Lezione 3.3: XML e JSON

Università degli Studi di Milano

Programmazione Web e _Mobile

Programmable Web

- Simile al web
  - La più grande differenza è che usa documenti XML o simili (non HTML + banner + loghi)
  - Non è per forza orientato agli utenti umani
  - Può fornire input a software
- Si basa su HTTP e XML
  - Alternative a XML: HTML, JSON, plain text, binary
- Se non si usa HTTP non si può parlare di programmable web

Programmable Web: Building blocks

- HTTP è il punto comune di tutti
- Method information
  - Come il client comunica con il server i suoi bisogni
  - HTTP method vs application-specific method name (nella richiesta HTTP o nell'URI)
- Scoping information
  - Come il client dice al server su quale parte dei dati deve agire
  - Nel percorso dell'URI vs nell'entity-body

XML

Perchè XML

- 1960-1980 Infrastruttura Internet
- 1986 SGML (Standard Generalized Markup Language) per definire e rappresentare documenti strutturati
- 1991 Introduzione WWW e HTML
- 1991 Business adottano la tecnologia WWW; espansione nell'utilizzo Internet
- 1995 Nuovi tipi di business basati sulla connettività delle persone in tutto il mondo e sulla connettività delle applicazioni costruite attraverso il software di diversi provider (B2C, B2B)

Bisogno urgente di un nuovo e comune formato dei dati per internet

Perchè XML

- Necessità di regole semplici e comuni semplici da comprendere per persone con diversi background (come HTML)
- Capacità di descrivere risorse Internet e loro relazioni (come HTML)
- Capacità di definire struttura delle informazioni per diversi domini di business (*non come* HTML, come SGML)

Perchè XML

- Formato abbastanza formale per computer e abbastanza chiaro per essere leggibile da uomini (come SGML)
- Regole semplici per permettere uno sviluppo software semplice (non come SGML)
- Supporto per diversi linguaggi naturali (non come SGML)

Cos'è XML

- XML = Extensible Markup Language

A set of rules for defining and representing information as structured documents for applications on the Internet; a restricted form of SGML

T. Bray, J. Paoli, and C. M. Sperberg-McQueen (Eds.), Extensible Markup Language (XML) 1.0, W3C Recommendation 10- February-1998, http://www.w3.org/TR/1998/REC-xml-19980210/.

T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler (Eds.), Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000, http://www.w3.org/TR/2000/REC-xml-20001006/.

Cos'è XML

- EXtensible Markup Language
- XML è un linguaggio di marcatura (markup) come HTML
- XML è stato progettato descrivere i dati
- I tag di XML non sono predefiniti
- XML usa DTD e/o Schemi per descrivere i dati
- XML insieme ad un DTD o uno Schema è progettato per essere auto-descrittivo

Cos'è XML

- Regola 1: informazioni rappresentate in unità chiamate documento XML
- Regola 2: un documento XML contiene uno o più elementi
- Regola 3: un elemento ha un nome, è denotato nel documento con un markup esplicito contiene altri elementi e può essere associato con attributi
- E molte altre regole...

XML non fa niente!

- XML è progettato per non fare niente!
- XML è stato creato per strutturare, memorizzare e trasmettere dati
- Esempio

```
<nota>
        <a>Gianni</a>
        <da>Monica</da>
        <titolo>Promemoria</titolo>
        <corpo>Ricordati di me questo week end</corpo>
</nota>
```

XML non fa niente!

- La nota ha una intestazione e un corpo, ha anche un mittente ed un destinatario
- Però continua a non fare niente!
- Abbiamo solamente delle informazioni *intrappolate* tra dei tag
- Qualcuno dovrà scrivere del software per spedire e ricevere il messaggio della nota

XML è gratis ed estensibile

- XML non ha tag predefiniti, si inventano
- I tag di HTML sono predefiniti e si possono usare solo quelli per scrivere documenti
- XML permette di definire i tag che ritiene necessari e la struttura di documento adeguata

XML è complementare a HTML

- XML non sostituisce HTML
- La tendenza è quella di rappresentare i dati con XML e mostrarli con HTML
- Una buona definizione di XML è : "**XML è uno strumento per trasmettere informazioni, indipendente dalla piattaforma, dal software e dall'hardware**"

XML può separare i dati da HTML

- Con XML i dati vengono memorizzati separatamente dai documenti HTML
- Solitamente quando si visualizzano dei dati con HTML i dati sono all'interno del documento HTML stesso
- Con XML possono essere memorizzati in file separati

XML può separare i dati da HTML

- Questa divisione permette di potersi concentrare sulla visualizzazione con la sicurezza che delle modifiche ai dati non richiederanno modifiche al layout HTML

- È anche possibile inserire dati XML all'interno di pagine HTML, tenendoli isolati

XML per lo scambio di dati

- Con XML è possibile scambiare dati fra sistemi incompatibili
  - Encoding di dati
  - Encoding di protocolli
    - Definizione di funzioni
    - Marshalling di argomenti
- Nel mondo reale diversi sistemi e diversi database contengono dati in formato non uniforme
- E' incalcolabile il tempo speso dai programmatori per fare dialogare sistemi incompatibili

