Représentation des données

Olivier Flauzac & Cyril Rabat

Licence 3 Info - Info0503 - Introduction à la programmation client/serveur

2020-2021

Cours n°4

Représentation des données - JSON

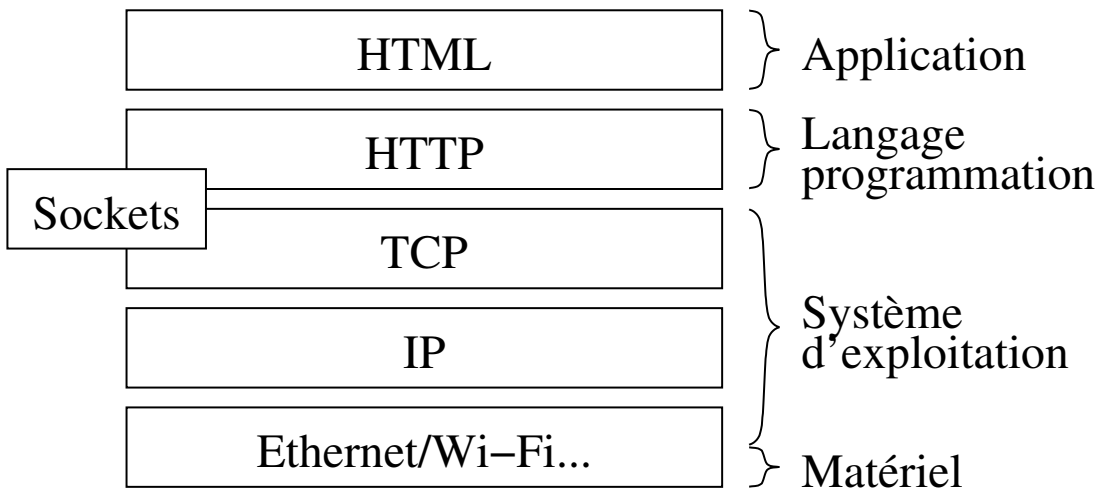
Version 20 septembre 2020

Table des matières

Communications dans les systèmes répartis

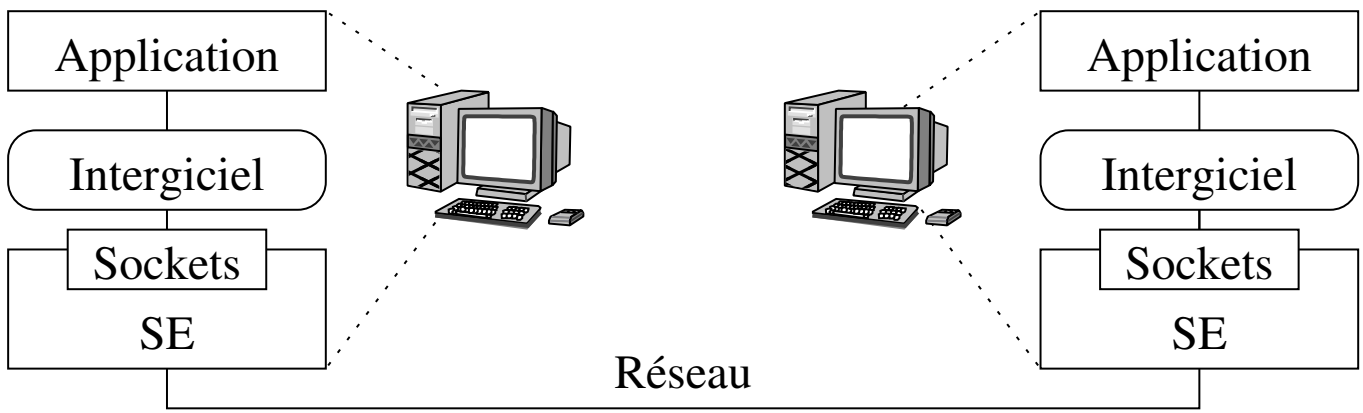
- Systèmes répartis = systèmes communicants
- Utilisation du réseau (réel ou virtuel) :
  - Protocoles réseaux sous-jacents (normalisés)
  - API fournies par le système d'exploitation
  - API fournies par le langage de programmation

Encapsulations du HTML



Les intergiciels

- Communications entre applications via le réseau :
  ↪ Utilisation des sockets (envois de messages)
- Utilisation d'un intergiciel pour envoyer/recevoir des données :
  - Files de messages (MOM)
  - Appels de méthodes distantes (RPC)
  - Objets distribués (*Java* RMI, CORBA)
  - Mémoire partagée
  - etc.



