Il protocollo

XML-RPC

e le sue applicazioni

Federico Caboni & Stefano Sanna

GULCh – Gruppo Utenti Linux Cagliari

Agenda

- XML e HTTP
- XML-RPC
  - Introduzione
  - Il protocollo XML-RPC
  - Pro e contro di XML-RPC
- XML-RPC e Python
- XML-RPC e Java
- Applicazioni d'esempio
- Bibliografia e risorse sul web

XML e HTTP

HTTP in a nutshell

- HyperText Transfer Protocol
- “metodi” di connesione:
  - GET
    - Per prendere le pagine ;-)
    - Per inviare dati con le “query string”
    -
  - POST (usato da XML-RPC)
    - Implementa la comunicazione bidirezionale
- Headers
  - Content-Length
- Stateless

XML in a nutshell

- eXtensible Markup Language
- ***può assomigliare*** a HTML, ma...
  - tag arbitrari : <pippo unattr=”unvalore”/>
  - più formalizzato, regole più strette
    - i tag devono essere chiusi
  - XML è un linguaggio che permette di definire a sua volta linguaggi come HTML (cfr. XHTML)
- utilizzato per definire formati di dati e documenti di qualsiasi tipo
- machine readable e human readable

XML-RPC

Introduzione

- XML-RPC: XML-based Remote Procedure Call
- XML-RPC è una specifica sviluppata nel 1998 da UserLand Software per l'invocazione remota di metodi tra diverse versioni di Frontier
- XML-RPC utilizza HTTP (POST) come livello di trasporto per documenti XML
- XML-RPC è indipendente dal linguaggio di programmazione: supporta un set minimo di tipi disponibili su tutti i moderni linguaggi

Chiamate sincrone

- Essendo basato su HTTP, un server XML-RPC invia una risposta contestualmente alla connessione TCP/IP della invocazione
- La richiesta è bloccante, nel senso che l'operazione non è conclusa finché il server non restituisce il documento XML di output
- È importante interfacciare tramite XML-RPC procedure che richiedono tempi di elaborazione predicibili e non eccessivamente lunghi

Chiamate “stateless”

- L'uso di HTTP rende scorrelate due invocazioni successive ad una interfaccia XML-RPC.
- Né il protocollo XML-RPC né HTTP contengono meccanismi per tracciare la sequenza delle invocazioni:
  - È compito del programmatore inserire metadati (indicatori di sessione) per conoscere la storia dell'accesso ad una certa interfaccia
  - È consigliabile progettare procedure remote “autocontenute”, che dunque non richiedano invocazioni successive o correlate

Invocazioni

- Esempio di invocazione XML-RPC:

```
<?xml version="1.0"?>
<methodCall>
 <methodName>NOME_DEL_METODO</methodName>
 <params>
   <param>
     <value><TIPO>VALORE</TIPO></value>
   </param>
   <param>
     <value><TIPO>VALORE</TIPO></value>
   </param>
 </params>
</methodCall>
```

Invocazioni

- L'associazione tra tipi XML-RPC e tipi supportati dal linguaggio in uso dipende dalla particolare implementazione: la specifica non definisce nulla a proposito, lasciando ai programmatori completa libertà sul mapping

Risposte

- Esempio di risposta:

```
<?xml version=”1.0”?>
<methodResponse>
 <methodName>NOME_DEL_METODO</methodName>
 <params>
   <param>
     <value><TIPO>VALORE
 RESTITUITO</TIPO></value>
   </param>
 </params>
</methodResponse>
```

Eccezioni

- In caso di errore il server risponde così:

```
<?xml version="1.0"?>
<methodResponse>
 <fault><value><struct>
    <member>
     <name>faultCode</name> <value><int>4</int></value>
    </member>
    <member>
     <name>faultString</name> <value><string>Too many
  parameters.</string></value>
    </member>
   </struct></value></fault>
</methodResponse>
```

Tipi supportati

- Dati semplici
  - String
  - Numerici: Int (32bit) e Double (64bit)
  - Boolean
  - Date Time (ISO
  - Binary
- Dati composti
  - Array
  - Struct (dizionari)
- Attenzione: non esiste *null*!

String

- String è il tipo di default: se all'interno di un value non è specificato nessun tipo la specifica impone di assumere come tipo la stringa di testo.
- Esistono due rappresentazioni:
  - **<value>***stringa XML valida***</value>**
  - **<value><string>***s XML v* **</string></value>**
- Per inserire caratteri speciali è necessario utilizzare le entità (> < , &)

Int

- È definito un unico tipo intero con segno di 32 bit con range compreso tra -2147483648 (2^{31}) e 2147483647(2^{31}) – 1.
- Esistono due rappresentazioni:
  - **<value><i4>***numero***</i4></value>**
  - **<value><int>***numero***</int></value>**
- L'intero deve essere rappresentato esclusivamente con cifre e il segno (+ opzionale) senza spazi o altri simboli (dunque non sono ammesse rappresentazioni come **0x3F4D**).

