Web services di Moodle per gestione e integrazione

Marco Ferrante
Università di Genova
marco@csita.unige.it

I web services di Moodle

I web services esposti da Moodle, oltre che all'app mobile, possono essere usati per l'integrazione di applicazioni e l'automazione di task amministrativi

Agenda

- generalità sui web services
- XML-RPC, SOAP, REST
- Moodle e web services
- attivazione di servizi
- esempi shell e PHP
- altre librerie

Nel seguito, gli esempi saranno riferiti a Moodle 3.7 e verranno descritti solo i punti meno ovvi

In principio era RPC

Il problema di eseguire (invocare) una funzione su un computer differente da quello che esegue il programma è stata affrontata a partire dagli anni '70

Il termine *Remote Procedure Call* (RPC) viene coniato ai laboratori Xerox di Paolo Alto nel 1982
Il primo standard è il Sun RPC (RFC 1831) del 1981 definito per NFS

Criticità RPC

- gestione degli errori remoti
- concorrenza
- differenze di architetture
- mancanza di variabili globali
- prestazioni
- autenticazione e sicurezza

Terminologia

serializzazione (o marshalling)

rappresentazione di un dato in modo che sia trasmissibile tra applicazioni/sistemi diversi

proxy (o stub)

un oggetto che appare locale all'applicazione ma che in realtà inoltra le richieste a un sistema remoto. Lo skeleton è la definizione di una classe che implementerà le funzioni di proxy

endpoint

indirizzo (es. IP:port o otrs@unige.it) con cui invocare un servizio

stateless

un sistema in cui ogni invocazione è indipendente dal chiamate dello stesso client che l'hanno preceduta. Contrario stateful

idempotenza

successive invocazioni della stessa funzione con gli stessi parametri non modificano lo stato del sistema

I web services, fatta semplice

Alla fine degli anni '90 viene in mente di serializzare i dati in formato testo, trasportarli su HTTP e definire gli endpoint come URL

Il primo standard diffuso è XML-RPC (1998)

XML-RPC richiesta

```
POST /RPC2 HTTP/1.0
...
Content-Type: text/xml

<?xml version="1.0"?>
<methodcall>
  <methodname>examples.getStateName</methodname>
  <params>
    <param>
        <value><i4>40</i4></value>

  </params>
</methodcall>
```

XML-RPC risposta

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodresponse>
   <params>
      <param>
         <value><string>South Dakota</string></value>

      </params>
   </methodresponse>
```

SOAP, i web services resi difficili...

Tra il 1998 e il 2001 appare SOAP (Simple Object Access Protocol)

- basato su messaggi (envelope, header e body)
- document style o RPC style
- trasporto via HTTP, SMTP, queue manager, ecc...
- estensibili (via Namespace)
- sicurezza built-in
- discovery automatico dei servizi

SOAP richiesta

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelop
  <soap:header>
  </soap:header>
  <soap:body>
    <m:getstockprice>
      <m:stockname>GOOG</m:stockname>
    </m:getstockprice>
  </soap:body>
</soap:envelope>
```

SOAP come visione

Alla base degli standard WS-* ci sono due obiettivi ideali:

- generazione di stub e skeleton direttamente dalle specifiche in modo trasparente e *language agnostic*
- autowiring a partire dalla descrizione semantica del servizio

REST

Nel 2000 Roy Fielding fa notare nella propria tesi di dottorato che:

- HTTP è già un sistema di operazioni predefinite con una semantica chiara
- gli URL sono già degli endpoint di servizi
- le risorse sono già il payload con i metadata trasportati da SOAP

Propone così i principi dello stile architetturale REST (REpresentational State Transfer)

I principi REST

- client–server
- stateless
- cacheable
- layered
- uniform interface

Le risorse

Una risorsa ha un nome (URL) e una rappresentazione (XML, SOAP, JSON, ...)

