

XML SUR HTTP – ARCHITECTURE REST

Eric van der Vlist (vdv@dyomedea.com)

XML sur HTTP – architecture REST

Web Services Convention

Juin 2004

HTTP (RAPPELS)

Définitions

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems (IETF).

HTTP, abréviation de hyper text transfer protocol, est un protocole informatique utilisé pour transférer des documents hypertextes ou hypermédias entre un serveur Web et un client Web. (source : Wikipédia)

Principes

Intégré dans le système d'adressage URI/URL

« http: » est l'un des services les plus utilisés dans les URL (Uniform Resource Locators) qui sont elles mêmes l'un des deux types d'URI (Uniform Resource Identifiers).

Ce système d'adressage permet non seulement d'identifier les ressources publiées sur le Web mais également de décrire les liens entre ressources.

Protocole client/serveur

HTTP est un protocole client/serveur (dont asymétrique) : un client envoie une requête et le serveur lui renvoie une réponse.

Protocole sans connexion et sans état

Le protocole HTTP par lui même est sans connexion et sans état, c'est à dire que chaque requête est considérée de manière indépendante sans qu'il ne soit demandé au serveur de mémoriser aucun contexte.

Cela n'interdit pas à un serveur Web de simuler une connexion et de maintenir un état, mais ces pratiques devraient être limitées au maximum afin de respecter les principes architecturaux du Web.

S'appuie sur les types mime (« Internet Media Types »)

Les documents transmis par les requêtes et les réponses sont décrits en suivant les types mime (rebaptisés « Internet Media Types »), ce qui permet de transmettre tout type de documents, y compris en formats binaires.

Exemple

HTTP repose sur des principes très simples comme nous pouvons le constater sur cet exemple :

Requête

La requête est exprimée par la première ligne :

La « méthode » est « GET » qui signifie « donne moi un document », le document est « / », la version du protocole HTTP utilisée ici est « 1.1 » et il se trouve sur la machine « poweredge.paris.dyomedeadea.com ».

Les autres lignes sont des « headers » qui apportent des informations complémentaires sur le contexte dans lequel cette requête est effectuée.

```
GET / HTTP/1.1 Host: poweredge.paris.dyomedeadea.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.6)
Gecko/20040413 Epiphany/1.0.8
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset:ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: Apache=10.0.0.2.139451059503845970
```

Réponse

La première ligne donne le compte rendu (ici, « 200 » signifie OK). Les lignes suivantes sont des headers donnant des informations complémentaires suivis, après un saut de ligne, par le contenu (ici un document HTML).

```
HTTP/1.1 200 OK
Date: Thu, 10 Jun 2004 12:19:17 GMT
Server: Apache/1.3.26 (Unix) Debian GNU/Linux PHP/4.1.2
ApacheJServ/1.1.2
Last-Modified: Tue, 27 Aug 2002 16:21:36 GMT
ETag: "3a802b-120-3d6ba710"
Accept-Ranges: bytes
Content-Length: 288
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
```

```
Content-Type: text /html; charset =iso-8859-1
```

```
<html> <head> <title>Under construction @ dyomedea.com</ title>
</head> <body> <h1 align=center>Under construction @<a
href="http://dyomedea.com">dyomedea.com</a></h1> < p align="center">
 <br> <a
href="mailto:vdv@dyomedea.com">vdv@dyomedea.com</a > </p> </body>
</html>
```

Les principales méthodes HTTP

GET

La sémantique de la méthode « GET » est le téléchargement d'un document.

La spécification HTTP (IETF rfc 2616) indique que cette méthode devrait être « sûre » et ne provoquer aucune action autre que le téléchargement d'une ressource et ne devrait en particulier pas provoquer d'effet de bord au niveau du serveur.

Beaucoup d'applications Web ignorent cette règle.

HEAD

La sémantique de la méthode « HEAD » est la même que celle de la méthode « GET » sauf que seuls les headers sont renvoyés dans la réponse.

Cette méthode permet donc de tester si un document existe sans le télécharger et de vérifier son type, sa date de modification, ...

POST

La sémantique de la méthode « POST » est le transfert d'information du client vers le serveur.

L'URL associée à la requête POST identifie l'action à effectuer et non l'emplacement auquel les informations transmises sont éventuellement publiées.

Si cette information est publiée sur le serveur et accessible au moyen d'une URL, le serveur peut indiquer cette adresse dans sa réponse. Dans le cas contraire, le serveur renvoi un document servant de compte rendu.

PUT

La sémantique de la méthode « PUT » est la mise à jour d'une ressource accessible sur le serveur.

La requête PUT est accompagnée du document à publier et l'URL associée est l'adresse à laquelle ce document doit être publié.

