Luca Cabibbo

Architettura dei Sistemi Software

Ci ascolta ma hai seguito

Servizi REST

dispensa asw525

ottobre 2024

The World Wide Web is arguably the world's largest distributed application. Understanding the key architectural principles underlying the Web can help explain its technical success and may lead to improvements in other distributed applications.

Roy Thomas Fielding

- Riferimenti

- Luca Cabibbo. **Architettura del Software**: Strutture e Qualità. Edizioni Efesto, 2021.
  - Capitolo 30, Servizi web SOAP e REST
- Fielding, R.T. Architectural Styles and the Design of Network-based Software Architectures. PhD Thesis, 2000.
- Newman, S. **Building Microservices**: Designing Fine-grained Systems. O'Reilly, second edition, 2021.
- Higginbotham, J. **Principles of Web API Design**. Addison-Wesley, 2022.
- Jin, B., Sahni, S. and Shevat, A. **Designing Web APIs**. O'Reilly, 2018.

Il documento introduce i servizi REST, uno stile architetturale leggero per sviluppare applicazioni distribuite e interoperabili. Di seguito il riassunto:

Definizione e concetti principali

- REST (Representational State Transfer) è uno stile architetturale utilizzato per progettare applicazioni web.
- I servizi REST (o RESTful web services) si basano sull'interazione tra risorse, identificate univocamente tramite URI, e sulle operazioni del protocollo HTTP (GET, POST, PUT, DELETE).
- Le risorse hanno rappresentazioni (es. JSON, XML) che disaccoppiano la memorizzazione interna dal modo in cui vengono condivise.

Caratteristiche dello stile REST

1. Client-server: Separazione delle responsabilità tra client e server.
2. Stateless: Nessuna informazione sullo stato della conversazione viene mantenuta lato server.
3. Caching: Possibilità di memorizzare risposte per migliorare le prestazioni.
4. Interfaccia uniforme: Operazioni standardizzate tramite HTTP.
5. Architettura a livelli: Comunicazione possibile tramite intermediari.
6. Code on demand (opzionale): Capacità di scaricare codice eseguibile per estendere le funzionalità.

Principi dei servizi REST

- Identificazione tramite URI: Ogni risorsa ha un URI unico.
- Messaggi auto-descrittivi: Richieste e risposte contengono tutte le informazioni necessarie.
- HATEOAS (Hypermedia as the Engine of Application State): I link ipertestuali nelle risposte guidano il client verso altre risorse e operazioni.
- Interazioni stateless: Le comunicazioni sono autonome e non dipendono da stati precedenti.

Vantaggi e limitazioni

Vantaggi:

- Semplicità e leggerezza, rispetto a tecnologie come SOAP.
- Scalabilità e interoperabilità grazie all'uso di standard web.
- Ampia disponibilità di strumenti e librerie per sviluppare servizi REST.
- Facilità di scoperta delle risorse grazie agli URI.

Limitazioni:

- Meno adatto a scenari complessi o con requisiti di qualità rigorosi.
- Operazioni HTTP limitate a un vocabolario ristretto.
- Mancanza di standard per la composizione e la descrizione delle interfacce.
- Gestione asincrona non nativa (supportata solo da soluzioni aggiuntive come WebHooks o WebSockets).

Confronto con SOAP

- REST: più semplice e leggero, adatto a contesti meno complessi.
- SOAP: più completo, con supporto standardizzato per descrizioni, scoperta e composizione dei servizi, oltre a una gestione più robusta degli attributi di qualità.