Una collezione di risorse è essa stessa una risorsa

Le risorse si manipolano con metodi HTTP

I riferimenti ad altre risorse all'interno di una risorsa sono URL

MoodleREST

tutti i corsi:

http://my.moodle/courses/

il corso MM19:

http://my.moodle/courses/MM19

gli utenti con ruolo globale manager:

http://my.moodle/users/managers/

CRUD mappato sui metodi GET, PUT, POST, DELETE

Ma per essere RESTful...

```
1  GET /courses/enroll/MM19 HTTP/1.1
2  Host: my.moodle
3
4  <?xml version="1.0" encoding="UTF-8"?>
5  <response>
6  <multiple>
7  <single>
8  <key name="id"><value>730</value>
9  </key>
10 <key name="username"><value>50000005</
11 </key>
12 <key name="firstname"><value>Marco</valu
13 </key>
14 <key name="lastname"><value>Ferrante</val
15 ...
16 </key></single></multiple></response>
```

... manca qualcosa

```
1  GET /courses/enroll/MM19 HTTP/1.1
2  Host: my.moodle
3
4  <?xml version="1.0" encoding="UTF-8"?>
5  <response>
6  <multiple>
7  <single>
8  <key name="id"><value>http://my.moodle/us
9  </key>
10 <key name="username"><value>50000005</
11 </key>
12 <key name="firstname"><value>Marco</valu
13 </key>
14 <key name="lastname"><value>Ferrante</val
15 ...
16 </key></single></multiple></response>
```

Protocolli Moodle

XML-RPC

alla base di Mnet (dominio di Single SingOn di Moodle/Mahara)

SOAP

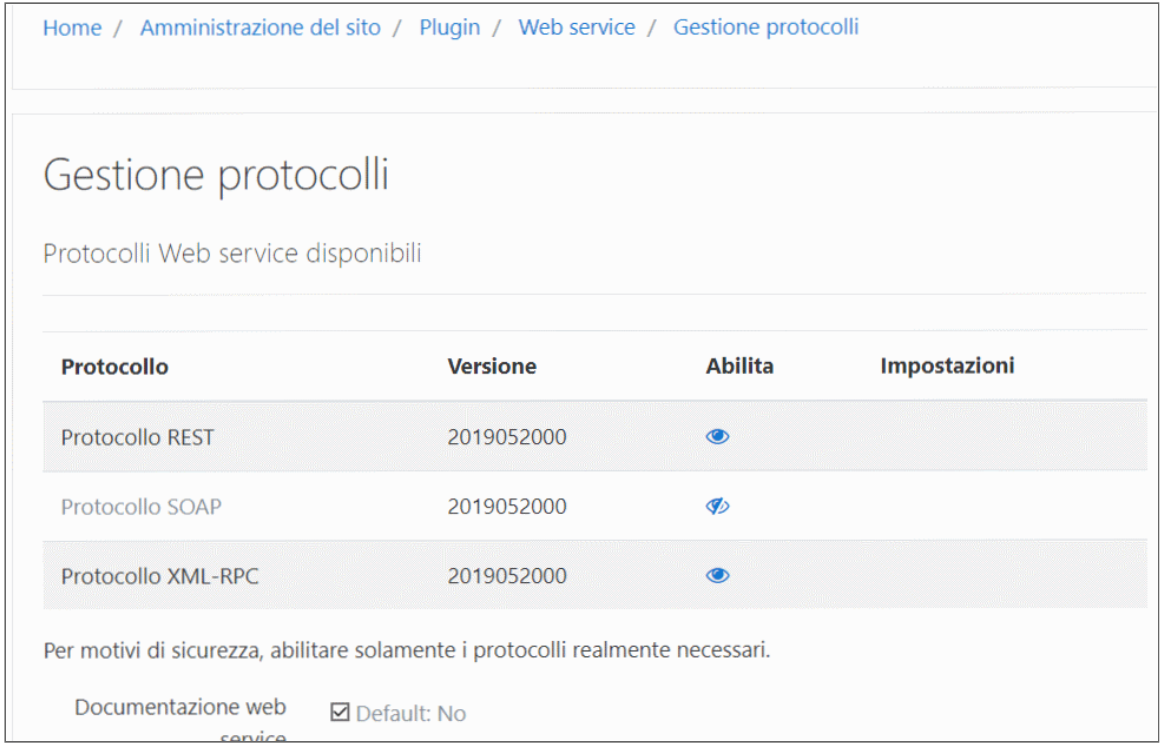
non ha il WSDL adatto per generare gli stub

un "protocollo REST"

in realtà non proprio RESTful...

