Introduction to REST and RestHUB

1. REST - Representational State Transfer

1.1. Overview - What is REST?

REST is an architectural style which is based on web-standards and the HTTP protocol. REST was first described by Roy Fielding in 2000.

In a REST based architecture everything is a resource. A resource is accessed via a common interface based on the HTTP standard methods.

In a REST based architecture you typically have a REST server which provides access to the resources and a REST client which accesses and modifies the REST resources.

Every resource should support the HTTP common operations. Resources are identified by global IDs (which are typically URIs).

REST allows that resources have different representations, e.g., text, XML, JSON etc. The REST client can ask for a specific representation via the HTTP protocol (content negotiation).

1.2. HTTP methods

The *PUT*, *GET*, *POST* and *DELETE* methods are typically used in REST based architectures.

The following table gives an explanation of these operations.

- GET defines a reading access of the resource without side-effects. The resource is never changed via a GET request, e.g., the request has no side effects (idempotent).
- PUT creates a new resource. It must also be idempotent.
- DELETE removes the resources. The operations are idempotent. They can get repeated without leading to different results.
- POST updates an existing resource or creates a new resource.

1.3. RESTFul web services

A RESTFul web services are based on HTTP methods and the concept of REST. A RESTFul web service typically defines the base URI for the services, the supported MIME-types (XML, text, JSON, user-defined, ...) and the set of operations (POST, GET, PUT, DELETE) which are supported.

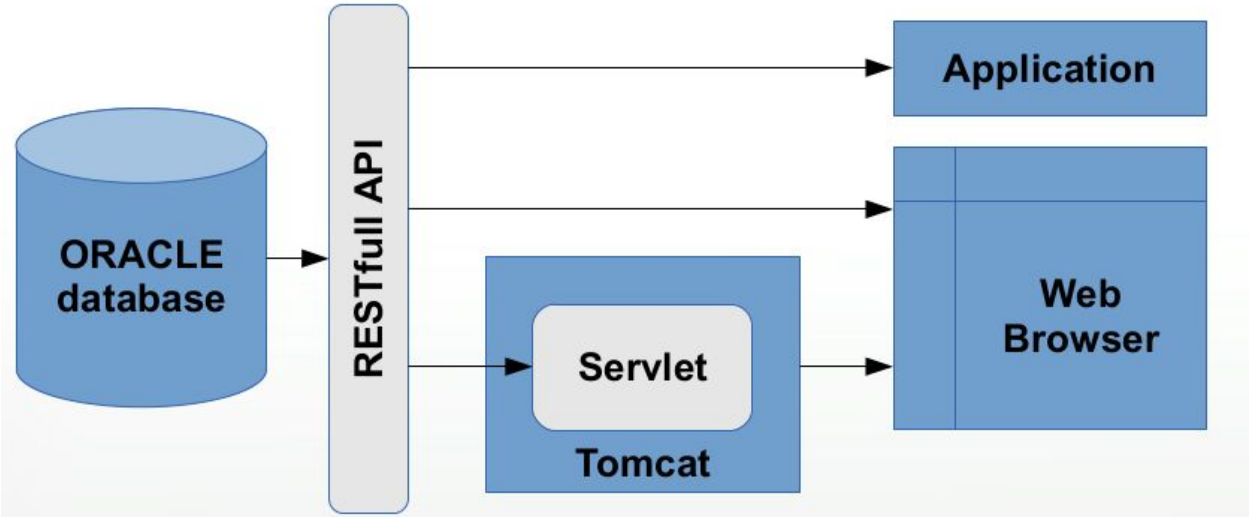
2. RestHUB - RESTful API for Oracle DB querying

2.1. Overview

RestHub was designed to provide a way to access data from the client application through RESTful API. In this manner developers will no longer have to implement difficult approaches for data retrieval from the database in representational layer and can separate application and representation codebase. They will be able to access data through the RestHub system by sending simple SQL queries.

