Laboratorio di Reti
Lezione 12
RMI MOBILITY
HTTP

10/12/2020

Laura Ricci

DYNAMIC CLASS LOADING: SCENARI

- thin client scaricano dinamicamente il loro codice dal server (lezione precedente)
  - semplificazione delle applicazioni
  - aggiornamento automatico in caso di nuove versioni/patch
- caricamento dinamico degli stub, da parte del client, richiesto nelle prime versioni di RMI
- polimorfismo: "il cuore" della programmazione ad oggetti.
  - caricamento dinamico delle classi essenziale per una vera programmazione ad oggetti remoti
    - lato server:
      - parametro di una invocazione remota può essere un oggetto di una sottoclasse di quella dichiarata nella interfaccia
    - lato client:
      - gli oggetti restituiti dal server al client possono essere di una sottoclasse del tipo dichiarato come parametro di ritorno

CARICAMENTO DINAMICO DI STUB

- nelle versioni antecedenti la 1.1
  - generare lo stub dell'oggetto remoto mediante `RMIC` lato server
  - quindi lo `STUB` deve essere reso disponibile al client.
- come rendere lo stub disponibile al client?
  - copiare off-line il file `.class` dello `STUB` in una directory riferita dal `CLASSPATH` del client
  - utilizzare dynamic class loading
- meccanismo automatico, evita la copia esplicita dello `STUB` dal client al server
- nelle ultime versioni non più necessario, in quanto lo `STUB` viene generato dinamicamente con il meccanismo delle reflection.

POLIMORFISMO

- nei linguaggi ad oggetti denota la possibilità di usare un oggetto di una sottoclasse `B` ogni volta viene indicato di usare un oggetto di classe `A`, con `B < A`.
- consideriamo un server remoto. passaggio di parametri al metodo remoto: cosa accade se...
  - un client passa al metodo remoto un oggetto di una sottoclasse della classe del parametro del metodo?
  - il metodo remoto restituisce un oggetto di una sottoclasse della classe che definisce il valore di ritorno del metodo?
- `JAVA RMI` supporta la possibilità di caricare dinamicamente ed automaticamente la implementazione della sottoclasse
- questa soluzione può essere utilizzato sia lato client che lato server

PASSAGGIO PARAMETRI A METODI REMOTI

Il client può passare al momento dell'invocazione di un metodo remoto:

- un oggetto di un tipo di dato primitivo (facile)
- un oggetto di un tipo (classe) sconosciuto al server remoto.
  - la (classe) dell'oggetto è passata dal client al server esplicitamente mediante altre applicazioni oppure off-line
- un oggetto di un tipo (classe) sconosciuto al server remoto. Come può accadere questo scenario?
  - il server conosce l'interfaccia, può essere compilato, attivato ed esportato
  - polimorfismo: sottoclasse di un tipo noto sia al client che al server. Il server conosce il tipo T (classe) del parametro, ma il client passa un oggetto di un sottotipo che estende T

POLIMORFISMO

```
public class Studente implements java.io.Serializable
   {
    private static final long serialVersionUID=1L;
    private String nome;
    private int matricola;
    private String corsoDiLaurea;

    public Studente (String n, int m, String c)
        {nome = n;
         matricola =m;
         corsoDiLaurea = c;}

    public String toString(){
       return ("Nome"+ nome + ", matricola=" + matricola +
                    ", corso di Laurea=" + corsoDiLaurea);}}
```

POLIMORFISMO

```
import java.rmi.*;
public interface DBStudentiServer extends Remote{
        public boolean insert (Studente s) throws RemoteException; }
```

POLIMORFISMO

```
import java.rmi.*; import java.rmi.server.*;
public class DBStudentiServerImpl extends UnicastRemoteObject implements
                                                        DBStudentiServer{
private static final long serialVersionUID = 1L;
   public DBStudentiServerImpl() throws java.rmi.RemoteException{ };
   public boolean insert (Studente s) throws RemoteException
     { // inserisce in un Data Base lo studente
       // se andato a buon fine restituisce  true, altrimenti false
       System.out.println("Inserito:"+s); return true;}
   public static void main(String args[])
      { // System.setSecurityManager(new SecurityManager());
       DBStudentiServerImpl server;
       try{ server = new DBStudentiServerImpl();
            Naming. rebind("DBServer",server);
            System.out.println("pronto per Connessioni al DB...");}
      catch (Exception e){e.printStackTrace();}}}
```