Double

- XML-RPC definisce un unico tipo di dato in virgola mobile a doppia precisione (64bit) con range $10^{-323.3}$-$10^{308.3.}$
- Esiste un'unica rappresentazione XML:
  - `<value><double>`*numero*`</double></value>`
- I numeri sono rappresentati da segno (+ opzionale), cifre, punto e altre cifre. Non e' ammessa altra rappresentazione (ad esempio, quella esponenziale)

Double

- Non esiste una rappresentazione per i valori INFINITE, NOT_A_NUMBER e INDETERMINATE. Per gestire tali valori è necessario ricorrere a codici d'errore (eccezioni).

Boolean

- XML-RPC definisce un tipo logico booleano:
  - `<value><boolean>1</boolean></value>`
  - `<value><boolean>0</boolean></value>`
- Non sono ammesse altre rappresentazioni!

Date e ore

- La codifica avviene secondo un *profilo* dello standard ISO8601.
- L'ora d'inizio di questo seminario dovrebbe essere:
  -
```
<value>
  <dateTime.iso8601>
    20031129T12:30:00
  </dateTime.iso8601>
</value>
```

  che indica, appunto, le 12:30 del 29 novembre 2003

Dati binari

- XML-RPC permette di inserire dati binari (ad esempio, un byte array) utilizzando la codifica base64.
- Questo permette di trasferire immagini, audio, eseguibili all'interno di una invocazione/risposta XML-RPC

```
<value>
 <base64>SGVsbG8gSjJNRSE=</base64>
</value>
```

Codifica base64

- base64 è basata su un alfabeto costituito da simboli di 6 bit.
- La codifica avviene prendendo 24 bit (3 byte), raggruppandoli in gruppi da 6 bit (4 simboli) e poi esprimendo tali simboli nel normale ASCII.
- Un file codificato con base64 è più grande rispetto al file in chiaro del 33% (si aggiungono 2 bit ogni 6).

Inserimento di un file XML...

- In base alla specifica attuale, non è possibile inserire documenti XML all'interno di una invocazione (o risposta) XML-RPC.
- Nel caso ciò fosse assolutamente necessario, è possibile:
  - specificare i caratteri speciali con le apposite entità
  - codificare il documento in base64 e poi trasferirlo come un qualsiasi altro valore binario.

Array

- Gli Array XML-RPC contengono sequenze di elementi di dati
- Gli elementi non sono necessariamente dello stesso tipo e possono essere dati semplici o composti (è dunque possibile avere array multidimensionali)

Array

- Se non specificato esplicitamente, ogni elemento è considerato di tipo String

```
<value>
 <array>
  <data>
   <value>Un valore stringa</value>
   <value><int>-354</int></value>
   <value><boolean>0</boolean></value>
  </data>
 </array>
</value>
```

Struct

- Le Struct XML-RPC permettono la codifica di array associativi (dizionari, hashtable).
- Una Struct contiene un insieme di membri, ciascuno dei quali contiene un nome e un valore
- Due particolarità:
  - I nomi (chiavi) non devono essere univoci, anche se è consigliato che lo siano
  - I membri costituiscono un insieme non ordinato (la sequenza degli elementi può essere casuale e non deve essere tenuta in considerazione)

Struct

```
<value>
 <struct>
  <member>
   <name>port</name>
   <value><string>ttyS0</string></value>
  </member>
  <member>
   <name>speed</name>
   <value><int>4800</int></value>
  </member>
 </struct>
</value>
```

Trasporto su HTTP

- La richiesta utilizza il metodo POST:
  - `POST /RPC2 HTTP/1.0`
- Sono richiesti due soli header:
  - Content-Type: `text/xml`
  - Content-Length: *lunghezza documento XML_RPC*
- Altri header possono essere utilizzati per funzioni particolari (selezione della lingua, user agent) ma non sono richiesti dalla specifica

Pro

- Permette di realizzare semplici web service in breve tempo
- In caso di scambio di dati semplici, costituisce una valida (e rapida) alternativa a sistemi quali RMI e DCOM.
- Esiste una implementazione XML-RPC per tutti i linguaggi
- Utilizzando HTTP come trasporto, è firewall-friendly :-)
- Supportato anche su telefoni cellulari

Contro

- Di fatto introduce un basso livello di astrazione nell'interfaccia tra due moduli
- Le definizioni di interfaccia sono poco flessibili (niente *implements* né *extends*):
  - Non è estendibile: si possono utilizzare solo i tipi definiti nella specifica
  - Non è ad oggetti (meglio SOAP): si torna alla programmazione procedurale!

Alternative

- Esistono valide alternative a XML-RPC:
  - SOAP
  - Corba
  - Sistemi proprietari (DCOM, RMI, Jini, Twisted PB...)