- Obiettivi e argomenti

- Obiettivi
  - presentare i servizi REST
  - confrontare i servizi SOAP con i servizi REST
- Argomenti
  - introduzione
  - servizi REST
  - discussione
  - confronto tra servizi SOAP e servizi REST

* Introduzione

- Nell'architettura a servizi, un servizio è un componente software
  - ha lo scopo di implementare una funzionalità o un servizio di business di un'organizzazione
  - può essere scoperto e invocato dai suoi consumatori mediante un'interfaccia aperta e tramite tecnologie standard del web
  - presentiamo ora i servizi REST

Ne abbiamo parlato per l'invocazione remota. Ma sono anche

* Introduzione ai servizi REST

- I *servizi REST* – *RESTful web service* o *Web API*
  - un'altra importante tecnologia a servizi
    - più leggera rispetto ai servizi web SOAP
  - i servizi REST
    - sostengono l'interoperabilità
    - offrono migliori garanzie su prestazioni e scalabilità
    - sono più semplici da programmare
    - ma presentano anche alcuni inconvenienti

Non stiamo più parlando dei servizi ora

- Stile architetturale REST

I servizi rest nascono dallo stile architetturale REST

- *REST* (*Representational State Transfer*) [Fielding] è uno stile architetturale che descrive l'architettura del world-wide web – e per guidare la progettazione di applicazioni web
  - "Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

Trasferimento di stato della rappresentazione

L'applicazione è un automa a stati con stati tra cui ci muoviamo tramite link.
Quando clicco su un link
- cambia il mio stato
- mi viene restituita la rappresentazione del nuovo stato in cui sono arrivata

Risorse

- Uno dei concetti fondamentali dello stile REST sono le risorse
  - una *risorsa* (*resource*) è un elemento informativo di interesse a cui può essere attribuito un nome
    - ad es., il “corso di Architettura dei Sistemi Software a Roma Tre” oppure “il tempo a Roma oggi”
  - un *identificatore di risorsa* (*resource identifier*) è un nome univoco utilizzato per identificare una specifica risorsa
    - ad es., l’URI http://www.uniroma3.it/corsi/asw per la risorsa “corso di Architettura dei Sistemi Software a Roma Tre”

Rappresentazioni

Le risorse vengono gestite da un servizio. L’implementazione del servizio utilizza una rappresentazione interna della risorsa, ma la stessa risorsa avrà una o più rappresentazioni esterne. Gli altri servizi potranno accedere solo alle rappresentazioni esterne e non a quella interna di un altro servizio

- Un altro concetto fondamentale è quello di rappresentazione
  - una *rappresentazione* è un gruppo di dati (e metadati) per una risorsa
    - ad es., un documento JSON oppure XML
  - le rappresentazioni vengono usate nella richiesta di azioni sulle risorse
    - quando un client accede a una risorsa, gli viene restituita rappresentazione della risorsa
    - per aggiornare una risorsa, un client può comunicare la rappresentazione desiderata della risorsa
  - una risorsa può avere più rappresentazioni diverse
  - l’uso delle rappresentazioni consente di disaccoppiare le modalità di memorizzazione interna delle risorse nei componenti dal modo in cui le risorse vengono condivise con l’esterno

Caratteristiche dello stile REST

- Caratteristiche generali dello stile architetturale REST
  - è uno stile architetturale di tipo client-server
  - i servizi sono di tipo stateless
  - è possibile fare caching delle risposte dei servizi
  - è un'architettura a livelli/strati – un client di solito non sa se sta comunicando con un server che eroga effettivamente il servizio oppure con un intermediario
  - uso di un'interfaccia uniforme tra componenti
  - code on demand (opzionale)

Distingui stile architetturale e servizi! Hai appena visto gli elementi fondamentali che definiscono lo stile!