XML per la condivisione

- Con XML semplici file di testo possono essere usati per condividere dati
- I dati memorizzati con XML sono in formato testo
  - XML fornisce un modo indipendente da hardware e software di condividere dati
- Ciò rende semplice la creazione di documenti di cui hanno necessità applicazioni diverse

XML per la memorizzazione

- Con XML i dati sono disponibili a un maggior numero di utilizzatori
- Altri client ed applicazioni possono accedere ai file XML come se fossero sorgenti di dati (database)

La sintassi di XML

Sintassi

- Le regole sintattiche di XML sono nel contempo semplici e rigide
- Sono semplici da imparare e ancora di più da usare
- È quindi molto facile creare software che possa leggere e manipolare file XML

Un documento d'esempio

- La sintassi è semplice ed auto-descrittiva

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nota>
        <a>Gianni</a>
        <da>Monica</da>
        <titolo>Promemoria</titolo>
        <corpo>Ricordati di passare a prendermi domani</corpo>
</nota>
```

Un documento d'esempio

- La prima riga, la dichiarazione XML, definisce la versione di XML e il tipo di codifica dei caratteri utilizzati nel file

- In questo caso il documento è conforme alla versione 1.0 di XML ed è codificato con lo standard ISO-8859-1 (latin/western europe)
  - <?xml version="1.0" encoding="ISO-8859-1"?>

Un documento d'esempio

- La riga successiva descrive l'elemento radice del documento (*note*)
- Le quattro righe che seguono descrivono gli elementi figli (*a*, *da*, *intestazione* e *corpo*) dell'elemento radice
- L'ultima riga definisce la fine dell'elemento radice

Tutti i tag devono essere chiusi

- In XML è illegale omettere il tag di chiusura
- In HTML tale obbligo non vale per tutti i tag
- Nell'esempio precedente la dichiarazione XML non è chiusa, infatti, essa non fa parte del documento XML stesso

Nomi dei tag

- I nomi dei tag XML sono *case-sensitive*
- HTML non è *case-sensitive*
- In XML il tag <Lettera> è diverso dal tag <lettera>

Annidamento dei tag

- Significa che ogni tag, deve avere al suo interno
  - L'apertura e la chiusura di un altro tag
  - Nessun altro tag

- Esempio
  - testo vario corretto!
  - testo vario errato!

Nodo radice

- Tutti i documenti XML devono avere un nodo radice
- Tutti gli altri elementi devono essere compresi nell'elemento radice
- Tutti gli elementi possono avere dei nodi figli
- Tutti i nodi figli devono essere correttamente annidati nel tag genitore

Attributi

- I valori di tutti gli attributi devono essere tra doppi apici
- Gli elementi XML possono avere degli attributi che devono essere in coppia attributo-valore
- Esempio
  - <nota data="01/02/2003"> corretto!
  - <nota data=01/02/2003> errato!

Gli spazi

- In XML gli spazi bianchi non vengono troncati

- In HTML la frase

  Ciao, il mio nome è Claudio

  Verrebbe visualizzata con

  Ciao, il mio nome è Claudio

Il ritorno a capo

- In XML il CR/LF viene convertito in LF
  - In XML il ritorno a capo è sempre memorizzato con un LF
  - LF indica “l’andare a capo”

- Normalmente nei sistemi Windows viene usato il CR/LF, nei sistemi Unix/Linux viene usato LF e nei Macintoch il CR

Commenti in XML

- È possibile inserire righe di commento in documenti XML
- La sintassi di un commento è la seguente

<!-- Questo è un commento -->

XML è semplice

- XML non è niente di speciale
- XML è un documento di testo in cui ci sono dei tag completamente liberi
- Ogni software in grado di manipolare file di testo, è in grado di manipolare file XML

Elementi XML

- Gli elementi in un documento XML possono essere estesi per rappresentare più informazioni

- Riprendiamo l'esempio della nota ed immaginiamo di avere un'applicazione che produca un output come il seguente

MESSAGGIO

DA Monica A Gianni

Ricordati di passare a prendermi domani

Elementi XML

- Supponiamo ora che l'autore aggiunga un tag data per aumentare le informazioni

```
<nota>
<data>21-11-15</data>
<a>Gianni</a>
<da>Monica</da>
<titolo>Promemoria</titolo>
<corpo>Ricordati di passare a prendermi domani</corpo>
</nota>
```

Elementi XML

- L'applicazione andrà in errore ?

- No perché sarà comunque in grado di trovare i tag *a, da, corpo, intestazione e nota*
  - Produrrà il medesimo output

Relazioni tra elementi

- Gli elementi sono in relazione padre-figlio
  - L'elemento **radice** deve essere unico
  - Gli elementi che nascono da esso sono detti nodi **figli**
  - Nodi **figli** che hanno lo stesso nodo **padre** sono detti **fratelli** (sibling)

Contenuto degli elementi

- Un elemento XML può contenere
  - Altri elementi
  - Un contenuto semplice
  - Un contenuto misto
  - Nessun contenuto

Nomi degli elementi

- Il nome degli elementi deve seguire alcune semplici regole
  - Può contenere lettere, cifre ed altri caratteri
  - Non può iniziare con un numero o con un carattere di punteggiatura
  - Non può iniziale con le tre lettere XML
  - Non può contenere spazi

Nomi degli elementi

- Quando si inventano dei nomi, è bene seguire le seguenti regole dettate dal semplice buon senso
  - I nomi devono essere *descrittivi*
  - Cercare di evitare caratteri *ambigui* come '-' e '.' e ':'
  - I nomi non hanno lunghezza massima, ma non esagerare

Attributi XML

- Gli elementi XML possono avere degli attributi nel tag iniziale
- Forniscono maggiori informazioni sul tag
- Il valore dell'attributo deve essere compreso tra doppi apici oppure tra singoli apici
- Nel caso in cui nel valore ci siano dei doppi apici, allora il valore deve essere compreso tra singoli apici e viceversa

Elementi vs Attributi

- Si considerino i due brani di codice

