The REBOL Documentation Project

-- FR - Documentation REBOL - Articles Techniques --

Articles Techniques

REBOL/Services pour les nuls (2) - Un peu plus loin

Philippe Le Goff

Première publication : 18 août 2006, et mis en ligne le vendredi 18 août 2006

Résumé :

Une petite introduction aux REBOL/services

1 Présentation générale

Les REBOL/Services ont été introduits en novembre 2005, peu après le DevCon 2005, la conférence des développeurs REBOL qui se déroulait à Milan.

Les REBOL/Services fournissent un moyen simple, élégant de partager de l'information entre les programmes informatiques. Ils peuvent être utilisés pour échanger de l'information entre des clients et des serveurs éloignés par de milliers de kilomètres, ou simplement entre applications fonctionnant localement sur votre ordinateur.

Les REBOL/Services implémentent un concept appelé "Service Orienté Architecture" (SOA). L'idée de base d'un SOA est que vous envoyez une requête à un autre programme ("un service") qui va tenter d'y répondre, et de vous renvoyer le résultat.

Il faut noter que dans les prochaines versions, les parties client et serveur seront complétement intégrées dans REBOL.

2 Installation

2.1 Récupération des codes client et serveur

Tout d'abord, récuperer les codes du serveurs et du client.

1- récupérer la partie client (avec le code proposé par coccinelle) :

```
x: skip load/header http://www.rebol.net/rebservices/client.r 3
x: head change/only/part x do copy/part x 3 3
```

```
save/header %client.r next x x/1
```

2- récupérer la partie serveur (avec le code proposé par coccinelle) :

```
x: skip load/header http://www.rebol.net/rebservices/server.r 3
x: head change/only/part x do copy/part x 3 3
save/header %server.r next x x/1
```

3- Créer un répertoire dans l'arborescence que vous avez choisie.

Par exemple, un répertoire appelé "tests-rebsrv" .

4- Y mettre les deux scripts : %server.r et %client.r

5- La structure de votre répertoire doit ressembler à cela :

```
tests-rebsrv/
    ----- server.r
    ----- client.r
```

6- Créer un fichier %reb-serveur.r qui contiendra le code suivant :

```
rebol []

do %server.r
service: start-service/options tcp://:8000
print "waiting..."
wait []
stop-service service
```

7- Créer un fichier %reb-client.r qui contiendra le code suivant :

```
rebol []

do %client.r
port: open-service tcp://localhost:8000
print result: do-service port [date]
ask "Press to quit"
```

8- Le répertoire devrait à présent ressembler à cela :

```
tests-rebsrv/
    ----- server.r
    ----- client.r
    ----- reb-client.r
    ----- reb-serveur.r
```

9- Exécutez le script %reb-serveur.r pour initier le serveur de Services.

Oh suprise, quelques petites choses apparaissent dans votre répertoire :

```
tests-rebsrv/
    ----- server.r
    ----- client.r
    ----- reb-client.r
    ----- reb-serveur.r
    ----- log.txt
    ----- public/
    ----- log-user.txt
    ----- userbase.rdb
```

Un fichier %error.txt pourra apparaître plus tard si des erreurs surviennent, dans l'appel des services.

Normalement, à ce stade, vous devez avoir une REBOL console ouverte, contenant divers messages "DEBUG ". Notamment le chargement des services par défaut est indiqué par les messages suivants :

```
DEBUG system: [start-service tcp://:8000 View 1.3.2.3.1 5-Dec-2005/18:22:54-8:00]
DEBUG service: [handle-service home "Home Service"]
DEBUG service: [handle-service admin "Administrative Service"]
DEBUG service: [handle-service file "Simple File Service"]
```

Si un service ne peut être chargé, un message d'erreur le signale.

Dans l'exemple, ici, seuls les services par défaut ont été chargés :

- home : qui fournit des services de base
- admin : les services d'administration pour gérer les connexions, la configuration, les utilisateurs.
- file : un service de gestion de fichiers assez simple, mais fort utile.

10- Exécutez à présent le script %reb-client.r pour tester votre premier client :

après quelques instants, la fenêtre de la console du client s'ouvre avec les informations suivantes :

```
[
    done [reply seq 1 service "Generic REBOL Service" commands 1 time 0:00:00.219]
    ok [date 12-May-2006/7:41:44+2:00]
]
Enter to quit
```

On peut voir que la commande [date] a bien été effectuée, par le service home (générique), en 0.219 sec, et que le résultat du traitement est "ok". La date est correctement renvoyée.

Voilà, il s'agit de votre premier appel de Rebol/service !

3 Les services génériques : home, admin, et file

Intéressons-nous à présent aux Services par défaut, pour comprendre leurs fonctionnalités, puis

nous créerons nos propres services.

