GET POST ORDS JSON: Web Services for APEX Decoded

SUMNER technologies

Welcome

About Me

scott@sumnertech.com

@sspendol

ORACLE

Sumner TECHNOLOGIES

accenture

sumnēva Application Success

ORACLE ACE Director

enkitec

SUMNER technologies

About Sumner Technologies

- Originally Established 2005
- Relaunched in 2015
  - Focused exclusively on Oracle APEX solutions
- Provide wide range of APEX related Services
  - Architecture Design & Reviews
  - Security Reviews
  - Health Checks
  - Education
    - On-site, On-line, On-Demand
    - Custom & Mentoring
  - Oracle Database Cloud Consulting
  - Curators of APEX-SERT

SUMNER technologies

ORACLE Gold Partner

Agenda

- Overview
- Definitions
- Demonstration
- Summary

SUMNER technologies

Overview

Our Happy Place…

- As PL/SQL developers, we're most comfortable when we are where we know the most
  - Inside the Oracle Database
- We can get and manipulate any data, so as long as we can use:
  - Views
  - Tables
  - SQL
  - PL/SQL

Our Sad Place…

- Once we leave our Happy Place (i.e. the Oracle Database), we're outside of our comfort zone
- Simple things - like reading and writing data - become insurmountable tasks
  - We just don't even know where to start
  - Even if we did, we don't speak the language
  - We feel lost
  - We are lost

Calling Procedures: Oracle to Oracle

- **Same schema**:
  - Refer to the name of the procedure
- **Different schema/same database**:
  - Create a grant
  - Refer to the schema.name of the procedure
- **Different schema/different database**:
  - Create a database link
  - Create a grant
  - Refer to the schema.name@link of the procedure

Calling Procedures: Oracle to Non-Oracle

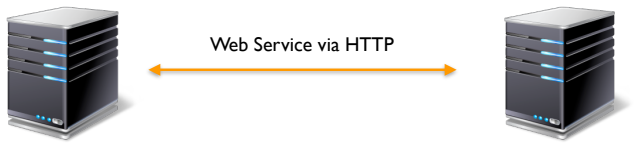
- Things get **complicated** when we have to **venture outside** of the Oracle Database
  - No more simple procedure calls are possible
  - Must turn to something else
  - This is our **Sad Place**
- To bridge the gap, we can turn to **Web Services**

Web Services

- For most Oracle APEX & PL/SQL developers, **Web Services** represent a **sad place**
  - A place where their skills do them no good
  - Major feelings of hopelessness and despair
- As IT and associated systems evolve, we can **no longer ignore web services** and hope they go away
  - They are here to stay and becoming more and more important
  - It's no longer **if**, but **when** you'll have to learn how to use them

Web Services

- Web Services are nothing more than a **procedure that lives on another server**
  - Typically used when two computers exchange data
  - Runs over **HTTP** or **HTTPS**
  - Results typically contain data formatted in either **XML** or **JSON**



Definitions

Definitions

- Most confusion surrounding Web Services lies in the **many, many acronyms** used to define them
  - We already used four!
- Let's take a moment to define a few:
  - **HTTP/HTTPS**
  - **XML**
  - **JSON**
  - **REST**
  - **Methods**
    - GET, POST, PUT, DELETE
  - **ORDS**

HTTP/HTTPS

- **H**yper**T**ext **T**ransfer **P**rotocol (**S**ecure)
  - Underlying protocol that is used by the web
  - HTTP is clear text; HTTPS is encrypted
- Web Services use HTTP/S to **communicate** with one another
  - No need for SQL*Net or other protocols
- **Note**:
  - On the Oracle side, may involve adding site certificates to **Oracle Wallet** and adding entries to the database **Access Control List (ACL)**

XML

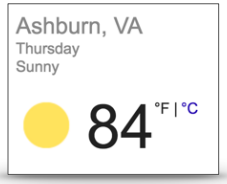
- XML is basically a **format for transmitting data**
  - It doesn't really do anything; but rather provides an envelope for and a description of the data which it transports
- Like HTML, XML is made up of **two components**:
  - **Tags**
  - **Data**
- Unlike HTML, you can **create any tag you feel like** in an XML document
  - Which is the source of much confusion

XML

- A simple XML document:

```
<weather>
  <location>Ashburn, VA</location>
  <temperature>84</temperature>
  <condition>Sunny</condition>
</weather>
```

- It's **self-describing** & **readable**
  - You can extract that it's Sunny and 84 degrees in Ashburn,VA
- The **challenge** is to write some code to **extract the data** and then display or use the results in your own applications