2.2. Benefits

- Separate data access from the representation
    - Data access via API
    - Application and representation on separate codebase
- Standard, reusable API
- Many ways to use the service (Browser, Python, Java, JS, ...)



2.3. Features

Self descriptive:

- Includes metadata
- HATEOAS (including hypermedia links with the responses).

Secure:

- Specialized multilevel query parsers
- Multithreading
- Timeouts

Fast:

- Caching
- Asynchronous prefetching

Flexible:

- SQL for querying

2.4. Types of resources

- Tables: Views with optional parameters and metadata, back-end control.
- Queries: Views on Tables, front-end control.
- Cache: Query result in different representation for fast retrieval.

2.5. RestHub raw API

This API is mainly used by RestHub plugins. Developers that want to use RestHub should use those plugins instead of this API. Plugins are described below.
Variables:

- {namespace} - group of tables
- {name} - table name
- {id} - automatically generated id number (via POST method)

URL	Method	Description
/tables	GET	List of tables in JSON
/table/{namespace}	GET	List of single namespace tables in JSON
/table/{namespace}/{name}	GET	Table description in JSON
/queries	GET	List of queries in JSON

/query	POST	Create query from provided entity SQL and return automatically generated {id}
/query/{id}	GET	Query description in JSON
/query/{id}	DELETE	Remove query
/query/{id}/data?{r}={v}	GET	Get query data. Media type is by Accept header. For example: “Accept” : “application/xml”. Variables: r - parameter name, v - parameter value.
/query/{id}/page/{pp}/{p}/data?{r}={v}	GET	Get query page data. Media type is by Accept header. Variables: pp - data rows per page, p - page number, r - parameter name, v - parameter value.
/table/{namespace}/{table}/data?{r}={v}	GET	Get table data. Media type is by Accept header. For example: “Accept” : “application/xml”. Creates query and redirects to it. Variables: r - parameter name, v - parameter value.
/table/{namespace}/{table}/page/{pp}/{p}/data?{r}={v}	GET	Get query page data. Media type is by Accept header. Variables: pp - data rows per page, p - page number, r - parameter name, v - parameter value. Creates query and redirects to it.
/query/{id}/cache	GET	Get query cache information
/query/{id}/cache	DELETE	Clean all query cache
/table/{namespace}/{name}/cache	GET	Get the list of query caches that use the table defined
/blacklist	GET	List of blacklisted tables in JSON

/blacklist/{namespace}	GET	List of blacklisted namespace tables in JSON
/blacklist/{namespace}/{name}	GET	Blacklisted table description in JSON
/blacklist	DELETE	Remove all tables from blacklist (refresh)
/blacklist/{namespace}	DELETE	Remove all namespace tables from blacklist (refresh)
/blacklist/{namespace}/{name}	DELETE	Remove table from blacklist (refresh)

3. RestHUB Tutorial - How to use it

3.1. Store your tables in RestHub

In order to start using RestHub you have to create an XML file where your database tables would be stored. This file should be created from the template and put in application classpath. When this XML file is uploaded to svn versioning system it is automatically picked up by the RestHub and all the necessary tables are created and saved. From this point you are ready to use RestHub in your applications.

Example of simplified tables XML file is listed below. There is only one table (hcal_parts) in this example, but it's possible to add as many tables as you need.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<TABLES>
        <TABLE    CACHE_TIME="120"    HIT_COUNT="1"    TIME_OUT="30"    ROWS_LIMIT="1000"
CONNECTION_NAME="default">
        <METADATA>
            <ENTRY KEY="description">Unique list of notebook parts</ENTRY>
        </METADATA>
        <NAMESPACE>hcal</NAMESPACE>
        <NAME>hcal_parts</NAME>
        <SQL> SELECT DISTINCT
                    P.PART_ID,
                    P.NAME_LABEL
              FROM CMS_HCL_CORE_CONSTRUCT.PARTS P </SQL>
        <COLUMNS>
```