Intérêts et fonctionnement d'un intergiciel

- Fournit des primitives d'envoi et de réception :
  - De messages
  - D'objets
  - D'appels de méthodes...
- Suivant l'intergiciel / l'API utilisé :
  - Données envoyées recopiées depuis la mémoire :
    ↪ Exemple : avec une socket, recopie à l'aide de `write`
  - Envois formatés suivant l'intergiciel :
    ↪ Exemple : avec *Java* RMI, l'appel est traduit par la souche

Remarque

Moins l'intergiciel est développé, plus la charge laissée à l'utilisateur pour l'envoi et la réception est importante.

Représentation des données en mémoire

- Les données sont stockées en mémoire en chaque site
  ↪ Un site : une architecture matérielle, un système d'exploitation, etc.
- Leur représentation dépend :
  - Du type des données (entiers, caractères, réels, etc.)
  - De l'architecture de la machine
  - Du système d'exploitation
  - Du langage de programmation...
- Suivant l'intergiciel ou le langage utilisé :
  ↪ Plus ou moins de contrôle sur les données échangées

Remarque sur les sites d'un système réparti

- Ils n'appartiennent pas nécessairement à la même organisation
- N'utilisent pas nécessairement le même intergiciel

Architecture processeur

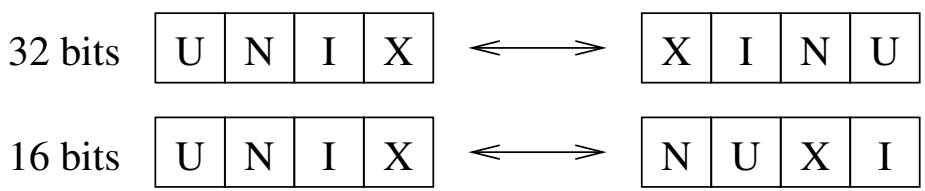
Le petit-boutiste et le grand-boutiste

- En anglais *little-endian* et *big-endian*
- Représentation des binaires dans un ordre différent
- Arrangements des groupes d'octets différents (mots de 16/32/64 bits)
- *Intel* x86 sont *little-endian*, *Motorola* sont *big-endian*
- *MacOS* est *big-endian*, *Windows* est *little-endian*

Exemples

Décimal	107							
Binaire	1101011							
little–endian	0	1	1	0	1	0	1	1
big–endian	1	1	0	1	0	1	1	0

Différence little-endian/big-endian



Différents arrangements d'octets

Les langages

- Suivant les langages, les types n'ont pas la même représentation
- Exemple de l'entier (type primitif `long`) :
  - En C : 4 octets en 32 bits, 8 octets en 64 bits
  - En *Java* : 8 octets
- Exemple de l'entier non signé :
  - En C : `unsigned int` (autant d'octets qu'un `int`)
  - En *Java* : n'existe pas !

Remarque

- Dans certains langages, le compilateur peut avoir une influence
  ↪ Exemple avec les structures en C

Envoi de données

Problématique

- Quand deux entités échangent des données :
  - Type des données envoyées ?
    ↪ Entier, caractère, chaîne de caractères...
  - Taille des données ?
  - Format des données ?
    ↪ Table des caractères

Particularités de la chaîne de caractères

- Une chaine possède une taille variable (ou non)
- Réception d'une chaîne depuis une socket :
  - Une ou plusieurs chaînes ?
  - Comment délimiter les deux chaînes ?

Les langages de description

- ASN.1 :
  ↪ Types primitifs prédéfinis, structures, représentations normalisées
- IDL (utilisé dans CORBA) :
  ↪ Description d'interfaces d'objets, types primitifs prédéfinis

Exemple d'IDL

```
module distributionCafe {
  exception debordementEau {};
  exception plusDEau {};
  interface ICafetiere {
    readonly attribute double quantiteEau;
    void remplirEau(in double quantiteEau) raises(debordementEau);
    boolean prendreCafe() raises(plusDEau);
  };
};
```

Le format TV

- Pour chaque donnée : Type + Valeur
- Problèmes :
  - Chaque lecture doit être analysée
  - Gestion des séparateurs
  - Types fixés

Exemple

```
int,12
str,Coucou tout le monde !
str,Bonjour\nComment vas-tu ?\n
```

Le format (Type) Valeur Séparateur (1/2)