```
<persona sesso="femmina">
<nome>Anna</nome>
<cognome>Rossi</cognome>
</persona>
```

```
<persona>
<sesso>femmina</sesso>
<nome>Anna</nome>
<cognome>Rossi</cognome>
</persona>
```

Elementi vs Attributi

- Non esistono regole che dicano quando usare i tag e quando usare gli attributi

- Tendenze
  - Evitare il più possibile di usare gli attributi
  - Usare gli attributi solo per delle istruzioni di controllo (simile ad HTML)

Problemi usando attributi

- Alcuni problemi che si possono incontrare usando gli attributi
  - Possono contenere un solo valore
  - Non sono estendibili in caso di future revisioni
  - Non possono descrivere strutture
  - Sono più difficili da manipolare da parte delle applicazioni

Validazione di documenti XML

Schema e DTD

- Sono documenti che definiscono la struttura di un documento XML
- Gli schemi sono a loro volta dei documenti XML
- I DTD hanno una sintassi un po' più complessa e sono obsoleti

XML Schema

- Standard per la validazione dei file XML
- Scritto in XML, definisce tipo e cardinalità degli elementi
- Basato sui concetti di tipi semplici e complessi

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://xml.netbeans.org/schema/travelXSD"
            xmlns:tns="http://xml.netbeans.org/schema/travelXSD"
            elementFormDefault="qualified">
    <xsd:complexType name="inputC">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="origin" type="xsd:string"/>
            <xsd:element name="destination" type="xsd:string"/>
            <xsd:element name="passengers" type="xsd:int"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="outputC">
        <xsd:sequence>
            <xsd:element name="ticket" type="xsd:string"/>
            <xsd:element name="voucher" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="faultC">
        <xsd:sequence>
            <xsd:element name="fault" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="input" type="tns:inputC"/>
    <xsd:element name="output" type="tns:outputC"/>
    <xsd:element name="fault" type="tns:faultC"/>
</xsd:schema>
```

Validazione

- Un documento XML sintatticamente corretto è detto ***ben formato***
- Un documento XML che rispetta uno schema (o un DTD) è detto ***valido***
- Un documento valido è anche ben formato, ma non è sempre vero il viceversa

XML 1.0 fundamentals

- Documento ben formato (sintassi)
  - Sintatticamente corretto
  - Tutti i tag di apertura e di chiusura corrispondono
  - I tag vuoti utilizzano una sintassi XML speciale
  - Tutti i valori degli attributi sono racchiusi tra virgolette
  - Tutte le entità sono dichiarate
- Valido (semantica)
  - Ben formato
  - Il documento rispetta le definizioni e la struttura proposte nello schema relativo

Visualizzazione di XML

- XML non ha informazioni di visualizzazione
- Per visualizzare file XML si usa una delle seguenti tecnologie
  - CSS (Cascading Style Sheets)
    - http://www.w3schools.com/xml/xml_display.asp
  - XSL (eXtensible Stylesheet Language)
    - http://www.w3schools.com/xsl/default.asp

Parser

- Per leggere, modificare o creare un documento XML è necessario un parser XML
- Il parser fornisce un modello che supporta
  - Javascript, VBScript, Perl, VB, Java, C++ ed altri
  - W3C XML 1.0 e XML DOM
  - DTD e validazione
- Esempio : per creare un oggetto di tipo documento XML con JavaScript si usa la seguente
  - var xmlDoc = new activeXobject(“Microsoft.XMLDOM”)

Esempio

- Note.xml
- CD.xml
- Plant_catalog.xml
- Simple.xml
- http://www.w3schools.com/xml/xml_examples.asp

XML e Javascript

XML Parser - Sincrono

- Tutti i browser moderni hanno integrato un XML parser
- Un XML parser converte un documento XML in un oggetto XML DOM, che può essere manipolato con il javascript

XML DOM

- XML DOM (Document Object Model) definisce un modo standard per accedere e manipolare documenti XML
- XML DOM rappresenta un documento XML come una struttura ad albero
- Tutti gli elementi possono essere acceduti attraverso l'albero DOM
  - Il loro contenuto (testo e attributi) possono essere modificati o cancellati, e nuovi elementi creati
  - Elementi, testo, attributi sono tutti rappresentati come nodi