POLIMORFISMO

```
import java.io.*;
import java.rmi.*;
public class DBStudentiClient {
  final String pippo = "localhost";
  public static void main(String [] args){
     BufferedReader in = new BufferedReader (new InputStreamReader(System.in));
     String nome ="";
     int matricola=0;
     String corsoDiLaurea="";
     try{ System.out.println("Inserire Nome");
          nome = in.readLine();
          System.out.println("Inserire Matricola");
          matricola = Integer.parseInt(in.readLine());
          System.out.println("Inserire Corso di Laurea");
         corsoDiLaurea = in.readLine();}
     catch (Exception e){System.out.println(e.getMessage()); System.exit(-1);}
```

POLIMORFISMO

```
// System.setSecurityManager(new SecurityManager());

DBStudentiServer server;
try {
    server = (DBStudentiServer) Naming.lookup("DBServer");
    Studente s=new Studente(nome, matricola, corsoDiLaurea);
    server.insert(s);
    } catch (Exception e){System.out.println(e.getMessage());
}
}
}
```

POLIMORFISMO

```
public class StudenteErasmus extends Studente {
    private static final long serialVersionUID = 1L;
    private String nazione;
    public StudenteErasmus(String n, int m, String c, String naz)
        {super (n, m, c);
         nazione= naz;
         }
    public String toString() {
         return (super.toString()+ ",nazione=" + nazione);}
            }
```

POLIMORFISMO

```
import java.io.*; import java.rmi.*;
public class DBStudentiClientErasmus {
 public static void main(String [] args){
  BufferedReader in = new BufferedReader (new InputStreamReader(System.in));
  String nome =" "; int matricola=0; String corsoDiLaurea="";
  String nazione=" ";
  try{ System.out.println("Inserire Nome");
       nome = in.readLine();
       System.out.println("Inserire Matricola");
       matricola = Integer.parseInt(in.readLine());
       System.out.println("Inserire Corso di Laurea");
       corsoDiLaurea = in.readLine();
       System.out.println("Inserire Nazione di Provenieneza");
       nazione = in.readLine();}
  catch (Exception e){e.printStackTrace(); System.exit(-1);}
```

POLIMORFISMO

```
//System.setSecurityManager(new SecurityManager());
DBStudentiServer server;
try{
   server = (DBStudentiServer) Naming.lookup("DBServer");
   server.insert(new StudenteErasmus(nome, matricola,
                 corsoDiLaurea,nazione));
} catch (Exception e){System.out.println(e.getMessage());
}}}
```

INDIVIDUAZIONE CLASS REPOSITORY

- pubblicazione dei file `.class` che devono essere scaricati dinamicamente su un `server web` o `ftp`.
- questo server viene riferito come `codebase`
- come passare al server un riferimento alla `URL` del `codebase` ?
  - la rappresentazione serializzata dell'oggetto che viene trasmesso al client o al server viene annotata automaticamente con la `URL` del codebase
  - quando l'oggetto viene deserializzato, la JVM può scaricare il bytecode della classe dell'oggetto da deserializzare in modo trasparente, utilizzando l'annotazione
- i due server, quello su cui è in esecuzione l'oggetto remoto e quello che pubblica i file `.class` possono essere ospitati o meno su macchine diverse

INDIVIDUAZIONE CLASS REPOSITORY

- meccanismo per l'individuazione del codebase
- settare la proprietà `java.rmi.server.codebase` della JVM
  - il valore della proprietà è un riferimento alla URL del codebase.

- le classi RMI che effettuano la serializzazione di un oggetto
  - leggono il valore di questa proprietà
  - incorporano la URL nella rappresentazione serializzata dell'oggetto.
  - una eccezione: se la JVM ha scaricato il .class file da un altro server, la JVM incorpora nell'oggetto serializzato la URL da dove la classe è stata originariamente caricata

PROGRAM E VM ARGUMENTS

- program arguments: passati al programma e disponibili nel vettore args del main:

```
public static void main(String[] args)
```

- VM arguments: passati alla macchina virtuale, vengono utilizzati dalla JVM
- ogni VM argument corrisponde ad una proprietà della VM, ad esempio:
  - impostare il caching degli indirizzi IP risolti mediante DNS. (visto nella lezione su InetAddress)
  - un riferimento al file di policy per la sicurezza utilizzato in caso di code mobility (vedi prossime slides).
  - controllo dello heap: -Xms controlla la dimensione iniziale heap e -Xmx controlla la dimensione massima dello heap.
  - molte altre opzioni utili...(vedere documentazione)

VM PROPERTIES

- i suoi valori, in generale, possono essere:
  - impostati da linea di comando, al momento del lancio del programma. Usare opzione -D, per distinguere gli argomenti del programma da quelli della JVM

```
java  -Dnomeprop=val  MyClass
```

    o specificati, in Eclipse, attraverso la finestra `RunConfiguration`

  - impostati attraverso dei metodi della classe `System`, che sono in effetti delle "System calls".

```
System.setProperty(prop, val)
```

    dove sia prop che val sono espressioni di tipo String. Il metodo `getProperty()` consente di reperire il valore della proprietà.