- Utilisation d'un séparateur (une chaîne, un caractère)
- Déclinaisons :
  - Valeur séparateur
  - Type valeur séparateur
    - Typage par donnée
    - Typage par en-tête
- Exemple : le CSV (*Comma-Separated Value*)
  $\hookrightarrow$ Non typé
- Premiers pas vers la structuration des données
- Problèmes :
  - Types fixés
  - Exclusion du séparateur

Le format (Type) Valeur Séparateur (2/2)

Exemples

```
88;Clint;Eastwood
71;Arnold;Schwarzenegger

int,88;str,Clint;str,Eastwood
int,71;str,Arnold;str,Schwarzenegger

int;str;str
88;Clint;Eastwood
71;Arnold;Schwarzenegger
```

Le format Type, Longueur, Valeur - TLV (1/2)

- Définition de toutes les informations
- Analyse de tous les champs pour exploitation
- Séparateur inutile
- Limites :
  - Pas de structuration
  - Pas de vérification de structure
  - Problématique de l'envoi des données

Le format Type, Longueur, Valeur - TLV (2/2)

Exemples

```
int,1,34
str,14,Salut les amis
str,14,Bonjour\nà\ntous
```

```
int,1,34
str,14,Salut les amis
str,14,Bonjour
à
tous
```

Solutions hybrides

- Cas des réponses HTTP
- En-têtes multi-méthodes

Exemples

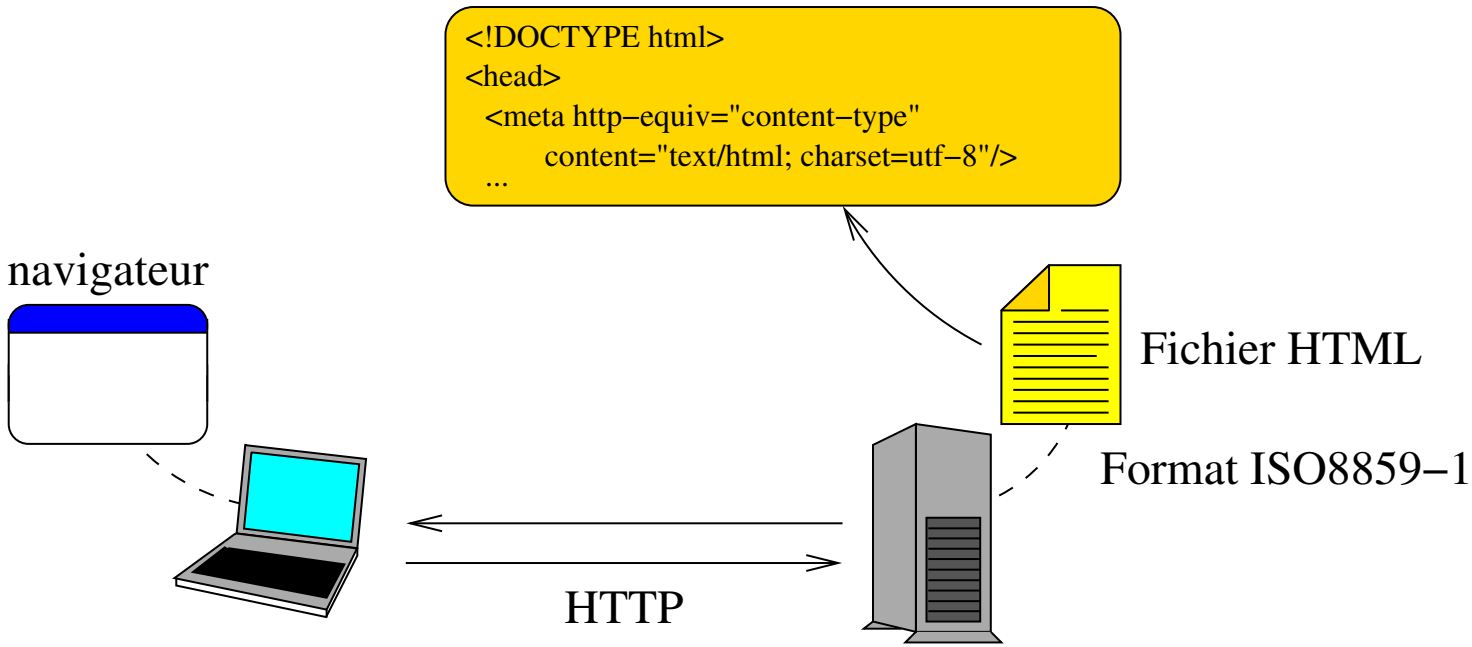
```
HTTP/1.1 200 OK
Date: Sat, 19 Sep 2020 18:05:15 GMT
Server: Apache
Last-Modified: Wed, 18 Sep 2020 06:15:32 GMT
Content-Length: 3447
Content-Type: text/html
Content-Language: fr

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
...
</body></html>
```