- Servizi REST

- Lo stile architetturale REST può essere usato anche per guidare la definizione di *servizi REST* (o *RESTful web service* o *Web API*)
  - presentiamo ora un'interpretazione comune dei servizi REST, basata su HTTP
    - in pratica, esistono diverse varianti dei servizi REST – ad es., l'uso di HTTP non è obbligatorio
  - per un esempio di servizio REST concreto e dettagliato, si veda https://api.stackexchange.com/docs
    - Stack Exchange è una rete di oltre 170 comunità Q&A – tra cui Stack Overflow

Principi per servizi REST

- Principi che guidano la definizione di un servizio REST
  - identificazione delle risorse tramite URI
    - un servizio REST espone un insieme di risorse, identificate mediante URI
  - interfaccia uniforme
    - le risorse vengono manipolate in modo uniforme, tramite le operazioni di HTTP – come PUT, GET, POST, PATCH e DELETE
  - messaggi auto-descrittivi
    - le risorse sono disaccoppiate dalle loro rappresentazioni – il loro contenuto può essere acceduto sulla base di più formati
    - ogni richiesta contiene informazioni sufficienti a descrivere come il servizio possa elaborare la richiesta
    - ogni risposta contiene informazioni sufficienti a descrivere come il client possa elaborare la risposta

Principi per servizi REST

- Principi che guidano la definizione di un servizio REST
  - interazioni stateful basate su collegamenti ipertestuali
    - le interazioni con le risorse sono stateless
    - è possibile realizzare interazioni stateful sulla base di un trasferimento dello stato delle conversazioni – ad es., riscrittura di URI, campi nascosti, cookie
  - rappresentazione ipermediale
    - una risposta contiene comunemente anche gli identificatori delle risorse correlate alla risorsa restituita, per consentire l'accesso a queste risorse
    - *HATEOAS*, *Hypermedia As The Engine Of Application State*

Esempio di servizio REST

- Un servizio REST potrebbe gestire alcune collezioni omogenee di risorse
  - ad es., un insieme di corsi e un insieme di docenti
  - per ciascuna collezione, il servizio definisce una *collection URI*
    - ad es., http://www.uniroma3.it/corsi e http://www.uniroma3.it/docenti
  - ciascuna istanza di risorsa ha un proprio *element URI*
    - ad es., http://www.uniroma3.it/corsi/asw e http://www.uniroma3.it/docenti/luca.cabibbo
  - le risorse possono avere una struttura annidata
    - ad es., http://www.uniroma3.it/docenti/luca.cabibbo/corsi e http://www.uniroma3.it/docenti/luca.cabibbo/corsi/asw

Esempio di servizio REST

- Un servizio REST potrebbe gestire alcune collezioni omogenee di risorse
  - il servizio offre, per ciascuna risorsa, delle operazioni in corrispondenza con le operazioni HTTP, come GET, PUT, POST, PATCH e DELETE – sulla base di un'interfaccia uniforme
  - le rappresentazioni potrebbero essere scambiate in formato JSON oppure XML

Esempio di servizio REST

- Interfaccia uniforme di un servizio REST
  - operazioni riferite a un element URI
    - GET – restituisce una rappresentazione di uno specifico elemento della collezione
    - PUT – crea o aggiorna un elemento della collezione
    - PATCH – aggiorna parzialmente uno specifico elemento della collezione
    - DELETE – cancella l'elemento della collezione
    - POST – aggiunge un elemento (annidato) a un elemento della collezione

Esempio di servizio REST

- Interfaccia uniforme di un servizio REST
  - operazioni riferite a un collection URI
    - GET – restituisce tutti gli elementi della collezione
    - POST – crea un nuovo elemento della collezione e gli assegna un URI
    - PATCH – modifica la collezione sulla base di una lista di cambiamenti specificati nella richiesta
    - PUT – sostituisce la collezione con un'altra collezione
    - DELETE – cancella l'intera collezione

HATEOAS

- Il principio *HATEOAS* suggerisce di modellare le relazioni tra risorse in modo ipermediale
  - in una risposta, la rappresentazione di una risorsa può includere anche dei collegamenti (*link*) a risorse correlate
    - link a risorse informative correlate (ad es., gli appelli di un corso) oppure a operazioni sulla risorsa richiesta (ad es., per prenotarsi ad un appello)
    - ad es., in XML, l'elemento link prevede gli attributi href (l'URI della risorsa) e rel (che indica il tipo di relazione)
  - l'interfaccia di un servizio può essere scoperta in modo progressivo
    - il client ha bisogno di conoscere solo un punto di ingresso a un servizio – l'uso del servizio può poi proseguire sulla base di questi collegamenti
    - riduce l'accoppiamento con il servizio – il servizio può evolvere in modo più flessibile