- `java.rmi.server.codebase` è una delle proprietà della JVM

CODE BASE SETTING

- ad esempio, considerando una classe nomecl, posso eseguire il programma come segue:

```
java -Djava.rmi.server.codebase=http://www.di.unipi.it/~ricci nomecl
```

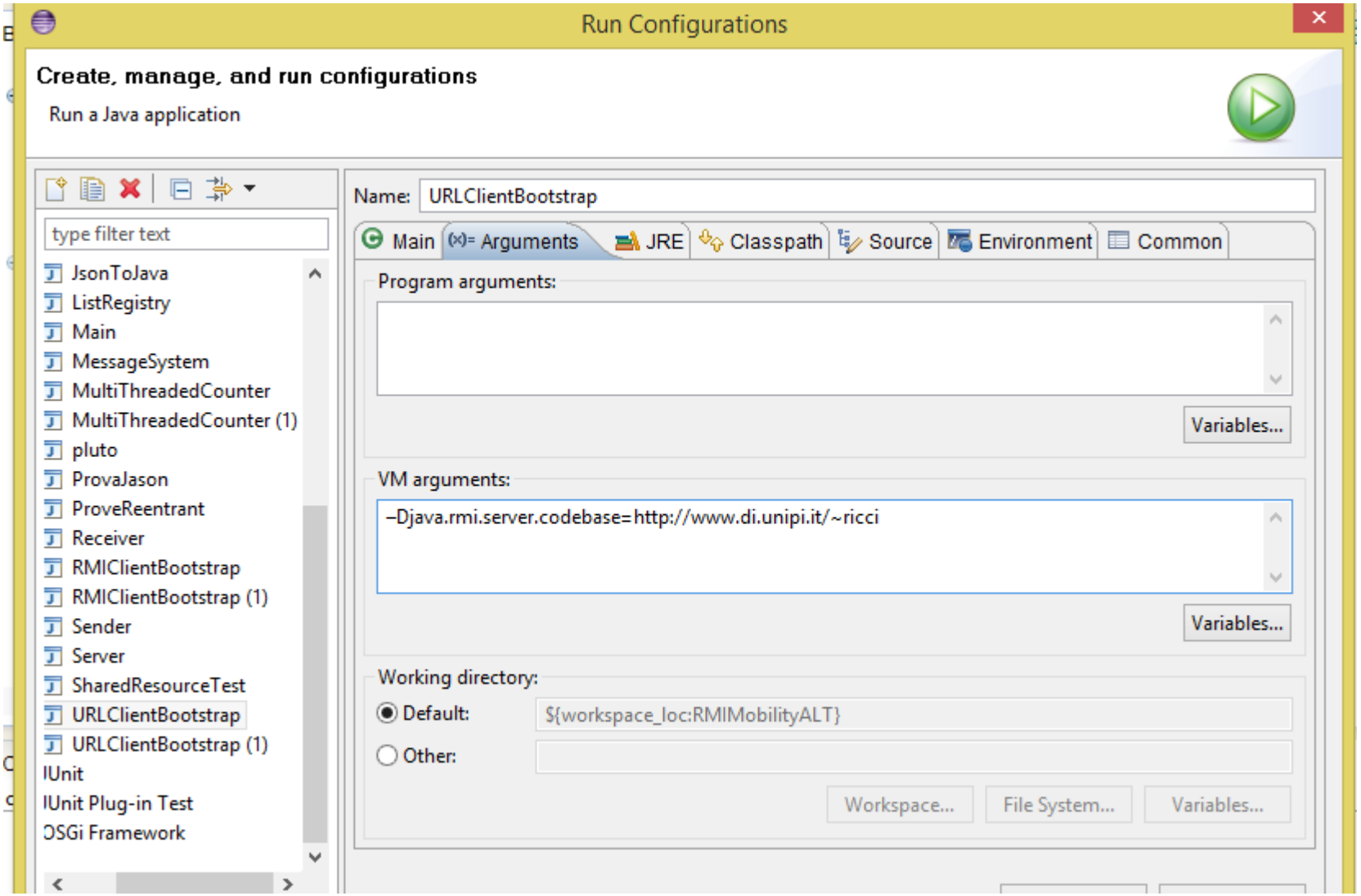
- questo comando setta la proprietà *java.rmi.server.codebase* a http://www.di.unipi.it/~ricci

- questa URL viene incorporata in ogni oggetto serializzato della classe nomecl

- un oggetto viene successivamente ricreato mediante una deserializzazione effettuata da una diversa JVM

- questa JVM non trova una copia locale del file .class associato a C (nel CLASSPATH)
  - la JVM automaticamente genera una richiesta di download alla URL specificata nell'oggetto serializzato

VM ARGUMENTS IN ECLIPSE



RICAPITOLANDO....