XML Parser - Sincrono

- Codice per il parsing di una stringa XML il un oggetto DOM
  - txt="<bookstore><book>";
    txt=txt+"<title>Everyday Italian</title>";
    txt=txt+"<author>Giada De Laurentiis</author>";
    txt=txt+"<year>2005</year>";
    txt=txt+"</book></bookstore>";

    if (window.DOMParser)
      {
      parser=new DOMParser();
      xmlDoc=parser.parseFromString(txt,"text/xml");
      }
    else // Internet Explorer
      {
      xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
      xmlDoc.async=false;
      xmlDoc.loadXML(txt);
      }

Esempio

- Caricamento XML File
- Parsing di un documento XML contenente alcune note in un oggetto XML DOM
  - <note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
    </note>

Esempio 1

- xmldoc_string.html
- Definizione struttura pagina HTML

```
<div>
    <b>To:</b> <span id="to"></span><br>
    <b>From:</b> <span id="from"></span><br>
    <b>Message:</b> <span id="message"></span>
</div>
```

Esempio 2

- Creazione della stringa con il contenuto XML
  - txt="<note>";
  - txt=txt+"<to>Tove</to>";
  - txt=txt+"<from>Jani</from>";
  - txt=txt+"<heading>Reminder</heading>";
  - txt=txt+"<body>Don't forget me this weekend!</body>";
  - txt=txt+"</note>";

Esempio 2

- Creazione oggetto DOM da stringa

```
if (window.DOMParser) {
                parser=new DOMParser();
                 xmlDoc=parser.parseFromString(txt,"text/xml");
}
else { // Internet Explorer
                xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
                xmlDoc.async=false;
                xmlDoc.loadXML(txt);
}
```

Esempio 2

- Caricamento dati da stringa XML

  - document.getElementById("to").innerHTML=xmlDoc.getElementsByTagName("to")[0].childNodes[0].nodeValue;
    document.getElementById("from").innerHTML=xmlDoc.getElementsByTagName("from")[0].childNodes[0].nodeValue;
    document.getElementById("message").innerHTML=xmlDoc.getElementsByTagName("body")[0].childNodes[0].nodeValue;

JSON

Overview

- JSON o JavaScript Object Notation è un text-based open standard leggero disegnato per lo scambio di dati human-readable
- Convenzioni usate da JSON sono conosciute a programmatori C, C++, Java, Python, Perl etc.
  - Specificato da Douglas Crockford
  - Definito per scambio di dati human-readable
  - Esteso da JavaScript scripting language
  - Facile da analizzare e generare per le macchine
  - Estensione dei file .json
  - JSON Internet Media type è application/json

Utilizzo di JSON

- Usato quando si scrivono applicazioni basate su JavaScript (include estensioni al browser e siti web)
- Usato per serializzare e trasmettere dati strutturati su una connessione di rete
- Primariamente usata per trasmettere dati tra server e applicazioni web
- Web Service e API usano JSON per fornire dati pubblici
- Può essere usato con linguaggi moderni

Caratteristiche di JSON

- Facile da leggere e scrivere
- Formato di scambio basato su testo e molto leggero
- Indipendente dal linguaggio

Sintassi

- La sintassi JSON è considerata come un sottoinsieme della sintassi JavaScript
  - Dati rappresentati come coppie nome/valore
  - Parentesi graffe contengono oggetti e ogni nome è seguito dal : (colon), le coppie nome/valore sono separate da , (comma)
  - Parentesi quadre definiscono array e sono separate da , (comma)

Esempio

```
{
   "book": [
   {
     "id":"01",
     "language": "Java",
     "edition": "third",
     "author": "Herbert Schildt"
   },
   {
     "id":"07",
     "language": "C++",
     "edition": "second"
     "author": "E.Balagurusamy"
   }]
}
```

Sintassi

- JSON supporta due strutture dati
  - Insieme di coppie nome/valore
    - Struttura dati supportata da diversi linguaggi di programmazione
  - Lista ordinate di valori
    - Array, liste, vettori, sequenze…

Datatype

Tipo	Descrizione
Number	Formato Javascript floating-point double-precision
String	Unicode double-quoted con backslash escaping
Boolean	True o false
Array	Una sequenza ordinata di valori
Value	String, number, true o false, null…
Object	Una collezione non ordinate di coppie nome:valore
Whitespace	Usato tra ogni coppia di token
null	Vuoto

Number

- È floating-point double-precision come in JavaScript e dipende dall'implementazione
- Formati ottale ed esadecimale non usati
- Nessun supporto per NaN o Infinity in Number

Tipo	Descrizione
Integer	Digits 1-9, 0 e positive o negative
Fraction	Frazioni: .3, .9
Exponent	Esponenti: e, e+, e-,E, E+, E-

- Sintassi
  - var json-object-name = { string : number_value, ...}
- Esempio
  - var obj = {"marks": 97}

String

- Sequenza di zero o più caratteri Unicode double quoted con backslash escaping
- Character è una stringa di lunghezza 1