XML-RPC e Python

Le implementazioni

- Varie implementazioni
  - xmlrpclib *(batteries)*
  - SimpleXMLRPCServer *(batteries)*
  - Twisted Matrix, Zope...
- L'implementazione delle libreria standard *non* supporta l'introspezione, SSL...
  - Ma è abbastanza facile introdurre delle modifiche...
- L'implementazione Twisted è sicuramente più avanzata....
  - Ma richiede l'installazione del framework

Tipi di dati

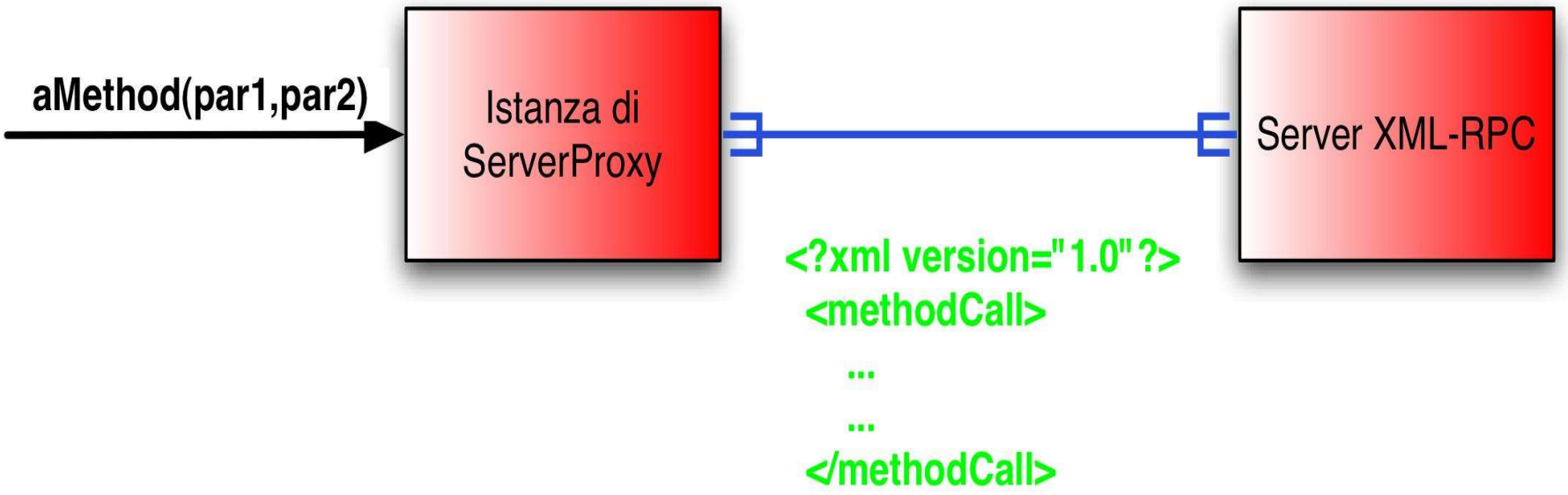
- I tipi XML-RPC vengono mappati sui corrispondenti tipi Python:

XML-RPC	Python
string	stringa unicode (Unicode)
int	intero (int)
double	float
boolean	xmlrpclib.Boolean
date.iso8601	xmlrpclib.DateTime
base64	xmlrpclib.Binary
array	lista (list)
struct	Dizionario (dict)

xmlrpclib

- Libreria fondamentale di XML-RPC per Python
- Aggiunge il supporto per il protocollo e alcuni tipi (Boolean, Binary, DateTime)
- Fornisce varie classi e factory functions
- Per scrivere un client XML-RPC, si istanzia xmlrpclib.ServerProxy, un *remote proxy* che si occuperà di connettersi e inoltrare le richieste al server.

ServerProxy



Esempio di un client

- Ecco un semplice client che usa un metodo del server *betty.userland.com*, che restituisce gli stati U.S. per codice

```
import xmlrpclib

proxy=xmlrpclib.ServerProxy
  ('http://betty.userland.com/RPC2')

try:

    print proxy.examples.getStateName(41)

except:

    print “Si è verificato un errore!”
```

-

xmlrpclib

- Oltre alla classe ServerProxy, il modulo xmlrpclib, contiene altre classi e funzioni factory utili per programmare client XML-RPC:
- **binary(bytestring)**
  - Factory, ritorna un'istanza della classe Binary che è possibile passare come tipo <base64>
- **boolean(espressione)**
  - Ritorna la valutazione booleana dell'espressione usando come istanza della classe Boolean
- **DateTime(X)**
  - x è un float che rappresenta i secondi da *Epoch*
  -
  -

SimpleXMLRPCServer

- Il modulo **SimpleXMLRPCServer**, disponibile nella libreria standard dalla versione 2.2 di Python *(prima era un download esterno)*, permette di implementare in modo semplice dei server XML-RPC in Python
- **SimpleXMLRPCServer**, come dice il nome stesso, serve a implementare dei server *semplici*: non c'è supporto per alcuna estensione di XML-RPC, per SSL, per l'introspezione...

SimpleXMLRPCServer (classe)

- Il modulo SimpleXMLRPCServer fornisce una classe con lo stesso nome, che viene istanziata fornendo l'indirizzo e la porta in cui si intende creare un server XML-RPC
- La classe fornisce a sua volta due metodi per definire i metodi da esportare:
  - **register_function(callable, name=None)**
    - Permette di aggiungere una funzione che verrà servita tramite XML-RPC. E' possibile dare alla funzione un nome differente da quello reale (ad esempio per supportare la notazione puntata)
  - **register_instance(instance)**
    - Permettere di esportare via XML-RPC l'istanza di una classe. E' possibile esportare *solo un'istanza alla volta*. L'istanza può definire un metodo **_dispatch** che viene chiamato ad ogni invocazione.

