Le langage Javascript (sérialisation en JSON)

HENALLUX

Technologies web

IG2 – 2015-2016

Au programme…

- **La syntaxe des objets en Javascript**
  - 3 syntaxes possibles
  - Les littéraux « objets »
- **La sérialisation**
  - Kézako ?
  - XML : un très bref aperçu
  - Sérialisation en JSON
- **Comment définir précisément une syntaxe ?**
  - Prototypes
  - Orienté objet en Javascript : le point (2)

3 syntaxes pour les objets

- Syntaxe des **tableaux associatifs**

```
var h = [];                 // tableau associatif vide
h["nom"] = "Simpson";
h["parle"] = function () { alert("Doh !"); };
```

- Syntaxe **"orienté objet"**

```
var h = {};                 // objet vide
h.nom = "Simpson";
h.parle = function () { alert("Doh !"); };
```

- Syntaxe **littérale** (littéraux de type "objet")

```
var h = {
   "nom" : "Simpson",
   "parle" : function () { alert("Doh !"); },
};
```

Littéraux de type « objets »

- Syntaxe **littérale** (littéraux de type "objet")

```
var h = {
   "prenom" : "Homer",
   "nom" : "Simpson",
   "parle" : function () { alert("Doh !"); },
   "toString" : function () {
                   return this.prenom + " " + this.nom;
                }
};
```

  - Paires au format `"clef" : valeur` séparées par des virgules
  - Si la clef n'est pas un mot réservé et ne comporte ni espace ni caractère non standard (ex : accentué) : guillemets optionnels.
  - Syntaxe à la base du format JSON (JavaScript Object Notation).

La sérialisation d'objets

- **Qu'est-ce que la sérialisation ?**
  - Littéraux de type « fonction »
- **Le standard XML**
  - Juste un aperçu : deux exemples
- **Le standard JSON**
  - Plus en détails

Ensuite : *Comment définir une syntaxe ?*

Sérialisation

Qu'est-ce que la sérialisation ?

- Quand un programme se termine, ses données disparaissent.
- Pour pouvoir les réutiliser plus tard, il faut les stocker quelque part (les rendre « **permanentes** »).

- Quelques cas simples…
  - Tableau de nombres → fichier séquentiel
  - Texte → fichier texte
  - Structures (en C) → fichier séquentiel
  - Mais quid des objets ?

- **Sérialiser** un objet, c'est le transformer en un format mieux adapté pour le stockage à long terme.

Sérialisation

Quelques possibilités pour sérialiser des objets…

- format binaire

  On recopie "bit à bit" l'endroit de la mémoire où l'objet est stocké.

  Avantage : facile

  Désavantage : peu/pas portable

- formats standards
  - **XML** (Extensible Markup Language)
    - = langage de balisage extensible
    - à la base du HTML
    - peut être adapté à toute une série de domaines
  - **JSON** (JavaScript Object Notation)

Sérialisation en XML

Exemple de document XML

- Balises (selon le domaine)
- Attributs
- Valeurs placées entre les balises
- Structure imbriquée

```
<?xml version="1.0" ?>

<endangered_species>
  <animal>
    <name language="English">Tiger</name>
    <name language="Latin">panthera tigris</name>
    <threat>poachers</threat>
    <weight>500 pounds</weight>
    <source sectionid="120" newspaperid="21"></source>
    <picture filename="tiger.jpg" x="200" y="197"/>
  </animal>
  <animal>
    …
  </animal>
</endangered_species>
```

Sérialisation en XML

Exemple de document XML (question à un Web Service)

```
<?xml version="1.0"?>

<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

- Structure définie dans des fichiers de définition
- Format standard utilisé dans de nombreux outils du web

Sérialisation en JSON

Autre standard qui gagne du terrain : JSON

- JavaScript Object Notation
- basé sur la syntaxe des littéraux de type "objet" en Javascript
- utilisable (et utilisé) pas seulement en Javascript

Exemple en JSON

