eBook Gratuit

APPRENEZ JSON

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#json

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: json

It is an unofficial and free JSON ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official JSON.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec JSON

Remarques

JSON (JavaScript Object Notation) est l'un des formats d'échange de données les plus populaires et les plus largement acceptés à l'origine spécifiés par Douglas Crockford.

Il est actuellement décrit par deux normes concurrentes, RFC 71592 et ECMA-404 . La norme ECMA est minimale, décrivant uniquement la syntaxe de grammaire autorisée, tandis que la RFC fournit également des considérations de sémantique et de sécurité.

- JSON est largement accepté dans les logiciels qui incluent l'architecture client-serveur pour l'échange de données entre le client et le serveur.
- JSON est facile à utiliser et purement basé sur du texte, léger et lisible par l'homme, et les gens se méprennent souvent comme le remplacement de XML.
- Bien que l'abréviation commence par JavaScript, JSON n'est pas un langage ou ne contient aucun littéral de langue, il s'agit simplement d'une spécification pour la notation des données.
- Il est indépendant de la plate-forme et du langage et intégré dans presque tous les langages / frameworks de première ligne, et la prise en charge du format de données JSON est disponible dans tous les langages populaires. et beaucoup plus.
- Le type de média Internet officiel pour JSON est application / json.
- L'extension de nom de fichier JSON est .json.

Versions

Comme JSON n'a pas de mises à jour, il n'ya qu'une version de JSON, le lien ci-dessous redirige vers le document RFC d'origine, à savoir RFC 4627.

Version	Date de sortie
Original	2006-07-28

Examples

Règles de syntaxe JSON

La syntaxe JSON (JavaScript Object Notation) est basée sur un sous-ensemble de JavaScript (voir aussi json.org).

Une expression JSON valide peut être l'un des types de données suivants

- types de données simples: String, Number, Boolean, Null
- types de données composites: valeur, objet, tableau

Types de données simples

Une chaîne JSON doit être placée entre guillemets et peut contenir zéro ou plusieurs caractères Unicode; les échappements antislash sont autorisés. Les nombres JSON acceptés sont en notation E. Le booléen est `true` , `false` . Null est le mot clé réservé `null` .

Type de données	Exemples de JSON valide
### Chaîne	`"apple"`
	`"□□"`
	`"\u00c4pfel\n"`
	`""`
### Nombre	`3`
	`1.4`
	`-1.5e3`
### Booléen	`true`
	`false`
### Nul	`null`

Types de données composites

Valeur

Une valeur JSON peut être l'une des suivantes: String, Number, Boolean, Null, Object, Array.

Objet

Un objet JSON est une collection non ordonnée, séparée par des virgules, de paires nom-valeur entre accolades, où nom est une chaîne et une valeur JSON.

Tableau

Un tableau JSON est une collection ordonnée de valeurs JSON.

Exemple de tableau JSON:

```
["home", "wooden"]
```

Exemples d'objets JSON:

```
{
    "id": 1,
```

```
    "name": "A wooden door",
    "price": 12.50,
    "tags": ["home", "wooden"]
}
```

```
[
  1,
  2,
  [3, 4, 5, 6],
  {
    "id": 1,
    "name": "A wooden door",
    "price": 12.50,
    "tags": ["home", "wooden"]
  }
]
```

Outils en ligne pour valider et formater les données JSON:

- http://jsonlint.com/
- http://www.freeformatter.com/json-validator.html
- http://jsonviewer.stack.hu/
- http://json.parser.online.fr/

Objet JSON

Un objet JSON est entouré d'accolades et contient des paires clé-valeur.

```
{ "key1": "value1", "key2": "value2", ... }
```

Les clés doivent être des chaînes valides, donc entourées de guillemets doubles. Les valeurs peuvent être des objets JSON, des nombres, des chaînes, des tableaux ou l'un des «noms littéraux» suivants: `false` , `null` ou `true` . Dans une paire clé-valeur, la clé est séparée de la valeur par deux points. Plusieurs paires clé-valeur sont séparées par des virgules.