Server XML-RPC: esempio

- Una volta aggiunte funzioni o un'istanza, si fa partire il server invocando il metodo **serve_forever**.
- Ecco un semplice esempio di server “Echo” XML-RPC:
-

```
from SimpleXMLRPCServer import SimpleXMLRPCServer
myServer = SimpleXMLRPCServer(('localhost',8080))
def echo(what):
    return what
myServer.register_function(echo)
myServer.serve_forever()
```

XML-RPC con Twisted – 1

- Twisted Matrix è un framework per il networking asincrono in Python
- E' molto usato per la sua grandissima potenza: con Twisted è possibile implementare protocolli in pochissime righe di codice
- Twisted ha funzionalità XML-RPC avanzate: supporta SSL, usa un framework event-based asincrono...

XML-RPC con Twisted – 2

- Per esportare i metodi di XML-RPC, basta includerli in una classe che erediti da xmlrpc.XMLRPC
- Tutti i metodi di questa classe che abbiano prefisso **xmlrpc_** saranno esportati
- E' possibile ritornare oggetti **Deferred** per i metodi con tempi di esecuzione non immediata
- E' possibile esportare i servizi XML-RPC con un server dedicato, o come parte di un sito con altre risorse

XML-RPC con Twisted – esempio

- Ecco un esempio di un semplicissimo server “orologio”, scritto usando Twisted

```
import time
from twisted.web import xmlrpc,server
class Example(xmlrpc.XMLRPC):
    def xmlrpc_time(self):
        return time.time()
from twisted.internet import reactor
r = Example()
reactor.listenTCP(8080,server.Site(r))
reactor.run()
```

L'*introspezione*

- L'introspezione di un server XML-RPC è la capacità di fornire informazioni utili sulle proprie caratteristiche
- Esiste una convenzione “informale” di XML-RPC che consiste nel fornire questi metodi
- **array system.listMethods ()**
  - Restituisce tutti i metodi esportati
- **string system.methodHelp (string methodName)**
  - Restituisce l'help per un metodo
- **array system.methodSignature (string methodName)**
  - Restituisce la “signature” (argomenti e tipi) di un metodo
- Questi metodi sono nel “subhandler” system

Server con introspezione (1 / 2)

```
from twisted.web import xmlrpc, server
class Example(xmlrpc.XMLRPC):
    def xmlrpc_echo(self, what):
        return what
    xmlrpc_echo.signature = [['string', 'string'],
                             ['int', 'int'],
                             ['double', 'double'],
                             ['array', 'array'],
                             ['struct', 'struct']]
    xmlrpc_echo.help = “Restituisce l'argomento”

... (continua)
```

Server con introspezione (2 / 2)

- `from twisted.internet import reactor`
- `r = Example()`
- `xmlrpc.addIntrospection(r)`
- `reactor.listenTCP(7080, server.Site(r))`
- `reactor.run()`
- `----------------------------------------`
- `>>> sp.system.listMethods()`
- `['echo', 'system.methodHelp', 'system.listMethods', 'system.methodSignature']`
- `>>> sp.system.methodHelp('echo')`
- `"Restituisce l'argomento"`
- `>>> sp.system.methodSignature('echo')`
- `[['string', 'string'], ['int', 'int'], ['double', 'double'], ['array', 'array'], ['struct', 'struct']]`
-

