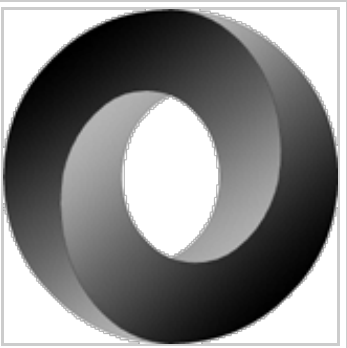
JavaScript Object Notation

aus Wikipedia, der freien Enzyklopädie

Die **JavaScript Object Notation**, kurz **JSON** ['dʒeɪsən], ist ein kompaktes Datenformat in für Mensch und Maschine einfach lesbarer Textform zum Zweck des Datenaustauschs zwischen Anwendungen. Jedes gültige JSON-Dokument soll ein gültiges JavaScript sein und per `eval()` interpretiert werden können. Aufgrund kleiner Abweichungen in der Menge der erlaubten Unicode-Zeichen ist es jedoch möglich, JSON-Objekte zu erzeugen, die von einem normkonformen JavaScript-Interpreter nicht akzeptiert werden.[1] Davon abgesehen ist JSON aber unabhängig von der Programmiersprache. Parser existieren in praktisch allen verbreiteten Sprachen. JSON wurde von Douglas Crockford spezifiziert.



JSON-Logo

Inhaltsverzeichnis

Einsatzgebiete

- In Verbindung mit *JavaScript on Demand (JOD)*, Ajax oder WebSockets zur Übertragung von Daten zwischen Client und Server.
- Ersatz für XML in Bereichen, wo Ressourcen (Speicherplatz, CPU-Leistung) sparsam eingesetzt werden sollen. Dies gilt im Besonderen bei der Entwicklung von desktopähnlichen Webanwendungen.

Datenstruktur und Formatdefinition

Die Daten können beliebig verschachtelt werden, beispielsweise ist ein Array von Objekten möglich. Als Zeichenkodierung benutzt JSON standardmäßig UTF-8. Auch UTF-16 und UTF-32 sind möglich.

JSON kennt folgende Datentypen:

Nullwert
: wird durch das Schlüsselwort **`null`** dargestellt.

boolescher Wert
: wird durch die Schlüsselwörter **`true`** und **`false`** dargestellt. Dies sind *keine* Zeichenketten. Sie werden daher, wie **`null`**, *nicht* in Anführungszeichen gesetzt.

Zahl
: ist eine Folge der Ziffern **`0`**–**`9`**. Diese Folge kann durch ein negatives Vorzeichen **`-`** eingeleitet und einen Dezimalpunkt **`.`** unterbrochen sein. Die Zahl kann durch die Angabe eines Exponenten **`e`** oder **`E`** ergänzt werden, dem ein Vorzeichen **`+`** oder **`-`** und eine Folge der Ziffern **`0`**–**`9`** folgt.

Zeichenkette
: beginnt und endet mit doppelten geraden Anführungszeichen (**`"`**). Sie kann Unicode-Zeichen und Escape-Sequenzen enthalten.

Array
: beginnt mit **`[`** und endet mit **`]`**. Es enthält eine durch Kommata geteilte, geordnete Liste von *Werten*, gleichen oder verschiedenen Typs. Leere Arrays sind zulässig.

Objekt
: beginnt mit **`{`** und endet mit **`}`**. Es enthält eine durch Kommata geteilte, ungeordnete Liste von *Eigenschaften*. Objekte ohne Eigenschaften ("leere Objekte") sind zulässig.

  Eigenschaft
  : besteht aus einem Schlüssel und einem Wert, getrennt durch einen Doppelpunkt (`Schlüssel:Wert`). Die Schlüssel aller Eigenschaften in einem Objekt müssen eindeutig, also paarweise verschieden sein.

