

JSON

formatos ligeros de intercambio de datos

INTRODUCCION- EL PROBLEMA

- * Los desarrolladores necesitan enviar y recibir datos de manera sencilla pero utilizando un formato común para estructuras complejas.
- * Se han desarrollado muchas soluciones ad-hoc donde se separan un conjunto de valores separados por comas, puntos y otros separadores pero de serialización y deserialización complicadas.
- * Hay que evitar tener que construir parsers cada vez que queremos intercambiar mensajes con el servidor.
- * Xml es opción valida pero no la mas adecuada por ser demasiado pesada.



INTRODUCCION - solución JSON

JSON(JavaScript Object Notation –Subconjunto ECMAScript)

- * Formato ligero de intercambio de datos independientes de cualquier lenguaje de programación.
- * Tiene forma de texto plano, de simple de lectura, escritura y generación.
- * Ocupa menos espacio que el formato XML
- * No es necesario que se construya parsers personalizados.



INTRODUCCION - JSON

- * JSON
 - * Independiente de un lenguaje específico
 - * Basado en texto
 - * De formato ligero
 - * Fácil de parsear
 - * NO Define funciones
 - * NO tiene estructuras invisibles
 - * NO tiene espacios de nombres (Namespaces)
 - * NO tiene validator
 - * NO es extensible
- * Su tipo MIME es → application/json



INTRODUCCION - JSON

- * Lenguajes que lo soportan:
 - * ActionScript
 - * c/c++
 - * .NET(C#,VB.NET...)
 - * Delphi
 - * Java
 - * JavaScript
 - * Perl
 - * PHP
 - * Python
 - * Ruby
 - * Etc...



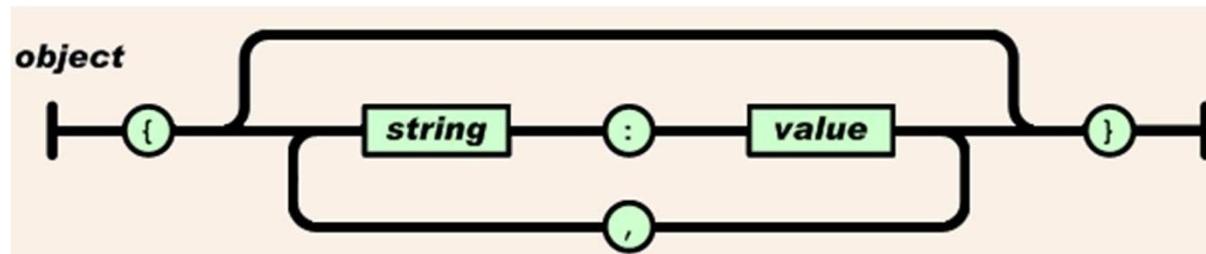
FORMAS DE REPRESENTACION

- * Sirve para representar objetos en el lado del cliente, normalmente en aplicaciones RIA (Rich Internet Application) que utilizan JavaScript.
- * Elementos:
 - * Object.- Conjunto desordenado de pares nombre/valor
 - * Array.- Colección ordenada de valores
 - * Value.- Puede ser un string, numero bool, objeto o array
 - * String.- Colección de cero o mas caracteres unicode
 - * Number.- Valor númerico sin comillas