Tipo	Descrizione
"	double quotation
\	reverse solidus
/	solidus
b	backspace
f	form feed
n	new line
r	carriage return
t	horizontal tab
u	four hexadecimal digits

String

- Sintassi
  - var json-object-name = { string : "string value", ...}
- Esempio
  - var obj = {"name": "Claudio"}

Boolean

- True o false
- Sintassi
  - var json-object-name = { string : true/false, ...}
- Esempio
  - var obj = {"name": "Claudio", "marks": 97, "distinction": true}

Array

- Collezione ordinate di valori
- Inclusi in parentesi quadre
  - Iniziano con '[' e finiscono con ']'
- Valori separati da , (comma)
- Indice dell'array parte da 0 o 1
- Array dovrebbero essere usati quando i nomi delle chiavi sono interi sequenziali

Array

- Sintassi
  - [value, ...]

- Example:
  - {

```
"books": [
 { "language":"Java" , "edition":"second" },
 { "language":"C++" , “edition":"fifth" },
 { "language":"C" , “edition":"third" }
]
```

}

Object

- Insieme non ordinate di coppie nome/valore
- Object sono racchiusi tra parentesi graffe
  - Iniziano con '{' e finiscono con '}'
- Ogni nome è seguito da : (colon) e coppie nome/valore sono separate da , (comma)
- Le chiavi devono essere stringhe e dovrebbero essere diverse tra loro
- Object dovrebbero essere usati quando le chiavi sono stringhe arbitrarie

Object

- Sintassi
  - { string : value, ...}

- Esempio
  - {

```
"id": "011A",
"language": "JAVA",
"price": 500,
```

}

Whitespace

- Può essere inserito tra ogni coppia di token
  - Utilizzato per rendere tutto più leggibile
- Sintassi
  - {string:" ",....}
- Esempio
  - var i= " sachin";
  - var j = " saurav"

NULL

- Tipo vuoto
- Sintassi
  - null
- Esempio
  -
```
var i = null;
if(i==1) {
    document.write("<h1>value is 1</h1>");
}
else {
    document.write("<h1>value is null</h1>");
}
```