    - der **Schlüssel** ist eine Zeichenkette.
    - der **Wert** ist ein *Objekt*, ein *Array*, eine *Zeichenkette*, eine *Zahl* oder einer der Ausdrücke `true`, `false` oder `null`.

Nicht signifikante Leerraum-Zeichen sind verwendbar.[2]

Einschränkungen

JSON unterstützt nicht alle von JavaScript unterstützten Datentypen. Daher werden bei der Serialisierung

- `NaN`, `Infinity` und `-Infinity` zu `null` serialisiert,
- `Date`-Objekte als String in das ISO-8601-Format konvertiert, und
- `Function`-, `RegExp`- und `Error`-Objekte verworfen.

Beispiel

```
{
  "Herausgeber": "Xema",
  "Nummer": "1234-5678-9012-3456",
  "Deckung": 2e+6,
  "Währung": "EURO",
  "Inhaber": {
    "Name": "Mustermann",
    "Vorname": "Max",
    "männlich": true,
    "Hobbys": [ "Reiten", "Golfen", "Lesen" ],
    "Alter": 42,
    "Kinder": [],
```

```
    "Partner": null
  }
}
```

Unterschied zu XML

Die Syntax von JSON ist einfacher gestaltet und erscheint daher oft lesbarer und insbesondere leichter schreibbar. In der Regel reduziert JSON auch den Overhead im Vergleich zu XML.

In XML könnten viele Werte und Eigenschaften potenziell sowohl als Attribute als auch Kindknoten beschrieben werden, was zu Problemen führen kann, wenn dies nicht durch sehr strikte Spezifizierung verhindert wird. In JSON kann dieses Problem nicht auftreten.

JSON-Daten sind im Gegensatz zu XML-Daten typisiert, wobei nur einige grundlegende Typen unterstützt werden. Eine Stärke von JSON ist die Tatsache, dass es sich bei der Definition selbst, bis auf wenige Einschränkungen[1] um valides JavaScript handelt. Damit lässt sich eine JSON-Definition in JavaScript direkt mit der `eval()`-Funktion in ein JavaScript-Objekt umsetzen. Bei Daten aus potentiell unsicheren Quellen sollte aber unbedingt ein Parser verwendet werden, da `eval` auch ggf. schädliche Programmanweisungen ausführt.

XML ist eine Auszeichnungssprache und somit vielseitiger einsetzbar als JSON, das ein Datenaustauschformat ist. XML ist weiter verbreitet, wird jedoch von JSON aufgrund seiner Einfachheit dort zurückgedrängt, wo keine komplizierten Auszeichnungen notwendig sind. Beide Formate sind nicht gut zum Repräsentieren von Binärdatenmengen geeignet, da beide keinen Binärdatentyp unterstützen.

Zum Vergleich das oben genannte Beispiel in einer XML-Form:

```
<Kreditkarte
  Herausgeber="Xema"
  Nummer="1234-5678-9012-3456"
  Deckung="2e+6"
  Waehrung="EURO">
  <Inhaber
    Name="Mustermann"
    Vorname="Max"
    maennlich="true"
    Alter="42"
    Partner="null">
    <Hobbys>
      <Hobby>Reiten</Hobby>
      <Hobby>Golfen</Hobby>
      <Hobby>Lesen</Hobby>
    </Hobbys>
    <Kinder />
  </Inhaber>
</Kreditkarte>
```

Nach Entfernung der optionalen Leerzeichen ist das JSON-Objekt 224 Byte, das XML-Objekt 289 Byte (ein Zuwachs um 29%) groß. Oftmals können Attribute auch als Kindknoten formuliert werden, das Beispiel könnte dann wie folgt aussehen:

```
<Kreditkarte>
  <Herausgeber>Xema</Herausgeber>
  <Nummer>1234-5678-9012-3456</Nummer>
  <Deckung>2e+6</Deckung>
  <Waehrung>EURO</Waehrung>
  <Inhaber>
    <Name>Mustermann</Name>
    <Vorname>Max</Vorname>
    <maennlich>true</maennlich>
    <Hobbys>
```

```
      <Hobby>Reiten</Hobby>
      <Hobby>Golfen</Hobby>
      <Hobby>Lesen</Hobby>
    </Hobbys>
    <Alter>42</Alter>
    <Kinder />
    <Partner>null</Partner>
  </Inhaber>
</Kreditkarte>
```

Dieses Objekt wäre mit Entfernung der Leerzeichen 362 Byte (ein Zuwachs um 62%) groß.

