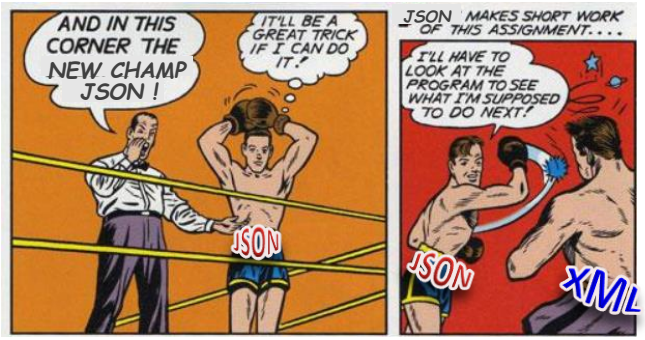
JSON

JSON

- JavaScript Object Notation
  - un format d'échange de données textuel "poids-léger" (lightweight)
    - plus léger que XML il est souvent plus rapide à lire et analyser (parse)
  - indépendant de tout langage de programmation
    - bien que basé sur un sous ensemble du langage JavaScript
  - auto-descriptif et facile à lire et comprendre

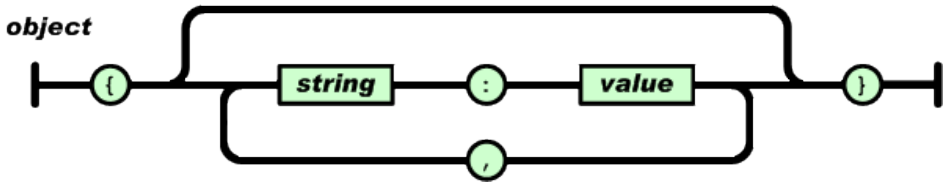
→ de plus en plus utilisé dans les API web où il a tendance à détrôner XML

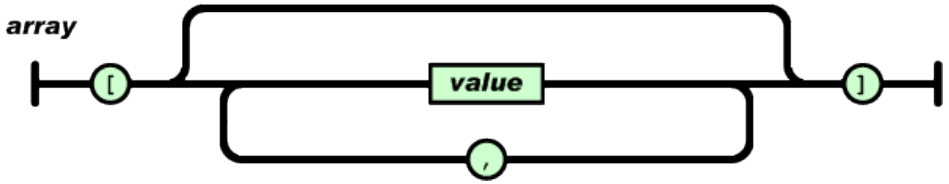
Douglas Crockford



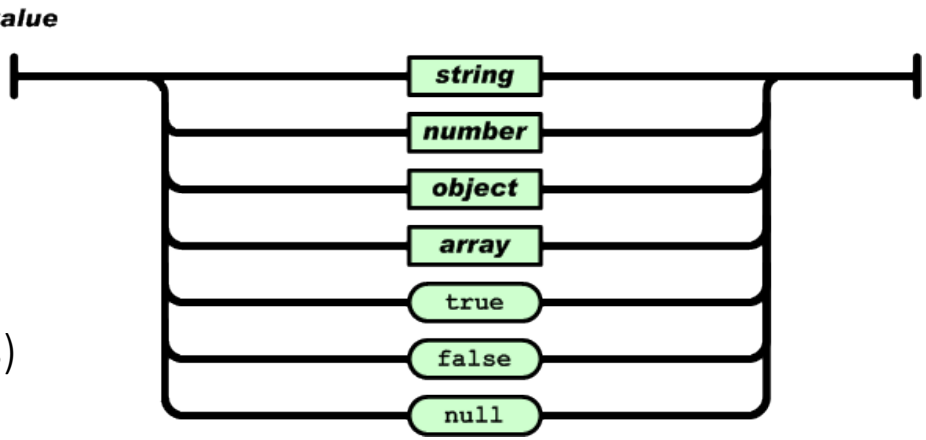
JSON

- Basé sur deux structures universelles quasiment présentes dans tous les langages de programmation
  - Collection de paires clé/valeur
    - selon les langages object, record, struct, dictionnaire, tableau associatif…
  - Liste ordonnées de valeurs
    - selon les langages tableau, vecteur, liste ou séquence…





