JSON Schema

Giuseppe Della Penna
Università degli Studi di L'Aquila

giuseppe.dellapenna@univaq.it
http://people.disim.univaq.it/dellapenna

Versione documento: 220124

JSON e JSON Schema

- Uno schema JSON **definisce formalmente la struttura e i tipi di dato all'interno di un documento JSON**.
  - Si tratta di un formalismo in fortemente ispirato a XML Schema, che viene utilizzato per definire la struttura di documenti XML.
- Uno schema JSON è a sua volta un documento JSON, la cui struttura è definita nella specifica all'indirizzo https://json-schema.org/specification.html
- Ricordiamo che in JSON si possono esprimere
  - quattro tipi di dato primitivi, cioè stringa, numero, booleano e nullo
  - due tipi di dato complessi, cioè array e oggetto
- Altri tipi più specifici possono essere definiti come sotto-tipo di quelli di base proprio utilizzando uno schema.
- Uno schema JSON è definito come un oggetto JSON avente un vocabolario ben preciso.
  - L'oggetto vuoto {} o la costante *true* rappresentano uno schema che permette qualsiasi tipo di documento JSON.
  - Al contrario, la costante *false* costituisce uno schema che non valida alcun valore JSON.

Perché formalizzare lo schema dei dati?

- Ci si potrebbe chiedere che vantaggi fornisca l'usare uno schema per validare un documento JSON. Spesso, infatti, JSON viene visto (al contrario, ad esempio, di XML) come un «linguaggio a schema libero».
- In verità, uno schema permette di ottenere numerosi vantaggi, ad esempio
  - E' possibile **verificare che un documento JSON rispetti lo schema** prima di usarlo/elaborarlo, in modo da dover effettuare *meno controlli nella logica vera e propria del programma* (e incorrere in meno situazioni di errore impreviste)
    https://json-schema.org/implementations.html#validators
    https://json-everything.net/json-schema
  - E' possibile **generare automaticamente il codice per serializzare e deserializzare in oggetti di vari linguaggi un documento JSON** valido rispetto a uno schema
    https://json-schema.org/implementations.html#code-generation
    https://app.quicktype.io/#l=schema
  - E' possibile **creare in maniera automatica editor intelligenti per riempire le strutture JSON** descritte dallo schema senza dover scrivere direttamente il codice JSON (che, diciamocelo, è tedioso e spesso porta a fare errori come omettere virgole o parentesi di chiusura!)
    https://json-schema.org/implementations.html#generators-from-schemas
    https://github.com/json-editor/json-editor

Schema di base

```
{
 "$schema": "https://json-schema.org/draft/2020-
12/schema",
 "$id": "https://example.com/product.schema.json",
 "title": "Product",
 "description": "A product in the catalog‟,
 ...
}
```

- Uno schema JSON **radice**, cioè *non nidificato in alcun altro schema*, deve contenere le seguenti proprietà:
  - **$schema**: la URI che identifica la versione della specifica adottata per lo schema.
  - **$id**: la URI dello schema (usata anche come base per le URI relative usate nello schema stesso). Non deve essere necessariamente la URL di una risorsa scaricabile, ma solo un **identificatore univoco**.
- Inoltre, è utile inserire le annotazioni generali *title* e *description*, che verranno introdotte più avanti.

Annotazioni

https://json-schema.org/understanding-json-schema/reference/generic.html

```
{
  "title": "Match anything",
  "description": "A schema that matches anything.",
  "default": "Default value",
  "examples": [
    "Anything",
    4035
  ],
  "deprecated": true
}
```

- Le annotazioni sono informazioni opzionali che possono essere inserite negli schemi e non vengono usate nel processo di validazione, ma possono essere utili ad altri tool che utilizzano lo schema.
- **default**: suggerisce il valore di default per il dato.
- **title**: fornisce un nome descrittivo per il dato
- **description**: fornisce una descrizione dettagliata del dato
- **deprecated**: un booleano che indica che questo dato non andrebbe utilizzato e potrebbe venir rimosso dallo schema in futuro
- **examples**: un array di valori di esempio per il dato

Tipi di dato

```
{
 "$schema": "https://json-schema.org/draft/2020-
12/schema",
 "$id": "https://example.com/product.schema.json",
 "title": "Product",
 "description": "A product in the catalog‟,
 "type" : "object",
 ...
}

{
 "$schema": "https://json-schema.org/draft/2020-
12/schema",
 "$id": "https://example.com/product.schema.json",
 "title": "Product",
 "description": "A product in the catalog‟,
 "type" : ["string","number"],
 ...
}
```