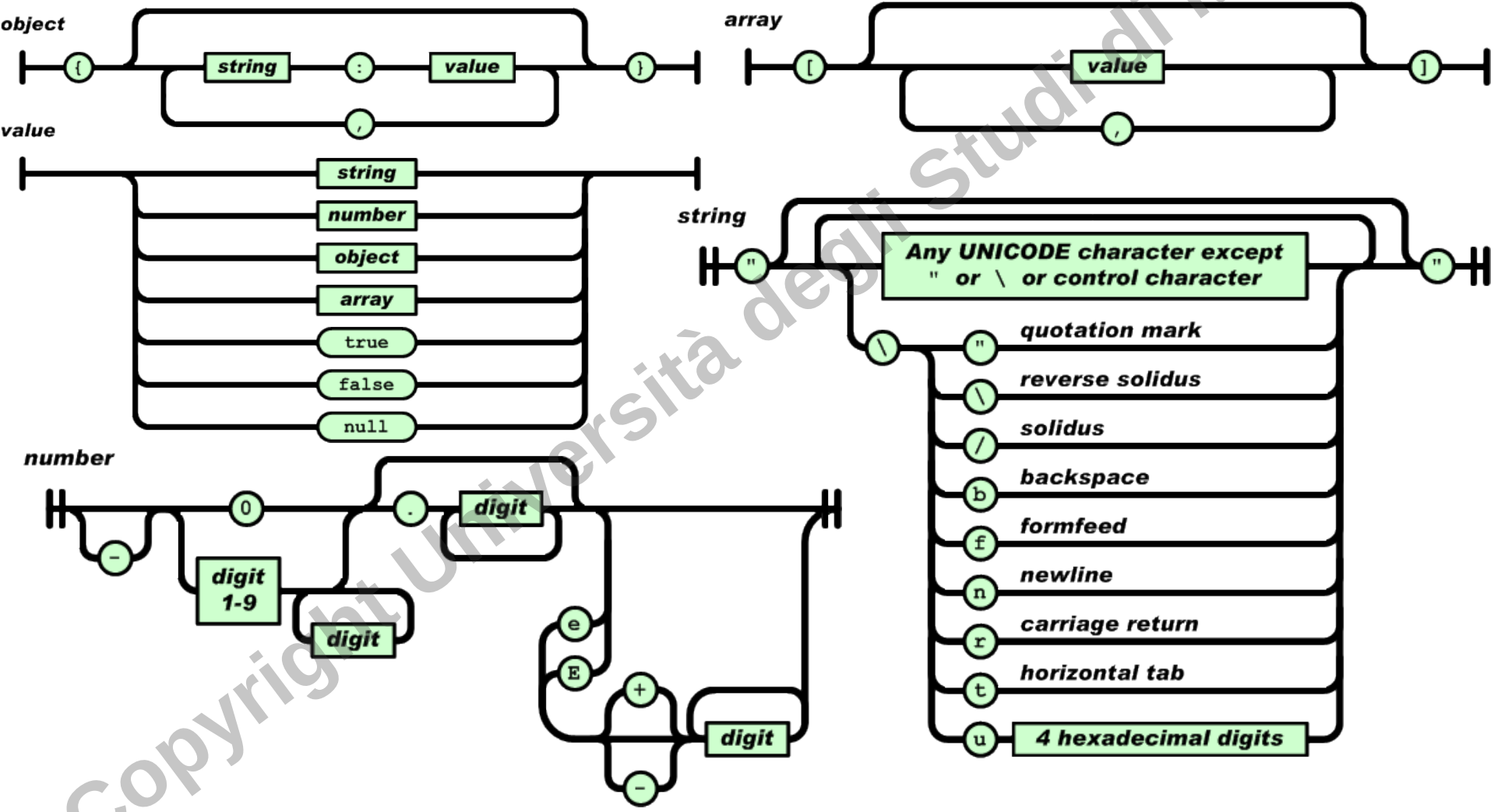
Value

- Include
  - number (integer o floating point)
  - string
  - boolean
  - array
  - object
  - null

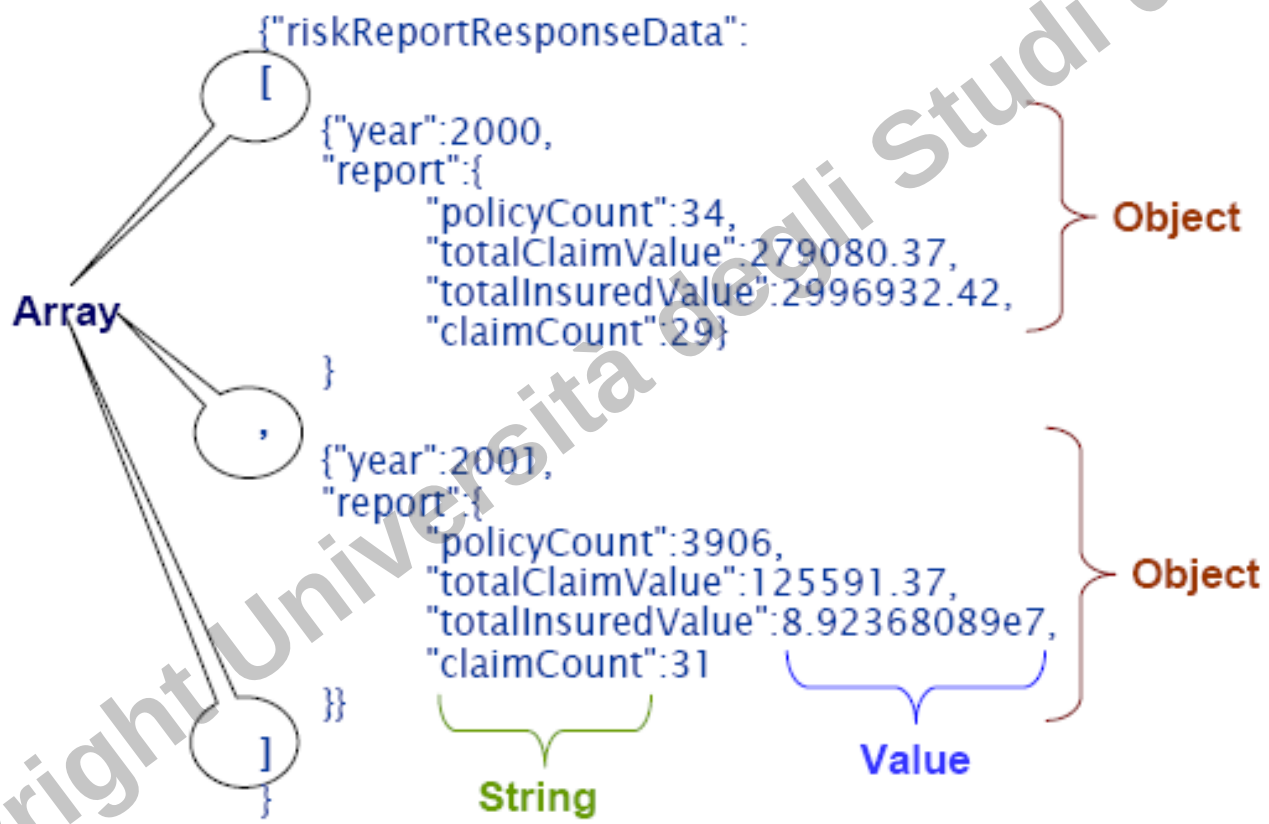
- Sintassi
  - String | Number | Object | Array | TRUE | FALSE | NULL

- Esempio
  - var i =1;
  - var j = "sachin";
  - var k = null;

Strutture dati



Esempio JSON



JSON Schema

- JSON Schema è una specifica per la definizione dei dati JSON

- Scritta in un IETF draft scaduto nel 2011
  - Descrive il formato dei dati
  - Documentazione semplice, human-readable e machine-readable
  - Validazione strutturale completa
    - Utile per testing automatico
    - Valida dati sottomessi dal client

JSON Schema: Librerie di validazione

- Diversi validatori per diversi linguaggi di programmazione
- JVS il validatore più completo e consistente

Languages	Libraries
C	WJElement (LGPLv3)
Java	json-schema-validator (LGPLv3)
.NET	Json.NET (MIT)
ActionScript 3	Frigga (MIT)
Haskell	aeson-schema (MIT)
Python	Jsonschema
Ruby	autoparse (ASL 2.0); ruby-jsonschema (MIT)
PHP	php-json-schema (MIT). json-schema (Berkeley)
JavaScript	Orderly (BSD); JSV; json-schema; Matic (MIT); Dojo; Persevere (modified BSD or AFL 2.0); schema.js.