Gli esempi che seguono sono sviluppati con questo

Scelta del protocollo


Home / Amministrazione del sito / Plugin / Web service / Gestione protocolli
Gestione protocolli
Protocolli Web service disponibili
Protocollo
Versione
Abilita
Impostazioni
Protocollo REST
2019052000
Protocollo SOAP
2019052000
Protocollo XML-RPC
2019052000
Per motivi di sicurezza, abilitare solamente i protocolli realmente necessari.
Documentazione web
Default: No

Autodocumentazione

I moduli che espongono API devono fornire una funzione di documentazione
Le funzioni e relative signature effettivamente disponibili su un'istanza dipendono da:

- versione
- plugin che espongono funzioni
- plugin che modificano il comportamento di altri plugin

Ad esempio, le funzioni core mod_assign_*() dipendono dalle attività valutabili presenti e quindi i parametri cambiano a seconda della configurazione

```
mod_assign_get_submission_status()
```

corrisponde in mod/assign/externallib.php

```
public static function get_submission_status_parameters() {
...
public static function get_submission_status($assignid, $userid =
...
public static function get_submission_status_returns() {
```

externallib.php

```
    $pluginsubmissionparams = array();
    foreach ($instance->get_submission_plugins()
        if ($plugin->is_visible()) {
            $pluginparams = $plugin->get_external_p
            if (!empty($pluginparams)) {
                $pluginsubmissionparams = array_merg
                        $pluginsubmissionparams, $pluginp
            }
        }
    }
    return new external_function_parameters(
        array(
            'assignmentid' => new external_value(PAR
                    'assignment id to operate on'),
            'plugindata' => new external_single_struct
                $pluginsubmissionparams
            )
        )
    );
```

Documentazione API

Dopo aver abilitato la funzione "Documentazione API", si può consultare la voce “Documentazione web service” e poi consultare

```
Amministrazione →  Plugin → Web service → Documentazione API
```

Documentazione web service

```
assignmentid= int
plugindata[onlinetext_editor][text]= string
plugindata[onlinetext_editor][format]= int
plugindata[onlinetext_editor][itemid]= int
plugindata[files_filemanager]= int
```

Parametri

Moodle si aspetta che gli argomenti vengano passati alle funzioni esposte via web service per nome, questo stile risulta fluente in linguaggi che lo prevedono come C# e Python, ma rende il codice un po' innaturale nei linguaggi che prevedono solo argomenti per posizione

Privilegi utente

Per invocare le API, l’utente deve avere il privilegio “usa web services <protocollo>”

La corretta assegnazione dei privilegi è fondamentale altrimenti possono prodursi risultati incoerenti senza avvertimenti

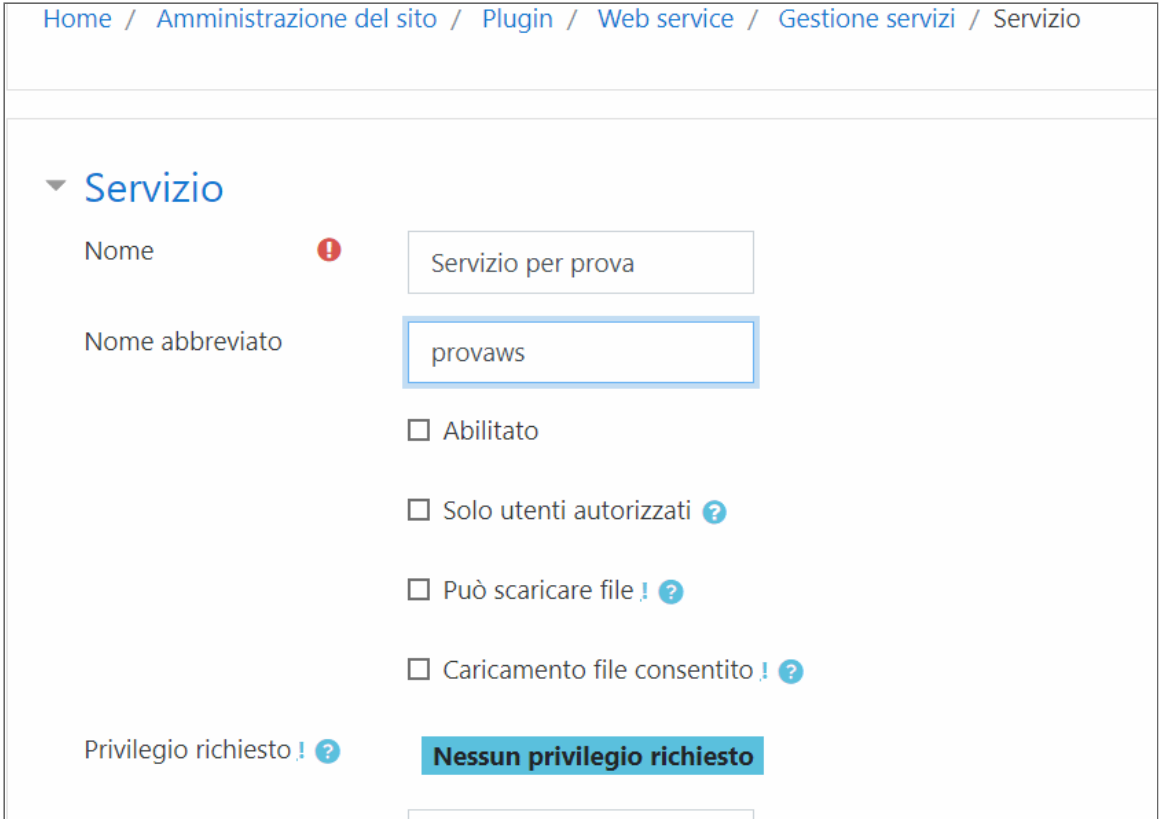
Esempio: alla richiesta dell'elenco utenti, vengono restituiti solo quelli iscritti a certo corsi