Problématiques liées aux caractères

- Caractères représentés suivant des tables de caractères :
  - ASCII : limitation à l'anglais (pas d'accent), sur un octet
  - UTF-8 : codage des caractères internationaux avec compatibilité ASCII
    ↪ représentation de 1 à 4 octets
  - UTF-16 : représentation par mots de 16 ou 32 bits
- Lors de l'envoi de caractères :
  ↪ Attention à la table de caractères utilisée !
- Pour les fichiers :
  ↪ Mêmes problèmes lors de la sauvegarde
  ↪ Idem pour les programmes (format du fichier source)

Problèmes fréquents



Autres exemples

- Encodage des champs dans une base de données
  ↪ Récupération des données : configuration du *driver* (exemple PDO)
- Normes pour certains langages ou protocoles (UTF-8 pour *Ajax*)

Exemple en PHP

Exemple de code

```
$toto1 = "Eleve";
$toto2 = "Élevé";
echo "Longueur de '$toto1' = " . strlen($toto1) . "<br/>";
echo "Longueur de '$toto2' = " . strlen($toto2);
```

Sortie (UTF-8)

```
Longueur de 'Eleve' = 5
Longueur de 'Élevé' = 7
```

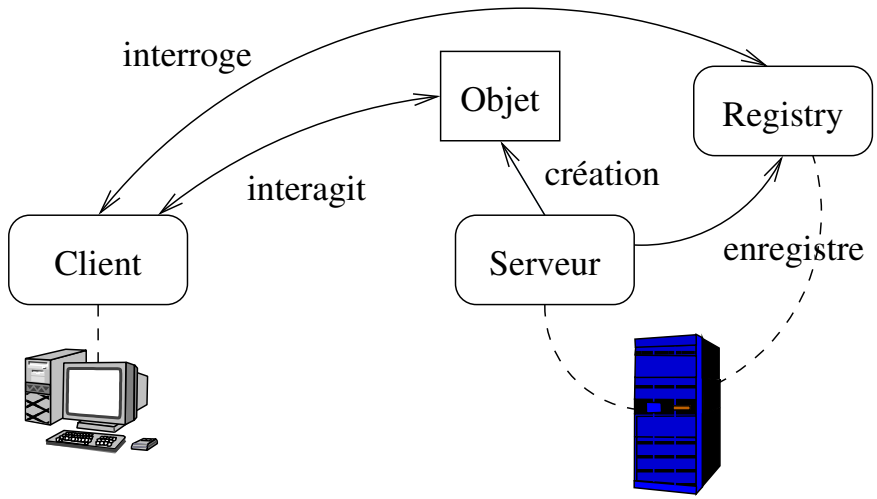
Sortie (ISO-8859-1)

```
Longueur de 'Eleve' = 5
Longueur de 'Ã?levÃ©' = 7
```

Attention

- Au format du fichier (lors de l'enregistrement)
- Au format spécifié dans l'en-tête HTML :
  `<meta http-equiv="content-type" content="text/html; charset=utf-8"/>`
- Possibilité en PHP : `strlen(utf8_decode($toto2))`
  ↪ Affichera bien 5

Problèmes de compatibilité : exemple avec *Java* RMI



- Communications = protocole RMI
  ↪ JRMP (spécifique *Java*), RMI-IIOP (interopérabilité avec CORBA)
- Interactions possibles via CORBA (langages objets)
- Interactions avec d'autres langages (non objets) ?

Problèmes de compatibilité : objets sérialisés en *Java*

- Sérialisation "automatique" via l'interface `Serializable`
  ↪ Stockage dans des fichiers, envoi via des flux
- Récupération de l'objet :
  ↪ Création de l'objet avec une référence de type `Object`
  ↪ Introspection et transtypage possibles
- Communications entre applications *Java* : OK
- Incompatibilité avec les autres langages :
  ↪ Pas de sérialisation normalisée
- Quid de la modification de la classe ?
  ↪ Utilisation de `serialVersionUID`
  ↪ Mise à jour à chaque modification de la classe