API guidate dagli eventi

- Un servizio può offrire anche un'interfaccia basata sulla comunicazione asincrona – per la notifica di eventi e lo scambio di messaggi
  - ad es., quando un servizio vuole essere informato dei cambiamenti avvenuti negli altri servizi, per aggiornare i propri dati
  - esistono diverse soluzioni per realizzare API guidate dagli eventi – ad es., *WebHooks*, *WebSockets* e *HTTP Streaming*

Sviluppo di servizi REST

- L'adozione dei servizi REST è favorita dall'ampia diffusione di infrastrutture per HTTP e per applicazioni web
  - REST si basa su standard semplici, spesso già noti agli sviluppatori
  - sono disponibili numerose librerie, API e framework per REST
    - ad es., nel framework Spring oppure librerie per Java come JAX-RS
    - questi strumenti possono semplificare sia la gestione di HTTP che la gestione delle trasformazioni tra risorse e rappresentazioni

Servizi REST e interoperabilità

- I servizi REST sono interoperabili
  - sono basati su protocolli standard del web – come HTTP e XML
  - è possibile usare diversi formati per l'interscambio dei dati che sono indipendenti dai linguaggi di programmazione e dalle piattaforme – come JSON e XML
  - l'interfaccia di un servizio REST è costituita da un insieme di URI e di operazioni HTTP su questi URI
    - di solito questa interfaccia non è descritta esplicitamente, ma può comunque essere scoperta dai client del servizio in modo progressivo, sulla base di HATEOAS e dei collegamenti tra risorse

* Discussione

- I servizi REST sono un'altra importante tecnologia a servizi e per l'interoperabilità
  - sono basati su alcuni standard fondamentali del web
  - l'adozione dei servizi REST è di solito semplice
  - è possibile scrivere servizi REST e i loro client in un'ampia varietà di linguaggi di programmazione
  - grazie all'uso di URI e dei collegamenti è possibile scoprire le risorse senza l'uso di un registry centralizzato
  - è possibile l'adozione di soluzioni per la scalabilità e per la sicurezza già sviluppate per il web

Discussione

- I servizi REST presentano però anche alcuni inconvenienti
  - sono adatti soprattutto a scenari con una bassa complessità – per gestire risorse informative abbastanza semplici
  - le infrastrutture per REST non supportano un insieme ampio di attributi di qualità – che devono dunque essere gestiti dagli sviluppatori dei servizi
  - l'accesso ai servizi REST avviene in genere mediante invocazioni remote sincrone – l'integrazione di applicazioni può essere problematica
  - le operazioni HTTP hanno un vocabolario limitato – può essere difficile stabilire una buona definizione dell'interfaccia di un servizio REST
  - i servizi REST non hanno un'interfaccia descritta esplicitamente – questo può rendere più difficile la loro fruizione
  - manca un supporto standardizzato per la composizione

* Confronto tra servizi SOAP e servizi REST

- Vantaggi dei servizi REST
  - l'adozione è più semplice – si basano su standard e tecnologie più semplici
  - è più semplice realizzare sia i servizi che i loro client
  - una tecnologia più leggera, con migliori soluzioni per la scalabilità
  - il vantaggio principale di REST vs. SOAP è la semplicità
- Vantaggi dei servizi SOAP
  - adatti anche a contesti applicativi complessi
  - supporto standardizzato per la descrizione delle interfacce dei servizi e per la scoperta dei servizi
  - supporto standardizzato per la composizione di servizi
  - miglior supporto per diversi attributi di qualità
  - il vantaggio principale di SOAP vs. REST è la completezza