I "servizi" Moodle

Moodle organizza le API in *servizi* definiti da:

- un nome
- un insieme di funzioni esposte
- un insieme di privilegi (conviene crearne uno *ad-hoc*)
- un insieme di utenti autorizzati

I servizi non usano un protocollo specifico, ma l'accesso può essere limitato dai privilegi assegnati al ruolo


Home / Amministrazione del sito / Plugin / Web service / Gestione servizi / Servizio
Servizio
Nome
Servizio per prova
Nome abbreviato
provaws
Abilitato
Solo utenti autorizzati
Può scaricare file
Caricamento file consentito
Privilegio richiesto
Nessun privilegio richiesto


Cerca

I token

Il token, specifico per utente e per servizio, può essere ottenuto in vari modi:

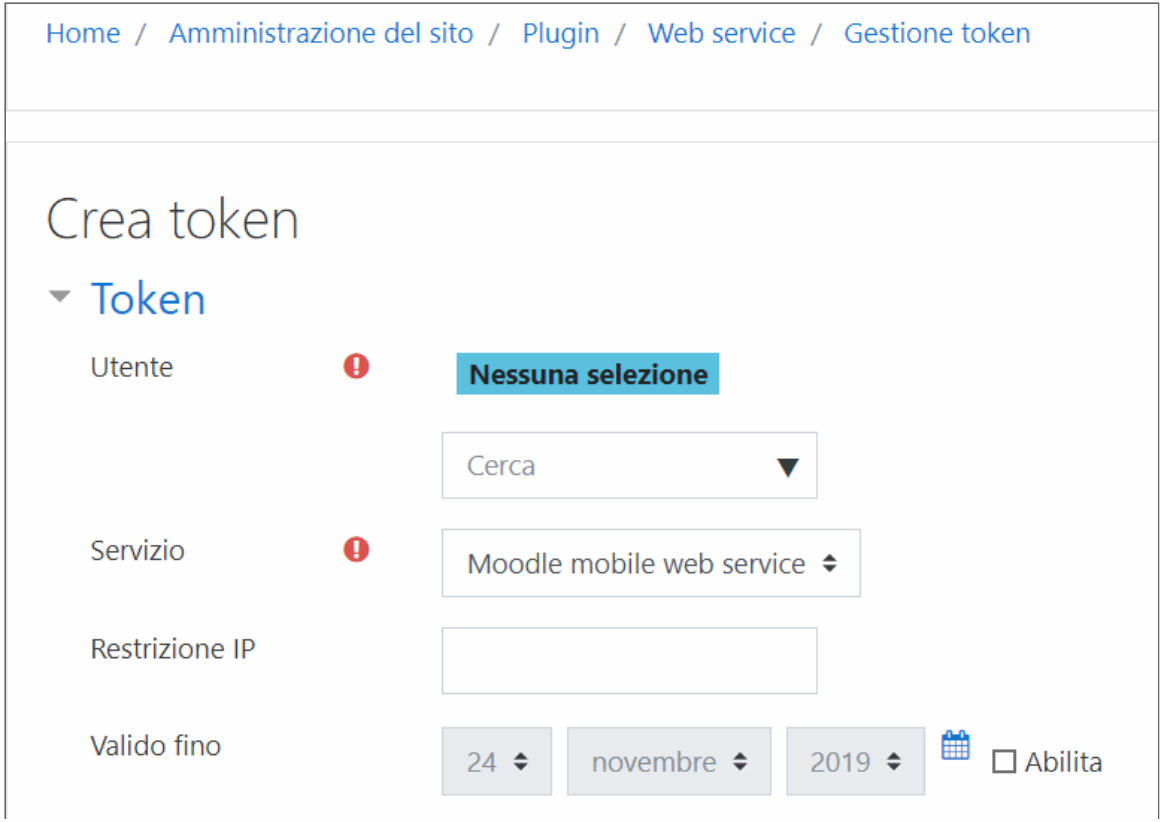
- `Amministrazione → Plugin → Web service → Gestione token`