- quando viene effettuato il marshalling dell'oggetto, l'oggetto serializzato viene annotato con una URL:
  - se la classe dell'oggetto serializzato viene caricata dal CLASSPATH locale, la URL incorporata è il valore della proprietà locale

    `java.rmi.server.codebase`
  - se la classe dell'oggetto serializzato è stata caricata dinamicamente da un altro server, la URL incorporata è la URL di quella directory

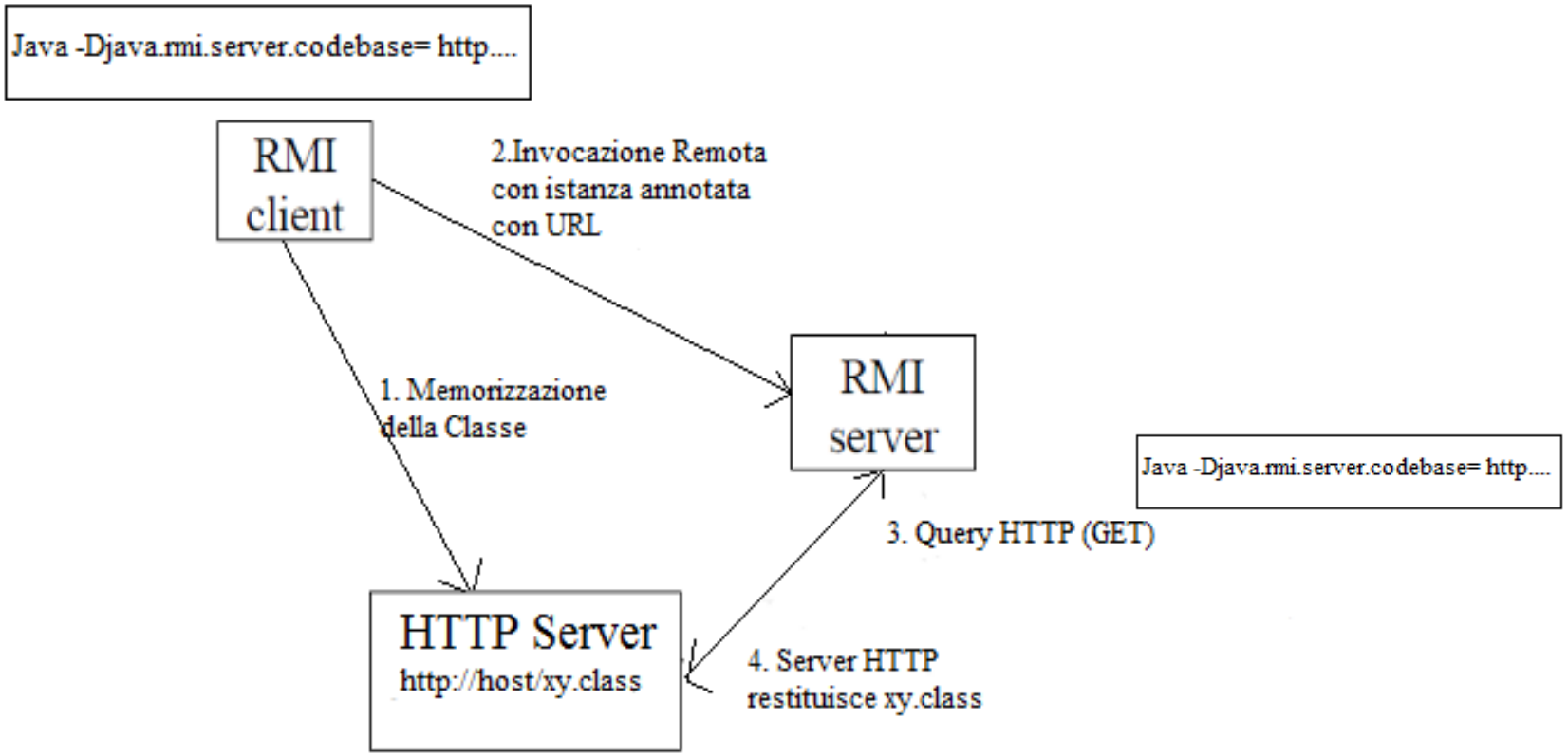
- questo avviene per tutti gli oggetti remoti serializzat
  - gli oggetti STUB
  - altri oggetti passati come argomenti a metodi remoti o restituiti come risultato di metodi remoti

RICAPITOLANDO....