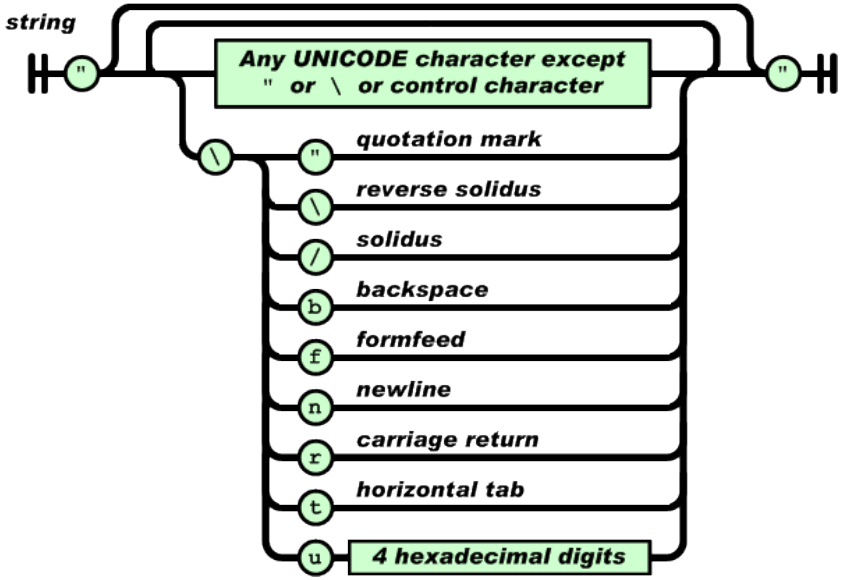
- Une valeur peut être :
  - une chaîne de caractères
  - un nombre
  - les valeurs `true`, `false` ou `null`
  - ou bien un objet ou tableau
    (ces structures peuvent être imbriquées)



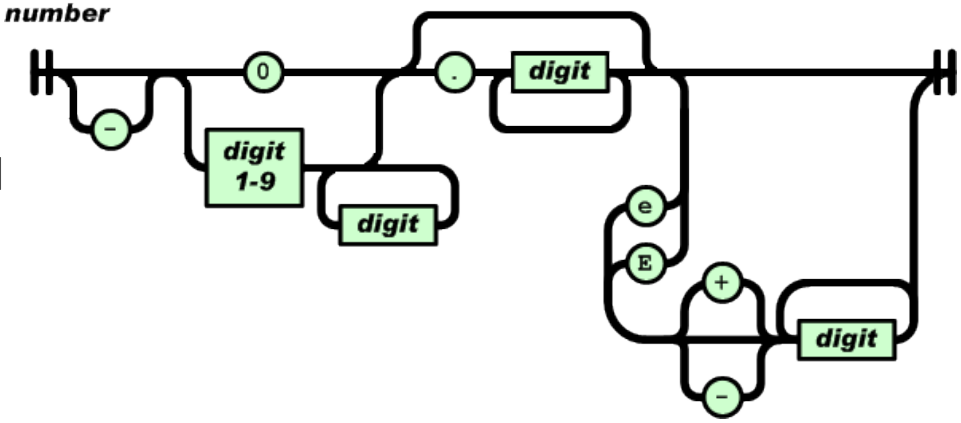
figures issues de http://www.json.org/

JSON

- **Chaines :** séquence de zéro ou plusieurs caractères Unicode entouré de double quotes (")
  - similaire aux String Java et C



- **Nombres:** similaires aux nombres en C et Java
  - sauf que les formats hexadécimal et octal ne sont pas utilisés)



JSON

Exemple

XML

```
<leschats>
      <chat nom = "Félicité">
         <age>10</age>
         <race>chat de gouttière<race>
         <aime>
             <item>manger du thon</item>
             <item>grimper aux arbres</item>
             <item>dormir</item>
         <aime>
         <poids>3.5</poids>
      </chat>
      <chat nom = "Felix">
         <age>6</age>
         <race>siamois<race>
         <aime>
             <item>se lécher</item>
             <item>manger des croquettes</item>
             <item>dormir</item>
         <aime>
         <poids>3.</poids>
      </chat>
</leschats>
```

moi j'aime
les croquettes

JSON

```
[
   {
      "nom": "Félicité",
      "age" : 10,
      "race" : "chat de gouttière",
      "aime" : [
                "manger du thon",
                "grimper aux arbres"
                "dormir"
               ],
      "poids": 3.5
   } ,
   {
      "nom": "Félix",
      "age" : 6,
      "race" : "siamois",
      "aime" : [
                "se lécher",
                "manger des croquettes",
                "dormir"
               ],
      "poids": 3.
   }
 ]
```

JSON

Echanges JSON sur HTTP

- JSON souvent utilisé comme format commun pour sérialiser et dé-sérialiser des données et les échanger entre des applications qui communiquent sur internet (en particulier via HTTP).