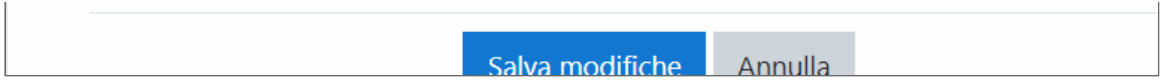
  è possibile restringere gli indirizzi IP validi dei clier
  e impostare una scadenza del token

- `Preferenze → Chiave di sicurezza`

  l'utente trova le proprie chiavi e può resettarle

- `https://yourmoodle/login/token.php?username=<...>&password=<...>&service=<`


Home / Amministrazione del sito / Plugin / Web service / Gestione token
Crea token
Token
Utente
Nessuna selezione
Cerca
Servizio
Moodle mobile web service
Restrizione IP
Valido fino
24
novembre
2019
Abilita


Salva modifiche
Annulla

Gestione token

E' possibile eliminare qualsiasi token ma puoi visualizzare solo i token che hai creato.

Token	Nome / Cognome	Servizio	Restrizione IP	Valido fino	Creatore	Oper
-	Servizi di Segreterie Privilegi mancanti: moodle/badges:viewotherbadges, moodle/competency:competencyview, moodle/competency:evidencedelete, moodle/competency:coursecompetencyview, moodle/site:deleteownmessage, moodle/user:manageownfiles, moodle/site:config, gradereport/overview:view	Moodle mobile web service			Servizi di Segreterie	Elimi

Esercitazione

Recuperiamo il nostro token
server **my.moodle**
nome utente **moodlemoot** password: ********
servizio: **demo_moodlemoot**

```
$ curl "https://my.moodle/login/token.php?username=moodlemoot↩
    &password=********&service=demo_moodlemoot"

{"token":"2869d667xxxxxxxxxxxxxxxxxxxxxxxx","privatetoken":"k2ItN
```