- quando viene effettuato l'unmarshalling dell'oggetto (deserializzazione), se la classe dell'oggetto non è già caricata nella JVM, allora:
  - se la classe può essere reperita via CLASSPATH, viene caricata localmente
  - altrimenti viene caricata dalla URL trovata nell'oggetto serializzato

il procedimento può essere eseguito ricorsivamente

DYNAMIC LOADING STEP BY STEP



CODE MOBILITY E SICUREZZA

- la mobilità del codice pone ovvi problemi di sicurezza
- da Java 2 in poi la politica di sicurezza di JAVA impone all'utente di definire esplicitamente i permessi di cui deve disporre un'applicazione scaricata dalla rete
- tali permessi definiscono una sandbox, ovvero uno "spazio virtuale" in cui l'applicazione deve esere eseguita.
  - una sandbox dovrebbe contenere il minimo numero di permessi che consentono l'esecuizione della applicazione
  - i permessi sono elencati in un file di policy
  - due usi diversi dei file di policy
    - amministratore di sistema: utilizza un file di policy per indicare cosa possono fare i programmi lanciati all'interno del sistema
    - sviluppatore dell'applicazione: distribuisce il file di policy che elenca i permessi di cui l'applicazione ha bisogno

PASSI NECESSARI PER LA SICUREZZA

- scrivere un file di policy.
  - un file di testo che contiene lespecifiche della "politica di sicurezza" che si intende adottare. Il file deve essere memorizzato all'interno della directory del client

- definire la proprietà java.security.policy, che deve fare riferimento al file di policy che contiene la specifica della politica di sicurezza

- lanciare il programma con il seguente comando:

```
java  -Djava.security.policy=policy
```

dove policy è il nome del file di policy

- oppure specificare da programma il valore della proprietà, mediante `System.setProperty().`

INTERAZIONE CON SERVIZI WEB

- programmazione di rete a livello trasporto: “low level networking”
- a livello applicazioni: strumenti specifici per l' interazione con siti e servizi web
  - web services
- un web service è un programma che:
  - utilizza una connessione web
  - richiede ad un server esterno di fornire dati o eseguire un algoritmo
- possibili tipologie:
  - RESTful
  - RPC style (o RMI)
  - Ibrido REST-RPC

CHE COSA E' REST?

REST (Representational State Transfer)

- uno stile architetturale per sistemi software distribuiti.
- termine introdotto e definito nel 2000 nella tesi di dottorato di Roy Fielding, uno dei principali autori delle specifiche del protocollo HTTP.
- indica una serie di principi architetturali per la progettazione di Web Services.
- introdotto per diminuire la complessità dei framework "heavyweight" come SOAP
- inizialmente descritto da Fielding nel contesto del protocollo HTTP è il protocollo a livello applicazione maggiormente utilizzato, anche se sono possibili altre implementazioni
- adottato da molti website che offrono una API per l'accesso ai loro servizi
  - diverse reti sociali (Twitter, Facebook GRAPH API, Doodle..)
  - PayPal
- interazione molto spesso utilizza JSON, piuttosto che XML

SERVIZI REST

un servizio web conforme alle specifiche REST deve avere alcune specifiche caratteristiche:

- **architettura client e server**
- **stateless** ogni ciclo di request-response deve rappresentare un'interazione completa del client con il server. In questo modo non è necessario mantenere informazioni sulla sessione di un utente, minimizzando l'uso di memoria del server e la sua complessità.
- **uniformemente accessibile.** ogni risorsa deve avere un'indirizzo univoco e ogni sistema presenta la stessa interfaccia per identificare le risorse, in genere individuata dal protocollo HTTP.

SERVIZI REST

REST (Representational State Transfer): elementi fondamentali di un web service basato su architettura REST sono i seguenti

- **risorse**
  - in un sistema di commercio elettronico: prodotti ed ordini.
  - un gioco multiplayer: elementi del gioco, ad esempio robot e space shuttle
  - una risorsa può rappresentare un insieme di altre risorse:
    - insieme degli ordini effettuati da un certo utente
    - insieme di tutti i robot del gioco
- **rappresentazione delle risorse**
- **operazioni sulle risorse**

RISORSE

- ogni risorsa è identificata attraverso una URL specifica
  - semplice trasferire l'indirizzo di una risorsa ad un'altra applicazione, creare collegamenti tra risorse,..