```
{
  "prénom" : "Homer",
  "nom" : "Simpson",
  "adresse" : {
                "numéro" : 742,
                "rue" : "Evergreen Terrace"
              },
  "enfants" : [ "bart", "lisa", "maggie" ]
};
```

Objet : entre accolades, des paires `"clef" : info` séparés par des virgules

Tableaux : entre crochets, éléments séparés par des virgules

Sérialisation en JSON

Structure d'un élément au format JSON

- Objet = suite de paires clefs/valeurs entre `{ }`
  - Paires écrites au format `"clef" : valeur`
  - Paires séparées par des virgules
- Valeurs autorisées :
  - booléen (`true` ou `false`)
  - nombres
  - chaîne de caractères entre guillemets
  - `null`
  - un objet
  - un tableau, c'est-à-dire
    - une suite de valeurs entre `[ ]`
    - séparées par des `virgules`

Sérialisation en JSON

Utilisation de l'API liée à JSON

- Pour transformer un objet Javascript au format JSON :
  `JSON.stringify(obj)`
- Pour transformer une description JSON en un objet Javascript :
  `var obj = JSON.parse(chaine);`
- Alternative (chaîne JSON -> objet Javascript) dangereuse :
  `var obj = eval(chaine);`
  - dangereuse car aucune vérification de ce qui est évalué !
  - donc, à éviter !

Comment définir une syntaxe ?

- **Backus-Naur Form**
  - Les définitions de type « BNF »
- **Railroad diagrams**
  - Un dessin vaut mieux qu'un long discours ?

Méthode BNF

Comment définir précisément une syntaxe ?

Premier standard : Backus-Naur Form (BNF)
(il en existe plusieurs variantes)

```
<chiffre> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<préfixe> ::= 0<chiffre> | 0<chiffre><chiffre>
            | 0<chiffre><chiffre><chiffre>

<numTelSimple> ::=
  <chiffre><chiffre>.<chiffre><chiffre>.<chiffre><chiffre>

<numTel> ::= [ <prefixe> / ] <numTelSimple>
```

Méthode BNF

Quelques éléments du format BFN…

- `<chiffre>` symbole non-terminal (qui est défini)
  `3` symbole terminal (à reprendre tel quel)
  `::=` symbole de la définition
  `|` sépare les choix ("ou")

- `[ … ]` partie optionnelle
  `{ … }` pour regrouper un ensemble de symboles
  `…*` partie qui peut être répétée 0, 1 ou plusieurs fois
  `…+` partie qui peut être répétée 1 ou plusieurs fois

```
<préfixe> ::= 0<chiffre>[<chiffre>[<chiffre>]]
<paire> ::= <clef> : <valeur>
<objet> ::= { <paire> { , <paire> }* }
```

Méthode BNF

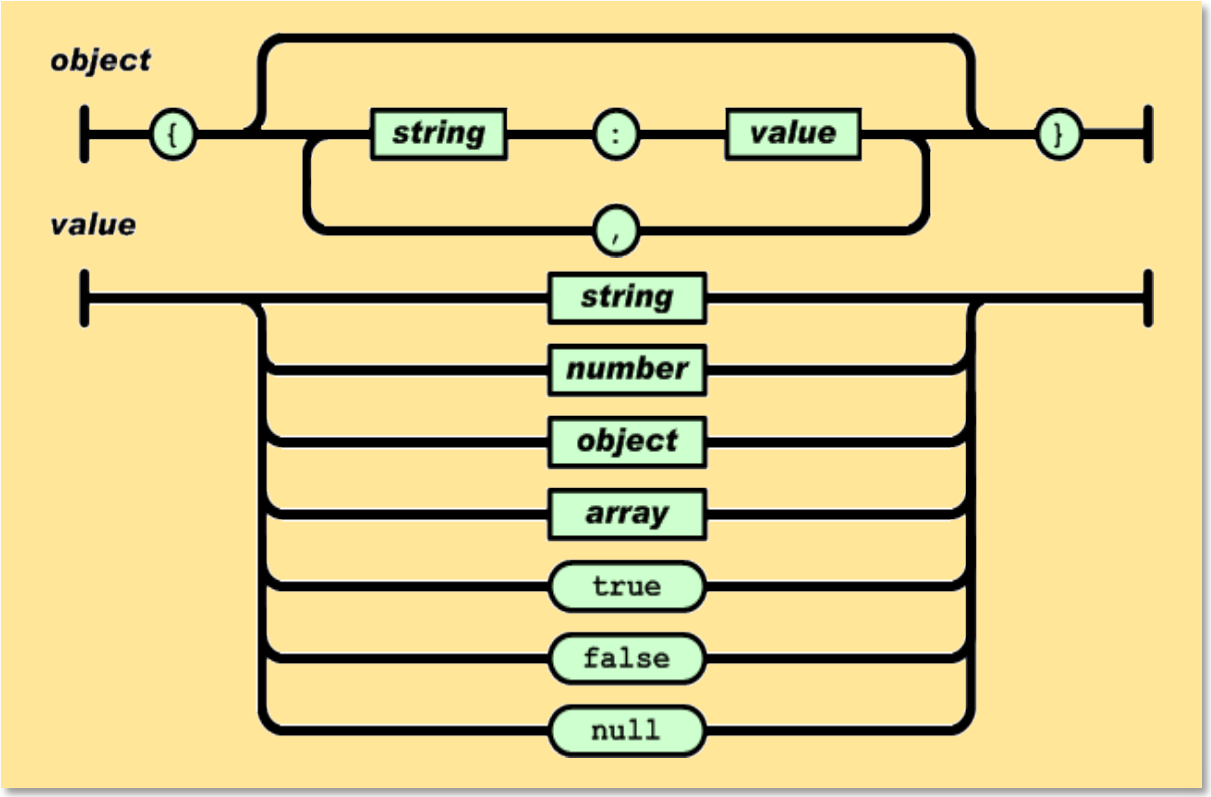
Définition de la syntaxe JSON en BNF