JSONP (JSON mit Padding)

JSONP (JSON mit Padding) ermöglicht die Übertragung von (JSON-)Daten über Domaingrenzen.

Üblicherweise erfolgen Ajax-Datenabfragen an Server über das XMLHttpRequest-Objekt eines Webbrowsers. Aufgrund der Same-Origin-Policy funktioniert das nicht, wenn die in einem Webbrowser angezeigte Webseite über dieses Objekt auf einen Server zuzugreifen versucht, der in einer anderen Domain als die angezeigte Webseite liegt. Das Problem kann durch JSONP umgangen werden.

Die Grundidee: JSON-Abfragen über Script-Tags

Im `src`-Attribut eines `<script>`-Elements ist es möglich, beliebige URLs anzugeben. Für dieses Attribut greift die Same-Origin-Policy nicht. Es ist also möglich, eine URL in einer anderen Domain anzugeben, die beispielsweise JSON-Daten zurückgibt. Dieses Script hätte aber keinen Effekt.

Padding

Um die JSON-Daten auf dem Client verarbeiten zu können, verpackt der Server diese als Parameter in eine JavaScript-Funktion, die im Webbrowser bereits definiert ist. Der Name dieser Funktion wird dem Server über einen Query String der URL mitgeteilt; beispielsweise:

```
<script type="text/javascript"
        src="http://example.com/getjson?jsonp=Rueckruf">
</script>
```

Script-Element-Injektion (Einfügen von Programmcode)

Für jeden JSONP-Aufruf ist ein eigenes `<script>`-Element erforderlich. Daher muss der Browser für jeden Aufruf ein neues `<script>`-Element in den DOM-Knotenbaum der aktuellen Webseite einfügen.

Sicherheitsrisiken

`<script>`-Elemente ermöglichen es einem Server, *beliebige* Inhalte (nicht nur JSON-Objekte) an den Webbrowser zu übermitteln. Dies kann dazu führen, dass ein bösartiger Web-Service über die zurückgesendeten Daten private Informationen im Webbrowser ausspäht oder in seinem Sinne verändert.

Cross-Site Request Forgery

Da das `<script>`-Element die Same-Origin-Policy nicht beachtet, kann eine bösartige Webseite JSONP-Daten anfordern und auswerten, die nicht für sie bestimmt sind (Cross-Site Request Forgery).[3] Das Problem tritt dann auf, wenn sensible Daten vor Dritten geschützt werden sollen.

Geschichte

JSONP wurde 2005 von Bob Ippolito vorgestellt[4] und wird jetzt von vielen Web-2.0-Anwendungen wie Dojo Toolkit, jQuery[5], Google Web Toolkit Applications[6] und Web Services unterstützt. Für dieses Protokoll wurden Erweiterungen vorgeschlagen, die zusätzliche Eingabeparameter ermöglichen, wie z. B. JSONPP[7].

Cross-Origin Resource Sharing

Mit Cross-Origin Resource Sharing (CORS) existiert eine vergleichbare Technologie, die den Zugriff über Domaingrenzen hinweg ermöglicht.