L'ordre dans les objets n'est pas important. L'objet JSON suivant est donc équivalent à ce qui précède:

```
{ "key2": "value2", "key1": "value1", ... }
```

En résumé, voici un exemple d'objet JSON valide:

```
{
  "image": {
    "width":  800,
    "height": 600,
    "title":  "View from 15th Floor",
    "thumbnail": {
      "url": "http://www.example.com/image/481989943",
      "height": 125,
      "width":  100
```

```
    },
    "visible": true,
    "ids": [116, 943, 234, 38793]
  }
}
```

Exemples courants d'objets JSON, avec des contreparties d'objet (Java) associées

Tout au long de cet exemple, il est supposé que l'objet «root» en cours de sérialisation en JSON est une instance de la classe suivante:

```
public class MyJson {
}
```

Exemple 1: Un exemple d'instance de `MyJson` , tel `MyJson` :

```
{}
```

c'est-à-dire que la classe n'a pas de champs, seules les accolades sont sérialisées. **Les accolades sont les délimiteurs courants pour représenter un objet** . Notez également que l'objet racine n'est pas sérialisé en tant que paire clé-valeur. Cela est également vrai pour les types simples (String, numbers, arrays) lorsqu'ils ne sont pas des champs d'un objet (externe).

Exemple 2: Ajoutons des champs à `MyJson` et initialisons-les avec certaines valeurs:

```
// another class, useful to show how objects are serialized when inside other objects
public class MyOtherJson {}

// an enriched version of our test class
public class MyJson {
  String myString = "my string";
  int myInt = 5;
  double[] myArrayOfDoubles = new double[] { 3.14, 2.72 };
  MyOtherJson objectInObject = new MyOtherJson();
}
```

Ceci est la représentation JSON associée:

```
{
  "myString" : "my string",
  "myInt" : 5,
  "myArrayOfDoubles" : [ 3.14, 2.72 ],
  "objectInObject" : {}
}
```

Notez que tous les champs sont sérialisés dans une structure clé-valeur, où la clé est le nom du champ contenant la valeur. Les délimiteurs courants pour les tableaux sont des crochets. Notez également que chaque paire clé-valeur est suivie d'une virgule, à l'exception de la dernière paire.

Tableau JSON

Un tableau JSON est une collection ordonnée de valeurs. Il est entouré d'accolades carrées, c'est-à-dire `[]` , et les valeurs sont délimitées par des virgules:

```
{ "colors" : [ "red", "green", "blue" ] }
```

Les tableaux JSON peuvent également contenir tout élément JSON valide, y compris des objets, comme dans cet exemple de tableau comportant 2 objets (extrait du document RFC):

```
[
  {
     "precision": "zip",
     "Latitude":  37.7668,
     "Longitude": -122.3959,
     "Address":   "",
     "City":      "SAN FRANCISCO",
     "State":     "CA",
     "Zip":       "94107",
     "Country":   "US"
  },
  {
     "precision": "zip",
     "Latitude":  37.371991,
     "Longitude": -122.026020,
     "Address":   "",
     "City":      "SUNNYVALE",
     "State":     "CA",
     "Zip":       "94085",
     "Country":   "US"
  }
]
```

Ils peuvent également contenir des éléments de types mixtes, par exemple:

```
[
    "red",
    51,
    true,
    null,
    {
        "state": "complete"
    }
]
```

Une erreur courante lors de l'écriture de tableaux JSON (et d'objets) consiste à laisser une virgule à la fin du dernier élément. Ceci est un modèle courant dans de nombreuses langues, mais n'est malheureusement pas valide en JSON. Par exemple, le tableau suivant n'est pas valide:

```
[
    1,
    2,
]
```

Pour que cela soit valide, vous devez supprimer la virgule après le dernier élément, en le transformant en:

```
[
    1,
    2
]
```

Modification de JSON à la main

JSON est un format très strict (voir http://json.org) . Cela facilite l'analyse et l'écriture pour les machines, mais surprend les humains lorsqu'une erreur discrète casse le document.

Problèmes communs

Virgule

Contrairement à la plupart des langages de programmation, vous n'êtes pas autorisé à ajouter une virgule de fin:

```
{
  a: 1,
  b: 2,
  c: 3
}
```

L'ajout d'une virgule après `3` produira une erreur de synchronisation.

Le même problème existe pour les tableaux:

```
[
  1,
  2
]
```

Vous devez prendre des précautions supplémentaires si vous devez réorganiser les articles.

Virgule manquante

```
{
  a: 1,
  b: 2,
  c: 3
  d: 4
}
```

Comme les virgules ne sont pas autorisées, il est facile d'oublier d'en ajouter une avant d'ajouter une nouvelle valeur (dans ce cas, après `3`).

commentaires

JSON n'autorise pas les commentaires, car il s'agit d'un format d'échange de données. C'est toujours un sujet brûlant, mais sans réponses claires, sinon de ne pas les utiliser.

Il existe plusieurs solutions de contournement:

- Utilisez les commentaires de style C, puis supprimez-les avant de les transmettre à l'analyseur
- Intégrer des commentaires dans les données

```
{
  "//": "comment",
  "data": 1
}
```

- Intégrer des commentaires et les remplacer par des données

```
{
  "data": "comment",
  "data": 1
}
```

La deuxième entrée de `data` écrasera le commentaire *dans la plupart des analyseurs* .