DELETE

La sémantique de la méthode « DELETE » est l'effacement physique ou logique d'une ressource accessible sur le serveur.

Sans connexion et sans état?

C'est vrai au niveau du protocole

Le protocole HTTP par lui même n'entretient aucune information de connexion ou d'état.

Pas au niveau des applications Web

Les application Web, comme toutes les application ont besoin de maintenir un contexte et si le protocole ne le gère pas, elles vont devoir le faire par elles-mêmes.

Stockage sur le poste client (URL ou cookies)

Le contexte peut être stocké sur le poste client et transmis au serveur à chaque requête notamment dans l'URL (méthode conseillée) ou sous forme de cookies (à éviter).

Stockage sur le serveur (POST, PUT, DELETE)

Le contexte peut également être stocké sur le serveur et le client de transmet plus alors dans ses requêtes qu'une information permettant d'identifier le contexte. Dans ce cas, il est conseillé d'utiliser les méthodes POST, PUT et DELETE pour gérer le contexte.

REST

Representational State Transfer (REST)

Thèse de Roy Fielding (2000)

REST a été introduit en 2000 par Roy Fielding, président de la fondation Apache et membre actif de l'IETF et éditeur de HTTP 1.1 dans le cadre d'une thèse en... philosophie!

Principes architecturaux pour applications Web

Il ne s'agit pas d'un standard, mais d'une série de principes architecturaux permettant aux applications Web de respecter les principes architecturaux du Web et donc de tirer pleinement partie de son potentiel.

Affirme le caractère sans état et sans connexion

Parmi ces principes, le plus notable est la promotion du caractère « sans état et sans connexion » du protocole HTTP au rang de principe de base que les applications ne devraient pas chercher à contourner.

REST != HTTP

REST < HTTP

Dans sa thèse, Roy Fielding liste plusieurs lacunes de HTTP qui sont gênantes pour la réalisation de systèmes REST et même une fonctionnalité (les cookies) qui est contraire à ses principes.

On peut donc écrire une application HTTP sans respecter les principes REST et REST peut donc être considéré comme un sous ensemble des fonctionnalités de HTTP.

REST = HTTP + URI + Mime

HTTP étant à la base des applications Web, REST est fortement associé à HTTP mais ne se restreint pas à HTTP.

En tant que principe architectural, REST influence également la manière d'utiliser les URI et les types mime.

REST ~ HTTP

En pratique, REST est néanmoins très fidèle à HTTP dans la mesure où il conduit à essayer d'utiliser au mieux les différentes méthodes (GET, POST, PUT, DELETE) mises à notre disposition par HTTP en respectant leur sémantique.

XML ET HTTP

XML shall be straightforwardly usable over the Internet (XML 1.0).

Le premier des « design goals » de XML est d'être facilement utilisable sur Internet et l'association de XML et de HTTP est plus que naturelle.

And it is!

Et c'est le cas puisque de nombreuses applications réalisent depuis des années des services web sans le savoir en utilisant XML sur HTTP.

XML et méthodes HTTP

Les différentes méthodes HTTP sont bien adaptées à un contenu XML.

POST : envoi d'un document XML

La méthode POST est typiquement utilisée pour envoyer un nouveau document XML d'un client vers un serveur (par exemple une commande). En réponse à cette requête, le serveur peut renvoyer un compte rendu sous forme d'un document XML mais également l'adresse à laquelle le document a été stocké et peut être retrouvé, modifié ou supprimé.

HEAD : informations sur un document XML

La méthode HEAD peut être utilisée pour récupérer des informations sur un document XML et vérifier notamment si le document a été modifié.

GET : récupération d'un document XML

La méthode GET peut être utilisée pour interroger un serveur. Elle peut servir à récupérer la commande après que le serveur ait rajouté un numéro de commande.

PUT : mise à jour d'un document XML

Lorsqu'il est nécessaire de modifier une information envoyée par POST et publiée par le serveur, on peut utiliser la méthode PUT. Elle peut servir, par exemple, à préciser l'adresse de livraison sur notre commande.

DELETE : suppression d'un document XML

Si l'on souhaite annuler la commande, pourquoi ne pas utiliser la méthode DELETE dont c'est la fonction?

EXEMPLE : SIRENE – XML

Un des premiers schémas publiés par l'ADAE (2000)

SIRENE -XML est l'un des premiers schémas à avoir été publié sur le répertoire des schémas XML de l'administration

<http://www.adae.gouv.fr/upload/documents/architecture.pdf>).