JSON Schema: Esempio

- Descrizione catalogo prodotti
  - {

```
"$schema": "http://json-schema.org/draft-04/schema#",
"title": "Product",
"description": "A product from Acme's catalog",
"type": "object",
"properties": {
    "id": {
        "description": "The unique identifier for a product",
        "type": "integer"
    },
    "name": {
        "description": "Name of the product",
        "type": "string"
    },
    "price": {
        "type": "number",
        "minimum": 0,
        "exclusiveMinimum": true
    }
},
"required": ["id", "name", "price"]
```

}

JSON Schema: Esempio

Keywords	Description
$schema	The $schema keyword states that this schema is written according to the draft v4 specification.
title	You will use this to give a title to your schema
description	A little description of the schema
type	The type keyword defines the first constraint on our JSON data: it has to be a JSON Object.
properties	Defines various keys and their value types, minimum and maximum values to be used in JSON file.
required	This keeps a list of required properties.
minimum	This is the constraint to be put on the value and represents minimum acceptable value.
exclusiveMinimum	If "exclusiveMinimum" is present and has boolean value true, the instance is valid if it is strictly greater than the value of "minimum".
maximum	This is the constraint to be put on the value and represents maximum acceptable value.
exclusiveMaximum	If "exclusiveMaximum" is present and has boolean value true, the instance is valid if it is strictly lower than the value of "maximum".
multipleOf	A numeric instance is valid against "multipleOf" if the result of the division of the instance by this keyword's value is an integer.
maxLength	The length of a string instance is defined as the maximum number of its characters.
minLength	The length of a string instance is defined as the minimum number of its characters.
pattern	A string instance is considered valid if the regular expression matches the instance successfully.

Istanza dello schema: Esempio

```
[
   {
      "id": 2,
      "name": "An ice sculpture",
      "price": 12.50,
   },
   {
      "id": 3,
      "name": "A blue mouse",
      "price": 25.50,
   }
]
```

JSON vs XML

- JSON è simile a XML ma più leggero
  - Formattano e strutturano il dato da trasferire
- Sono entrambi formati human readable e indipendenti dal linguaggio
- Ogni valore in XML richiede apertura e chiusura di un tag, in JSON solo il valore

JSON vs XML

- Possibile confrontare JSON e XML sulla base di tre fattori principali
  - Verbosità
  - Utilizzo degli array
  - Parsing

JSON vs XML

- Verbose
  - XML è più verboso di JSON
  - Più veloce scrivere JSON per esseri umani
- Utilizzo degli array
  - XML è usato per descrivere dati strutturati che non includono array
  - JSON include array
- Parsing
  - Il metodo eval di JavaScript fa il parsing di JSON
  - Quando applicato a JSON, eval ritorna l'oggetto descritto

JSON vs XML

- JSON è leggero, quindi semplice da leggere e scrivere
- JSON supporta gli array
- I file JSON sono più leggibili dall'uomo
- JSON non ha capacità di visualizzazione
- Fornisce tipi di dati scalari e la capacità di esprimere dati strutturati tramite array e oggetti
- Supporto per oggetti nativi
- XML è meno semplice di JSON

- XML non supporta array
- I file XML sono meno leggibili
- XML fornisce la capacità di visualizzare i dati perché è un linguaggio di markup
- Non fornisce alcuna nozione di tipi di dati. Bisogna fare affidamento a XML Schema per aggiungere informazioni sul tipo
- Gli oggetti devono essere espressi per convenzioni, spesso attraverso un uso misto di attributi ed elementi.

- Somiglianze tra JSON e XML
  - Entrambi sono semplici e aperti
  - Entrambi supportano unicode
  - Entrambi rappresentano dati autodescrittivi
  - Entrambi sono interoperabili e indipendenti dalla lingua

Formato di rappresentazione delle risorse: XML vs. JSON

- XML
  - Plain Old XML (PO-XML): XML base
  - SOAP (WS-*)
  - RSS, ATOM
- Sintassi testuale standard per dati semi-strutturati
- Molti tool disponibili
  - XML Schema, DOM, SAX, XPath, XSLT, XQuery
- Chiunque può parsare XML
- Lento e verboso

Formato di rappresentazione delle risorse: XML vs. JSON

- JSON (JavaScript Object Notation)
  - Formato introdotto per applicazioni Web AJAX (comunicazioni browser-web server)
  - Sintassi testuale per la serializzazione di strutture dati non ricorrenti
  - Supportato da molti linguaggi (non solo JavaScript)
  - Non estendibile
  - “JSON has become the X in AJAX”

