.Read();
            if (reader.TokenType != JsonTokenType.String)
            {
                throw new JsonException();
            }
            string type = reader.GetString();

            //Check type exists
            var tp=Type.GetType(type);

            reader.Read();
            if (reader.TokenType != JsonTokenType.PropertyName)
            {
                throw new JsonException();
            }

            propertyName = reader.GetString();

            if (propertyName != "__object")
            {
                throw new JsonException();
            }

            var rt= (G) JsonSerializer.Deserialize(ref reader, tp, options);

            while (reader.Read())
            {
                if (reader.TokenType == JsonTokenType.EndObject) return rt;
            }
            return rt;
        }

        public static T Cast<T>(object o)
        {
            return (T)o;
        }

        public override void Write(Utf8JsonWriter writer, G value, JsonSerializerOptions options)
        {
            writer.WriteStartObject();
            writer.WriteString("__type", value.GetType().AssemblyQualifiedName);
            writer.WritePropertyName("__object");
            JsonSerializer.Serialize(writer, value, value.GetType());
            writer.WriteEndObject();
        }
    }
```

PolymorphicJsonConverter.cs hosted with ♥ by GitHub view raw

.NET Development C#

RECENT POSTS

CodeArt ApS
Teknikerbyen 5, 2830 Virum, Denmark
Email: info@codeart.dk
Phone: +45 26 13 66 96
CVR: 39680688

Copyright © 2024