XML-RPC e Java

GULCh – Gruppo Utenti Linux Cagliari

Libreria XML-RPC per Java

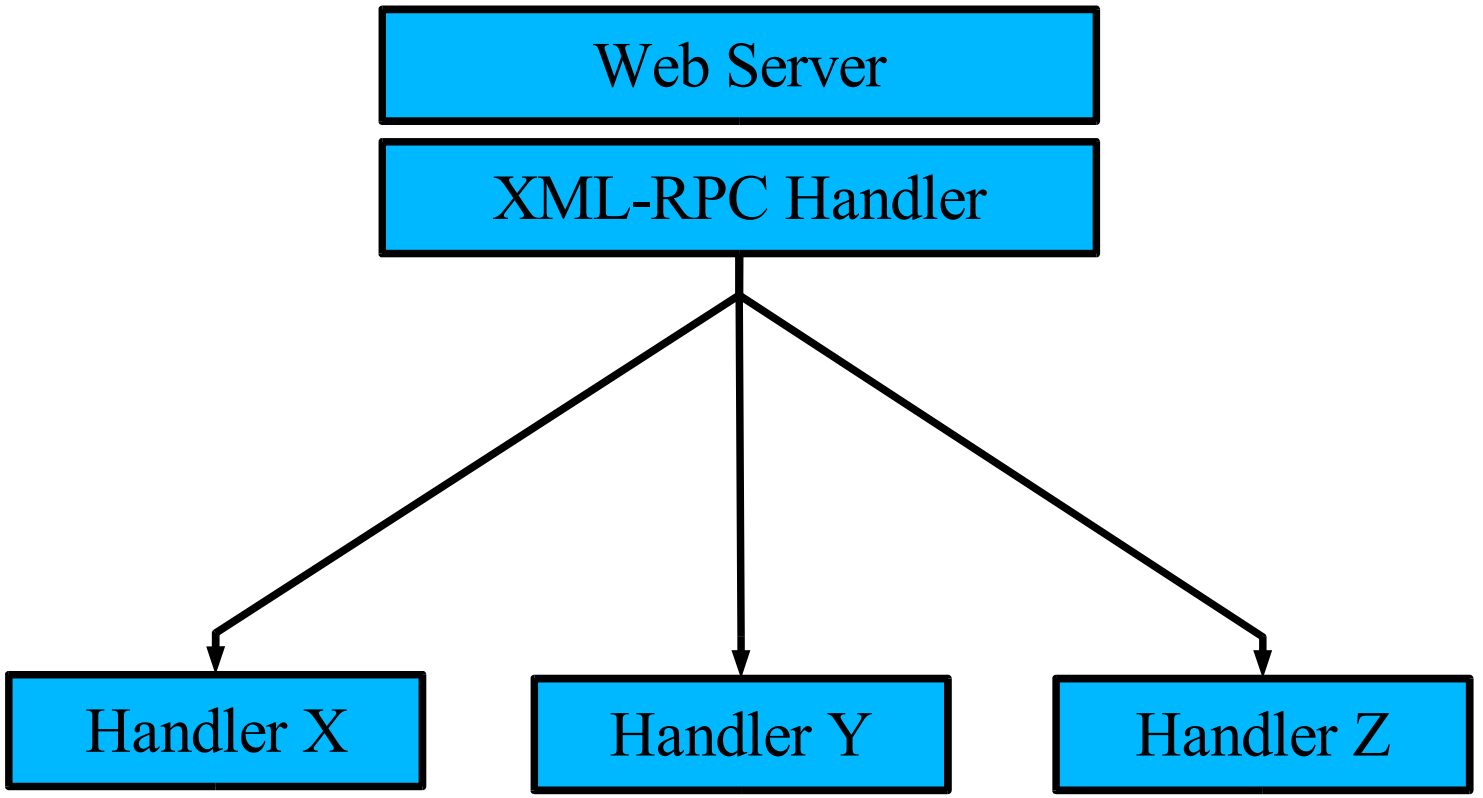
- Il progetto è stato iniziato da Hannes Wallnofer ed è poi diventato parte dell'Apache XML Project.
- La libreria fornisce:
  - Classi per realizzare di server e client XML-RPC
  - Un web server minimale per sistemi non dotati di application server dedicato
  - Classi per l'integrazione in un servlet container
  - Classi per l'integrazione di XML-RPC in Applet

Tipi di dati supportati

- I tipi XML-RPC vengono mappati sui corrispondenti tipi Java:

XML-RPC	**Java**
string	String
int	Integer (int)
double	Double (double)
boolean	Boolean (boolean)
date.iso8601	Date (java.util)
base64	java.lang.byte[]
array	Vector (java.util)
struct	Hashtable (java.util)