JSON

- **JSON** - or **JavaScript Object Notation** - is another popular format for data interchange
  - Less payload than XML and fast becoming the "go to" alternative for XML for this reason
- JSON uses **key/value** pairs to encapsulate data
  - Value pairs are enclosed by **"** and delimited by a **colon**
  - Document or arrays within document enclosed by **{ }** and/or **[]**

JSON Example

- Simple example of a JSON document:

```
"band":
  [
    {"firstName":"John",  "lastName":"Lennon"},
    {"firstName":"Paul",  "lastName":"McCartney"},
    {"firstName":"Ringo", "lastName":"Star"}
    {"firstName":"George","lastName":"Harrison"}
  ]
```

- It's **self-describing** & **readable**
  - You can extract that the names of the band members
- The **challenge** is again to write some code to **extract the data** and then display or use the results in your own applications

XML vs. JSON

- Similar to XML in that:
  - Both are "self describing" and easy to read
  - Both can be hierarchical
  - Both can easily be parsed and used by lots of programming languages
  - Both JSON and XML can be fetched with an XMLHttpRequest

Winner: JSON

- JSON is different from XML in that:
  - No closing/end tags
  - Shorter
  - Quicker to read & write
  - Supports Arrays
- With most modern web applications, it just makes sense to use JSON over XML

REST

- **REST** (Representational State Transfer)
  - Exposes a named resource via HTTP/S
    - Example: **https://servername/service/emp/7839**
  - Data Format returned can be anything (CSV, JSON, XML, Text)
- REST **eliminates some of the complexity** that came with SOAP-based web services
  - Can just really refer to a URL and get data
  - What you do with the data depends on a number of things
- Web Services based on the REST architecture that use HTTP methods are referred to **RESTful Web Services**

HTTP Methods

- There are a **number of ways or methods** to interact with a resource over HTTP

Method	SQL	Idempotent
GET	SELECT	Y
POST	INSERT	N
PUT	UPDATE or INSERT	Y
DELETE	DELETE	Sort Of

GET

- A **GET** transaction is when you **request a resource** on the server
  - As simple as entering a URL/URI
  - Possible to pass parameters, but not required
- Example:
  - **http://server/action.do?p_value=123&p_name=scott**

GET

- **GETs** requests:
  - Can be cached
  - Will remain in your browser's history
    - Until you delete it or are running in incognito mode
  - Can be bookmarked
  - Should never be used when dealing with sensitive data
    - As the data is readable in multiple places
  - Should only be used to retrieve - or GET - data from a server
    - Never for anything transactional
  - Can be tampered with

POST

- A **POST** occurs when data is sent to a server
  - Typically when a user clicks a button and submits a form
  - Possible and likely to pass parameters, but not required
- Example:
  - **http://server/action.submit**
  - Item name & value pairs are sent in the body of the post

POST

- **POST** requests:
  - Are never cached
  - Do not remain in the browser's history
  - Cannot be bookmarked

Other Methods

- **PUT**
  - Performs an update of the specified resource
- **DELETE**
  - Deletes specified resource
- **HEAD**
  - Same as GET, but only returns HTTP Header information
- **OPTIONS**
  - Returns the methods that the HTTP server supports

ORDS

- **O**racle **R**ESTful **D**ata **S**ervices
  - Formerly called **Oracle APEX Listener**
  - **Fully Supported** feature of the Oracle Database since 2010
    - Can log SRs against a corresponding Database License Provides HTTP/S Access to Oracle Databases (and other databases)
  - **Maps HTTP(S) RESTful GETS and POSTS to SQL and PL/SQL**
  - Declaratively returns results in JSON or CSV format
- Enables virtually every platform to easily and securely access an Oracle Database