3.1 home

Ce service, comme les autres génériques, est déclarés dans le code du serveur (%server.r) :

Dans le code de votre client, ajoutez en plus de la date, la commande : [commands] qui sert à afficher les commandes disponibles :

```
Rebol []

do %client.r
port: open-service tcp://localhost:8000
print result: do-service port [[date] [commands]]
ask "Press to quit"
```

Et relancez votre client. Dans la console devrait s'afficher à présent :

```
[
done [reply seq 1 service "Generic REBOL Service" commands 2 time 0:00:00.609]
ok [date 12-May-2006/7:50:51+2:00]
ok [commands service "Change the service command context" set arg opt word! "Name of
service"

    | login "Login into a user account (force authentication)"
    string! "Username" opt integer! "Encrypt strength"
    | logout "Logout out of a user account"
    | echo "Echo whatever values were passed" to end
    | client "Download the standard service client"
    | end "Terminate session (for CGI efficiency)"
    | info "Get general info about the service server" [all | any word! |
    none ] "Optional info field names"
    | commands "Return command summary for this service"
    | date "Return the server's current date/time/zone"
    | system "Get REBOL system product and version info"
    | language "Set client's target language (EN FR ...) [session]"
    set arg opt word! "Language id"
    | feedback "Send feedback to the service operator"
    | invite "Ask for an account on this service"
    | options "Pending - not implemented" to end
    | fatal "Test option - ignore"
]
]
Enter to quit
```

Les diverses commandes sont détaillées. Certaines, comme feedback ou options, ne sont pas encore implémentées.

3.2 Exemples de commandes du service home et admin :

En modifiant le bloc de commandes passés dans le script %test-client.r :

```
 [date]      ;  [login "admin"]      ;                          accéder aux fonctions
d'admin)
 [system]        ;  [admin/log info]      ;
```

La console client renverra quelque chose ressemblant à ceci.

```
 [
     done [reply seq 1 service "Generic REBOL Service" commands 4 time 0:00:00.844]
     ok [date 12-May-2006/18:26:11+2:00]
     ok [login #{E9A270A221.....754587F81202038} #{D9D2F5AB1D.....D111871F20} 3 0]
     ok [system REBOL View 1.3.2.3.1 5-Dec-2005/18:22:54-8:00]
     ok [admin/log
         file: %log.txt
         watch: true
         size: 87375
         date: 12-May-2006/18:26:14+2:00
         ]
 ]
```

En modifiant le bloc de commandes passées dans le script %test-client.r :

```
 [date]          ;
 [login "admin"]  ;
 [admin/files list]   ;  [admin/log info]    ;  [admin/user-add ["toto" "Administrator"
"password" [admin]
   toto.toto@toto.fr 5-Feb-2009/22:45:02+2:00]]
```

renverra entre autres :

```
 ...
 ok [admin/files %services/ %log.txt %log-user.txt %server.r
 %test-client.r %test-server.r   %userbase.rdb %commandes-rebol-services.txt
 %error.txt %public/ %client.r %functions.txt %objets.txt]
 ok [admin/log
     file: %log.txt
     watch: true
     size: 159451
     date: 26-Jun-2006/17:30:58+2:00
 ]
 ...
```

A noter que les services génériques sont des contextes du point de vue REBOL, d'où la nécessité de fournir le path vers les fonctions adéquates.

3.3 le service file

3.3.1 Récuperer un fichier :

En modifiant à présent les commandes de votre fichier %test-client.r, avec :

```
[file/list  ]
[file/get %taratata.txt]
```

