WebService di tipo REST

TEPSIT

Sono sempre più diffuse nel *World Wide Web* risorse non destinate alla visualizzazione da parte di un utente umano, ma all'uso esclusivo da parte di applicazioni software che, eventualmente, ne ricostruiscono successivamente una rappresentazione adatta a un utente umano. Le risorse direttamente fruibili dagli esseri umani sono ovviamente le pagine dei siti web, mentre gli insiemi di risorse consumabili da un sistema software sono definiti *web-service*.

Google rende disponibili alcuni *web-service* di supporto al servizio *Maps*; è per esempio possibile ottenere le coordinate geografiche (latitudine e longitudine) corrispondenti a un indirizzo semplicemente utilizzando un qualsiasi *browser* per inviare al server **http://maps.googleapis.com** un URL come il seguente:

```
http://maps.googleapis.com/maps/api/geocode/xml?
address=...
```

La richiesta

```
http://maps.googleapis.com/maps/api/geocode/xml?address=via+
Irnerio+34+Bologna+Italia
```

ottiene come risposta il seguente documento in formato XML (del quale è stata omessa per brevità una parte) che comprende le coordinate geografiche della sede di Zanichelli a Bologna (dà anche altre informazioni come il CAP, la Provincia e la Regione in cui si trova l'indirizzo richiesto):

I parametri della richiesta inviata al *web-service* sono inseriti nell'URL dopo il simbolo «?», in una lista, denominata *query-string*, che utilizza come separatore il simbolo «&» e in un formato che prevede una coppia nome/valore composta utilizzando il simbolo «=». Per rispettare i rigidi criteri imposti nella formulazione degli URL che non prevedono la presenza di spazi, questi sono sostituiti da simboli «+».

In generale i caratteri non permessi nella formulazione degli URL sono sostituiti dal loro codice ASCII espresso con due cifre esadecimali preceduto dal simbolo «%», come riportato nella TABELLA **1**.

Il processo di codifica di una stringa nel formato previsto per un URL è noto come *URL-encoding*.

- un *web-service* espone l'accesso a «risorse» di vario tipo (dati, procedure, dispositivi, sensori, ...) identificate mediante URL;
- le risorse sono rese disponibili mediante rappresentazioni che possono essere differenziate (per esempio: XML, JSON, ...);
- le operazioni che si effettuano sulle risorse sono quelle espresse dai metodi del protocollo HTTP (GET, POST, PUT, DELETE, ...) rispettandone la semantica;
- il *web-service* non memorizza lo stato dell'interazione con il client (*stateless* server) e di conseguenza ogni richiesta da parte del client deve contenere tutte le informazioni necessarie per essere servita;
- lo stato dell'interazione tra client e server è rappresentato e gestito mediante *link*, cioè URL restituiti dal *web-service* al client, che identificano le rappresentazioni delle risorse: questo principio è definito HATEOAS (*Hypermedia As The Engine Of Application State*).

OSSERVAZIONE Dato che è possibile realizzare un *web-service* utilizzando il protocollo HTTP senza rispettare tutti i principi individuati da Fielding, i *web-service* che implementano l'architettura REST in modo integrale sono definiti ***RESTful***. I sostenitori dell'architettura *RESTful* sostengono che solo i *web-service* che la implementano presentano le stesse caratteristiche di estendibilità e di capacità di diffusione su larga scala che il web ha dimostrato di avere. Il termine REST viene infatti spesso utilizzato per definire *web-service* che si limitano a utilizzare HTTP come protocollo per l'invocazione di un servizio.

Con il termine **web API** sono usualmente denominate le specifiche di composizione degli URL che permettono di accedere a un determinato *web-service* e i formati delle risposte restituite (ed eventualmente dei dati associati alle richieste). L'insieme dei *web-service* disponibili in rete viene spesso definito ***programmable web***: questa definizione enfatizza la possibilità di sviluppare applicazioni software che interagiscono con le risorse e i servizi disponibili in rete e che eventualmente espongono a loro volta risorse e servizi ad altre applicazioni mediante una web API.

Interagire con un *web-service* REST mediante un programma sviluppato con il linguaggio di programmazione Java è relativamente semplice ricorrendo alle funzionalità della classe *URLConnection*, i cui metodi fondamentali sono descritti nella TABELLA **2**[1] a pagina seguente.
La classe derivata *HttpURLConnection* estende *URLConnection* aggiungendo alcuni metodi per la gestione delle caratteristiche specifiche del protocollo HTTP, tra cui quelli riportati nella TABELLA **3** a pagina seguente.
Dalla classe *HttpURLConnection* deriva la classe *HttpsURLConnection* che gestisce il protocollo sicuro HTTPS anziché HTTP: normalmente non è necessario operare in modo diverso utilizzando l'una o l'altra classe.
La classe *HttpURLConnection* definisce come attributi statici tutti i possibili valori dello *status-code* di una risposta del protocollo HTTP.