JSON vs XML: Esempio

- JSON

```
{
  "company": Volkswagen,
  "name": "Vento",
  "price": 800000
}
```

- XML

```
<car>
  <company>Volkswagen</company>
  <name>Vento</name>
  <price>800000</price>
</car>
```

Javascript e JSON

Parser JSON

- Un numero crescente di web service ritornano dati serializzati come JSON
- Più semplice per un browser ottenere una struttura dati javascript da un JSON
  - Lavora allo stesso modo per qualunque browser
  - JSON è un alternativa leggera a XML
  - http://www.json.org fornisce implementazioni in molti linguaggi
  - È un meccanismo semplice e indipendente dal linguaggio per formattare strutture dati

Parser JSON

- Esempio JSON

```
[3, "three"]
```

- Esempio XML

```
<value>
  <array>
    <data>
      <value><i4>3</i4></value>
      <value><string>three</string></value>
    </data>
  </array>
</value>
```

Parser JSON

- Un programma javascript che analizza una stringa JSON (http://json.parser.online.fr/)
  - Basato su un sottoinsieme del JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999
  - Si può parsare un JSON semplicemente chiamando la funzione parse sulla stringa
    - Vecchie versioni dei browser possono usare la funzione eval
      - https://www.w3schools.com/js/js_json_eval.asp

Parser JSON

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<p id="demo2"></p>

<script type="text/javascript">
   var d = [{"count": 1, "id": 2, "message": "ciao"},{"count": 2, "id": 3, "message": "hello"}];
   var n = JSON.stringify(d); // trasforma un oggetto JavaScript in una stringa JSON
   document.getElementById("demo").innerHTML = n;
   var array=JSON.parse(n); // trasforma una stringa JSON in un oggetto JavaScript
   document.getElementById("demo2").innerHTML = array[0].count;
</script>

</body>
</html>
```


Creare un semplice oggetto JSON

- Oggetto JavaScript in formato JSON può essere creato con Javascript
- Creazione di un oggetto vuoto
  - var JSONObj = {};
- Creazione di un nuovo oggetto
  - var JSONObj = new Object();
- Creazione di un oggetto con attributo bookname di tipo stringa e con attributo price di tipo numero
  - var JSONObj = { "bookname ":"VB BLACK BOOK", "price":500 };
  - Attributi accedibili usando '.'

Creare un semplice oggetto JSON: Esempio

- Esempio della creazione di un oggetto in JavaScript usando JSON

```
<html>
<head>
<title>Creating Object JSON with JavaScript</title>
<script language="javascript" >
  var JSONObj = { "name" : "tutorialspoint.com", "year"  : 2005 };
  document.write("<h1>JSON with JavaScript example</h1>");
  document.write("<br>");
  document.write("<h3>Website Name="+JSONObj.name+"</h3>");
  document.write("<h3>Year="+JSONObj.year+"</h3>");
</script>
</head>
<body>
</body>
</html>
```

- Salvare il codice come json_array.htm e aprire il file

Creazione di un array di oggetti

- Creazione di un array di oggetti in JavaScript usando JSON

```
<html>
<head>
<title>Creation of array object in javascript using JSON</title>
<script language="javascript" >
document.writeln("<h2>JSON array object</h2>");
var books = { "Pascal" : [
     { "Name"  : "Pascal Made Simple", "price" : 700 },
     { "Name"  : "Guide to Pascal", "price" : 400 }
  ],
  "Scala"  : [
     { "Name"  : "Scala for the Impatient", "price" : 1000 },
     { "Name"  : "Scala in Depth", "price" : 1300 }
  ]
}
```

Creazione di un array di oggetti

```
var i = 0
document.writeln("<table border='2'><tr>");
for(i=0;i<books.Pascal.length;i++)
{
  document.writeln("<td>");
  document.writeln("<table border='1' width=100 >");
  document.writeln("<tr><td><b>Name</b></td><td width=50>"
  + books.Pascal[i].Name+"</td></tr>");
  document.writeln("<tr><td><b>Price</b></td><td width=50>"
  + books.Pascal[i].price +"</td></tr>");
  document.writeln("</table>");
  document.writeln("</td>");
}
```

Creazione di un array di oggetti

```
for(i=0;i<books.Scala.length;i++)
{
  document.writeln("<td>");
  document.writeln("<table border='1' width=100 >");
  document.writeln("<tr><td><b>Name</b></td><td width=50>"
  + books.Scala[i].Name+"</td></tr>");
  document.writeln("<tr><td><b>Price</b></td><td width=50>"
  + books.Scala[i].price+"</td></tr>");
  document.writeln("</table>");
  document.writeln("</td>");
}
document.writeln("</tr></table>");
</script>
</head>
<body>
</body>
</html>
```

- Salvare il codice come json_table.htm e aprire il file

Altro esempio

- storage-JSON.html
  - Creazione file JSON
  - Aggiunta a web storage
  - Parsing del codice JSON

Conclusioni

- XML e JSON come linguaggi di definizione e rappresentazione delle informazioni
- XML e JSON come sorgenti di dati e informazioni per applicazioni web
- JSON standard più utilizzati nelle applicazioni moderne