- ad esempio:
  - un particolare avatar

    `http://a-multiplayer.com/game/robots/four-hand-george`

  - una particolare space shuttle

    `http://a-multiplayer.com/game/spaceships/destoyers/98`

  - tutti gli space shuttles

    `http://a-multiplayer.com/game/spaceships`



  - ed anche riferire la stessa risorsa da più di una URL

`http://a-multiplayer.com/game/robots/four-hand-george`

`http://a-multiplayer.com/game/spaceships/987/crew/four-hand-george`

RISORSE: RAPPRESENTAZIONE

- le risorse sono entità astratte caratterizzate da uno stato che può cambiare nel tempo
- lo stato può essere statico o dinamico

  `/game/spaceships/upgrades/567` la posizione della navicella cambia nel tempo

- client e server comunicano scambiandosi una rappresentazione dello stato delle risorse
  - giustifica il nome **RE**presentational **S**tate **T**ransfer
- il formato della rappresentazione deve essere concordato tra client e server
- ciascuna risorsa può avere rappresentazioni diverse
  - `HTML`: per la visualizzazione in un browser
  - `XML`: per l'elaborazione da parte di altre applicazioni
  - `JSON`: per l'elaborazione di un `Javascript` in una pagina web
  - `PDF`: per la stampa

RISORSE: RAPPRESENTAZIONE

- nel protocollo HTTP, lo stato di una risorsa, viene identificato attraverso la codifica definita dallo standard

  MIME (Multimedia Internet Mail Extension)

- MIME TYPE: specificato da una stringa che ha come formato
  - Type/subtype
  - Type/subtype; options

- nomi dei tipi MIME assegnati da IANA
  - text/plain formato testo semplice
  - text/plain; charset=UTF-8 testo semplice, con codifica
  - text/html formato HTML
  - application/xml formato XML
  - image/jpeg formato JPEG
  - ……
  - oltre 1000 MIME TYPEs

HTTP REQUEST

- GET /index.html HTTP 1.1

```
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:20.0)
          Gecko/20100101 Firefox/20.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

- Nessun body!
  tipico del metodo GET

HTTP RESPOND

```
HTTP 1.1 200 OK
Date: Mon, 28 Oct 2013 14:00:00 GMT
Server: Apache
Content-Length: 6448
Content-Type: text/html
Content-Encoding: gzip
Connection: Keep-Alive
<!DOCTYPE html>
<html lang="en">

<head>…
<title>…
```

OPERAZIONI

- nell'approccio REST si usa un numero limitato di operazioni per leggere o modificare lo stato delle risorse

- le operazioni corrispondono ai "metodi" o "verbi" definiti nel protocollo HTTP

Metodo	Semantica
POST	Crea una sottorisorsa
GET	Restituisce una rappresentazione della risorsa
PUT	Inizializza o aggiorna lo stato di una risorsa
DELETE	Elimina una risorsa

- Altre operazioni
  - HEAD: legge i metadati di una risorsa (GET senza response body)
  - OPTIONS: elenco di operazioni possibili su una risorsa
  - PATCH: modificare una parte di una risorsa

I METODI: SEMANTICA

- POST consente di creare risorse come subordinate ad una risorsa esistente
  - in pratica è un'operazione di "append"
  - spesso, il server risponde con: `201 Created`
  - nel response header Location è indicato l'URI della risorsa creata
    - può essere utilizzato per future `GET`, `PUT`, `DELETE`
- GET è un'operazione di sola lettura
  - può essere ripetuta senza generare effetti sullo stato della risorsa (*idempotente*) e cachata
  - questo non vuol dire che ogni volta sarà restituita la stessa rappresentazione
- PUT è una operazione di lettura e scrittura e potrebbe
  - modificare lo stato della risorsa
  - provocare effetti collaterali sul server

CLIENT REST IN JAVA

- la definizione di un client che accede ad un server REST può essere effettuata in JAVA sfruttando le seguenti classi:
  - `URL`
  - `URLConnection`
  - `HTTPURLConnection`
- più complessa la definizione di un server `REST` (non la vedremo)
- struttura di una URL

CLIENT REST IN JAVA

- le classi `URL` ed `URLConnection` incapsulano la maggior parte della complessità relativa al ritrovamento di una informazione da un sito remoto
- per specificare una URL

```
URL url= new URL("https://docs.oracle.com/javase/tutorial/
                  networking/urls/creatingUrls.html")
```

- se si richiede solamente di scaricare i contenuti di una risorsa, è possibile usare il metodo `openStream` della classe URL.