Solutions

Pour faciliter l'écriture de JSON, utilisez un IDE qui vérifie les erreurs de syntaxe et fournit une coloration de la syntaxe. Les plugins sont disponibles pour la plupart des éditeurs.

Lorsque vous développez des applications et des outils, utilisez JSON en interne et en tant que protocole, mais essayez de ne pas l'exposer dans des endroits où un humain serait susceptible de le modifier manuellement (sauf pour le débogage).

Évaluez d'autres formats mieux adaptés à cette utilisation, tels que:

- Hjson , peut être converti de manière transparente depuis et vers JSON
- TOML , similaire aux fichiers INI
- YAML , plus de fonctionnalités mais au prix d'une complexité accrue

Justification pour Array vs Object (c.-à-d. Quand utiliser quoi)

Les tableaux JSON représentent une collection d'objets. Dans JS, il ya un tas de fonctions de collection telles que `slice` , `pop` , `push` . Les objets ont juste plus de données brutes.

Un **JSONArray** est une séquence *ordonnée* de *valeurs* . Sa forme textuelle externe est une chaîne entre crochets avec des virgules séparant les valeurs.

Un objet **JSONObject** est une collection *non ordonnée* de paires *nom / valeur* . Sa forme externe est une chaîne entre accolades avec les deux points entre les noms et les valeurs et les virgules

entre les valeurs et les noms.

Object - key et value, Array - chiffres, chaînes, booléens. Quand utilisez-vous ceci ou cela?

Vous pouvez considérer les tableaux comme "est un / un" et les objets comme "a un". Utilisons par exemple "Fruit". Chaque article dans la gamme de fruits est un type de fruit.

```
array fruit : [orange, mango, banana]
```

Les tableaux peuvent contenir des objets, des chaînes, des nombres, des tableaux, mais ne traitons que des objets et des tableaux.

```
array fruit : [orange:[], mango:{}, banana:{}]
```

. Vous pouvez voir que l'orange est aussi un tableau. Cela implique que tout élément qui devient int orange est un type d'orange, disons: bitter_orange, mandarin, sweet_orange.

pour les fruits, tout objet est un attribut du fruit. ainsi le fruit a un

```
object fruit :{seed:{}, endocarp:{},flesh:{}}
```

Cela implique également que tout élément de l'objet seed doit être la propriété de la graine, par exemple: color, ..

JSON est principalement un langage qui permet de sérialiser des objets javascript en chaînes. Donc, lors de la désérialisation d'une chaîne JSON, vous devriez obtenir une structure d'objet javascript. Si votre json se désérialise en un objet qui stocke 100 objets appelés object1 à object100, cela va être très gênant. La plupart des désérialiseurs s'attendent à ce que vous ayez des objets et des tableaux connus d'objets connus afin qu'ils puissent convertir les chaînes dans la structure réelle de l'objet dans le langage que vous utilisez. C'est aussi une question que la philosophie du design orienté objet vous répondrait.

crédits à tous les participants Quelles sont les différences entre l'utilisation de tableaux JSON et les objets JSON?

Lire Démarrer avec JSON en ligne: https://riptutorial.com/fr/json/topic/889/demarrer-avec-json

Chapitre 2: Analyse de la chaîne JSON

Examples

Analyse la chaîne JSON à l'aide de la bibliothèque com.google.gson en Java

`com.google.gson` bibliothèque `com.google.gson` doit être ajoutée pour utiliser ce code.

Voici l'exemple de chaîne:

```
String companyDetails = {"companyName":"abcd","address":"abcdefg"}
```

Les chaînes JSON peuvent être analysées en utilisant la syntaxe ci-dessous en Java:

```
JsonParser parser =  new JsonParser();
JsonElement jsonElement = parser.parse(companyDetails);
JsonObject jsonObj = jsonElement.getAsJsonObject();
String comapnyName = jsonObj.get("companyName").getAsString();
```

Parse chaîne JSON en JavaScript

En JavaScript, l'objet `JSON` est utilisé pour analyser une chaîne JSON. Cette méthode est uniquement disponible dans les navigateurs modernes (IE8 +, Firefox 3.5+, etc.).