Service Web sécurisé HTTP(S)/XML

Il s'agit d'un service web utilisant HTTP sécurisé par SSL (HTTPS) pour donner accès au répertoire SIRENE. Les requêtes et les réponses sont formalisées sous forme de documents XML.

Pourrait être « plus REST »

Je pense que l'on peut dire que bien que cette application utilise XML sur HTTP, elle pourrait être d'avantage conforme aux principes REST qu'elle ne l'est.

En attribuant des URLs aux établissements

Les informations les plus détaillées descendent au niveau de l'établissement. En attribuant une URL à chaque établissement et en publiant les informations concernant l'établissement à cette adresse, on réaliserait une meilleure intégration du système sur le Web.

POST ou GET pour les interrogations?

Dans la mesure où les interrogations sont des opérations « sûres » (sans effet de bord) leur sémantique se rapproche d'avantage de celle d'une méthode GET. L'utilisation de GET interdirait d'exprimer l'interrogation sous forme d'un document XML, mais elle donnerait une URL à chaque requête et rendrait le système facilement utilisable avec un simple navigateur Web.

EXEMPLE : SONDAGES XMLFR

Distribué et accessible en XSLT

Parmi le cahier des charges du système de gestion des sondages de XMLfr, ce système devait être distribué sur deux machines et, XMLfr étant implémenté en XSLT, il devait être facilement accessible à partir de XSLT.

Service Web utilisant HTTP GET

La solution technique retenue est une implémentation sous forme de service web utilisant des méthodes HTTP GET et renvoyant les résultats sous forme de documents XML (<http://xmlfr.org/documentations/articles/021115-0002>).

La transformation XSLT constitue l'adresse à appeler en fonction des informations saisies par les utilisateurs et appelle le traitement en utilisant la fonction « document() ».

Cette solution présente donc le gros avantage d'être immédiatement intégrable dans l'architecture XSLT du site XMLfr.

Pourrait être « plus REST »

En utilisant une méthode POST ou PUT pour le vote

Le fait d'utiliser une méthode GET pour le vote a facilité l'intégration dans XSLT, mais la sémantique d'un vote est plus proche de celle d'une méthode POST ou PUT puisqu'il s'agit d'envoyer une information au serveur.

Cet écart à une utilisation optimale de HTTP semble donc le prix à payer pour la facilité d'intégration à XSLT.

REST ET GESTION DES ÉTATS

Exemples de Paul Prescod et des Monty Python

La question la plus fréquente à propos de REST est sans aucun doute celle de la gestion des états et Paul Prescod en a fourni une très bonne démonstration dans une de ses publications (http://www.prescod.net/rest/state_transition.html).

Son exemple reprend l'épreuve du passage de pont de la mort dans la quête du graal des Monty Python...

Gestion des états dans les URL

Initialisation

Le premier échange est juste une phase d'initialisation décrivant ce qui va se passer par la suite :

```
-->
GET /cross_bridge

<--
200 OK
<challenge><p>Stop!
Who would cross the Bridge of Death must answer me
these questions three,
ere the other side he see.
Do you agree?</p>
If so, <method>GET</method> your next question from
<uri>/questions/name_question</uri></challenge>
```

Première question

La première question contient la description de la question ainsi qu'une indication sur la manière de poursuivre l'échange :

```
-->
GET /questions/name_question

<--
200 OK

<challenge>
<p>Very well. What... is your name?</p>
<method>GET</method> the next question by combining your answer
with this URI: <uri>/questions/quest_question</uri>
</challenge>
```

Deuxième question

Suivant les instructions données dans l'échange précédent, la réponse à la première question est transmise dans l'URL demandant la deuxième question. Comme la première, la deuxième question contient la description de la question et une indication sur la manière de poursuivre les échanges.

```
-->
GET /questions/quest_question?name=Sir+Launcelot+of+Camelot

<--
200 OK

<challenge">
<p>What... is your quest?
<method>GET</method> the next question by combining your answer with
this URI:
<uri>/questions/hard_question?name=Sir+Launcelot+of+Camelot</uri></c
hallenge>
```

Troisième question

La troisième question suit le même principe (patience, nous approchons du but)...

```
-->
GET /questions/hard_question?name=Sir+Launcelot+of+Camelot
      &quest=To+Seek+The+Holy+Grail

<--
200 OK

<challenge>
<p>What... is the air-speed velocity of an unladen swallow?
<method>GET</method> the next question by combining your answer with
<uri>/questions/hard_question?name=Sir+Launcelot+of+Camelot
      &quest=To+Seek+The+Holy+Grail</uri></challen
ge>
```

Réponse finale

La réponse finale se fait sur le même principe et l'on note que Paul Prescod va jusqu'à utiliser un code erreur HTTP spécifique pour signifier que la réponse est erronée.

```
-->
GET /questions/hard_question?name=Sir+Launcelot+of+Camelot
    &quest=To+Seek+The+Holy+Grail&air_speed=unknown

<--
406 Not Acceptable

<p>Answer not correct!</p>
```

L'état n'est stocké ni sur le serveur ni sur le client

Un des avantages de cette stratégie est que l'état n'est stocké ni sur le serveur ni sur le client. Il n'y a donc aucun effet de bord puisque le contexte est transféré entre le client et le serveur à chaque échange.