```
            <COLUMN NUM="1">
                <METADATA>
                    <ENTRY KEY="description">Part ID</ENTRY>
                </METADATA>
                <NAME>PART_ID</NAME>
                <TYPE>STRING</TYPE>
            </COLUMN>
            <COLUMN NUM="2">
                <METADATA>
                    <ENTRY KEY="description">Part name</ENTRY>
                </METADATA>
                <NAME>NAME_LABEL</NAME>
                <TYPE>STRING</TYPE>
            </COLUMN>
        </COLUMNS>
    </TABLE>
</TABLES>
```

Table has several options. The most important are listed below:

- CACHE_TIME - RestHub will save query data in cache for 120 seconds.
- ROWS_LIMIT - the maximum number of data rows returned from one query.
- CONNECTION_NAME - there can be multiple connections to the database(s) which are identified by name.

3.2. RestHub Javascript plugin

For example we want to create a simple HTML + Javascript page. There are several plugins already created that will help us. We have to put them in our page directory and (or) to include a path to those plugins.

A brief description of plugins:

- rhserver.js - this file stores a path to RestHub machine.
- resthub.js - main plugin that will help us to use RestHub.
- resthub.table.js - plugin that draws a table and uses RestHub.

All these plugins are created with jQuery library. So we have to include it as well.

If we are using Java language for our application there are also similar plugins created.

3.2.1 resthub.js API

Before using resthub.js plugin we have to make sure that a variable called rhserver in rhserver.js points to RestHub machine. We would use reshub.js API through this variable.

Consumer API:

- rhserver.query(sql) - assigns sql to a query. Later on if we want to modify a specific query we could change sql and refresh it like this:
    - rhserver.query.sql += " ORDER BY someColumn ASC" ;
    - rhserver.query.refresh();
- rhserver.data() - executes query and returns data. This method has parameters:
    - type - content type (json, xml, ...)
    - ppage - data rows per page
    - page - page number
    - params - query SQL parameters. Parameter name in SQL should begin with : symbol, for example :parameter.

Example:

```
var q = rhserver.query("SELECT * FROM hcal.hcal_parts q WHERE q.PART_ID = :part_id");
var query_data = q.data({ type: "application/xml", ppage: 20,
page: 5, params: {part_id: 5}});
```

3.2.2 resthub.table.js API

This plugin draws a datatable to a specific div element by passing a query and various parameters to it. It was made by implementing datatables (http://datatables.net) and resthub.js plugins. To use it we just have to select (through jQuery selectors) a specific div and then call rhTable method with parameters. Parameter descriptions:

- query - assign a query
- params - query parameters
- showColumns - how many columns you want to show on a table. Others will be hidden
- tableOptions - various table options (more info on http://datatables.net).

Example:

```
// Variable that stores datatable option - to add a link on ID column
var tableOptions = {
       "fnRowCallback": function(nRow, aData, iDisplayIndex) {
              $('td:eq(0)', nRow).html('<a href=info.html?id=' + aData[0] + '>' +
aData[0] + '</a>');
               return nRow;
       }
};
// assign a query to variable q
var q = rhserver.query("SELECT * FROM hcal.hcal_parts q WHERE q.PART_ID = :part_id");
// draw a table to a hcal_parts_table div
$('#hcal_parts_table).rhTable({query: q, params: {part_id: 5}, showColumns: 5,
tableOptions: tableOptions});
```

3.3. Building a simple web page that uses RestHub

Now that we know how to use resthub plugins we can start coding our applications. Here is the code for a simple page that displays a datatable from querying hcal_part table:

```
<!DOCTYPE html>
<html>
    <head>
       <title>HCAL example</title>
       <style type="text/css" title="currentStyle">
           @import "css/themes/smoothness/jquery-ui-1.8.4.custom.css";
       </style>
       <script type="text/javascript" src="js/jquery.js"></script>
       <script type="text/javascript" src="js/jquery.dataTables.js"></script>
       <script type="text/javascript" src="js/resthub.js"></script>
       <script type="text/javascript" src="js/resthub.table.js"></script>
       <script type="text/javascript">
            $(document).ready(function() {
```