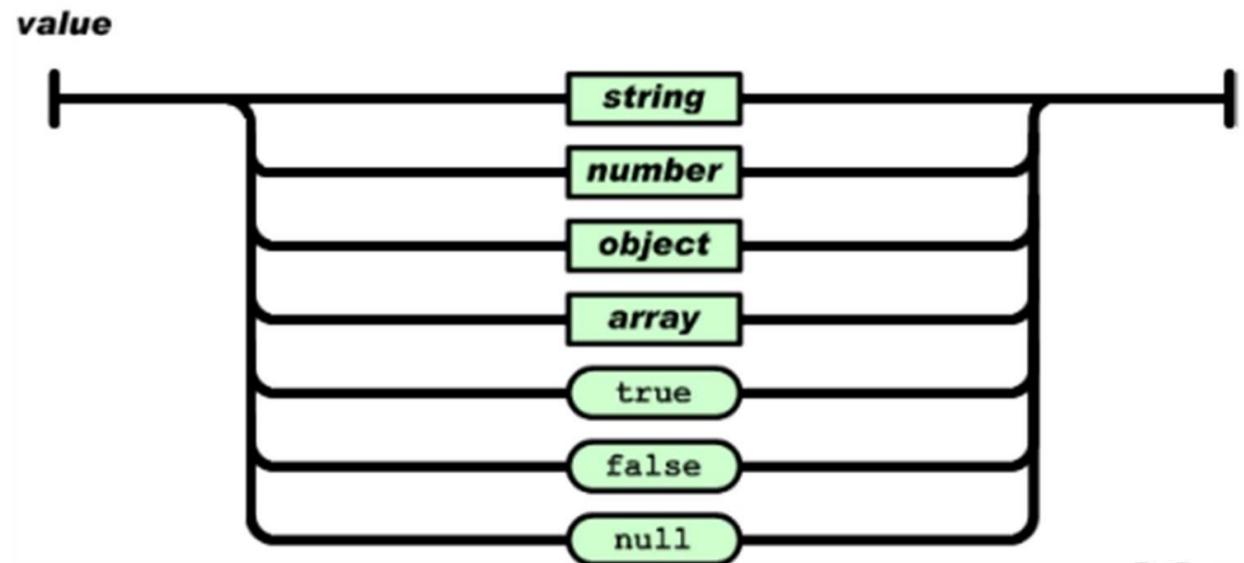
FORMA DE OBJETO/CLASE

- * Es un conjunto de propiedades, cada una con su valor
- * Notación
 - * Empieza con una llave de apertura {
 - * Termina con una llave de cierre }
 - * Sus propiedades
 - * Se separan con comas
 - * El nombre y el valor están separados por dos puntos:



FORMA DE VALUE

- * Puede ser
 - * Una cadena de caracteres con comillas dobles
 - * Un numero
 - * true, false, null
 - * Un objeto
 - * Un array



FORMA DE OBJETO/CLASE

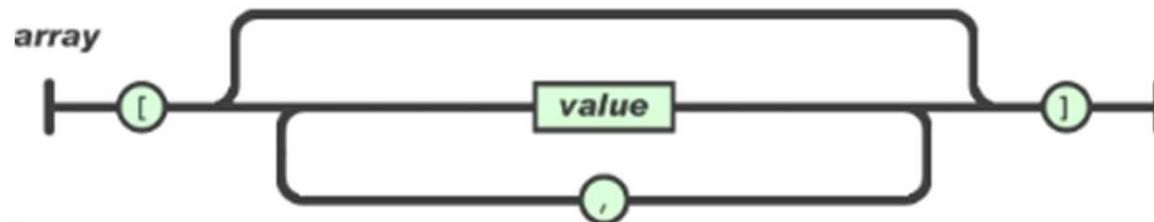
```
[  
  {  
    "country": "New Zealand",  
    "population": 3993817,  
    "animals": ["sheep", "kiwi"]  
  },  
  {  
    "country": "Singapore",  
    "population": 4353893,  
    "animals": ["merlion", "tiger"]  
  }]  
]
```

```
{  
  "Nombre": "Pedro",  
  "Contraseña": "Pedro123",  
  "Direccion": {  
    "Calle": "9 de octubre 235",  
    "Ciudad": "Quito"  
  },  
  "Pais": "Ecuador",  
  "Edad": 21,  
  "Hijos": ["Ivan", "Silvia"],  
  "esEmpleado": true  
}
```



FORMA DE ARRAY

- * Colección ordenada de valores u objetos
- * Notación
 - * Empieza con un corchete izquierdo [
 - * Termina con un corchete derecho]
 - * Los valores se separan con una coma ,



FORMA DE ARRAY

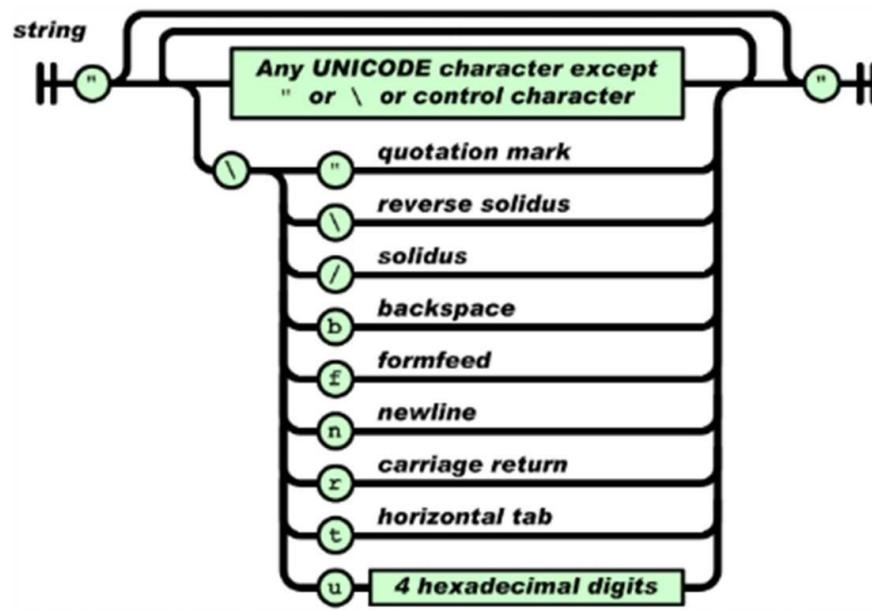
```
[  
{  
  "country": "New Zealand",  
  "population": 3993817,  
  "animals": ["sheep", "kiwi"]  
},  
{  
  "country": "Singapore",  
  "population": 4353893,  
  "animals": ["merlion", "tiger"]  
}  
]
```

```
["texto", 89, true, false, null, {}]
```