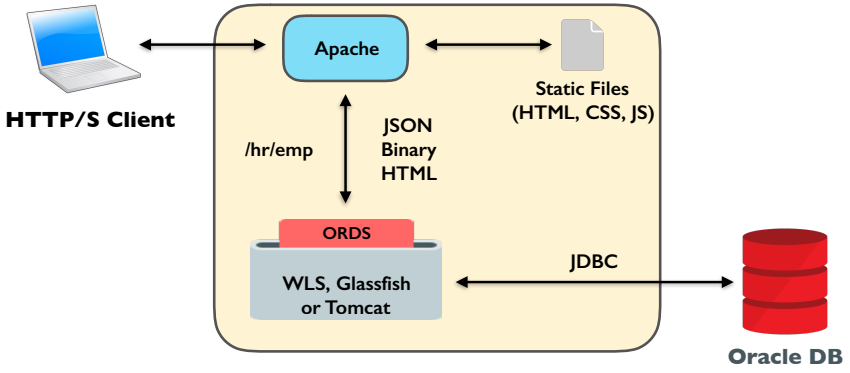
ORDS

SUMNER technologies

Architecture

- Standard Web Server layout
- Implements Java Servlet
- Supported deployment for:
  - Oracle WebLogic Server (OWS)
  - Glassfish
  - Tomcat
- Embedded Jetty for standalone operation
  - New in Release 4.0

Architecture



Demonstration

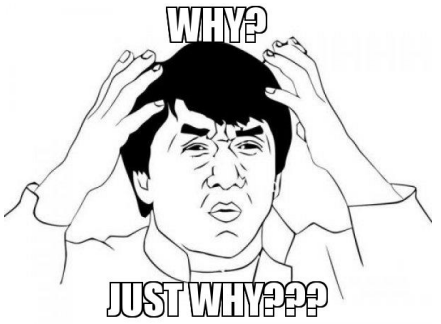
Examples

- **ORDS**
  - Same Server
  - Different Server
- **External Sites**
  - Weather
  - Zip Code

ORDS: Same Server

ORDS: Same Server

- **Why?**
  - Much easier to just create a grant to get data from another schema
  - Can utilize all of the built-in functionality of APEX (DML Processes, lost update detection, etc)



ORDS: Different Servers

ORDS: Different Servers

- Create a report & form based on data hosted on AWS via ORDS
  - Start with a **Data Grid**
  - Build modal page for **CRUD transactions**

Step to Implement

- Write some **PL/SQL code** to process the transaction
  - One procedure for each DML transaction type
- Create a **web service** via ORDS for each transaction type
  - Can use APEX or SQL Developer
- Consume the web service in APEX
- Secure the web services

Web Service via ORDS

- Create a simple web service
  - **Method**: GET
  - **Source Type**: Query
  - **Format**: JSON
  - **SQL**:

```
SELECT ename, empno, deptno, job, mgr, comm, sal,
  TO_CHAR(hiredate,'DD-MON-YYYY') hiredate
FROM emp
```

- Test the web service
  - http://server.com/apex/ords/emp/listEmp

Testing Web Services

- Use tools such as **Postman** to facilitate testing
  - Postman: https://www.getpostman.com/
  - **Free** Chrome extension or standalone application
    - Runs on Mac, Windows or Linux
    - Pro version available with more features
- Facilitates **testing of web services**
  - Support for authentication, header variables, scripts, etc.
  - Essential when it comes to testing POST/PUT/DELETE

DATEs are Dumb

- All values in a JSON document are **VARCHAR**
  - Thus, values that are DATEs need to be cast to VARCHAR first
- Date format needs to be **as precise as you need it**
  - If you need seconds, then you better use the proper date format mask to preserve them
- With ORDS, we can control the date format mask in our SQL
  - But many sites will use the JavaScript default mask:

```
YYYY-MM-DDTHH:MI:SS.SSSZ
```


Consuming the Web Service in APEX

- APEX contains a couple of APIs that are designed for use with web services
- **APEX_WEB_SERVICE**
  - Used to call and parse results of a web service
  - Support for SOAP & RESTful web services
- **APEX_JSON**
  - Parse & extract data stored in a JSON document

MAKE_REST_REQUEST

- Part of the **APEX_WEB_SERVICE** API
- Makes a simple REST request to the web service provided
  - Result is returned as a CLOB
- Support for
  - Basic Authentication
  - OAuth
  - Parameters
  - Oracle Wallet Path
- Can also call **MAKE_REST_REQUEST_B** to get back BLOB data

get_data

```
PROCEDURE get_data
IS
  l_response                CLOB;
BEGIN

-- Remove the existing record
DELETE FROM ws WHERE key = 'EMP';

-- Get the REST response
l_response := apex_web_service.make_rest_request
  (
  p_url         => 'http://server.com/apex/ords/listEmp',
  p_http_method => 'GET'
  );

-- Save the response to the WS table
INSERT INTO ws (key, response) VALUES ('EMP', l_response);

<< Repeat process for DEPT >>

END get_data;
```