```
<objet> ::= { [ <paire> { , <paire> }* ] }
<paire> ::= <chaîne> : <valeur>
<tableau> ::= [ [ <valeur> { , <valeur> }* ] ]

<valeur> ::= <chaîne> | <nombre> | true | false | null
           | <tableau> | <objet>

<chaîne> ::= " <caractère>* "
<caractère> ::= <caractère Unicode standard>
              | \" | \\ | \n | \t | …
<nombre> ::= [-] <chiffre>+ [.<chiffre>+]
               [ {e|E} [+|-] <chiffre>+ ]
```

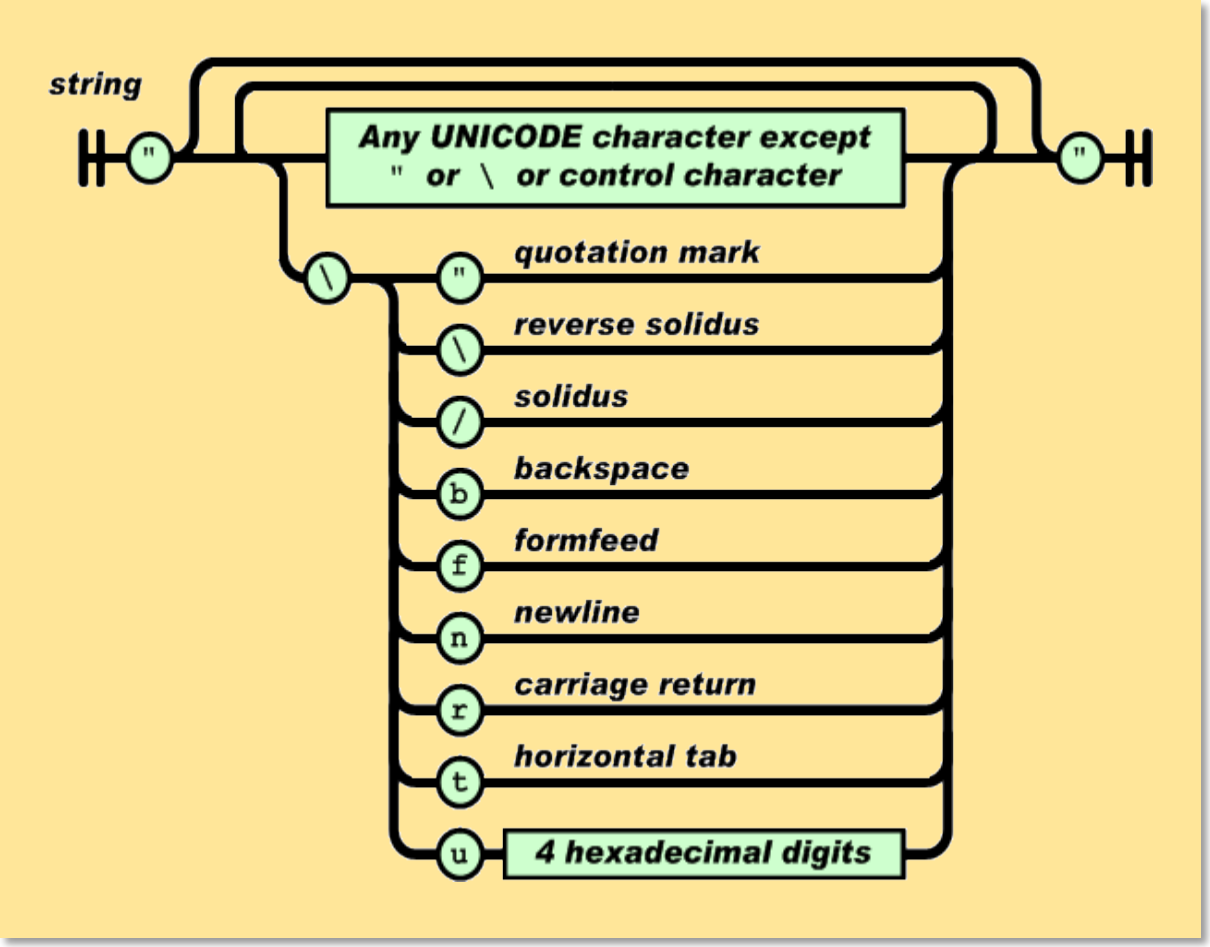