Présentation de JSON

- JSON pour *JavaScript Object Notation*
  ↪ Utilisé à la base pour les classes *JavaScript*
- Utilisation de la syntaxe *JavaScript* :
  ↪ Indépendant du langage !
  ↪ API existantes dans de nombreux langages
- Similarité avec XML :
  - Syntaxe permettant d'échanger ou stocker des données
  - Format texte
  - Lisible par un humain
  - Hiérarchisation des données

Avantages de JSON

- Vu comme une alternative à XML
- Plus léger (pas de balise fermante)
- Plus rapide à lire et écrire
- Peut utiliser des tableaux
- *Parsing* direct en *Javascript*

Inconvénients de JSON par rapport à XML

- Limitation du format de données
  ↪ Peu de types, types simplistes
- Pas d'utilisation de schéma :
  ↪ Validation impossible
  ↪ Vérification uniquement de la syntaxe
- Tous les outils XML non disponibles :
  - Langages de transformation (XSLT)
  - Recherche de données (X-Path)
  - etc.

Premier exemple

Exemple JSON

```
{"contacts":[
    { "nom" : "Schwarzenegger", "prenom" : "Arnold" },
    { "nom" : "Eastwood", "prenom" : "Clint" }
]}
```

Équivalent XML

```
<contacts>
  <contact>
    <nom>Schwarzenegger</nom><prenom>prenom</prenom>
  </contact>
  <contact>
    <nom>Eastwood</nom><prenom>Clint</prenom>
  </contact>
</contacts>
```

Syntaxe JSON

- Donnée : utilisation de couples clef/valeur
- Séparation des couples par des ','
- Utilisation des accolades pour les objets
- Utilisation des crochets pour les tableaux

Exemples de couples

```
"nom" : "Schwarzenegger"
"acteur" : {
    "prenom" : "Arnold",
    "nom" : "Schwarzenegger"
  }
"films" : [
    "Terminator 1", "Terminator 2", "Terminator 3"
  ]
```

Types de valeurs possibles

- Un nombre : entier ou réel
- Une chaîne de caractères : utilisation des '"' :
  ↪ Attention aux caractères spéciaux
- Un booléen : valeurs *true* ou *false*
- Un tableau : définition entre '[' et ']'
- Un objet : définition entre '{' et '}'

Les objets

- Définition entre '{' et '}'
- Correspond à un ensemble de pairs
- Chaque pair est séparée par une virgule

Exemple d'un objet `contact`

```
{ "nom" : "Schwarzenegger", "prenom" : "Arnold" }
```

Les tableaux

- Définition entre '[' et ']'
- Valeurs quelconques
- Séparation par des virgules
- Un document JSON peut être un tableau

Exemple d'un tableau d'objets

```
[
  {"nom" : "Schwarzenegger", "prenom" : "Arnold"},
  {"nom" : "Stalone", "prenom" : "Sylvester"},
  {"nom" : "Eastwood", "prenom" : "Clint"}
]
```

Échanges JSON "classiques" sur le Web

- **Serveur.** Utilisation d'un script PHP :
  - Récupération de données d'une base de données
  - Formatage en JSON
- **Client.** Côté *Javascript* :
  - Connexion HTTP et récupération du JSON via *AJaX*
  - Mise en forme des données et affichage

Exemples d'utilisation

- Création de listes déroulantes dynamiques
- Auto-complétion dans les formulaires
- Ajout de parties dynamiques

Générer du JSON en PHP

- Par défaut, PHP génère du HTML
- Nécessite de spécifier le type MIME :
  ↪ Avant l'écriture de données
  ↪ Utilisation de `header`
  ↪ Champ `Content-Type`
- La sortie est une chaîne de caractères contenant le JSON

Construction manuelle (non recommandé)

```php
const DONNEES = [
        [ 'id' => 1, 'nom' => 'Stalone', 'prenom' => 'Sylvester' ],
        [ 'id' => 2, 'nom' => 'Eastwood', 'prenom' => 'Clint' ]
    ];

$json = "[";
foreach(DONNEES as $contact) {
    if($json != "[")
      $json .= ", ";
    $json .= '{"id":"'.$contact['id'].'",'.
             '"prenom":"'.$contact['prenom'].'",'.
             '"nom":"'.$contact['nom'].'"}';
  }
$json .= ']';

header("Content-type: application/json; charset=UTF-8");
echo $json;
```