Selecting JSON Data

- To query data from the web service, we can use the API **APEX_JSON.TO_XMLTYPE**
  - And treat the JSON if it were XML
  - Which allows us to us the **XMLTABLE** SQL command and extract data from the table

to_xmltype

```
SELECT
  x.*
FROM
  xmltable
    (
    '/json/items/row'
    PASSING apex_json.to_xmltype(apex_web_service.make_rest_request
      (
      p_url         => :G_URL || '/listEmp',
      p_http_method => 'GET'
      )
    )
    COLUMNS
      ename    VARCHAR2(4000) PATH 'ename',
      empno    NUMBER         PATH 'empno',
      job      VARCHAR2(255)  PATH 'job',
      mgr      NUMBER         PATH 'mgr',
      hiredate VARCHAR2(255)  PATH 'hiredate',
      sal      NUMBER         PATH 'sal',
      deptno   NUMBER         PATH 'deptno'
    ) x
```


Real Time vs. Cached

- There are a couple of ways to build a report on the results of the web service
  - Call the web service in **real time**
  - Call the web service once and **store a local copy** of the data
    - Either in an APEX collection on a CLOB/XML column
- Which you use depends on **specific requirements**
  - Real time will be more up to date, but slower
  - Cached will be faster, but only as recent as the last pull

DML Transactions on JSON Data

- Since a row of our data is nested inside a JSON document, which is stored as a CLOB, we can't rely on the APEX built-in DML processes
  - Thus we have to create our own using web services

Web Service	Web Service Type	PL/SQL Procedure
getEMP	GET	emp_pkg.get_emp
insEMP	POST	emp_pkg.ins_emp
updEMP	PUT	emp_pkg.upd_emp
delEMP	DELETE	emp_pkg.del_emp

get_emp: Web Service

- **Method**: GET
- **Source Type**: Query One Row
- **Format**: JSON
- **URI**:
  - `/getEmp/{empNo}`
- **PL/SQL**:
  - `SELECT * FROM emp WHERE empno = :empno`

APEX_JSON

- A new API, **APEX_JSON** extends a number of JSON-oriented functions to Oracle
  - Works in 11g as well as 12c
- Can both **consume & generate** JSON documents
  - As well as parse and extract values
- This is used to parse the JSON document that we received from the web service
  - And also to extract the data from it

APEX_JSON: Parse, Count & Get

- **PARSE**
  - Parses a JSON document and returns the values into a PL/SQL array where we can more easily inspect and use them in PL/SQL
- **GET_COUNT**
  - Returns the number of members in the array
- **GET_NUMBER**
  - Returns a specific array value as a NUMBER
- **GET_VARCHAR2**
  - Returns a specific array value as a VARCHAR2
- Similar functions for BOOLEAN, CLOB, DATE, etc.

get_emp: PL/SQL

```
PROCEDURE get_emp
  (
  p_empno                    IN NUMBER
  )
IS
  l_response                 CLOB;
  l_val                      apex_json.t_values;
BEGIN

-- Get the REST response
l_response := apex_web_service.make_rest_request
  (
  p_url         => 'http://server.com/apex/ords/getEmp/' || p_empno,
  p_http_method => 'GET'
  );

-- Parse the results
apex_json.parse(l_val, l_response);

-- Set the page items
apex_util.set_session_state('P5_ENAME',  APEX_JSON.get_varchar2(p_path => 'ename',  p0 => 1, p_values => l_val));
apex_util.set_session_state('P5_SAL',    APEX_JSON.get_varchar2(p_path => 'sal',    p0 => 1, p_values => l_val));
apex_util.set_session_state('P5_JOB',    APEX_JSON.get_varchar2(p_path => 'job',    p0 => 1, p_values => l_val));
apex_util.set_session_state('P5_DEPTNO', APEX_JSON.get_varchar2(p_path => 'deptno', p0 => 1, p_values => l_val));
apex_util.set_session_state('P5_MGR',    APEX_JSON.get_varchar2(p_path => 'mgr',    p0 => 1, p_values => l_val));

END get_emp;
```


del_emp: Web Service

- **Method**: DELETE
- **Source Type**: PL/SQL
- **URI**:
  - `/delEmp`
- **PL/SQL**:
  -
```
BEGIN
DELETE FROM EMP WHERE EMPNO = :EMPNO;
END;
```