Railroad diagrams

Comment définir précisément une syntaxe ?

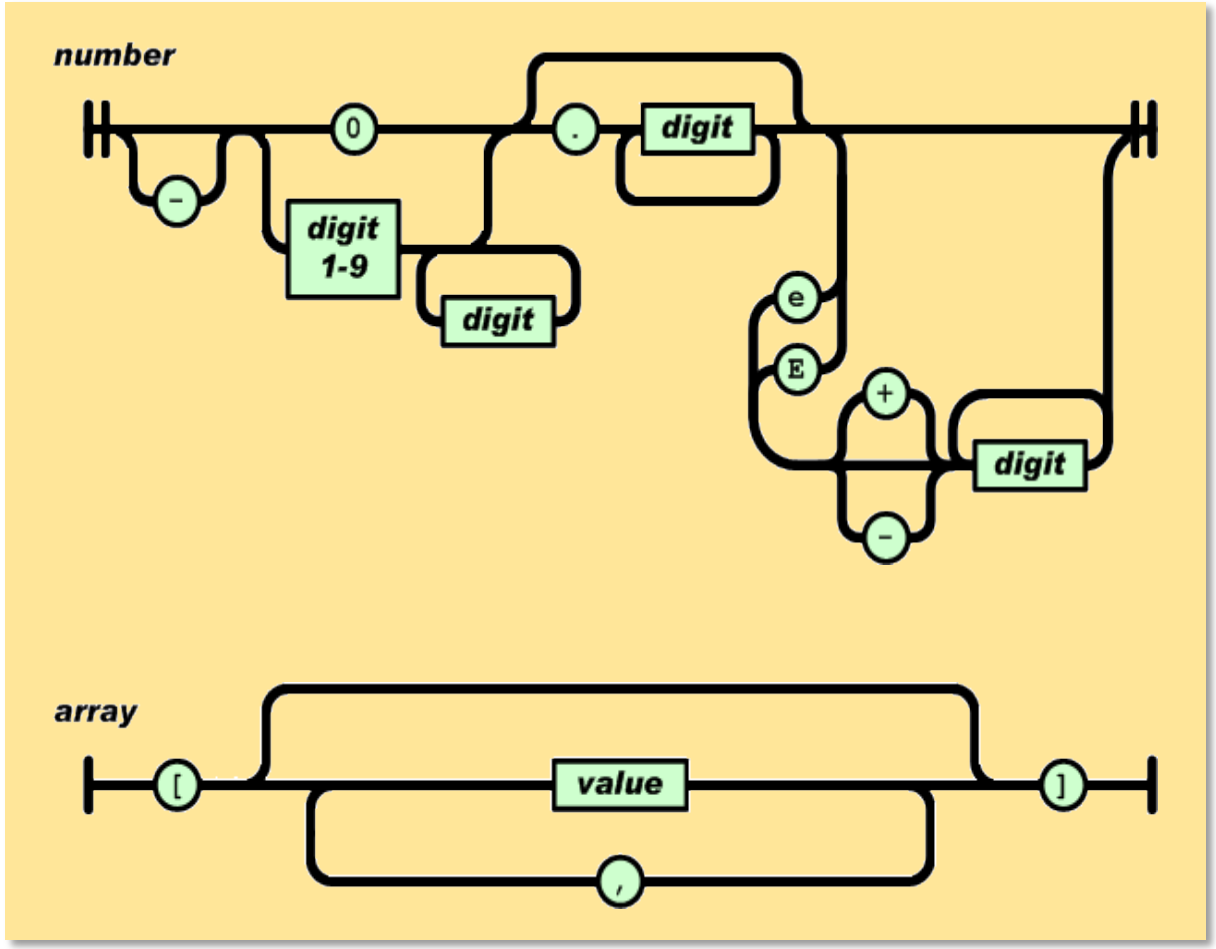
Second standard : diagramme syntaxique (source : json.org)



Railroad diagrams



Railroad diagrams



Comment définir une syntaxe ?

Railroad diagrams

- Autre exemple

```
<lettre> ::= a | b | … | z | A | B | … | Z
<chiffre> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<ident> ::= { <lettre> | $ | _ }
              { <lettre> | <chiffre> | $ | _ }*
```