```
InputStream stream= url.openStream();
```

  - restituisce un oggetto `InputStream` object.
  - usando questo oggetto, si possono leggere I contenuti di una risorsa

- `url.openStream()` è la forma abbreviata di

```
url.openConnection().getInputStream()
```

COSTRUIRE URL

```
import java.io.BufferedReader; import java.io.IOException;
import java.io.InputStreamReader; import java.net.MalformedURLException;
import java.net.URL;
public class JavaNetURLExample {
  public static void main(String[] args) {
       try {// Generate absolute URL
            URL url1 = new URL("http://www.gnu.org");
            System.out.println("URL1: " + url1.toString());
            // stampa URL1: http://www.gnu.org
            // Generate URL for pages with a common base URL = www.gnu.org
            URL url2 = new URL(url1, "licenses/gpl.txt");
            System.out.println("URL2: " + url2.toString());
            // stampa URL2: http://www.gnu.org/licenses/gpl.txt
            // Generate URL from different pieces of data
            URL url3 = new URL("http", "www.gnu.org", "/licenses/gpl.txt");
            System.out.println("URL3: " + url3.toString());
            // stampa URL3: http://www.gnu.org/licenses/gpl.txt
```

COSTRUIRE URL

```
URL url4 = new URL("http", "www.gnu.org", 80, "/licenses/gpl.txt");
System.out.println("URL4: " + url4.toString() + "\n");
//stampa URL4: http://www.gnu.org:80/licenses/gpl.txt
// Open URL stream as an input stream and print contents to command line
try (BufferedReader in = new BufferedReader(new
                              InputStreamReader(url4.openStream())))
    {String inputLine;
     // Read the "gpl.txt" text file from its URL representation
     System.out.println("/***** File content (URL4) *****/\n");
     while((inputLine = in.readLine()) != null) {
           System.out.println(inputLine);   }
     } catch (IOException ioe) { ioe.printStackTrace(System.err); }
  } catch (MalformedURLException mue) {
           mue.printStackTrace(System.err); }
 }}
```

GET GPL.TXT...

/***** File content (URL4) *****/

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

LEGGERE CONTENUTO DA UNA URL

```
InputStream uin = url.openStream();
BufferedReader in = new BufferedReader(new InputStreamReader(uin));
String line;
while ((line = in.readLine()) != null) {
            //    process line;
```

- invia un richiesta `HTTP GET`, la risposta è disponibile nello stream

URLCONNECTION

- se sono richieste ulteriori informazioni su una risorsa, oppure si vuole modificare una risorsa e crearne una nuova occorre usare la classe `URLConnection`
  - fornisce più funzionalità rispetto alla classe URL
- invocazione del metodo `openConnection` per ottenere un oggetto `URLConnection`

```
URLConnection connection = url.openConnection();
```

- impostare proprietà della connessione come
  - `SetDoInput`
  - `SetDoOutput`
  - `SetIfModifiedSince`
  - `SetUseCaches`
  - `SetAllowUserInteraction`
  - `SetRequestProperty`
- connettersi alla risorsa remota invocando il metodo connect

```
connection.connect();
```

URLCONNECTION

```
try {
      URL u = new URL("http://www.bogus.com");
      URLConnection uc = u.openConnection();
      InputStream raw = uc.getInputStream();
       // read from URL ...
      }
catch (MalformedURLException ex) {
      System.err.println(ex);}
catch (IOException ex) {
              System.err.println(ex);
      }
```

- apre una connessione HTTP con quella URL
- vengono inviate un insieme di informazioni che descrivono il contenuto reperibile a quella URL

URLCONNECTION

- dopo aver effettuato la connessione, si ricevono diverse informazioni che descrivono il contenuto
- due metodi per reperire il valore degli header

```
public String getHeaderFieldKey (int n)
public String getHeaderField (int n)
```

- per convenienza, esistono inoltre dei metodi per reprire il valore di alcuni header standard:
  - `getContentType:` restituisce il `MIME media type` dei dati contenuti nel corpo del messaggio
    - `null`, se il content type non è disponibile
    - se il content type è `text/*`, l'header può anche contenere informazioni sul charset utilizzato,
  - `getContentLength`
  - `getContentEncoding`
  - `getDate`
  - `getExpiration`

STAMPA DEGLI HEADER

