OOIT.com

RemoteQuery

Rapid services for web and mobile

Toni Bünter, OOIT.com AG © März 2014

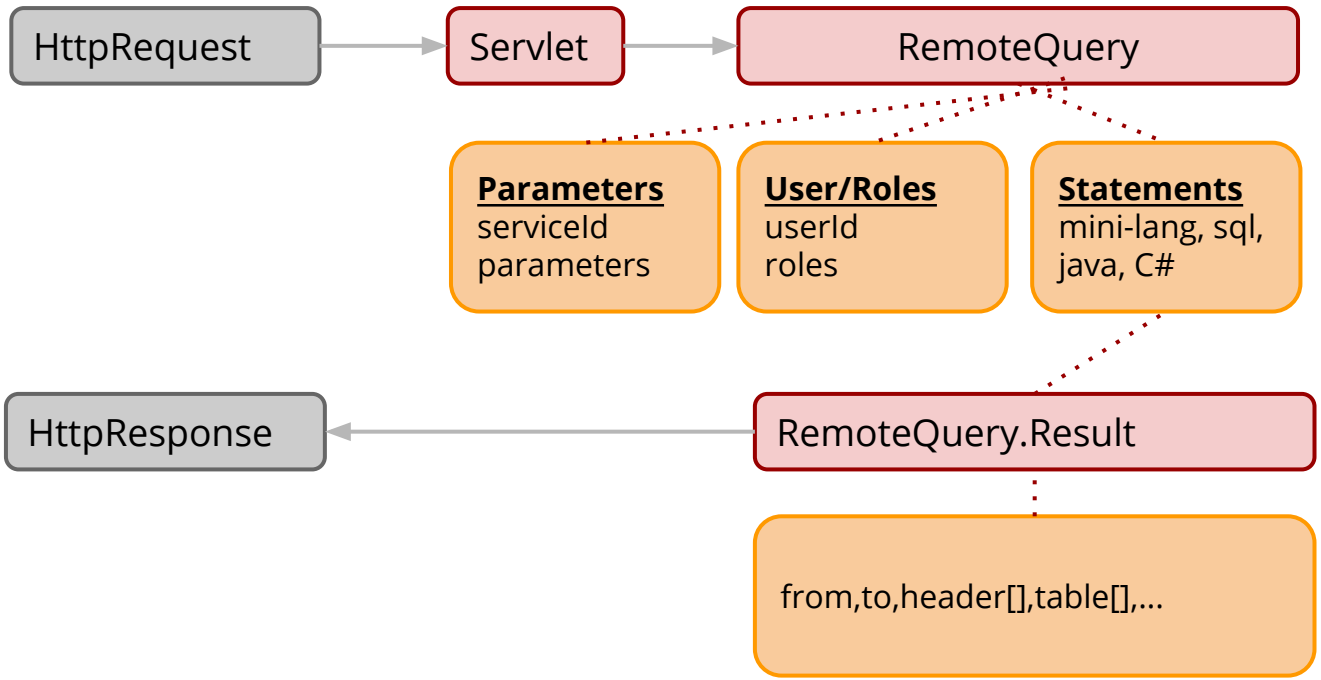
Goal of RemoteQuery

- rapid construction of services
- best possible performance
- reduced technology stack for high productivity
- platforms: Java, ASP.NET/C# and SQL DBs
- core: one source file (like jQuery ;-)
- primary support for SQL and Java programs

RemoteQuery Technology Stack

- **Client**: Web (HTML5, ...), Mobile via HTTP
- **Middle** Tier: HttpHandler, **RemoteQuery** (java, csharp and PHP *)
- **Backend**: SQL scripts and programs (e.g. in SQL, Java)
- **Communication** : JSON over HTTP(S)
- **Security** : Service access via Roles

RemoteQuery Web



Parameter Levels

RemoteQuery supports **parameter levels**. When injecting values into statements, values are taken from the parameters starting with the level 0 and working up to the last level. Values of a parameter level 0 have priority over a parameter of level 1. Primary usage is security such as the user principle. Parameters can be set in the statements and programmatically.