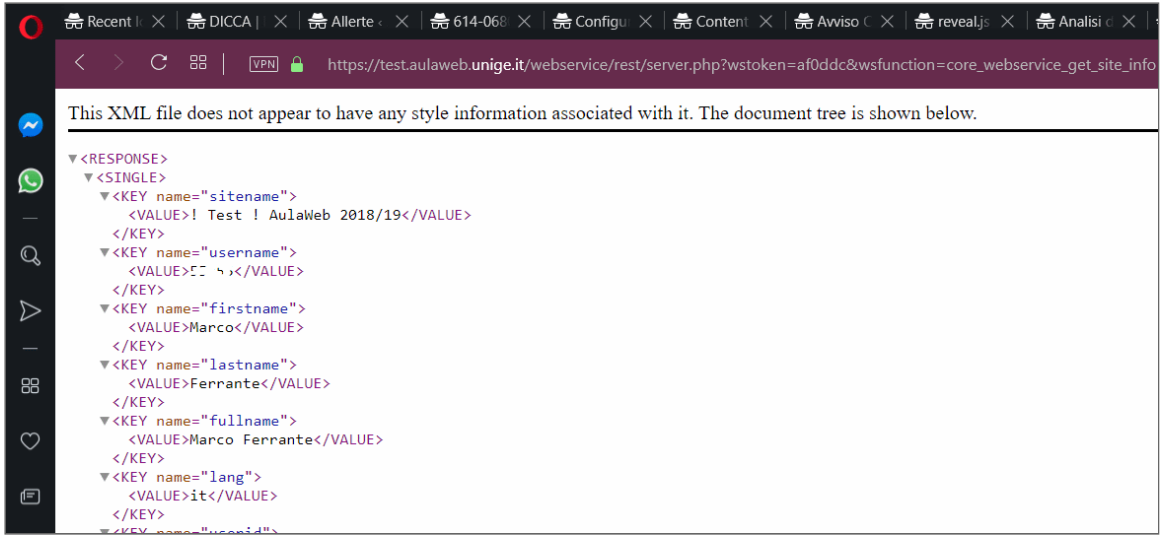
Verifica del funzionamento

Visto che REST non è veramente RESTful e usa interscambiabilmente i metodi HTTP POST e GET

```
$ curl "https://my.moodle/webservice/rest/server.php?↩
    wstoken=<...>&↩
    wsfunction=core_webservice_get_site_info"
```

Per default risultato è serializzato in XML; per JSON, aggiungere il parametro

```
&moodlewsrestformat=json
```


Recent
DICCA
Allerte
614-068
Configu
Content
Avviso
reveal.js
Analisi
VPN
https://test.aulaweb.unige.it/webservice/rest/server.php?wstoken=af0ddc&wsfunction=core_webservice_get_site_info
This XML file does not appear to have any style information associated with it. The document tree is shown below.
<RESPONSE>
<SINGLE>
<KEY name="sitename">
<VALUE>! Test ! AulaWeb 2018/19</VALUE>
</KEY>
<KEY name="username">
<VALUE></VALUE>
</KEY>
<KEY name="firstname">
<VALUE>Marco</VALUE>
</KEY>
<KEY name="lastname">
<VALUE>Ferrante</VALUE>
</KEY>
<KEY name="fullname">
<VALUE>Marco Ferrante</VALUE>
</KEY>
<KEY name="lang">
<VALUE>it</VALUE>
</KEY>

Policy sito

Piccolo problema per i siti con le sitepolicy abilitate: al primo accesso (o quando le policy vengono modificate) viene chiesto di accettarle anche via web services :-(
ma non c'è modo di farlo così (e comunque un task amministrativo non dovrebbe esserci sottoposto)

Errori

Gli errori di protocollo vengono gestiti con normali stati HTTP (es. 403 per Autorizzazione negata a livello di rete)

Gli errori di invocazione delle API restituiscono uno stato 200 con un oggetto contenente l'errore (ad esempio, autorizzazione negata per indirizzo IP non permesso dal token).

Errori, esempio

Autorizzazione negata per indirizzo IP non permesso dal token

```
<exception class="webservice_access_exception">
<errorcode>accessexception</errorcode>
<message>Eccezione nel controllo accesso</message>
<debuginfo>Invalid service - IP:10.186.20.30 is not supported↩
    - check this allowed user</debuginfo>
</exception>
```

Trasferimento file

Varie funzioni permettono l'upload di file, ad esempio core_files_upload(), ma il contenuto va codificato in *base64*, con un aumento di dimensioni di circa il 30% e un significativo sovraccarico computazionale

Sono disponibili due endpoint specifici *fuori protocollo* per il download e l'upload

Upload

L'upload è un POST con payload *multipart/form-data* i campi con nome, percorso e contenuto del file

Il file viene caricato in nell'area draft non visibile dalla propria interfaccia web e restituito un **id** utilizzabile come argomento di altre funzioni, come mod_assign_save_submission()

Upload da PHP

Attenzione che il demo ufficiale mostra:

```
$params = array('file_box' => "@".$imagepath,
    'filepath' => $filepath,
    'token' => $token);
```

ma la forma “@” è deprecata da PHP 5.6 e non più supportata da PHP 7. Usare invece la forma

```
'file_box' => new \CURLFile($imagepath)
```

perché quella vecchia fallisce, inviando un contenuto nullo, senza dare alcun tipo di errore