en tête HTTP pour indiquer que le contenu d'une requête ou d'une réponse sont des données JSON

JSON

convertir un objet JavaScript en JSON et inversement

- un usage très courant de JSON est la récupération de données depuis un serveur Web, et d'ensuite les convertir en un objet JavaScript pour les utiliser dans une page web

felecite

nom : "Félicité"
age : 10
race : "chat de gouttière"
aime : [
"manger du thon",
"grimper aux arbres"
"dormir"]

Objet JavaScript

var felicite = JSON.parse(data);

data = JSON.stringify(felicite);

data

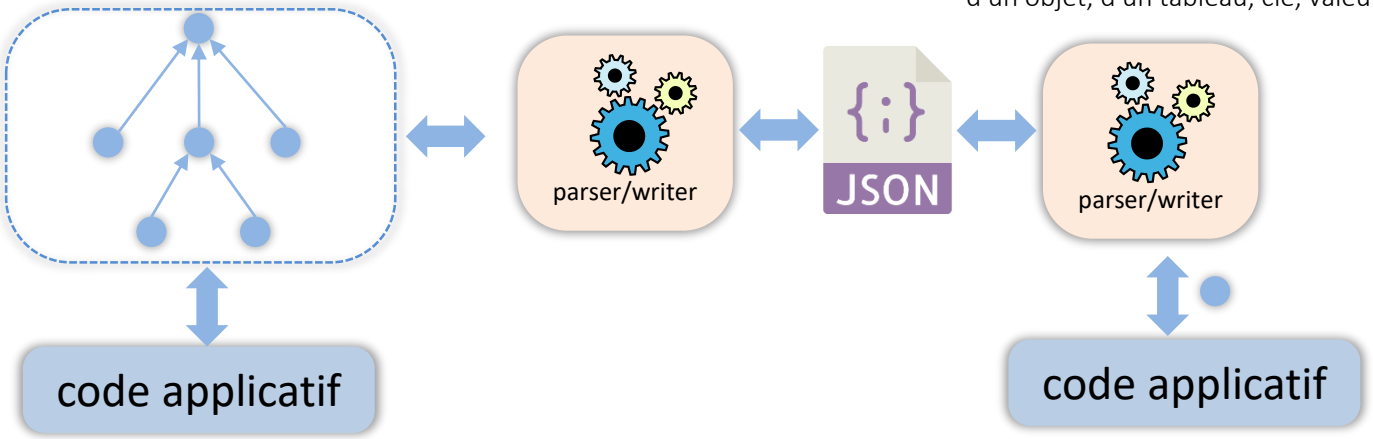
```
{
   "nom": "Félicité",
   "age" : 10,
   "race" : "chat de gouttière",
   "aime" : [
             "manger du thon",
             "grimper aux arbres"
             "dormir"
            ],
   "poids": 3.5
}
```

String (JSON) récupérée depuis un serveur

- deux modèles de programmation pour générer et parser des données JSON

Modèle Objet

un arbre d'objets en mémoire représente les données JSON



Modèle Streaming

les données JSON sont traitées à la volée élément par élément (début/fin d'un objet, d'un tableau, clé, valeur)

✓ flexibilité,
✓ possibilité de traitements nécessitant l'accès à l'ensemble de l'arbre