- Lo schema JSON può contenere la parola chiave **type**, che definisce il tipo di dato associato.
  - In base al tipo specificato, altre parole chiave nello stesso oggetto permetteranno di raffinarne meglio la forma e i vincoli.
- I tipi disponibili sono *null*, *boolean*, *object*, *array*, *number*, *string* e *integer* (come number ma senza decimali).
- E' anche possibile inserire un *array di nomi di tipo*, nel qual caso tutti i tipi elencati saranno considerati validi (sempre che il valore rispetti gli eventuali altri vincoli!)
- Se si omette la parola chiave **type**, spesso questo viene interpretato come «ogni tipo è accettabile», tuttavia si tratta di una questione semantica aperta e quindi è meglio non abusarne…

Il tipo string

https://json-schema.org/understanding-json-schema/reference/string.html

```
{
  "type": "string",
  "minLength": 2,
  "maxLength": 3
}

{
   "type": "string",
   "pattern": "^(\\([0-9]{3}\\))?[0-9]{3}-[0-9]{4}$"
}
```

- Uno schema di tipo **string** può essere arricchito tramite le seguenti proprietà vincolanti:
  - **minLength**, **maxLength**: vincolano la lunghezza della stringa
  - **pattern**: viene utilizzata per vincolare una stringa a una particolare espressione regolare.
  - **format**: richiede la validazione semantica della stringa in base a determinati tipi di valori stringa comunemente utilizzati. I più utili sono:
    - **date-time**: data e ora insieme, ad esempio 2021-09-06T20:20:39+00:00.
    - **time**: ora, ad esempio 20:20:39+00:00
    - **date**: data, ad esempio 2021-09-06
    - **e-mail**: indirizzo e-mail
    - **hostname**: nome host
    - **ipv4**: indirizzo IPv4
    - **ipv6**: indirizzo IPv6
    - **uri**: una URI

I tipi number e integer

https://json-schema.org/understanding-json-schema/reference/string.html

```
{
  "type": "number",
  "minimum": 0,
  "exclusiveMaximum": 100
}

{
    "type": "number",
    "multipleOf" : 10
}
```

- Uno schema di tipo **number** (o **integer**, che sottintende già alcuni voncoli come già descritto) può essere arricchito tramite le seguenti proprietà vincolanti:
  - **minimum**, **exclusiveMinimum**, **maximum**, **exclusiveMaximum**: definiscono l'intervallo (inclusivo o esclusivo) dei numeri accettabili.
  - **multipleOf**: vincola il numero a essere multiplo di un altro numero.

I tipi boolean e null

https://json-schema.org/understanding-json-schema/reference/boolean.html
https://json-schema.org/understanding-json-schema/reference/null.html

```
{
  "type": "boolean",
}

{
   "type": "null",
}
```

- Uno schema di tipo **boolean** o **null** non accetta ulteriori proprietà vincolanti.

Il tipo object

https://json-schema.org/understanding-json-schema/reference/object.html

```
{
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "email": { "type": "string" },
    "address": { "type": "string" },
    "telephone": { "type": "string" }
  },
  "required": ["name", "email"]
  "additionalProperties": false
}

{
  "type": "object",
  "patternProperties": {
    "^S_": { "type": "string" },
    "^I_": { "type": "integer" }
  }
}

{
  "type": "object",
  "properties": {
    "builtin": { "type": "number" }
  },
  "patternProperties": {
    "^S_": { "type": "string" },
    "^I_": { "type": "integer" }
  },
  "additionalProperties": { "type": "string" }
}
```

- Uno schema di tipo **object** valida ogni oggetto JSON. Per definire con più dettaglio la struttura degli oggetti validi, è possibile usare le seguenti proprietà vincolanti:
  - **properties**: è un oggetto che elenca nomi di proprietà che possono apparire nell'oggetto definito, associando a ciascuna uno schema JSON nidificato.
    - Di default le proprietà così definite sono *opzionali*, cioè possono anche non comparire negli oggetti. Il vincolo **required** può essere usato per elencare le proprietà obbligatoria dell'oggetto.
  - **patternProperties**: come *properties*, ma in questo caso i nomi delle proprietà sono espressioni regolari che permettono di associare la stessa definizione di tipo a tutte le proprietà il cui nome fa match con l'espressione.
  - L'oggetto definito potrà avere anche proprietà non in elenco (*properties* o *patternProperties*), a meno che non si specifichi il vincolo **additionalProperties** ponendolo a false. E' anche possibile associare ad additionalProperties lo schema che dovranno seguire tutte le proprietà non in elenco.
  - **minProperties**, **maxProperties**: permettono di limitare il numero di proprietà inserite negli oggetti.