Lorsqu'une chaîne JSON valide est analysée, le résultat est un objet JavaScript, un tableau ou une autre valeur.

```
JSON.parse('"bar of foo"')
// "bar of foo" (type string)
JSON.parse("true")
// true (type boolean)
JSON.parse("1")
// 1 (type number)
JSON.parse("[1,2,3]")
// [1, 2, 3] (type array)
JSON.parse('{"foo":"bar"}')
// {foo: "bar"} (type object)
JSON.parse("null")
// null (type object)
```

Les chaînes non valides lanceront une erreur JavaScript

```
JSON.parse('{foo:"bar"}')
// Uncaught SyntaxError: Unexpected token f in JSON at position 1
JSON.parse("[1,2,3,]")
// Uncaught SyntaxError: Unexpected token ] in JSON at position 7
JSON.parse("undefined")
// Uncaught SyntaxError: Unexpected token u in JSON at position 0
```

La méthode `JSON.parse` inclut une fonction `JSON.parse` facultative qui peut limiter ou modifier le résultat de l'analyse

```
JSON.parse("[1,2,3,4,5,6]", function(key, value) {
  return value > 3 ? '' : value;
})
// [1, 2, 3, "", "", ""]

var x = {};
JSON.parse('{"a":1,"b":2,"c":3,"d":4,"e":5,"f":6}', function(key, value) {
  if (value > 3) { x[key] = value; }
})
// x = {d: 4, e: 5, f: 6}
```

Dans le dernier exemple, `JSON.parse` renvoie une valeur `undefined` . Pour éviter cela, renvoyez la `value` dans la fonction de réanimation.

Parse JSON avec Groovy

Supposons que nous ayons les données JSON suivantes:

```
{
    "TESTS":
    [
         {
              "YEAR": "2017",
              "MONTH": "June",
              "DATE": "28"
         }
    ]
}
```

import groovy.json.JsonSlurper

classe JSONUtils {

```
private def data;
private def fileName = System.getProperty("jsonFileName")

public static void main(String[] args)
{
    JSONUtils jutils = new JSONUtils()
    def month = jutils.get("MONTH");
}
```

Voici l'analyseur:

```
private parseJSON(String fileName = "data.json")
{
    def jsonSlurper = new JsonSlurper()
    def reader

    if(this.fileName?.trim())
    {
```

```
        fileName = this.fileName
    }

    reader = new BufferedReader(new InputStreamReader(new FileInputStream(fileName),"UTF-8"));
    data = jsonSlurper.parse(reader);
    return data
}

def get(String item)
{
    def result = new ArrayList<String>();
    data = parseJSON()
    data.TESTS.each{result.add(it."${item}")}
    return  result
}
```

}

Lire Analyse de la chaîne JSON en ligne: https://riptutorial.com/fr/json/topic/3878/analyse-de-la-chaine-json

Chapitre 3: Stringify - Convertit JSON en chaîne

Paramètres

Param	Détails
Objet	(Object) L'objet JSON
Remplaçant	(Fonction | Tableau <chaîne | nombre> - optiotinal) filtre Fonction | Tableau
Espace	(Number | string - optionnel) Quantité d'espace blanc dans le JSON

Examples

Convertir un objet JSON simple en chaîne simple

```
var JSONObject = {
   stringProp: 'stringProp',
   booleanProp: false,
   intProp: 8
}

var JSONString = JSON.stringify(JSONObject);
console.log(JSONString);
/* output
 * {"stringProp":"stringProp","booleanProp":false,"intProp":8}
 */
```

Stringify avec filtre

```
var JSONObject = {
   stringProp: 'stringProp',
   booleanProp: false,
   intProp: 8
}

var JSONString = JSON.stringify(JSONObject, ['intProp']);
console.log(JSONString);
/* output
 * {"intProp":8}
 */
```

Stringify avec des espaces blancs

```
var JSONObject = {
   stringProp: 'stringProp',
```

```
   booleanProp: false,
   intProp: 8
}

var JSONString = JSON.stringify(JSONObject, null, 2);
console.log(JSONString);
/* output:
 *  {
 *    "stringProp": "stringProp",
 *    "booleanProp": false,
 *    "intProp": 8
 *   }
 */
```

Lire Stringify - Convertit JSON en chaîne en ligne:
https://riptutorial.com/fr/json/topic/7824/stringify---convertit-json-en-chaine

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec JSON	AndriuZ, Asaph, ccprog, Community, depperm, francesco foresti , gandreadis, Gavishiddappa Gadagi, itzmukeshy7, laktak, Mirec Miskuf, Nilanchala Panigrahy, pickypg, reidzeibel, user2314737, Vitor Baptista
2	Analyse de la chaîne JSON	Kruti Patel, Mahbub Rahman, Mottie, Vitor Baptista
3	Stringify - Convertit JSON en chaîne	Mosh Feu