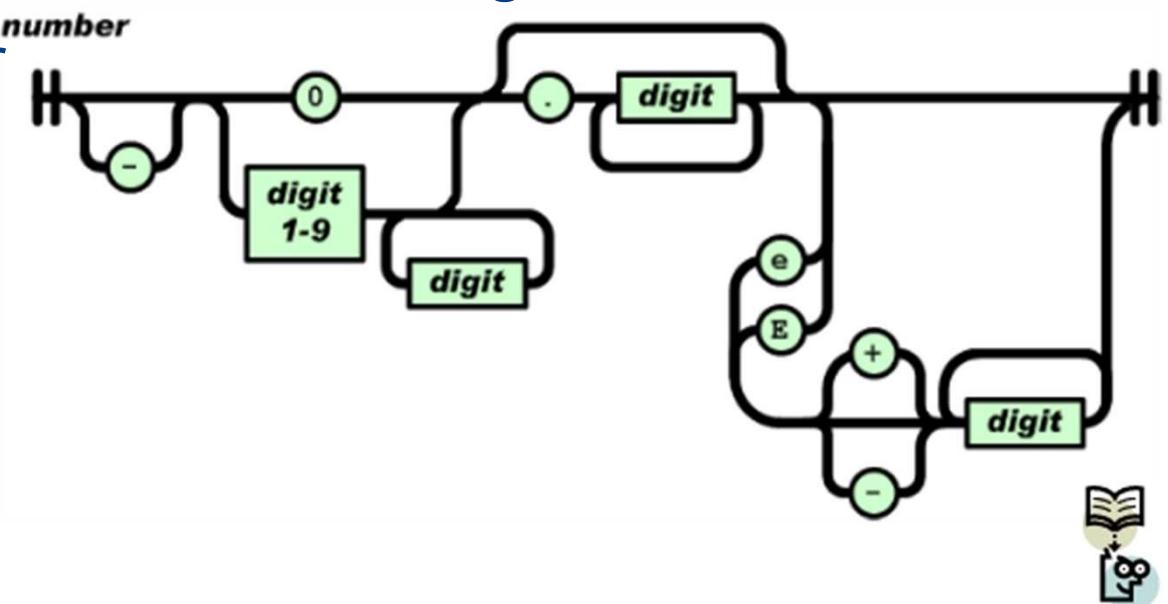
FORMA DE STRING

- * colección de cero a mas caracteres Unicode encerrados entre comillas dobles
- * Los caracteres de escape utilizan la barra invertida
- * Es parecida a una cadena de caracteres en C o Java



FORMA DE NUMBER

- * Similar a los numeros de C o Java
- * No usa formato octal o hexadecimal
- * No puede ser **Nan o Infinity**, en su lugar se usa **Null**
- * Puede representar
 - * Integer
 - * Real
 - * scientific



FORMA DE VALUE

```
[  
  {  
    "country": "New Zealand",  
    "population": 3993817,  
    "animals": ["sheep", "kiwi"]  
  } ,  
  {  
    "country": "Singapore",  
    "population": 4353893,  
    "animals": ["merlion", "tiger"]  
  }  
]
```

The diagram illustrates the structure of a JSON array. It shows two objects separated by a comma. The first object contains three properties: 'country' (String), 'population' (Number), and 'animals' (Array). The second object also contains three properties: 'country' (String), 'population' (Number), and 'animals' (Array). Lines connect the text labels to the corresponding blue boxes.