TABELLA **2**

Metodo	Descrizione
`void setRequestProperty(String Key, String value)`	Imposta mediante una coppia chiave/valore un *header* di richiesta.
`void setConnectTimeout(int ms)`	Imposta il tempo massimo di attesa in millisecondi per la richiesta di connessione.
`void setReadTimeout(int ms)`	Imposta il tempo massimo di attesa in millisecondi per la ricezione di dati.
`void setDoInput(boolean active)`	Abilita/disabilita la ricezione dati dalla risorsa web.
`void setDoOutput(boolean active)`	Abilita/disabilita la trasmissione dati verso la risorsa web.
`void connect()`	Esegue la connessione alla risorsa web identificata dall'URL fornito come argomento del costruttore.
`int getContentLength()`	Restituisce il valore dello *header content-length* della risposta.
`String getContentType()`	Restituisce il valore dello *header content-type* della risposta.
`String getContentEncoding()`	Restituisce il valore dello *header content-encoding* della risposta.
`long getDate()`	Restituisce il valore dello *header date* della risposta (ms trascorsi da 00:00:00 1/1/1970).
`long getExpiration()`	Restituisce il valore dello *header expires* della risposta (ms trascorsi da 00:00:00 1/1/1970).
`long getLastModified()`	Restituisce il valore dello *header last-modified* della risposta (ms trascorsi da 00:00:00 1/1/1970).
`InputStream getInputStream()`	Restituisce lo *stream* per la ricezione del contenuto del *body* della risposta.
`OutputStream getOutputStream()`	Restituisce lo *stream* per la trasmissione del *body* della richiesta.

TABELLA **3**

Metodo	Descrizione
`void setRequestMethod (String method)`	Imposta il metodo HTTP di richiesta (GET, POST, HEAD, OPTIONS, PUT, DELETE o TRACE).
`int getResponseCode()`	Restituisce lo *status-code* della risposta HTTP.
`String getResponseMessage()`	Restituisce il messaggio relativo allo *status-code* della risposta HTTP.
`void disconnect()`	Esegue la disconnessione dalla risorsa web.

ESEMPIO

Il costruttore della seguente classe interroga il servizio di *geocoding* esposto da Google e salva in un file di testo la risposta ricevuta in formato XML:

```
import java.net.*;
import java.io.*;

public class Geocoding {

   private String prefix = "http://maps.googleapis.com/maps/api/geocode/xml?address=";

   private String url;
   private String filename;
   private boolean saved = false;

   public Geocoding(String address, String filename) {
      URL server;
      HttpURLConnection service;
      BufferedReader input;
      BufferedWriter output;
      int status;
      String line;

      this.filename = filename;
      try {
```

```
// costruzione dello URL di interrogazione del web-service
url = prefix + URLEncoder.encode(address, "UTF-8");
server = new URL(url);
service = (HttpURLConnection)server.openConnection();
// impostazione header richiesta
service.setRequestProperty("Host", "maps.googleapis.com");
service.setRequestProperty("Accept", "application/xml");
service.setRequestProperty("Accept-Charset", "UTF-8");
// impostazione metodo di richiesta GET
service.setRequestMethod("GET");
// attivazione ricezione
service.setDoInput(true);
// connessione al web-service
service.connect();
// verifica stato risposta
status = service.getResponseCode();
if (status != 200) {
  return; // non OK
}
// apertura stream di ricezione da risorsa web
input = new BufferedReader(
            new InputStreamReader(service.getInputStream(), "UTF-8"));
// apertura stream per scrittura su file
output = new BufferedWriter(new FileWriter(filename));
// ciclo di lettura da web e scrittura su file
while ((line = input.readLine()) != null) {
  output.write(line);
  output.newLine();
}
```

```
            input.close();
            output.close();
            saved = true;
        }
        catch (IOException exception) {
        }
    }

    public boolean isSaved() {
        return saved;
    }

    public static void main(String[] args) {
        if (args.length < 2)
            System.err.println("Invocazione con argomenti errati.");
        else {
            Geocoding geocoding = new Geocoding(args[0], args[1]);
            if (geocoding.isSaved())
                System.out.println("Risposta servizio salvata nel file XML.");
            else
                System.err.println("Errore interrogazione servizio.");
        }
    }
}
```

Il metodo *main* esemplifica l'uso della classe: ne crea un'istanza che automaticamente crea un file in formato XML solo se si è ricevuta una risposta da parte del servizio; nel codice dell'esempio sia l'indirizzo sia il nome del file sono forniti come argomenti della riga di comando.

In molti casi reali – e in particolare nel caso tipico in cui si tratti di dati memorizzati in un database[2] – le risorse rappresentate da un *web-service* possono essere completamente gestite con le operazioni previste dal modello **CRUD** (***Create***, ***Read***, ***Update*** e ***Delete***):

Operazione	Descrizione
Create	Creazione di una nuova risorsa.
Read	Acquisizione della rappresentazione attuale della risorsa.
Update	Aggiornamento dello stato della risorsa.
Delete	Eliminazione della risorsa.

I metodi fondamentali del protocollo HTTP – che è stato fin dalla sua origine progettato per la gestione di risorse rappresentate in un linguaggio simbolico (come HTML, o XML, o JSON, ma anche le rappresentazioni MIME per i documenti, le immagini, i suoni, ...) – sono direttamente collegabili alle operazioni CRUD:

Operazione	Metodo HTTP
Create	POST
Read	GET
Update	PUT
Delete	DELETE

In questi casi è immediato realizzare *web-service* per i quali le operazioni sono richieste dai client rispettando la semantica dei metodi HTTP:

- il metodo **GET** permette di acquisire la rappresentazione di una risorsa;
- il metodo **POST** permette di creare una nuova risorsa;
- il metodo **PUT** permette di modificare una risorsa esistente;
- il metodo **DELETE** permette di eliminare una risorsa esistente.

Per aggiungere un nuovo punto di interesse è sufficiente inviare con il metodo POST del protocollo HTTP al *web-service* Google *Places* utilizzando il seguente URL (la parte mancante corrisponde alla API *key*):

```
https://maps.googleapis.com/maps/api/place/add/xml?key=.....
```

un documento XML