XML-RPC Server

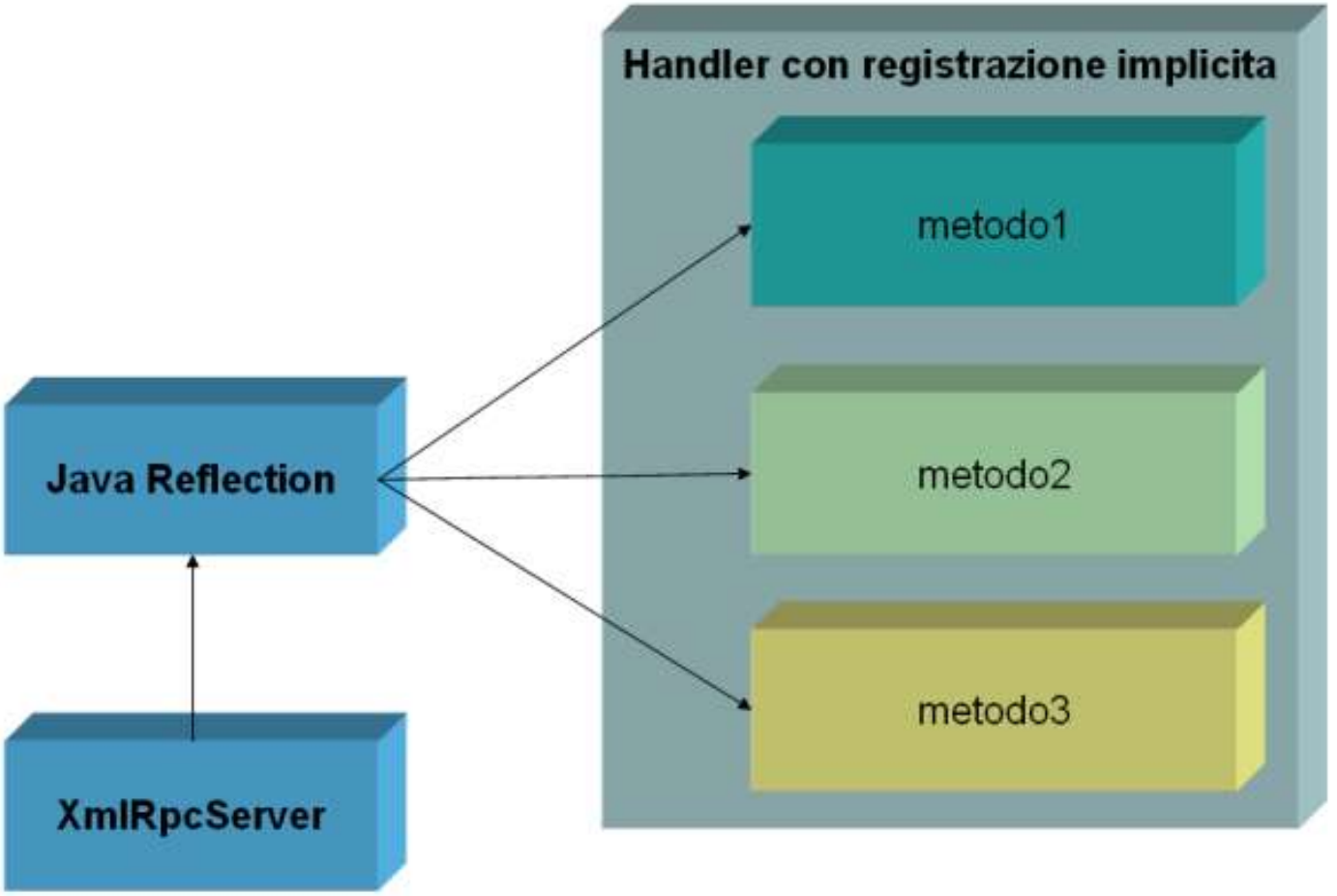


XML-RPC Server

- Un server XML-RPC risponde a invocazioni XML-RPC attraverso richieste su HTTP POST
- Le invocazioni devono essere interpretate:
  - Trasformare i parametri secondo la mappa dei tipi locale
  - Individuare oggetto e metodo al quale l'invocazione si riferisce
- L'esecutore di una invocazione è detto *handler*.

Registrazione implicita

- Il WebServer mappa automaticamente le invocazioni XML-RPC sui metodi pubblici della classe handler (registrazione implicita)