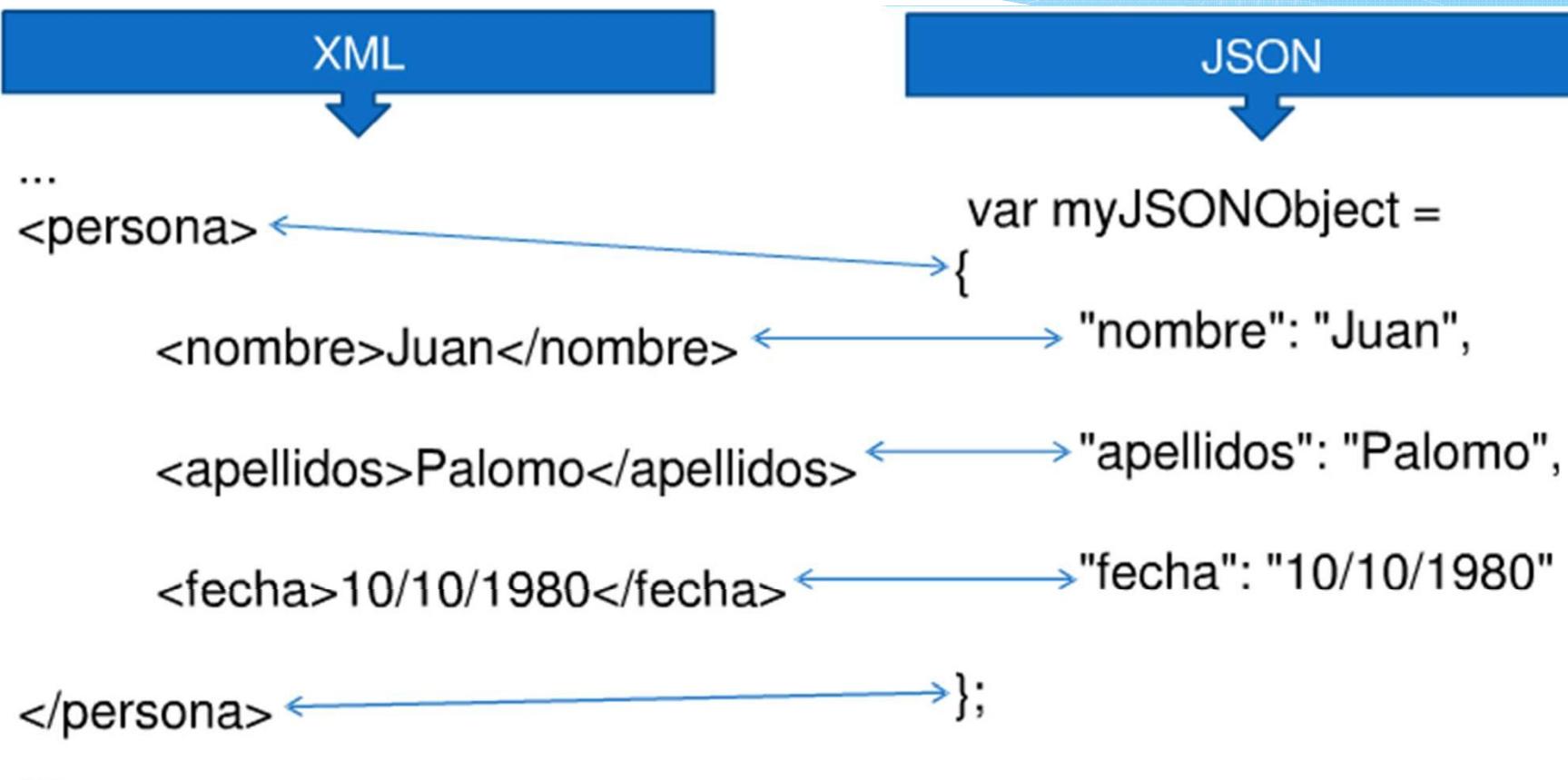


OTRAS FORMAS

- * BOOLEAN → true/false
- * null



JSON vs XML (CLASE)



JSON vs XML (Similitudes)

- * Ambos son legibles por los humanos
- * Tienen una sintaxis muy simple
- * Son jerárquicos
- * Son independientes del lenguaje de programación
- * Se pueden usar empleando Ajax



JSON vs XML (Diferencias)

- * Sintaxis dispar
- * JSON
 - * Es mas compacto
 - * Puede ser parseado usando el metodo eval() de JavaScript
 - * Puede incluir Arrays
 - * Los nombre de las propiedades no pueden ser palabras reservadas
- * XML
 - * Los nombres son mas extensos
 - * Puede ser validado bajo conjunto de reglas
- * JavaScript es normalmente utilizado en el lado del cliente