Possibilités de l'API

- Fonctions proposées dans PHP :
  - `json_encode` : transforme une valeur en JSON
    ↪ types primitifs, tableaux, classes
    Quelques options pour le deuxième paramètre :
    - `JSON_PRETTY_PRINT` : formate la chaîne de sortie
    - `JSON_FORCE_OBJECT` : force la création d'un objet
  - `json_decode` : transforme une chaîne JSON en valeur
    ↪ création d'un tableau associatif ou d'un objet
- D'autres fonctions pour gérer les erreurs générées pendant le codage/décodage :
  - `json_last_error` : dernière erreur
  - `json_last_error_msg` : dernier message d'erreur
  - Gestion des exceptions
- Interface `JsonSerializable` : permet de personnaliser la représentation JSON d'un objet
  ↪ Méthode `jsonSerialize` qui retourne la valeur à sérialiser

Exemple d'utilisation de `json_encode`

```php
const DONNEES = [
        [ 'id' => 1, 'nom' => 'Stalone', 'prenom' => 'Sylvester' ],
        [ 'id' => 2, 'nom' => 'Eastwood', 'prenom' => 'Clint' ]
    ];

header("Content-type: application/json; charset=UTF-8");
echo json_encode(DONNEES);
```

`json_encode` : exemples avec les tableaux

- `json_encode([1, 2])`
  `↪ [1, 2]`
  - Tableaux non associatifs transformés en tableaux JSON
- `json_encode([1, 2], JSON_FORCE_OBJECT)`
  `↪ {"0":1,"1":2}`
  - Possible de forcer la transformation en objet
- `json_encode(['a' => 1, 'b' => 2])` :
  `↪ {"a" : 1, "b" : 2}`
  - Tableaux associatifs transformés en objets

json_encode : exemples avec les objets

```php
class Personne {
    private string $nom, $prenom;

    public function __construct(string $nom, string $prenom) {
        $this->nom = $nom;
        $this->prenom = $prenom;
    }
    public function getNom() : string {
        return $this->nom;
    }
    public function getPrenom() : string {
        return $this->prenom;
    }
}
```

- `json_encode(new Personne('Rabat', 'Cyril'))`
  ↪ `{}`
- Une fois les attributs déclarés `public` :
  ↪ `{"nom":"Rabat","prenom":"Cyril"}`

`json_decode` : exemples

- `json_decode("[1, 2]")`
  ↪ `Array ( [0] => 1 [1] => 2 )`
  - Tableaux transformés en tableaux
- `json_decode('{"a":1,"b":2}')`
  ↪ `stdClass Object ( [a] => 1 [b] => 2 )`
  - Objets transformés par défaut en "objets standards"
    ↪ Attributs publiques, pas de constructeur, pas de *getters/setters*
- `json_decode('{"a":1,"b":2}', true)`
  ↪ `Array ( [a] => 1 [b] => 2 )`
  - Le deuxième paramètre permet de forcer la transformation en tableau associatif

json_decode : un peu plus loin

- Problème avec les propriétés des objets standards lors de l'utilisation de caractères spéciaux :

```
$json = '{"c-nul" : 1}';
$obj = json_decode($json);
echo $obj->{'c-nul'};
```

- Pour éviter que le décodage échoue :
  - Utilisation des guillemets doubles obligatoire
  - Nom entouré de guillemets doubles (contrairement à *Javascript*)
  - Virgule de fin non autorisée (contrairement à PHP)

Exemple d'utilisation de `JsonSerializable` (1/2)

```php
class Personne implements JsonSerializable {
    private string $nom, $prenom;

    public function __construct(string $nom, string $prenom) {
        $this->nom = $nom;
        $this->prenom = $prenom;
    }

    public function getNom() : string {
        return $this->nom;
    }

    public function getPrenom() : string {
        return $this->prenom;
    }

    public function jsonSerialize() : array {
        return ['nom' => $this->nom, 'prenom' => $this->prenom];
    }
}
```

Exemple d'utilisation de `JsonSerializable` (2/2)

```php
<?php
require_once("Personne.php");
$personne = new Personne("Schwarzenegger", "Arnold");
$json = json_encode($personne);

print_r($json);
```

Résultat obtenu

```
{"nom":"Schwarzenegger","prenom":"Arnold"}
```

Manipulation du JSON en *Java*

- Pas de support dans l'API standard (pour le moment ?)
- Nécessite l'utilisation d'une API externe
- Bibliothèque (simpliste) utilisée pour ce cours : `org.json`
  ↪ *Javadoc* :
  `http://stleary.github.io/JSON-java/index.html`
- Quelques objets :
  - `JSONObject` : objet JSON
    ↪ `getString(String)` : récupère une chaîne de caractères
    ↪ `getJSONObject(String)` : récupère un objet JSON
    ↪ `getInt(String)` : récupère un entier, etc.
  - `JSONArray` : tableau JSON
    ↪ `getJSONObject(String)`
    ↪ `getInt(String)`, `getString(String)`, etc.