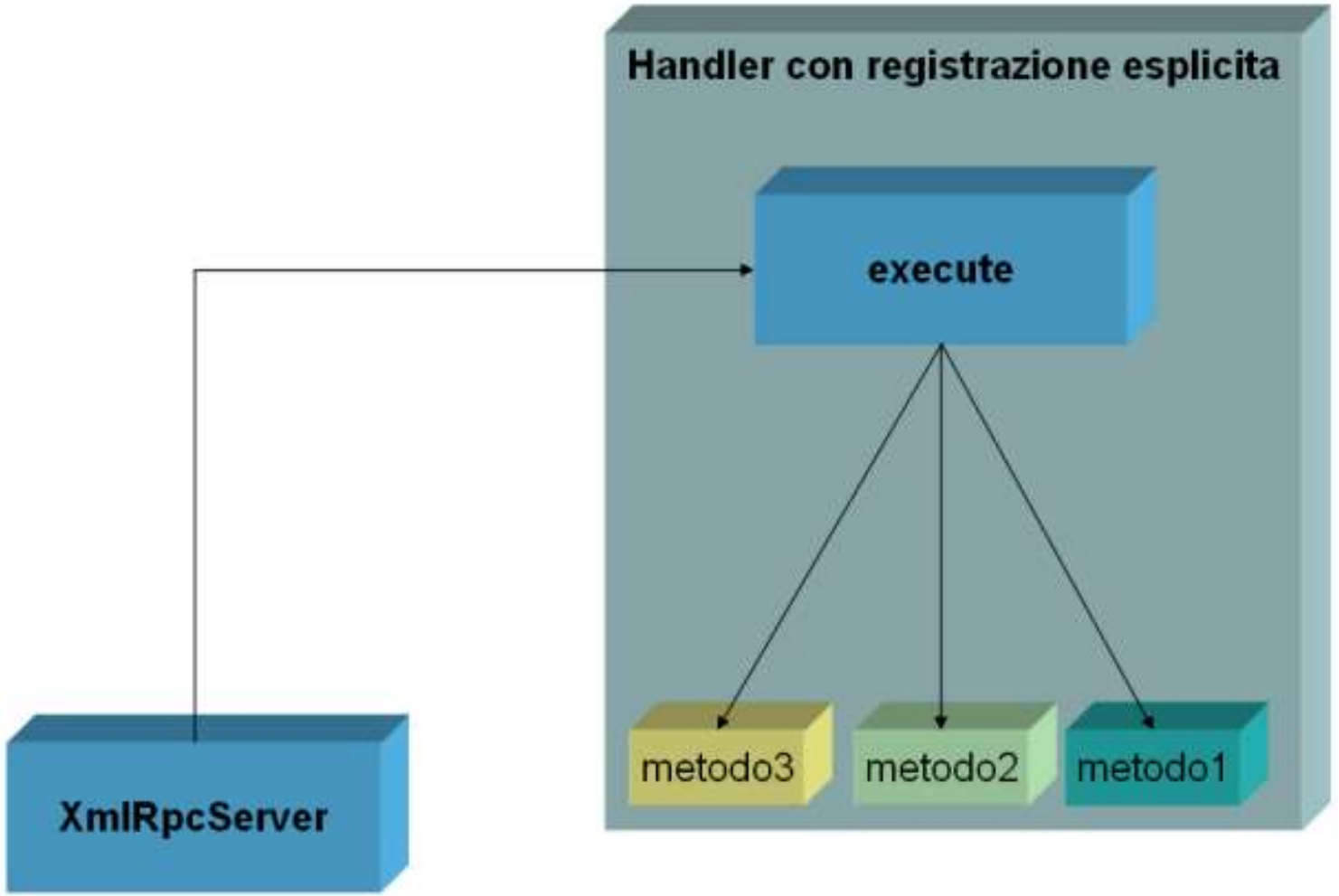


Registrazione implicita

- Può capitare che si voglia disabilitare questo meccanismo:
  - per non avere una corrispondenza 1:1 tra metodi della classe handler e signature dei metodi invocabili via XML-RPC
  - per nascondere alcuni metodi pubblici della classe handler, che devono essere accessibili da altre classi dell'applicazione ma non attraverso l'interfaccia XML-RPC

Registrazione esplicita

- Il pacchetto fornisce l'interfaccia XmlRpcHandler per la registrazione esplicita



Registrazione esplicita

- La registrazione esplicita è utile:
  - per supportare metodi con numero arbitrario di parametri (?)
  - per “arricchire” l'interfaccia dell'handler con i metodi pubblici di altre classi
  - modificare a runtime i nomi dei metodi

Scrittura di un server (1)

- Prima di tutto è necessario predisporre gli handler:

```
public class HandlerImplicito {
    public HandlerImplicito() {}
    public String getSystemInfo() {
        return System.getProperty("os.name");
    }
    public String delay() throws InterruptedException {
        Thread.currentThread().sleep(2000);
        return "I have stolen 2 seconds of your life!";
    }
}
```

Scrittura di un server (2)

- L'handler esplicito ha un solo metodo:

```
public class HandlerEsplicito implemens XmlRpcHandler {
    public HandlerEsplicito() {}
    public Object execute(String method, Vector params)
       throws Exception {
            System.out.println("Metodo: " + method);
            for (int i =0; i < params.size(); i++) {
                System.out.println("Parametro " + i + ": "
  + params.elementAt(i).toString());
            }
            return "Well done!";
       }
}
```

Scrittura di un server (3)

- Si istanzia il server, aggiungendo gli handler, ai quali è assegnato un nome:

```
WebServer server = new WebServer(3569);
server.addHandler("implicito", new HandlerImplicito());
server.addHandler("esplicito", new HandlerEsplicito());
```

- Ora il server è pronto ad accettare richieste sulla porta 3569...

Server Security

- La classe WebServer accetta connessioni provenienti da qualsiasi IP address.

- Attivando la modalita' “paranoid” è possibile limitare l'accesso al server in base all'IP del client.

Server Security

- **`setParanoid(true)`**
  - attiva la modalita' “paranoid” e tutte le connessioni in ingresso sono rifiutate
- **`addClient(“192.168.10.15”)`** oppure **`addClient (“192.168.11.*”)`**
  - accetta le connessioni da un indirizzo singolo o da un certo range
- **`denyClient(“192.168.0.15”)`** oppure **`denyClient(“192.168.11.*”)`**
  - nega l'accesso da un indirizzo singolo o da un certo range
- in caso di conflitto tra addClient e denyClient, il WebServer attribuisce priorità al denyClient.

XML-RPC Client

- Un client XML-RPC invia richieste XML ad un server remoto e riceve un XML di risposta
- La libreria gestisce automaticamente le eccezioni, trasformando i messaggi di errore XML-RPC in eccezioni Java.
- I metodi da invocare sono individuati da:
  - URL del server (http://ws.gulch.it/RPC2)
  - Nome dell'oggetto (dbsoci, mailserver...)
  - Nome del metodo (getList(), showStat())

Scrittura di un client

- XmlRpcClient incapsula le richieste:

```
XmlRpcClient client = new XmlRpcClient
                                ("http://ws.gulch.it:3569/RPC2");
Vector paramsEmpty = new Vector();
Object result = client.execute("implicito.getSystemInfo",
  paramsEmpty));
```

- I parametri sono contenuti all'interno del vettore. Anche se il metodo non accetta parametri, deve essere fornito un vettore vuoto.

