REST:
architettura per Web Services

Gianluca Pironato

Introduzione al REST

- **REST (Representational State Transfer)** è uno stile architetturale per progettare Web Services.
- Utilizza HTTP come base di tutto (per produrre e trasferire le richieste e le risposte) e risulta estremamente più semplice dello standard basato su SOAP.
- Ampiamente adottato da aziende come Google e Meta.

Le risorse

- Ogni risorsa è univocamente identificata da un URI (Uniform Resource Identifier) all'interno del web.
- Per risorsa si intendono dati, servizi o altre entità accessibili dal web e identificabili con URI.

Principi base di REST

REST si fonda su quattro principi fondamentali:

1. uso esplicito dei metodi HTTP;
2. statelessness (assenza di stato lato server);
3. esposizione di URI gerarchiche (simili a directory);
4. trasferimento di dati in JSON/XML.

1. Uso esplicito dei metodi HTTP

- REST mappa le operazioni essenziali CRUD (Create, Read, Update, Delete) ai metodi HTTP:
  - **POST**: creazione di risorse;
  - **GET**: ottenimento di risorse (no side effects);
  - **PUT**: aggiornamento o sostituzione di risorse;
  - **DELETE**: eliminazione di risorse.

1. Uso esplicito dei metodi HTTP

Listing 1. Before

```
GET /adduser?name=Robert HTTP/1.1
```

Listing 2. After

```
POST /users HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Robert</name>
</user>
```

2. Statelessness

- Ogni richiesta deve contenere tutte le informazioni necessarie per essere processata, senza che l'app lato server debba svolgere altre operazioni legate a sessione/contesto dell'elaborazione.
- Non si mantiene stato lato server tra richieste successive, così ciascuna richiesta risulta indipendente dalle altre.
- Vantaggi:
  - Semplicità nella progettazione e distribuzione.
  - Scalabilità migliorata (le richieste possono essere inoltrate attraverso intermediari, es. proxy, perché trasportano tutte le info necessarie).

2. Statelessness

Figure 1. Stateful design

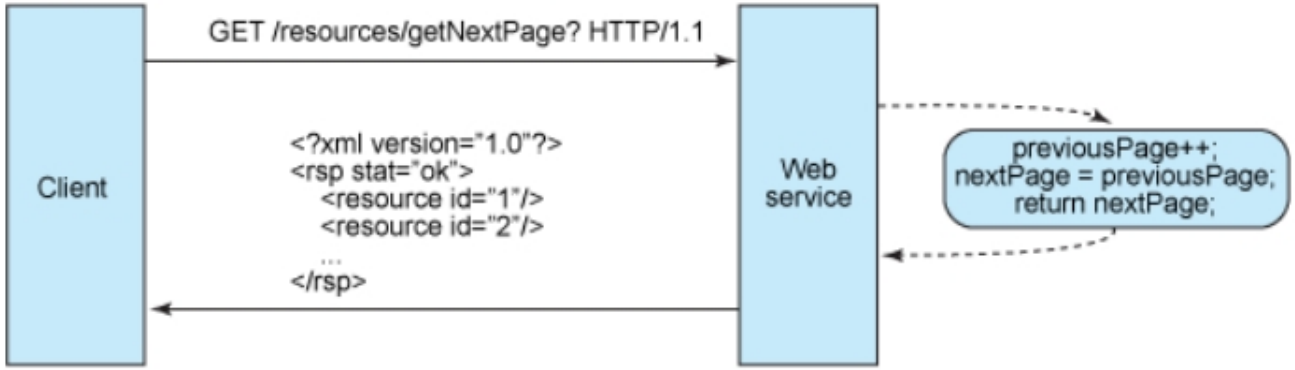
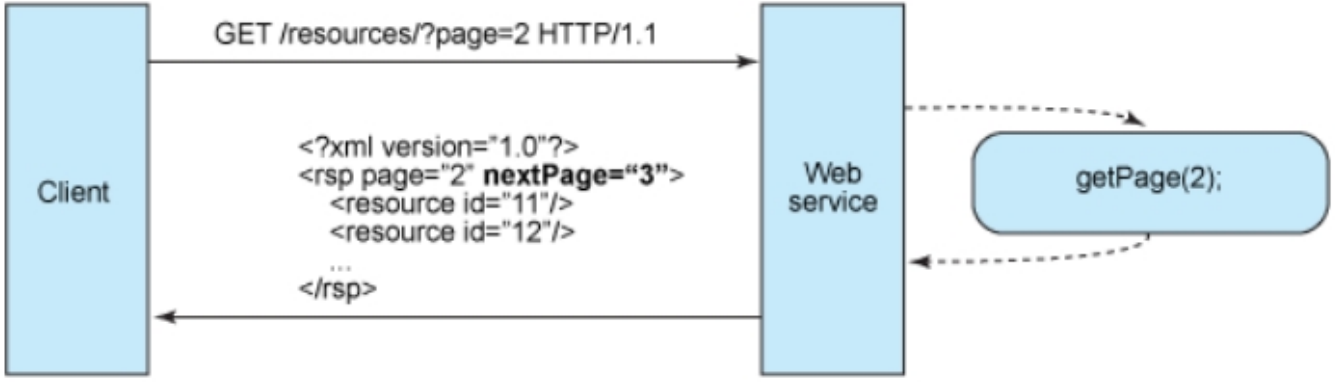