Esempi

Tutte le chiamate iniziano con:

```
$ curl [-L] "https://my.moodle/webservice/rest/server.php?↩
    wstoken=<...>&wsfunction=<...>"
```

Nota **curl** di default abilita il gobbling; per cui le parentesi quadre nell'URL vanno premesse dal carattere di escape oppure va invocato con **-g**:

```
&values\[0\]=<valore 1>&values\[1\]=<valore 2>&...
```

Informazioni sul sito

```
...=core_webservice_get_site_info
```


```
{"sitename":"! Test ! AulaWeb 2018\/19","username":"50000005","fi
"Marco","lastname":"Ferrante","fullname":"Marco Ferrante","lang":
"userid":730,"siteurl":"https:\/\/my.moodle","functions":[{"name"
"core_badges_get_user_badges","version":"2019052001"},{"name":
"core_block_get_course_blocks","version":"2019052001"},{"name":
"core_calendar_create_calendar_events","version":"2019052001"},
...
```

Recuperare id utente

Trova l'id interno Moodle per le successive operazioni sull'utente:

```
...=core_user_get_users_by_field&field=username&↩
    values[0]=<matricola>"
...=core_user_get_users_by_field&field=email&↩
    values[0]=demo@aulaweb.unige.it"
```

Recuperare id utente

Torna un array:

```
<?xml version="1.0" encoding="UTF-8" ?>
<response>
<multiple>
<single>
<key name="id"><value>612</value>
</key>
<key name="username"><value>xxxxxx</value>
</key>
<key name="firstname"><value>...</value>
</key>
<key name="lastname"><value>...</value>
...
</key></single></multiple></response>
```

Parametri e risultati

L'invocazione via web service ha un overhead di setup significativo, quindi quando possibile è meglio aggregare più richieste in una sola:

```
...&values[0]=<indirizzo mail 1>&↩
    values[1]=<indirizzo mail 2>&...
```

Attenzione che i risultati non sono necessariamente nello stesso ordine dei parametri

GET e POST

Il web server può avere un limite sulle dimensioni dell'URL o della query string è utile poter invocare le chiamate anche in POST con i parametri nel body codificati application/x-www-form-urlencoded

Messaggi personali

L'id utente è usato in tutte le funzioni relative ad essi, ad esempio per inviare messaggi privati:

```
$ curl -d "messages[0][touserid]=730&messages[0][text]=↩
    Testo messaggio"↩
    -H "Content-Type: application/x-www-form-urlencoded"↩
    -X POST "...=core_message_send_instant_messages"
```


```
[{"msgid":6,"text":"<p>Testo messaggio<\/p>",
"timecreated":1570549908,
"conversationid":5,"useridfrom":730,
"candeletemessagesforallusers":false}]
```

Codice e librerie

Moodle.org offre del codice dimostrativo PHP, JavaScript, Perl, Java e C#

https://github.com/moodlehq/sample-ws-clients

L'aspetto più complesso è la serializzazione dei parametri, che per supportare così tanti protocolli risulta poco intuitiva

Serializzazione PHP

Esempio: *enrol_manual_enrol_users()* richiede come parametro un array di terne

```
[ruolo, utente, corso]
```

ma non come intuitivo

```
[[r1, u1, c1], [r2, u2, c2], …]
```

bensì come unico elemento di un array con chiave 'enrolments':

```
['enrolments' => [
    ['roleid' => $r1, 'userid' => $u1, 'courseid' => $c1],
```

```
    ['roleid' => $r2, …],
]
```

Funzioni di serializzazione

Nel sample di PHP ci sono un paio di funzioni apposite da studiare

```
function format_array_postdata_for_curlcall(...

function format_postdata_for_curlcall(...
```

Sample Moodle

Fornisce un wrapper a cURL

```
$serverurl = 'https://my.moodle/webservice/rest/server.php'
$tokenurl = $serverurl . '?wstoken=' . $token;
$functionurl = $tokenurl . '&wsfunction='.$functionname;
require_once('curl.php');
$curl = new curl();
...
$resp = $curl->post($serverurl, $params);
...
```