Chiamate bloccanti

- Le richieste ad un server XML-RPC sono bloccanti
- Su reti veloci e per richieste semplici questo non e' un problema, ma vi sono dei casi (elaborazioni complesse) in cui è necessario implementare un modello asincrono
- Tra le tante, due strategie sono facilmente implementabili:
  - Chiamata multithread e notifica locale
  - Callback remota e server RPC locale

Chiamata multithreading

1. Il client effettua l'invocazione XML-RPC su un thread separato: l'applicazione può proseguire la normale esecuzione
2. Al termine dell'elaborazione il thread invoca un metodo dell'applicazione per restituire il valore di ritorno dell'invocazione XML-RPC

Chiamata multithreading

- Questa soluzione è ottimale per sistemi wireless:
  - Il codice non bloccante permette di offrire sempre un feedback istantaneo all'utente
  - Non è necessario avere un server XML-RPC sul dispositivo mobile (peraltro non sempre possibile...)

Invocazione asincrona

- La libreria fornisce una interfaccia che permette di invocare l'esecuzione asincrona di un metodo.
- In maniera analoga a quanto visto prima, si invoca *execute()* con in aggiunta un riferimento al *listener* della connessione.

Invocazione asincrona

```
public class ClientAsincrono implements AsyncCallback {

    public void handleError(Exception exception, URL src,
  String str) {

        System.out.println("Errore");

        exception.printStackTrace();

    }


public void handleResult(Object obj, URL src, String str) {

        System.out.println("Il metodo " + str + " e' stato
  eseguito su " + uRL.toString());

        System.out.println("Risultato: " + obj.toString());

        }

}
```

Invocazione asincrona

- Terminata l'esecuzione del metodo richiesto, la classe XmlRpcClient invocherà l'opportuno metodo sull'interfaccia AsyncCallback:

```
client.executeAsync("implicito.delay", paramsEmpty, new
                                        ClientAsincrono());
```

Callback e server locale

1. Il client effettua l'invocazione XML-RPC che ritorna immadiatamente (ad esempio, *true* se la richiesta è processabile)
2. Il server esegue l'elaborazione e effettua a sua volta una invocazione XML-RPC sul client per restituire il risultato.

Callback e server locale

- Il modello è simmetrico e particolarmente adatto in ambito LAN o su rete privata.
- Questa soluzione presenta alcuni svantaggi:
  - È necessario disporre di un server XML-RPC anche sul client (non sempre possibile su J2ME)
  - È necessario predisporre opportune politiche di sicurezza sul client
  - In ambito wireless, potrebbe capitare che il client cambi indirizzo IP durante l'elaborazione remota, rendendo impossibile l'invocazione XML-RPC da parte del server

Serializzazione di oggetti

- XML-RPC definisce un set ridotto di tipi supportati e non permette il trasporto di tipi arbitrari
- Per trasportare tipi complessi, dotati di campi interni semplici (ad esempio: appuntamento, contatto, messaggio) è possibile utilizzare delle classi wrapper basate sui dizionari.

Serializzazione di oggetti Java

- Nel caso fosse indispensabile trasferire un oggetto Java serializzato si può ricorrere al trasferimento di byte array
- Tale tecnica è indispensabile quando l'oggetto da trasferire
  - Non è composto da tipi elementari
  - Ha molti campi al suo interno
  - È di un tipo non noto in fase di design (e dunque non è praticabile la tecnica del wrapper)

Conclusioni

- XML-RPC permette di rendere “accessibile da remoto” una applicazione in maniera semplice
- È un soluzione rapida, economica e... funzionante anche in ambito wireless: molti telefonini supportano solo HTTP (niente socket!), per cui l'uso di un protocollo stile XML-RPC è praticamente obbligata.
- KDE può esportare le inferfacce dei propri componenti attraverso un server XML-RPC dedicato

Applicazioni d'esempio

Bibliografia e risorse sul web

- XML-RPC
  - http://www.xmlrpc.com
- XML-RPC Howto
  - http://www.tldp.org/HOWTO/XML-RPC-HOWTO
- Programming Web Services with XML-RPC
  - St. Laurent, Johnston, Dumbill – O'Reilly

Bibliografia e risorse sul web

- XML-RPC e Python:
  - PythonWare (xmlrpclib, SimpleXMLRPCServer):
    - http://www.pythonware.com/products/xmlrpc/index.htm
  - Twisted Matrix:
    - http://www.twistedmatrix.com/
- XML-RPC e Java:
  - Apache XML Project
    - http://ws.apache.org/xmlrpc/
  - kXML-RPC
    - http://kxmlrpc.enhydra.org

Bibliografia e risorse sul web

- Standard ISO 8601
  - http://www.iso.ch
  - ftp://ftp.informatik.uni-erlangen.de/pub/doc/ISO/ISO8601.ps.Z
- Codifica base-64 (RFC 2045)
  - http://www.ietf.org/rfc/rfc2045.txt

Questa presentazione...

... e' stata scritta con OpenOffice Impress:

OpenOffice.org 1.1
Office Suite

GULCh – Gruppo Utenti Linux Cagliari