Ähnliche Techniken

Mit YAML existiert eine ähnliche Technik. Allerdings ist YAML eine Markup-Sprache zur reinen Serialisierung und in keiner Sprache gültiger Code. Aber auch hierbei handelt es sich um einen „Document Object Model"-Dateityp. YAML kann als Obermenge von JSON angesehen werden, da jedes JSON-Dokument auch ein valides YAML-Dokument ist.[8]

Mit BSON (Binary JSON) existiert eine binäre JSON-Variante, die unter anderem von MongoDB verwendet wird.[9] Einen ähnlichen Ansatz verfolgen Googles Protocol Buffers (protobuf), denen im Vergleich zu JSON bzw. BSON ein Schema zugrunde liegt.[10][11]

NextSTEP bzw. MacOS X kennt eine ähnliche Technik, um einfache Objektbäume zu laden oder zu speichern, sie heißen dort „Property Lists". Diese erlauben ebenfalls die Speicherung von Werten der Typen Array, Dictionary, boolescher Wert, Binärdaten (Base64-kodiert), Datum, Zahl und Zeichenketten, entweder als XML, als kompaktes Binärformat oder als ASCII bzw. UTF-8.[12]

Symbolische Ausdrücke in Lisp-Notation beschreiben sowohl Daten als auch Code.

BERT ist ein Binärformat, das auf Erlang basiert, und hat somit eine ähnliche Herkunft wie JSON. Ähnlich zu JSON gibt es auch ein auf BERT basierendes RPC-Format.[13]

Weblinks

- json.org (http://json.org/json-de.html) deutsche Einführung auf der offiziellen JSON-Seite (weitere Sprachen verfügbar)
- JSON als XML-Alternative (http://www.webmasterpro.de/coding/article/json-als-xml-alternative.html) Kurze Einführung
- Speeding Up AJAX with JSON (http://www.developer.com/lang/jscript/article.php/3596836) Einführung in JSON, bei der die Unterschiede zu XML herausgearbeitet werden (englisch)
- RFC 4627 *application/json*, ein zusätzlicher MIME-Typ für JSON, zur Unterscheidung von JavaScript
- JSON Formatter & Validator (http://jsonformatter.curiousconcept.com/) Online Formatter und Validator (englisch)
- jsonp.eu, die deutsche JSONP Seite (http://jsonp.eu/) Erklärungen im Detail und Programmierbeispiele zu

JSON with Padding

Einzelnachweise

1. http://timelessrepo.com/json-isnt-a-javascript-subset
2. *RFC 4627* (http://tools.ietf.org/html/rfc4627). 2. JSON Grammar
3. Jeremiah Grossman: *Advanced Web Attack Techniques using GMail (http://jeremiahgrossman.blogspot.com/2006/01/advanced-web-attack-techniques-using.html)*. 27. Januar 2006. Abgerufen am 23. Januar 2011.
4. *Remote JSON - JSONP (http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/)*. In: *from __future__ import **. Bob.pythonmac.org. 5. Dezember 2005. Abgerufen am 23. Januar 2011.
5. *jQuery API (http://api.jquery.com/jQuery.getJSON/)*. Abgerufen am 23. Januar 2011.
6. *GWT Tutorial: How to Read Web Services Client-Side with JSONP (http://www.gwtapps.com/?p=42)*. In: *Google Web Toolkit Applications*. 6. Februar 2008. Abgerufen am 23. Januar 2011.
7. Jonas Almeida: *JSON, JSONP, JSONPP? (http://sites.google.com/a/s3db.org/s3db/documentation/mis/json-jsonp-jsonpp)*. S3DB. 11. Juni 2008. Abgerufen am 23. Januar 2011.
8. YAML Ain't Markup Language (YAML™) Version 1.2 (http://yaml.org/spec/1.2/spec.html#id2759572)
9. bsonspec.org (http://bsonspec.org)
10. What Are Protocol Buffers? (https://developers.google.com/protocol-buffers/)
11. Protocol Buffers - Google's data interchange format (https://code.google.com/p/protobuf/)
12. *Introduction to Property Lists.* (http://developer.apple.com/documentation/Cocoa/Conceptual/PropertyLists/Introduction/Introduction.html) Abgerufen am 6. November 2011 (englisch).
13. http://bert-rpc.org/