JSON - JavaScript Object Notation

Ingo Köster

Diplom Informatiker (FH)

JavaScript Object Notation

- Datenformat für Datenaustausch zwischen Anwendungen
- Daten liegen in einfach lesbarer Textform vor
  - Sowohl für Mensch und Maschine einfach auszuwerten
- JSON ist Sprachunabhängig und viele Sprachen bieten Parser für JSON

JSON vs. XML

- Leichtgewichtige Alternative zu XML
  - Schneller und einfacher

- JSON
  - `{ "vorname" : "Klaus", "nachname" : "Thaler" }`

- XML

```
<?xml version="1.0" encoding="utf-8" ?>
<Person>
<vorname>Klaus</vorname>
<nachname>Thaler</nachname>
</Person>
```

Aufbau von JSON

- Name/Wert Paare
  - Wie Wörterbuch (Dictionary) oder Hash-Tabelle
- Geordnete Liste von Werten
  - Wie Arrays oder Listen
- Inhalte von JSON
  - Nullwerte, boolesche Werte, Zahlen, Zeichenketten, Arrays, Objekte

Daten

Datentypen	Inhalt
Nullwerte	null
Boolesche Werte	true oder false
Zahlen	0-9 mit Dezimalpunkt und Vorzeichen
Zeichenketten	Mittels doppelter Anführungszeichen ""
Arrays	Mittels []
Objekte	Mittels {}

JSON Beispiel

```
{
"vorname"  : "Klaus",
"nachname": "Thaler",
"alter"    : 62,
"telefon"  : "1234567",
"männlich": true,
"hobbys"   : [ "Reiten", "Lesen", "Schwimmen" ],
"kinder"   : [],
"partner"  : null,
"adresse"  : { "strasse" : "Im Tal 1", "ort" : "12345 Stadt" }
}
```

JSON-Array

```
{
"teilnehmer":
 [
   { "vorname":"Dieter" , "nachname":"Müller" },
   { "vorname":"Anna" , "nachname":"Schmidt" },
   { "vorname":"Peter" , "nachname":"Klein" }
 ]
}
```

JSON in C#

JSON in C#

- .NET enthält eine Klasse für die JSON Serialisierung
  - `JsonSerializer`
- Namensraums `System.Text.Json`
- Es existieren weitere JSON Parser von Drittanbietern für .NET
  - Z.B. von Newtonsoft (NuGet Paket `Newtonsoft.Json`)

Serialisieren

```
› Person tim = new Person() { Email = "tim@home", Name =
  "Tim", Notizen = "Noch keine" };

› string json_tim = JsonSerializer.Serialize(tim);

› Console.WriteLine(json_tim);

› // Ausgabe
› {"Email":"tim@home","Name":"Tim","Notizen":"Noch keine"}
```

Deserialisieren

```
› string json = "{ \"Email\": \"lisa@home\",
  \"Name\":\"Lisa\", \"Notizen\":\"keine\" }";
```

```
› Person lisa = JsonSerializer.Deserialize<Person>(json);
```

Hinweise Serialisierung

› Serialisiert werden alle öffentlichen (`public`) Properties

› Öffentliche oder private Felder werden nicht serialisiert!

```
class Person
{
  public string Name { get; set; }    // Wird serialisiert
  public int zahl = 5;                // Wird nicht serialisiert!
  private int x = 23;                 // Wird nicht serialisiert!
}
```

Hinweise Deserialisierung

- Serialisiert werden alle Properties
- Deserialisiert werden nur die Properties die lesend und schreibend sind
- Für die Deserialisierung muss ein leerer Konstruktor vorhanden sein
- Hintergrund:
  - Beim Deserialisieren wird ein neues „leeres“ Objekt erzeugt
  - Über die Properties werden die Daten aus der Datei in die Properties gesetzt

Namen einer Eigenschaft ändern

› Um den Namen einer Eigenschaft im JSON-Dokument zu ändern, wird das Attribut `[JsonPropertyName]` verwendet

```
public class WeatherForecast
{
    public DateTimeOffset Date { get; set; }
    [JsonPropertyName("Temperature")]
    public int TemperatureCelsius { get; set; }
    public int WindSpeed { get; set; }
}
```

Ignorieren von Eigenschaften

› Beim Serialisieren von Objekten werden alle öffentlichen Properties serialisiert

› Durch das Attribut `[JsonIgnore]` kann eine einzelne Eigenschaft ignoriert werden

```
public class WeatherForecast
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    [JsonIgnore]
    public int WindSpeed { get; set; }
}
```

JSON Validierungsbeispiel

- JSON-Daten in ein `Dictionary` überführen, um zu validieren
  - `string json = "{ \"Email\": \"lisa@home\", \"Name\"...;`
  - `var result = JsonSerializer.Deserialize<Dictionary<string, object>>(json);`

- Bei ungültigem JSON wird eine `ArgumentException` ausgelöst

- Erzeugt ein `Dictionary<string, object>`
  - `[0] Key: Email    Value: lisa@home`
  - `[1] Key: Name     Value: Lisa`
  - `[2] Key: Notizen  Value: keine`

Ignorieren von Groß- und Kleinschreibung

› Groß- und Kleinschreibungsunterschiede zwischen JSON und Klasse ignorieren

```
string json = "{ \"name\":\"Lisa\"}";

class Person { public string Name { get; set; } }

var options = new JsonSerializerOptions {
  PropertyNameCaseInsensitive = true
};

Person lisa = JsonSerializer.Deserialize<Person>(json, options);
```

Hinweise zur Vererbung

› Vor der Version 7 von .NET gibt es keine Serialisierung von polymorphen Typhierarchien

› Ab .NET 7 wird die Serialisierung durch Attribute gesteuert

```
[JsonDerivedType(typeof(B))]
public class A
{
    public string AAA { get; set; }
}

public class B : A
{
    public string BBB { get; set; }
}
```

Aus JSON-Daten Klasse generieren