Parameter Levels

Level	***Description***	
initial	**Level 0.** of a parameter value. This is used for the parameter $USERID in a web application representing the authenticated user id.	select * from ADDRESS where ADDRESS_TID = bigint(:addressTid)
request	**Level 10.** Inspired by HTTP requests.	java:address.InsertStreet
header	**Level 20.** Inspired by HTTP requests.	java:address.InsertStreet
inter_request	**Level 30.**	
session	**Level 40.** In a web application this is used for String type entries in the HTTP session. Actually at start the parameters are read from the session after the request they are written back.	
application	**Level 50.** In a web application the entries in the ServletContext are read	select * from ADDRESS where ADDRESS_TID = bigint(:addressTid)

ML : Mini language for statements

RemoteQuery supports a **mini language (ML)** for efficient building of services. This includes:

1. Indication of SQL and Java code
2. Combination of services
3. Parameter handling such as setting values, default values or removing values
4. Setting debug levels
5. Transaction control

Code Type Prefix

sql-statement	A regular sql statements (select, update, delete, create …)	select * from ADDRESS where ADDRESS_TID = bigint(:addressTid)
java	Create a request and run the java class	java:address.InsertStreet
sql	Runs an sql statement. Exactly like no prefix.	select * from ADDRESS where ADDRESS_TID = bigint(:addressTid)

Combination of services

include	This includes the statement from another service referenced by the serviceId. This help to reuse statements in various services. The included statement will not be checked for access as specified in the reference service.	`include: InsertAddress;...`
serviceId	This runs a sub service referenced by the serviceId.	`serviceId:InsertAddress`

Parameter handling

set **set-LEVEL**	Sets one or more parameter values on a specific level. For LEVEL a number or a predefined level name such as `initial`, `request`, `inter_request`, `session`, `application` are supported. Http request parameter are set as **`request`** parameters.	`set-initial:a=3,b=12` `set-request:app=HR`
set-if-empty **set-if-empty-LEVEL**	Set parameter only if there is no value set	`set-if-empty:name=%;` `street=%`
set-if-null **set-if-null-LEVEL**	Set parameter if parameter is null or not available	`set-if-empty:name=%;` `street=%`
set-null	removes a parameter (or list of parameters) from all levels	`set-null:street,city`

Parameter handling (2)

set-by **set-by-LEVEL**	`Set-by` expects a list of `serviceIds` that are executed. Their ***results*** are used for creating parameters. The *header* is used as parameter names and the *first row* as values. Important **only the first row** is used!	`set-by:SelectAddress` Explanation: Assuming SelectAddress is a SQL statement such as '`select FIRST_NAME, LAST_NAME, STREET, CITY from ...`' the first row is used for setting parameters like `firstName=..., lastName=..., ...`

Debug and log (planned)

debug-start:*loggerName*	Setting the debug mode and starts debugging detailed processing messages.	set-initial:a=3,b=12 set-parameter:app=HR
debug-end:*loggerName*	Ends the debug mode.	
log:*message* **log-debug:***message*	Log message to output. If used as ‘log-debug’, the message will only be displayed if a ‘debug-start’ was called before.	log:start processing …;

Transaction control (planned)

RQ Example: Address Search

Server Side: ServiceEntry

serviceId	searchAddress
statements	sql:select STREET, CITY ... from T_ADDRESS where STREET like :street or CITY like :city
accessRoles	sales, admin

RQS : the RemoteQueryServlet

1. Connector between RemoteQuery and HTTP Requests
2. Translates HTTP request and header parameters, session parameters, application parameters to different parameter levels
3. Support of Access with RemoteQuery Entry:

```
<servlet>
        <servlet-name>RemoteQueryServlet</servlet-name>
        <servlet-class>org.remotequery.RemoteQueryServlet</servlet-class>
        <init-param>
            <param-name>accessServiceId</param-name>
            <param-value>RQ-UserAction</param-value>
        </init-param>
</servlet>
```

RQS : Access Service RQ (ASRQ)

1. If an ASRQ is defined (web.xml) the RQS executes it before it runs the (main) RQ defined with the HTTP request.
2. If the result of the ASRQ has ***no exception*** the result (first row) is saved in the HTTP session and the main RQ starts running.
3. If the result shows an exception only the exception will be returned.

RQS : Client-side Access via Ajax

Parameter	{street : 'Bahnhof%', city : 'Zuer%'}

Result	{
	from: 0, to: 3,
	header : ['street', 'city', ...],
	table : [
	['Karl', 'Schmitt', 'Bahnhofstrasse 32', 'Zuerich'], ...
	],...
	}

Features

Next to SQL scripts also Java Classes can be used

Multi-commands (separated with semicolons)

Special parameters : $USERID (principal name)

Multi connection / data source support

Transaction control

Appendix : ObjectStore

The class **ObjectStore** is a convenience API which offers a type-aware persistency tool using RQ:

```
class ObjectStore <Type>
    // create a new object
    Type newInstance(...)
    // search for objects
    List<Type> search(serviceId, ...)
    // get an object
    Type get(serviceId, ...)
    // update data with data of an object
    void update(serviceId, Type)
```