On obtient la réponse suivante :

```
ok [file/list %./ [size: 0 date: 11-May-2006/11:04:38+2:00 type: directory
name: %uploads/]
[size: 1685 date: 27-Jun-2006/9:22:10+2:00 type: file name: %taratata.txt]]
ok [file/get %taratata.txt #{
200920090D0A09090D0A2020090D0A0D0A6E6F7576656C6C6573206F7074696F
6E73206D756C74696D656469610D0A3232206A75696E2032303036205B207072
6F6475697473206574207365727669636573205D0D0AE0206E65207061732072
6174657220737572204F72616E676520576F726C6420210D0A09200D0A202009
0D0A0D0A70726F6D6F204D4D532063617274652070 6F7374616C650D0A313920
6A75696E2032303036205B2070726F6475697473206574207365727669636573
205D0D0A456E20756E20636C69632C20656E766F79657A20756E20636C696E20
64929C696C206D756C74696DE9646961852064657075697320766F747265206D
6F62696C6520210D0A09200D0A2020090D0A0D0A6C652066696C6D206465206C
6120436F757065206475204D6F6E64650D0A3132206A75696E2032303036205B
20636F6D6D756E69636174696F6E205D0D0A746F7574207361766F6972207375
72206C652073706F74206469666667573E92070656E64616E74206C6120436F75
7065206475204D6F6E6465204649641203230303 60D0A09200D0A2020090D0A
0D0A74656D707320642761747465 6E74650D0A31206A75696E2032303036205B
2072656C6174696F6E20636C69656E7473205D0D0A4F72616E67652C20316572
206F70E97261746575722206D6F62696C6520E0206D657474726520656E20706C
616365206C652074656D707320649261747465 6E746520772617475697420 73
757220736F6E2053657276696 36520436C69656E7473204772616E6420507562
6C69630D0A09200D0A2020090D0A0D0A746F75746573206C6573206163747561
6C6974E9730D0A09200D0A0D0A090D0A090D0A2020090D0A0D0A726F616D696E
670D0A0D0A0D0A4C65207765627A696E65206EB033342C0D0A61696D65206C65
73207472656D706C696E732C20726F756C6520646573206DE963616E69717565
73206974616C69656E6E65732C0D0A6469766572676520737572206C6120636F
6E76657267656E63652C0D0A6661697420736F6E206175746F70726F6D6F2C20
6C65207072E966E87265206175206C6169742C0D0A65737420756E20626F6E20
746F75746F752C0D0AE9636F75746520456C746F6E204A6F686E206574206261
69676E652064616E73206C6520636F6E74657874652E0D0A0D0A090D0A202009
0D0A0D0A436F757065206475204D6F6E646520646573204A65757820566964E9
6F0D0A3231206A75696E2032303036205B2070617274656E6172696174732065
742073706F6E736F72696E67205D0D0A333735206761676E616E747320647520
7469726167652061752073 6F72742E2E2E206C61206C6973746520636F6D706C
E87465206574206C657320626F6E6E65732072E9706F6E736573206475207175
697A0D0A09200D0A2020090D0A0D0A41726D6F722043757020323030360D0A32
31206A75696E2032303036205B2070617274656E617269617473206574207370
```

```
6F6E736F72696E67205D0D0A6574206C65206772616E64206761676E616E7420
6573742E2E2E0D0A09200D0A2020090D0A0D0A4F72616E6765536F6C69646172
6974E9730D0A3230206A75696E20323030360D0A617070656C20E02070726F6A
65742064616E73206C6520646F6D61696E65206465206C612064E96669636965
6E6365207669737565656C6C65203A2064617465206C696D6974652064652064E9
70F4742064657320646F737369657273206C65203331206A75696C6C65740D0A
09200D0A2020090D0A0D0A64E9636F757672657A206C6573206E6F7576656175
74E9732064752070726F6772616D6D65206163740D0A3136206A75696E203230
3036205B20726573736F7572636573732068756D61696E6573205D0D0A6C697265
206C6520636F6E6E65637427666C617368206EB036370D0A09200D0A2020090D
0A0D0A6C2741464F4D20656E20616374696F6E0D0A38206A75696E2032303036
205B206A7572696469717565206574207 2E9676C656D656E7461697265205D0D
0A6C697265206C61206E6577736C657474657220706F7572206D696575782063
6F6E6E61EE747265206C61206D697373696F6E206465206C274173736F636961
74696F6E204672616EE7616973652064657320 4F70E9726174657573204D6F
62696C65730D0A09200D0A2020090D0A0D0AE976616C756174696F6E204E4578
54206D6172717565730D0A3331206D6169203230303620 5B206D617271756520
5D0D0A766F7573206176657A207061727469636970E920E020756E6520736573
73696F6E2064652073656E736962696C6973617469
} 27-Jun-2006/9:22:10+2:00]
]
...
```

A noter que l'usage du service file va générer, s'ils n'existent pas, deux répertoires : le répertoire de base : %public/ et un sous-répertoire à celui-ci : %uploads.

Dans l'exemple ci-dessus, le fichier %taratata.txt se trouvait dans le répertoire de base (%public/). Il faudrait sinon spécifier le *path* complet en partant de celui-ci.

Le contenu du fichier texte %taratata.txt est téléchargé sous forme binaire. L'usage de la fonction to-string sur la valeur binaire permet de récupérer le code du texte.

3.3.2 Uploader un fichier :

La commande à utiliser est la suivante (la commande put ne nécessite pas d'être "admin"). Elle nécessite deux arguments : le nom du fichier à créer sur le serveur, et un binaire correspondant au contenu à transmettre :

```
[file/put %apachemodule1.txt (to-binary read %apachemodule0.txt)]
```

Ici, le fichier %apachemodule0.txt (situé dans le même répertoire que votre %test-client.r) est lu, mis en binaire et transmis. Côté serveur, il prendra le nom : %apachemodule1.txt et sera généré - par défaut - dans le répertoire : %public/uploads.

et voici le résultat de la commande :

```
[
    done [reply seq 1 service "Generic REBOL Service" commands 2 time 0:00:00.687]
```

```
    ok [date 27-Jun-2006/9:52:23+2:00]
    ok [file/put %uploads/apachemodule1.txt 1907]
]
```

où 1907 est la taille en octets du fichier.