Tableaux de valeurs

```
// Depuis un string
String entree = "{ \"valeurs\" : [ 1, 2, 3, 4] }";

JSONObject objet = new JSONObject(entree);
JSONArray tableau = objet.getJSONArray("valeurs");
for(int i = 0; i < tableau.length(); i++)
  System.out.print(tableau.getInt(i) + " ");

// Depuis un tableau
int t[] = {1, 2, 3, 4};

JSONObject objet = new JSONObject();
objet.put("valeurs", new JSONArray(t));
OutputStreamWriter output = new OutputStreamWriter(System.out);
try {
    objet.write(output);
    output.flush();
} catch(Exception e) {
    System.err.println("Erreur : " + e);
    System.exit(0);
}
```

Tableau d'objets

```
Personne p[] = {
    new Personne("John", "Smith", 30),
    new Personne("Cyril", "Rabat", 25)
  };
JSONObject objet = new JSONObject();
objet.put("valeurs", new JSONArray(p));

FileWriter fs = null;
try {
  fs = new FileWriter("fichier.json");
} catch(Exception e) {
  System.err.println("Erreur : " + e);
  System.exit(0);
}

try {
  objet.write(fs);
  fs.flush();
} catch(Exception e) {
  System.err.println("Erreur : " + e);
  System.exit(0);
}
```

Récupération depuis un URL (1/2)

```
// Création de l'URL
URL url = null;
try {
  url = new URL("http://localhost/JSON/index.php");
} catch(Exception e) {
  System.err.println("Erreur : " + e);
  System.exit(0);
}

// Récupération du JSON depuis l'URL spécifié
String json = "";
try {
  byte[] contenu = Files.readAllBytes(Paths.get(args[0]));
  json = new String(contenu);
} catch(IOException e) {
  System.err.println("Erreur : " + e);
  System.exit(0);
}
```

Récupération depuis un URL (2/2)

```
// Analyse du JSON reçu
JSONObject objet = new JSONObject(json);
JSONArray tableau = objet.getJSONArray("contacts");
for(int i = 0; i < tableau.length(); i++) {
  JSONObject element = tableau.getJSONObject(i);
  System.out.print("id=" + element.getString("id"));
  System.out.print("nom=" + element.getString("nom"));
  System.out.println("prenom=" + element.getString("prenom"));
}
```

Qu'est-ce que le parsing ?

- Rôle du parsing : parcourir les données d'un document
  ↪ Au format JSON, XML...
- Permet l'accès aux données par l'application
- parseur = lien entre les données et l'application :
  ↪ Permet la lecture des données contenues dans le fichier
- Importance du format des données :
  ↪ Complexité de l'écriture du parseur
  ↪ Nombre de traitements au niveau de l'application

Quelles règles à suivre ?

- Comme pour le XML, deux règles :
  ↪ Document bien formé
  ↪ Document valide
- Document bien formé : respect des règles d'écriture JSON
  ↪ Accolades, crochets, guillemets, clef-valeur
- Document valide : respect d'une structure donnée
  - Dépend de l'application
  - Nécessite de mettre en place des vérifications

Comment choisir une structure adaptée ?

- Prise en compte du traitement par l'application :
  ↪ Utiliser une structure qui ne soit pas trop complexe à parser
  ↪ Trop de couples clef/valeur = augmentation de la taille du message
- Identifier les données importantes :
  ↪ Chaque donnée nécessite un couple
- Quels seront les traitements sur les données ?
- Comment faire évoluer l'application ?

Choix de la représentation

Possibilité 1

```
{ "entrees" : [{
  "nom" : "Rabat",
  "prenom" : "Cyril",
  "adresse" : {
    "numeroRue" : 1,
    "nomRue" : "rue du Paradis",
    "codePostal" : 51100,
    "ville" : "Reims",
    "pays" : "France"
  }
 }]
}
```

Possibilité 2

```
{ "entrees" : [{
  "nom" : "Cyril Rabat",
  "adresse" : "1, rue du Paradis, 51100 Reims, France"
  }]
}
```

Problématique du choix de la représentation

Exemple des régions, départements et villes

```
{ "regions" : [{
    "id" : "GE",
    "nom" : "Grand-Est",
  }],
  "departements" : [{
    "id" : "_51",
    "nom" : "Marne",
    "numero" : 51,
    "region" : "GE"
  }],
  "villes" : [{
    "nom" : "Reims",
    "departement" : "_51"
  }]
}
```