Figure 2. Stateless design



3. Struttura delle URI

- Le URI in REST devono essere:
  - Intuitive e autoesplicative, organizzate gerarchicamente, come i path di un file system ad albero (le directory).
  - Statici nel tempo, per supportare i segnalibri (se cambiano, i link salvati serviranno ben poco al client).
- Esempio di URI correttamente strutturata:

`http://www.myservice.org/discussion/2024/12/07/topic`

- Buone pratiche:
  - Evitare estensioni specifiche (.jsp, .php).
  - Usare solo lettere minuscole.
  - Sostituire spazi con trattini o underscore.

4. Scambio di dati: XML e JSON

- Le risorse vengono rappresentate in formati standard:
  - **XML**: formato machine-readable e strutturato.
  - **JSON**: formato leggero e ideale per applicazioni web (notazione oggetti di JavaScript).
- Supporto alla negoziazione del contenuto tramite l'header HTTP `Accept`.
- Esempio: una richiesta per ottenere una rappresentazione JSON.

```
GET /users/john HTTP/1.1
Accept:  application/json
```

Conclusioni

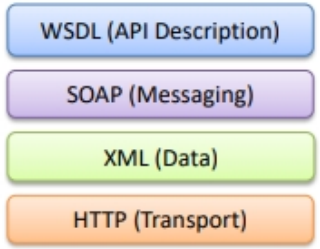
- REST è uno stile architetturale semplice e scalabile per Web Services.
- Enfatizza l'uso di standard aperti e consolidati come HTTP, URI, XML e JSON.

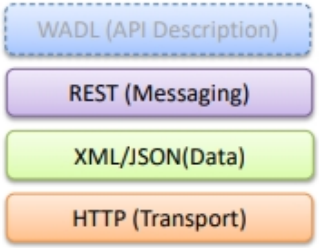
Conclusioni

SOAP vs. REST

SOAP:

- A service architecture
- XML based
- Runs on HTTP but envelopes the message
- Slower than REST
- Very mature, a lot of functionality
- Not suitable for browser-based clients
- More complicated





REST:

- A service architecture
- Uses the HTTP headers to hold meta information
- Can be used with XML, JSON or whatever necessary
- Usually used with JSON due to the easily parsable content
- Faster than SOAP

Fonti

- **IMPORTANTE:** Introduction to RESTful Web Services, `https://developer.ibm.com/articles/ws-restful/`
- Una panoramica generale sui Web Services, `https://www.halvorsen.blog/documents/programming/software_engineering/resources/resources/Web%20Services%20Overview.pdf`