G_REQUEST_HEADERS

- To set values when calling a POST, PUT, or DELETE, use the global array **G_REQUEST_HEADERS**
  - Which is part of the **APEX_WEB_SERVICE** API
- To set a variable, **two calls** are necessary
  - One for the value name, and one for the value itself
  -
```
apex_web_service.g_request_headers(1).name := 'ename';
apex_web_service.g_request_headers(1).value := p_ename;
```


del_emp: PL/SQL

```
PROCEDURE del_emp
  (
  p_empno                    IN NUMBER
  )
IS
  l_response                 CLOB;
BEGIN

-- Set the EMPNO
apex_web_service.g_request_headers(1).name := 'empno';
apex_web_service.g_request_headers(1).value := p_empno;

-- Get the REST response
l_response := apex_web_service.make_rest_request
  (
  p_url         => 'http://server.com/apex/ords/delEmp',
  p_http_method => 'DELETE'
  );

END del_emp;
```

ins_emp: Web Service

- **Method**: POST
- **Source Type**: PL/SQL
- **Format**: JSON
- **URI**:
  - `/getEmp/{empNo}`
- **PL/SQL**:

```
BEGIN
INSERT INTO emp (ename, sal, job, deptno, mgr)
VALUES (:ename, :sal, :job, :deptno, :mgr);
htp.prn('OK');
END;
```


ins_emp: PL/SQL

```
PROCEDURE ins_emp
  (
  p_ename                   IN VARCHAR2,
  p_sal                     IN NUMBER,
  p_job                     IN VARCHAR2,
  p_deptno                  IN NUMBER,
  p_mgr                     IN NUMBER DEFAULT NULL
  )
IS
  l_result                  CLOB;
BEGIN

-- Set the ENAME
apex_web_service.g_request_headers(1).name := 'ename';
apex_web_service.g_request_headers(1).value := p_ename;

-- Set the JOB
apex_web_service.g_request_headers(2).name := 'job';
apex_web_service.g_request_headers(2).value := p_job;

…

--Invoke the Web Service
l_result := apex_web_service.make_rest_request
  (
  p_url         => 'http://server.com/apex/ords/insEmp',
  p_http_method => 'POST'
  );

END ins_emp;
```


upd_emp: Web Service

- **Method**: PUT
- **Source Type**: PL/SQL
- **URI**:
  - `/updEmp`
- **PL/SQL**:

```
BEGIN
UPDATE emp SET ename = :ename, sal = :sal,
  job = :job, deptno = :deptno, mgr = :mgr
WHERE empno = :empno;
htp.prn('OK');
END;
```


upd_emp: PL/SQL

```
PROCEDURE upd_emp
  (
  p_empno                   IN NUMBER,
  p_ename                   IN VARCHAR2,
  p_sal                     IN NUMBER,
  p_job                     IN VARCHAR2,
  p_deptno                  IN NUMBER,
  p_mgr                     IN NUMBER DEFAULT NULL
  )
IS
  l_result                  CLOB;
BEGIN

-- Set the EMPNO
apex_web_service.g_request_headers(1).name := 'empno';
apex_web_service.g_request_headers(1).value := p_empno;

-- Set the ENAME
apex_web_service.g_request_headers(2).name := 'ename';
apex_web_service.g_request_headers(2).value := p_ename;

…

--Invoke the Web Service
l_result := apex_web_service.make_rest_request
  (
  p_url         => 'http://52.21.56.67/apex/ords/updEmp',
  p_http_method => 'PUT'
  );

END upd_emp;
```

External Sites: GET

Weather

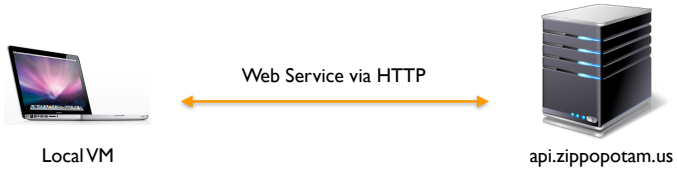
External Site/GET: Weather

- Create a web service call to return the weather for a given ZIP code
  - Parse results and set APEX page items
  - Refresh regions to update content

ZIP Code

External Site/GET: Zip Code

- Create a web service call to get the city/state from a ZIP code
  - Create two **Dynamic Actions**:
    - Use **PL/SQL** to set **OUT** parameters to values
    - Use **JavaScript** and **apex.server.process** to set values



Summary

Summary

- Your **Happy Place** should now include **Web Services!**
  - They are no longer scary things, but rather another tool or method to exchange data across unlike services
- There are **several different methods** to work with data which they provide
  - Choose the one you're most comfortable with
- Consider the **newer APIs** when consuming web services
  - Less code; more upgradable

SUMNER technologies