Hiérarchisation : une solution

Exemple des régions, départements et villes

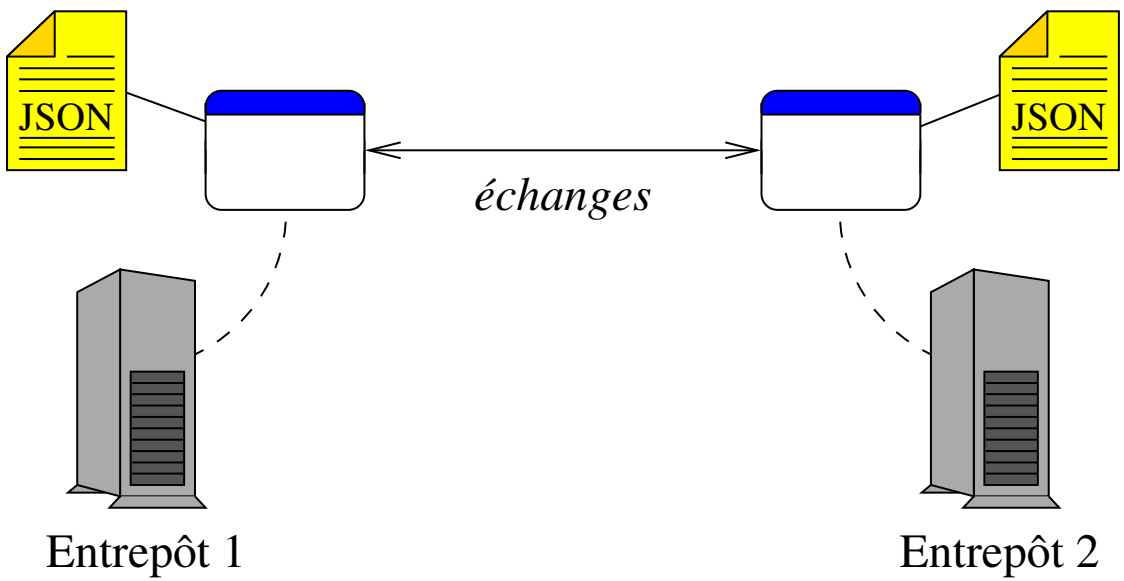
```
{ "regions" : [{
    "id" : "GE",
    "nom" : "Grand-Est",
    "departements" : [{
      "id" : "_51",
      "nom" : "Marne",
      "numero" : 51,
      "villes" : [{
        "nom" : "Reims"
      }]
    }]
  }]
}
```

Structuration en fichiers

- Avantage d'un seul fichier : lecture simplifiée
- Problèmes : la taille en mémoire, le temps de lecture...
- Solution : découper le fichier
  ↪ Construction d'une arborescence complète
  ↪ Choix des noms de fichier pour un accès rapide
- Cette réflexion dépend de la nature des données :
  - Objets légers mais en très grand nombre
    ↪ Fichiers contenant des parties des objets
  - Objets plus lourds
    ↪ Un fichier par objet
- Cas de l'historisation
  ↪ Les fichiers correspondent à des intervalles de temps

Étude de cas : gestion de conteneurs

- Application permettant de gérer des entrepôts de conteneurs
  ↪ Stockage dans des fichiers JSON
- Communication entre différents entrepôts :
  ↪ Échanges de conteneurs, demandes d'informations



Modélisation d'un conteneur

- Un conteneur possède :
  - Un numéro d'identification
  - Un poids
  - Un propriétaire (nom + adresse)
  - Une feuille de route
- Différents types :
  - Simple (classiques)
  - Réfrigérés (température, autonomie)
  - Liquide (volume)
  - Vrac

Question

- Fichiers nécessaires ?
- Donnez la représentation au format JSON d'un conteneur.

Modélisation du contenu d'un entrepôt

- Conteneurs positionnés dans un entrepôt
- Entrepôt = un nombre de rangées, de piles de conteneurs

Question

- Fichiers nécessaires ?
- Donnez la représentation au format JSON d'un entrepôt.

Modélisation des requêtes

- Une application = un entrepôt
  ↪ Réception de requêtes JSON + réponses JSON
- Différents types de requêtes :
  ↪ Ajout/suppression d'un conteneur
  ↪ Liste des conteneurs
  ↪ Transfert d'un conteneur
  ↪ Recherche d'un conteneur

Question

Donnez la représentation des requêtes au format JSON.