MoodleRest

Più in "The Right Way" la libreria di Lawrence Lagerlof

```
$MoodleRest = new MoodleRest($serverurl, $token);
$groups = $MoodleRest->request('core_group_get_groups',
        array('groupids' => array(1,2)));
```

UniGe MoodleWSClient

Abbiamo sviluppato una libreria che sfrutta i *metodi magici*

```
$moodle = new \UniGe\MoodleWSClient('https://my.moodle', $token);
$site_info = $moodle->core_webservice_get_site_info();
echo "Sei collegato al sito {$site_info->sitename}\n";
```

Gli errori sono trasformati uniformemente in eccezioni
Si può clonare da GitHub
https://github.com/UniGe/MoodleWSClient
oppure installare via composer come package
unige/moodlewsclient

Installare con composer

https://getcomposer.org/download/

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup
php -r "if (hash_file('sha384', 'composer-setup.php') === '...
php composer-setup.php
php -r "unlink('composer-setup.php');"

php composer.phar require "unige/moodlewsclient dev-master"
```

Iniziare lo script con:

```
<?php
require_once 'vendor/autoload.php';
```

```
$moodle = new \UniGe\MoodleWSClient($site, $token);
...
```

Tutti i badge conseguiti

```
$user = new stdClass();
$user->userid = 3;
$badges = $moodle->core_badges_get_user_badges($user);

print_r($badges);

stdClass Object
(
    [badges] => Array
        (
            [0] => stdClass Object
                (
                    [id] => 33
                    [name] => Formazione specifica per il ...
                    [description] => Questo badge ...
                    [badgeurl] => ...
                    [timecreated] => 1561628473
                    ...
```

Recuperare un corso

```
...->$core_course_get_courses(['options' => ['ids' => [13, 14]]])

Array (
    [0] => stdClass Object (
            [id] => 13
            [shortname] => Corso base Medicina&Farmacia
            [categoryid] => 4
            [fullname] => Corso base sulla sicurezza ...
            [displayname] => Corso base sulla sicurezza ...
            [idnumber] =>
            [summary] => Il corso è attual...

...->core_course_get_courses_by_field(['field' => 'category', 'va
```

Creare o modificato un corso

```
$diff = (object)['id' => 111, 'fullname' => 'prova'];
...->core_course_create_courses(['courses' => [$course]]);  // tu

...->core_course_update_courses(['courses' => [$diff]]);  // solo
```

Iscrivere uno studente o un docente

```
$enrolments[] = (object)[
    'roleid' => $role->id, 'userid' => $user->id, 'courseid' => $
];
...->enrol_manual_enrol_users(['enrolments' => $enrolments]);
```

Il corso deve ammettere l'iscrizione manuale

Supporto ai web services nei plugin

È possibile supportare i web service in qualunque plugin
Di qui vengono i "servizi predefiniti"
La documentazione relativa è utile anche agli sviluppatori di client web services

Differenze da Moosh

Moosh https://moosh-online.com/ è una shell per Moodle.
Moosh lavora in locale sullo stesso sistema in cui è installato Moodle e ha i "superpoteri"

JavaScript

Moodle offre codice dimostrativo JavaScript di un client REST basato su jQuery

La scrittura dei parametri risulta più naturale di PHP

JavaScript

Per node.js è disponibile il modulo https://github.com/mudrd8mz/node-moodle-client

```
moodle_client.init({
    wwwroot: "http://localhost/moodle/",
    token: "d457b5e5b0cc31c05ccf38628e4dfc14"
}).then(function(client) {
    return do_something(client);
});

function do_something(client) {
    return client.call({
        wsfunction: "core_message_unblock_contacts",
        args: {
            userids: [1, 2, 3, 4, 5]
        }
```

```
    }).then(function(info) {
        console.log("Hello %s, welcome to %s", info.fullname, info.sitename
        return;
    });
}
```

Java e .net

Gli esempi in Java e C# forniti da Moodle risultano macchinosi perché REST non è il protocollo "naturale" per l'RPC in linguaggi compilati

L'ideale sarebbe generare degli *stub* dal WSDL per accedere ai web services via SOAP, ma al momento Moodle non fornisce un file di descrizione compatibile con i generatori di *stub* Java o .Net

Grazie per l'attenzione

Le domande sono gradite