× occupe plus de mémoire
× plus lent

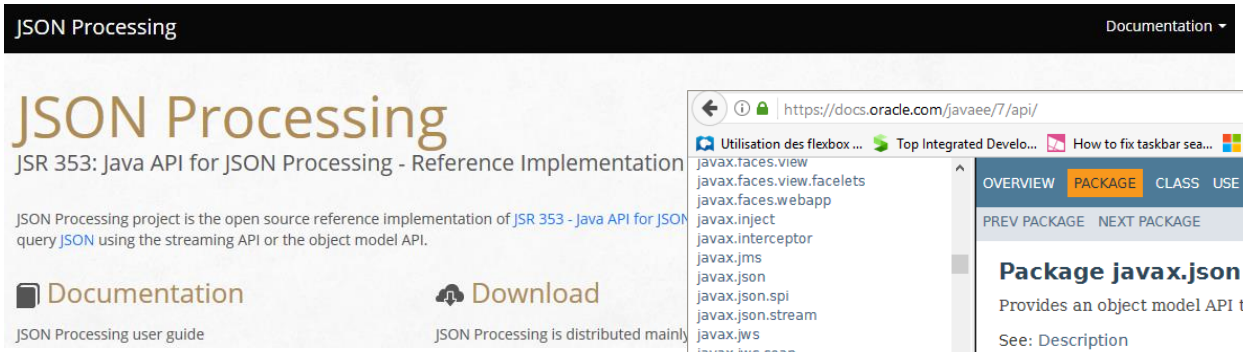
✓ rapidité
✓ faible empreinte mémoire

× rigidité
× ne permet que le traitement local d'un élément

JSON

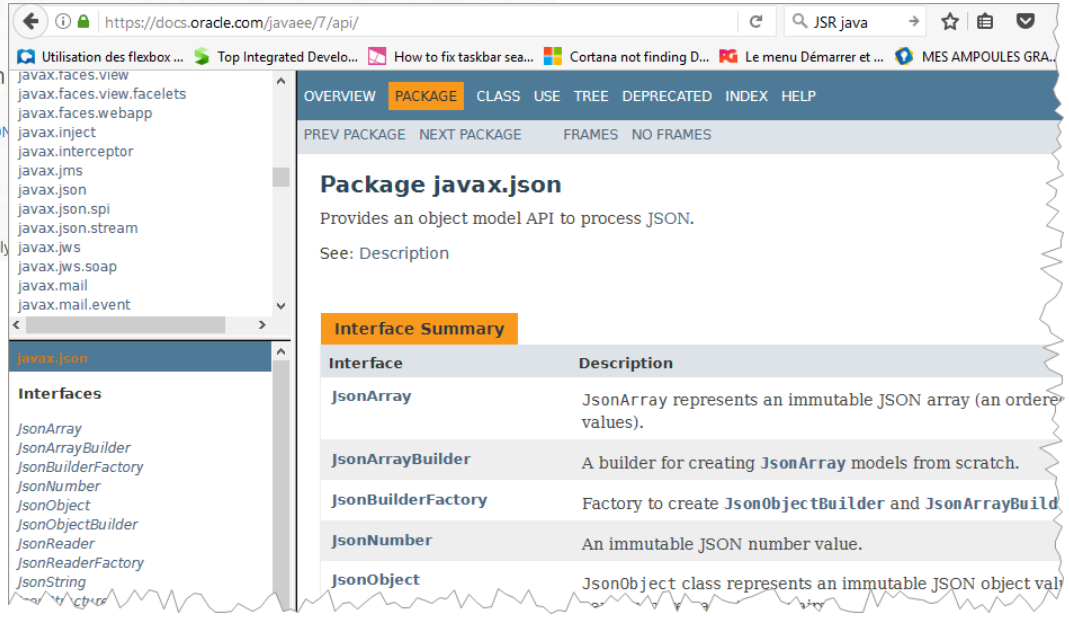
Traiter du JSON en Java

- nombreuses API
  - Gson (https://github.com/google/gson)
  - Jackson (https://github.com/FasterXML/jackson)
  - JSON Processing (https://jsonp.java.net/)
  - …



JSR: Java Specification Requests

intégré à la spécification JEE7

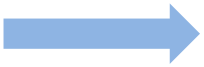


JSON

JSONP – streaming API javax.json.stream

- générer du JSON

```
FileWriter writer = new FileWriter("test.txt");
JsonGenerator gen = Json.createGenerator(writer);
gen.writeStartObject()
   .write("firstName", "Duke")
   .write("lastName", "Java")
   .write("age", 18)
   .write("streetAddress", "100 Internet Dr")
   .write("city", "JavaTown")
   .write("state", "JA")
   .write("postalCode", "12345")
   .writeStartArray("phoneNumbers")
      .writeStartObject()
         .write("type", "mobile")
         .write("number", "111-111-1111")
      .writeEnd()
      .writeStartObject()
         .write("type", "home")
         .write("number", "222-222-2222")
      .writeEnd()
   .writeEnd()
.writeEnd();
gen.close()
```



```
{
   "firstName": "Duke",
   "lastName": "Java",
   "age": 18,
   "streetAddress": "100 Internet Dr",
   "city": "JavaTown",
   "state": "JA",
   "postalCode": "12345",
   "phoneNumbers": [
      { "Mobile": "111-111-1111" },
      { "Home": "222-222-2222" }
   ]
}
```