JSON vs XML (ARRAYS)

XML

```
...  
<listado> ←  
  <persona>  
    <nombre>Juan</nombre>  
    <apellidos>Palomo</apellidos>  
    <fecha>10/10/1980</fecha>  
  </persona>  
  <persona>  
    <nombre>Juan</nombre>  
    <apellidos>Palomo</apellidos>  
    <fecha>10/10/1980</fecha>  
  </persona>  
</listado> ← ]  
...  
...
```

JSON

```
...  
var myJSONObject = {"listado": [  
  {  
    "nombre": "Juan",  
    "apellidos": "Palomo",  
    "fecha": "10/10/1980"  
  },  
  {  
    "nombre": "Juan",  
    "apellidos": "Palomo",  
    "fecha": "10/10/1980"  
  }]  
};  
...  
...
```



EJEMPO DE XML => JSON

XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<poblaciones>
    <poblacion id="0">Alcobendas</poblacion>
    <poblacion id="1">Miraflores de la Sierra</poblacion>
    <poblacion id="2">San Fernando de Henares</poblacion>
</poblaciones>
```

Equivalente en JSON:

```
{"poblaciones": [
    {"poblacion": { "@id": "0", "#text": "Alcobendas" }},
    {"poblacion": { "@id": "1", "#text": "Miraflores de la Sierra" }},
    {"poblacion": { "@id": "2", "#text": "San Fernando de Henares" }}
]}
```

JSON

- * Librería para convertir objetos Java a JSON y vice-versa
 - * <http://sites.google.com/site/gson/Home>
 - * <http://code.google.com/p/google-gson>
- * Objetivos
 - * Proporcionar mecanismos sencillos para convertir los objetos
 - * Dar capacidad de utilizar representaciones personalizadas de objetos.



Acceso a campos JSON mediante JS

- En Javascript se accede a los valores de un objeto JSON mediante el operador punto (.) o empleando el símbolo de corchetes ([]).
- Para mostrar por consola los valores se emplea la sentencia **console.log()**

```
var myJson = { "name": "Chris", "age": "38", "parents": ["John", "Margaret"] };
```

```
var name1 = myJson["name"];
```

```
var name2 = myJson.name;
```

```
var parents1 = myJson["parents"];
```

```
var parents2 = myJson.parents;
```

```
console.log(name1);
```

```
console.log(parents2);
```

Definición variables en JS

- Para definir variables en Javascript se emplean las palabras reservadas **let** y **var**.

```
let i = 0;  
var objeto = { "direccion": {"calle": "Dr Clará", "numero": "22" } };  
let arr = [1, 2, "tres", [4, 5], {"seis": 6} ];  
  
console.log(objeto.direccion.calle); //Dr Clará  
console.log(arr[2]); //tres  
console.log(arr[4].seis); //6
```

Arrays en JS

- Los elementos del **array** están numerados comenzando desde cero.

```
let fruits = ["Apple", "Orange", "Plum"];
console.log( fruits[0] ); // Apple
```

- El número total de elementos en el **array** es su longitud **length**

```
let fruits = ["Apple", "Orange", "Plum"];
console.log( fruits.length ); // 3
```

- Para recorrer los elementos del **array** se puede emplear el bucle **for**.

```
let fruits = ["Apple", "Orange", "Plum"];
for (var i = 0; i < fruits.length; i++) {
    console.log( fruits[i] ); //Apple, Orange, Plum
}
```

Funciones en JS

- En Javascript las funciones se definen mediante la expresión **function**:

```
function calculaPrecioTotal(precio, porcentajeImpuestos) {  
    var gastosEnvio = 10;  
    var precioConImpuestos = (1 + porcentajeImpuestos/100) * precio;  
    var precioTotal = precioConImpuestos + gastosEnvio;  
    return precioTotal;  
}
```

```
var precioTotal = calculaPrecioTotal(23.34, 16);  
var otroPrecioTotal = calculaPrecioTotal(15.20, 4);
```

Funciones en JS

- Escribir el código de una función a la que se pasa como parámetro un número entero y devuelve como resultado una cadena de texto que indica si el número es par o impar. Mostrar por pantalla el resultado devuelto por la función:

```
function parImpar(numero){  
    if(numero % 2 == 0){  
        return "par";  
    }  
    else {  
        return "impar";  
    }  
}
```

```
var numero = prompt("Introduce un número entero");  
var resultado = parImpar(numero);  
console.log("El número "+numero+" es "+resultado);
```

FIN