```
                // specify a query to RestHub
                var q = rhserver.query("SELECT * FROM hcal.hcal_parts c");
                 // draw a table using this query
                $('#hcal_parts_table).rhTable({query: q});
            });
        </script>
    </head>
    <body>
       <div>
           <h1>HCAL parts table</h1>
           <div id="hcal_parts_table"></div>
       </div>
    </body>
</html>
```

3.4. RestHub Java plugin

If we want to create a Java application that uses Resthub there are several classes that will help us. We have to put them in our classpath and import them:

```
import net.resthub.client.RestHubServer;
import net.resthub.model.DataResponse;
import net.resthub.model.QueryManager;
```

The usage of these plugins are very similar to earlier described jQuery plugins.

Example:

```
//  this class stores a path to RestHub machine.
RestHubServer rh = new RestHubServer("http://localhost:8080/api");
//  assign sql to a query
QueryManager  qm  =  rh.newQueryManager("SELECT  *  FROM  hcal.hcal_parts  q  WHERE
q.PART_ID = :part_id");
qm.addParameter("part_id", 5);
//  executes query and returns data
DataResponse dr = qm.getData("application/xml");
```

4. How to start the service

Resthub application is based on the Restlet framework so Resthub is a normal Restlet Application (org.restlet.Application) which can be easily integrated into your backend application stack. The easiest way to start the application is to start it from the main method, i.e.

```
	private static final int PORT = someport;

	public static void main(String[] args) throws Exception {

		ConnectionFactory cf = new MyConnectionFactory();
		TableFactory tf = new MyTableFactory();

		ServerApp app = new ServerApp(cf, tf);
		Component comp = new Component();
		comp.getServers().add(Protocol.HTTP, PORT);
		comp.getDefaultHost().attach(app);
		comp.start();
	}
```

This starts the RestHub service on PORT. Of course, you have to implement your net.resthub.ConnectionFactory and net.resthub.TableFactory interfaces which provide access to Oracle databases (aka connections) and tables metadata. We provide net.resthub.factory.DbTableFactory which extracts metadata from the Oracle database and net.resthub.factory.XmlClasspathFactory which extracts metadata from XML files in the classpath.

5. How to write queries

RestHub queries are simple SQL statements to combine tables into the single output.

Limitations:

- tables referred by namespace.table
- all tables used must have the unique alias in a query
- only tables that share the same connection can be used in a query
- named parameters are supported, i.e. :name
- by default named parameter is considered of string type
- named parameter type can be changed with prefix:
    - s__{parameter name} - string type, i.e. s__name, s__country
    - n__{parameter name} - number type, i.e. n__id, n__voltage
    - d__{parameter name} - date type, i.e. d__from, d__to

Example queries:

```
select * from shop.customers c

select * from shop.customers c order by c.name
```

```
select c.* from shop.customers c, shop.sales s where c.id = s.customer_id order by s.sale_date

select c.name, s.sale_date from shop.customers c join shop.sales s on c.id = s.customer_id
order by s.sale_date

select c.name, s.sale_date from shop.customers c join shop.sales s on c.id = s.customer_id
where c.country = ‘USA’ order by s.sale_date

select c.name, s.sale_date from shop.customers c join shop.sales s on c.id = s.customer_id
where c.country = :country order by s.sale_date
// parameter ?country=USA

select c.name, s.sale_date from shop.customers c join shop.sales s on c.id = s.customer_id
where c.country = :s__country order by s.sale_date
// parameter ?country=USA

select c.name, s.sale_date from shop.customers c join shop.sales s on c.id = s.customer_id
where s.price >= :n__price_from and s.price <= :n__price_to order by s.sale_date
// parameter ?price_from=10&price_to=100
```