JSON

JSONP – streaming API javax.json.stream

- lire du JSON

```
import javax.json.Json;
import javax.json.stream.JsonParser;
...
JsonParser parser = Json.createParser(new StringReader(jsonData));
while (parser.hasNext()) {
   JsonParser.Event event = parser.next();
   switch(event) {
      case START_ARRAY:
      case END_ARRAY:
      case START_OBJECT:
      case END_OBJECT:
      case VALUE_FALSE:
      case VALUE_NULL:
      case VALUE_TRUE:
         System.out.println(event.toString());
         break;
      case KEY_NAME:
         System.out.print(event.toString() + " " +
                          parser.getString() + " - ");
         break;
      case VALUE_STRING:
      case VALUE_NUMBER:
         System.out.println(event.toString() + " " +
                           parser.getString());
         break;
   }
}
```

```
{
   "firstName": "Duke",
   "lastName": "Java",
   "age": 18,
   "streetAddress": "100 Internet Dr",
   "city": "JavaTown",
   "state": "JA",
   "postalCode": "12345",
   "phoneNumbers": [
      { "Mobile": "111-111-1111" },
      { "Home": "222-222-2222" }
   ]
}
```

```
START_OBJECT
KEY_NAME firstName - VALUE_STRING Duke
KEY_NAME lastName - VALUE_STRING Java
KEY_NAME age - VALUE_NUMBER 18
KEY_NAME streetAddress - VALUE_STRING 100 Internet Dr
KEY_NAME city - VALUE_STRING JavaTown
KEY_NAME state - VALUE_STRING JA
KEY_NAME postalCode - VALUE_STRING 12345
KEY_NAME phoneNumbers - START_ARRAY
START_OBJECT
KEY_NAME type - VALUE_STRING mobile
KEY_NAME number - VALUE_STRING 111-111-1111
END_OBJECT
START_OBJECT
KEY_NAME type - VALUE_STRING home
KEY_NAME number - VALUE_STRING 222-222-2222
END_OBJECT
END_ARRAY
END_OBJECT
```

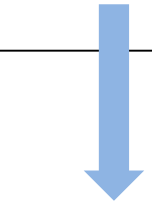
JSON

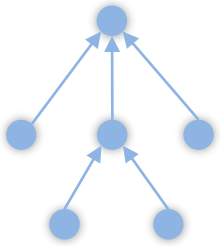
JSONP – object model API javax.json

- création d'un modèle objet à partir de données JSON

```
import java.io.FileReader;
import javax.json.Json;
import javax.json.JsonReader;
import javax.json.JsonStructure;
...
JsonReader reader = Json.createReader(new FileReader("jsondata.txt"));
JsonStructure jsonst = reader.read()
```

```
{
   "firstName": "Duke",
   "lastName": "Java",
   "age": 18,
   "streetAddress": "100 Internet Dr",
   "city": "JavaTown",
   "state": "JA",
   "postalCode": "12345",
   "phoneNumbers": [
      { "Mobile": "111-111-1111" },
      { "Home": "222-222-2222" }
   ]
}
```





- création d'un modèle objet par un programme Java

```
import javax.json.Json;
import javax.json.JsonObject;
...
JsonObject model = Json.createObjectBuilder()
   .add("firstName", "Duke")
   .add("lastName", "Java")
   .add("age", 18)
   .add("streetAddress", "100 Internet Dr")
   .add("city", "JavaTown")
   .add("state", "JA")
   .add("postalCode", "12345")
   .add("phoneNumbers", Json.createArrayBuilder()
      .add(Json.createObjectBuilder()
         .add("type", "mobile")
         .add("number", "111-111-1111"))
      .add(Json.createObjectBuilder()
         .add("type", "home")
         .add("number", "222-222-2222")))
   .build();
```

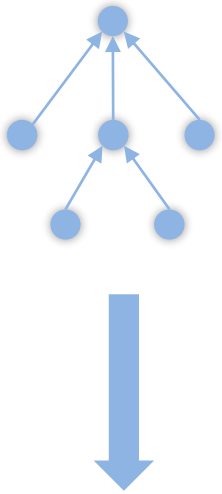
JSON

JSONP – object model API javax.json

- écrire un modèle objet JSON vers un Stream

```
import java.io.StringWriter;
import javax.json.JsonWriter;
...
StringWriter stWriter = new StringWriter();
JsonWriter jsonWriter = Json.createWriter(stWriter);
jsonWriter.writeObject(model);
jsonWriter.close();

String jsonData = stWriter.toString();
System.out.println(jsonData);
```



```
{
   "firstName": "Duke",
   "lastName": "Java",
   "age": 18,
   "streetAddress": "100 Internet Dr",
   "city": "JavaTown",
   "state": "JA",
   "postalCode": "12345",
   "phoneNumbers": [
      { "Mobile": "111-111-1111" },
      { "Home": "222-222-2222" }
   ]
}
```

JSONP

- To be done