```
import java.io.*; import java.net.*;
public class AllHeaders {
   public static void main(String[] args) {
    for (int i = 0; i < args.length; i++) {
      try {
         URL u = new URL(args[i]);
         URLConnection uc = u.openConnection();
         boolean fine=false; int j=1;
         while (!(fine)) {
           String header = uc.getHeaderField(j);
           if (header == null) fine=true;
           else System.out.println(uc.getHeaderFieldKey(j) + ": " + header);
           j++;}
      } catch (MalformedURLException ex) {
          System.err.println(args[i] + " is not a URL I understand.");
      } catch (IOException ex) { System.err.println(ex);}
          System.out.println();}}}
```

HEADER: CON URL http://www.internic.net

```
Date: Wed, 11 Dec 2019 23:11:40 GMT
Server: Apache
Content-Security-Policy: upgrade-insecure-requests
Vary: Accept-Encoding
Last-Modified: Sun, 11 Jun 2017 00:01:00 GMT
ETag: "23ad-551a3e5c58700"
Accept-Ranges: bytes
Content-Length: 9133
Cache-Control: max-age=3600
Expires: Thu, 12 Dec 2019 00:11:40 GMT
X-Frame-Options: SAMEORIGIN
Referrer-Policy: origin-when-cross-origin
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
Content-Language: en
```

JSON API CLIENT

- JSONPlaceholedr: http://jsonplaceholder.typicode.com/
- Semplice JSON API REST, free ed online per accedere ad una serie di oggetti JSON, utile per
  - CodeSandbox
  - imparare ad tradurre oggetti JSON in JAVA
- contiene 6 tipi di risorse, rappresentano entità raccolti da una social network
  - posts
  - commenti
  - album
  - foto
  - utenti
  - varie

JSON API CLIENT: UN ESEMPIO DI POST

Json Input from URL: http://jsonplaceholder.typicode.com/posts/1


```
{
  "userId": 1,
  "id": 7,
  "title": "magnam facilis autem",
  "body": "dolore placeat quibusdam ea quo vitae"
}
```

JSON API CLIENT

```
public class Post {
        private int userId;
        private int id;
        private String title;
        private String body;
        public int getUserId() {
                return userId; }
        public void setUserId(int userId) {
                this.userId = userId;}
        public int getId() {
                return id;}
        public void setId(int id) {
                this.id = id;}
        public String getTitle() {
                return title;}
        public void setTitle(String title)
                 this.title = title; }
```

JSON API CLIENT

```
public String getBody() {
     return body;
     }

 public void setBody(String body) {
     this.body = body;
     }

 public String toString(){
     return this.userId+" | "+this.id +" | "+this.title+" | "+this.body;
    }
 }
```

JSON API CLIENT

```
import java.io.IOException;
import java.net.URL;
import com.fasterxml.jackson.databind.ObjectMapper;

public class ReadJsonEx {
    public static void main(String a[]){
      ObjectMapper mapper = new ObjectMapper();
      try {
       Post usrPost = mapper.readValue(
          new URL("http://jsonplaceholder.typicode.com/posts/7"), Post.class);
       System.out.println(usrPost);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

ENCODING AWARE CONTENT VIEWER

- accedere ad una risorsa tramite URL
  - un file HTML
- visualizzare il contenuto del file
  - tenendo presente della codifica utilizzata per I caratteri del file
- utilizza l'header HTTP `ContentType`
- HTTP `ContentType`
  - media type della risorsa reperita
  - immagini `image/png` `image/jpg`
  - testo `text/html; charset=UTF-8`

ENCODING AWARE CONTENT VIEWER

```
import java.io.*; import java.net.*;
public class EncodingAwareSourceViewer {
  public static void main (String[] args) {
    for (int i = 0; i < args.length; i++) {
      try {
        // set default encoding
        String encoding = "ISO-8859-1";
        URL u = new URL(args[i]);
        URLConnection uc = u.openConnection();
        String contentType = uc.getContentType();
        System.out.println("contenttype"+contentType);
        int encodingStart = contentType.indexOf("charset=");
        if (encodingStart != -1) {
            encoding = contentType.substring(encodingStart + 8);
        }
        System.out.println("encoding"+encoding); }
```

ENCODING AWARE CONTENT VIEWER

```
    InputStream in = new BufferedInputStream(uc.getInputStream());
    Reader r = new InputStreamReader(in, encoding);
    int c;
    while ((c = r.read()) != -1) {
        System.out.print((char) c);
        }
  r.close();
} catch (MalformedURLException ex) {
                System.err.println(args[0] + " is not a parseable URL");
} catch (UnsupportedEncodingException ex) {
                 System.err.println(
                "Server sent an encoding Java does not support: " +
                                                    ex.getMessage());
} catch (IOException ex) {
                    System.err.println(ex);
     }}}}
```