Bien adapté à des contextes peu volumineux

Ce mécanisme élégant est facile à mettre en oeuvre et bien adapté dans le cas de contextes peu volumineux.

Stockage de l'état sur le serveur

Autorisé en faisant bon usage de HTTP

REST n'interdit pas de stocker l'état sur le serveur mais recommande dans ce cas de faire bon usage de HTTP et notamment d'utiliser les méthodes suivant leur sémantique.

Initialisation

Le dialogue s'engage de manière similaire à ce que nous avons déjà vu, mais l'on demande au client d'utiliser une méthode « POST » pour publier la confirmation de son engagement dans cet échange.

```
-->
GET /cross_bridge

<--
200 OK

<challenge><p>Stop!
Who would cross the Bridge of Death must answer me
these questions three,
ere the other side he see.
Do you agree?</p>
<method>POST</method> your answer in
<uri>/sessions</uri></challenge>
```

Première question

Lorsque la confirmation est postée, le serveur répond en indiquant l'adresse à laquelle elle serait disponible pour consultation et engage à poster la réponse à la première question :

```
-->
POST /sessions

<answer continue="true">Ask me the questions, bridgekeeper. I am not
afraid.</p>

<--
201 Created
Location: /sessions/42

<challenge session="/sessions/42">
<p>Very well. What... is your name?</p>
<method>PUT</method> your answer in
<uri>/sessions/42/name</uri>
</challenge>
```

Réponse à la première question

Paul Prescod propose de dissocier la réponse à la première question de la demande de la deuxième question :

```
-->
PUT /sessions/42/name

<answer continue="true">My name is 'Sir Launcelot of
Camelot' </answer>

<--
201 Created
Location: /sessions/42/name
...
```

Ici, /sessions/42/name est l'adresse à laquelle le nom pourrait être consulté ou modifié et il ne serait pas très propre d'utiliser cette adresse pour publier la deuxième question.

Deuxième question

La deuxième question est communiquée par un nouvel accès à /sessions/42 :

```
-->
GET /sessions/42

<--
200 OK
<challenge session="/sessions/42">
<name href="/sessions/42/name"/>,
<p>what... is your quest?
<method>PUT</method> your answer in
<uri>/sessions/42/quest</uri></challenge>
```

Il y a donc bien gestion d'un état au niveau du serveur puisque le document renvoyé à l'adresse /sessions/42 dépend des documents déjà postés sur le serveur. Ce n'est pas interdit par REST dans la mesure où nous avons utilisé les différentes méthodes conformément à leur sémantique.

Notons également que pour faciliter les développements, la session donne toutes les informations nécessaires pour récupérer les informations.

Réponse à la deuxième question et troisième question

Nous avons montré comment dissocier la réponse et l'interrogation de la session, mais HTTP n'interdit pas de renvoyer la description de la session en réponse à un « PUT » et nous pouvons le faire pour optimiser le dialogue :

```
-->
PUT /sessions/42/quest

<answer continue="true">To seek the Holy Grail.</answer>

<--
201 Created
Location: /sessions/42/quest

<challenge session="/sessions/42">
<name href="/sessions/42/name"/>,
<quest href="/quest="/sessions/42/quest"/>
<p>What... is the air-speed velocity of an unladen swallow?
<method>PUT</method> your answer in
<uri>/sessions/42/hard_question</uri></challenge>
```

Dernière réponse

```
-->
PUT /sessions/42/hard_question

<answer continue="confused">Err....What do you mean? An African or
European swallow?</answer>

<--
406 Not Acceptable

<p>Answer not correct!</p>
```

L'état est proprement géré par le serveur

L'état est stocké et géré par le serveur conformément aux principes REST et en respectant la sémantique des méthodes HTTP.

La session a son URL

Chaque session a son URL (dont la durée de vie peut être limitée) et l'interrogation de cette URL permet de connaître en détail l'état de la session.

Chaque réponse a son URL

Chaque réponse a également son URL qui peut être consultée pour vérification.

Cet exemple est généralisable

Cet exemple est facilement généralisable, par exemple à celui d'une gestion de commandes.