Il tipo array

https://json-schema.org/understanding-json-schema/reference/array.html

```
{
  "type": "array",
  "items": {
    "type": "number"
  }
}

{
  "type": "array",
  "prefixItems": [
    { "type": "number" },
    { "type": "string" },
    { "enum": ["Street", "Avenue", "Boulevard"] },
    { "enum": ["NW", "NE", "SW", "SE"] }
  ]
}

{
  "type": "array",
  "prefixItems": [
    { "type": "number" },
    { "type": "string" },
    { "enum": ["Street", "Avenue", "Boulevard"] },
    { "enum": ["NW", "NE", "SW", "SE"] }
  ],
  "items": false
}

{
  "type": "array",
  "minItems": 2,
  "maxItems": 3,
  "uniqueItems": true
}
```

- Uno schema di tipo **array** valida ogni array JSON. Per definire con più dettaglio la struttura degli array validi, è possibile usare le seguenti proprietà vincolanti:
  - **items**: è uno schema JSON nidificato che indica il tipo degli elementi dell'array.
  - **prefixItems**: permette di validare l'array come una **tupla**, fornendo una lista ordinata di schemi JSON, che verranno usati per validare, nell'ordine, ciascun elemento dell'array.
    - *In alcune versioni della specifica JSON schema questa proprietà è unita a items.*
  - Per specificare se e di che tipo possono essere gli elementi non gestiti da *prefixItems* è possibile inserire anche una proprietà *items* oppure, se questa è stata usata in vece di *prefixItems*, è possibile usare il vincolo **additionalItems**, il cui calore è sempre uno schema JSON nidificato.
  - **minItems**, **maxItems**: permettono di limitare il numero di elementi nell'array
  - **uniqueItems**: permette di richiedere che ogni elemento nell'array sia unico (se posto a *true*)

Enumerazioni

https://json-schema.org/understanding-json-schema/reference/generic.html

```
{
 “type”: string,
 "enum": ["red", "amber", "green"]
}

{
 "enum": ["red", "amber", "green", null, 42]
}
```

- E’ possibile vincolare un valore JSON ad essere esattamente uno di quelli elencati in un array tramite la proprietà **enum.**
- enum può essere usato in associazione con qualsiasi *type*, e in tal caso i valori inseriti nell’enumerazione dovranno essere validi rispetto al tipo così dichiarato.
- E’ anche possibile usare enum senza specificare anche type, e in tal caso i valori ammessi dall’enumerazione potranno essere arbitrari, e anche di tipi diversi.

Composizione di Schemi

https://json-schema.org/understanding-json-schema/reference/combining.html

```
{
  "allOf": [
    { "type": "string" },
    { "maxLength": 5 }
  ]
}

{
  "anyOf": [
    { "type": "string", "maxLength": 5 },
    { "type": "number", "minimum": 0 }
  ]
}

{
  "oneOf": [
    { "type": "number", "multipleOf": 5 },
    { "type": "number", "multipleOf": 3 }
  ]
}

{ "not": { "type": "string" } }

{ "not": {} }
(equivale allo schema false)
```

- E' possibile **costruire uno schema JSON combinando altri schemi secondo determinate regole.**
- In questo caso, lo schema non conterrà un *type*, ma una delle seguenti proprietà
  - **allOf**: i documenti devono essere validi **rispetto a tutti** gli schemi elencati nell'array associato a questa proprietà.
  - **anyOf**: i documenti devono essere validi **rispetto ad almeno uno** degli schemi elencati nell'array associato a questa proprietà.
  - **anyOf**: i documenti devono essere validi **rispetto ad esattamente uno** (uno e uno solo) degli schemi elencati nell'array associato a questa proprietà.
  - **not**: i documenti ***non* devono essere validi rispetto allo schema** associato a questa proprietà.

Nota: nel primo esempio, il secondo sottoschema non contiene un type, quindi si può considerare, come abbiamo visto, che qualsiasi valore (di qualsiasi tipo) con lunghezza massima 5 sia valido rispetto ad esso.

Riferimento a Schemi

https://json-schema.org/understanding-json-schema/structuring.html#ref

```
{
 "$id": "https://example.com/schemas/customer",
 "type": "object",
 "properties": {
  "first_name": { "type": "string" },
  "last_name": { "type": "string" },
  "shipping_address": { "$ref": "/schemas/address" },
  "billing_address": { "$ref": "/schemas/address" }
 },
 "required": ["first_name", "last_name",
"shipping_address", "billing_address"]
}
```

- E' possibile **far riferimento a uno schema tramite la sua URI** e la proprietà **$ref** (il valore $id dello schema) per modularizzare la definizione di uno schema composto.
  - In questo caso, lo schema non conterrà nessun *type* o keyword di composizione, ma potrà contenere elementi di annotazione.
- Le URI dei riferimenti, se incomplete, sono risolte completandole con la URI di base dello schema che, come già detto, corrisponde al suo attributo *$id*.