Pour effacer un fichier, il est nécessaire d'être en mode admin :

```
[login "admin"]
[file/delete %uploads/apachemodule1.txt ]
```

et voici la réponse du serveur :

```
[
    done [reply seq 1 service "Generic REBOL Service" commands 3 time 0:00:00.906]
    ok [date 27-Jun-2006/10:12:20+2:00]
    ok [login #{E28EF2E4F.....78B7BEDB165E7} #{D9D2F5AB1D5....111871F20} 3 0]
    ok [file/delete true]
]
```

4 Créer votre propre service

Après ces quelques exemples, il est temps de créer votre propre service :

1. Créer un sous-répertoire %services dans votre répertoire. Soit :

```
tests-rebsrv/
    ----- server.r
    ----- client.r
    ----- reb-client.r
    ----- reb-serveur.r
    ----- public/
    ----- services/
```

3. Le code du service :

```
rebol []

name: 'time-server   ;=> ceci devient le nom du CONTEXTE !
title: "Precise Time Server"
commands: [
    'precise-time ;=> le nom du service
    "gives time" ;=> commentaire libre
    (result: mold now/time/precise) ;=> l'action réalisée par le service
]
```

4. Arréter et relancer le serveur pour que le service soit pris en compte.

La console côté serveur devrait afficher quelque chose comme cela :

```
DEBUG config: [load-config [services: [
            time-server

DEBUG userbase: [load-userbase %userbase.rdb true]
DEBUG system: [start-service tcp://:8000 View 1.3.2.3.1 5-Dec-2005/18:22:54-8:00]
DEBUG service: [handle-service home "Home Service"]
DEBUG service: [handle-service admin "Administrative Service"]
DEBUG service: [handle-service file "Simple File Service"]
DEBUG service: [handle-service time-server "Precise Time Server"]
DEBUG connect: [opening-tcp-server 8000]
waiting...
```

5. Côté client, les commandes à passer seraient les suivantes :

```
[date]
[time-server/precise-time]
```

et on obtiendrait à l'exécution du client :

```
[
    done [reply seq 1 service "Generic REBOL Service" commands 2 time 0:00:00.437]
    ok [date 27-Jun-2006/10:27:30+2:00]
    ok [time-server/precise-time "10:27:30.418"]
]
```

Quelques remarques pour la création d'un service :

1. le "nom" du service ("name" dans le script du service) servira à donner un nom au contexte (object !) dans lequel sera chargé le service. Dans notre cas, la commande sera appelée par "time-server/precise-time".

2. la commande d'un service (un handle-service) peut utiliser deux variables : 'arg et 'result, qui serviront respectivement à capter les arguments passés avec la commande et le résultat de l'exécution de la commande.

3. L'action qui doit être réalisée par la commande doit être mis entre parenthèses, car dans le dialecte des "services", chaque commande subit un 'parse.

4. Il est possible de hiérarchiser et d'organiser des services en créant des sous-répertoires dans %services.

Typiquement, créer un sous-répertoire %autres-services, dans votre répertoire %services :

```
tests-rebsrv/
    ----- server.r
    ----- client.r
    ----- reb-client.r
    ----- reb-serveur.r
```

```
    ----- public/
    ----- services/
    -------------- autres-services/
```

Créez y un script appelé %basic-password.r et copiez y le code suivant :

```
REBOL []

name: 'basic-password
title: "basic tool to compute passwords"
description: "Just returns mixed arg"
commands: [
    'reverse set arg string! (result: reverse arg)
    |
    'mix set arg string! (result: join arg reverse arg)
    ]
```

Arrétez et relancez le serveur pour prendre en compte ce nouveau service.

et passez par le client les commandes :

```
[date]
[time-server/precise-time]
[basic-password/reverse "password"]
[basic-password/mix "password"]
```

Voici le résultat :

```
[
    done [reply seq 1 service "Generic REBOL Service" commands 4 time 0:00:01]
    ok [date 27-Jun-2006/10:46:49+2:00]
    ok [time-server/precise-time "10:46:49.82"]
    ok [basic-password/reverse "drowssap"]
    ok [basic-password/mix "drowssapdrowssap"]
]
```

On voit que le service 'basic-password est bien pris en compte, bien qu'il soit déclaré dans un sous-répertoire.

Voilà donc une approche des REBOL/Services.

A noter que dans le code actuel du serveur certaines fonctions ne sont pas encore implémentées.