## C:\Users\kaveti_s\Desktop\icons-facebook-facebook-icon-social-media-icon-ios-7-facebook-png--9.png

Advanced **Java**

/[smlcodes](https://twitter.com/smlcodes) [/smlcodes](https://www.facebook.com/SmlCodes-216971275364320/) [/smlcodes](https://www.youtube.com/channel/UCO0JjZa8BcRvc3rTe0BL9HA)



***A*** [***SmlCodes.Com***](http://www.smlcodes.com/) ***Small presentation***

In Association with Idleposts.com

## Satya Kaveti

## So called TITLE

**SQL, MongoDB, JDBC, Servlets, JSP, WebServices**



**Small Codes**

Programming Simplified



Copyright © 2016 SmlCodes.com All rights Reserved.

**For more tutorials & Articles visit** [**SmlCodes.com**](http://www.smlcodes.com/)

**Copyright © 2016 Smlcodes.com**

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, **without the prior written permission** of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

**Every effort has been made in the preparation of this book to ensure the accuracy of the information presented**. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, SmlCodes.com, nor its dealers or distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

**Smlcodes.com has endeavored to provide trademark information** about all the companies and products mentioned in this book by the appropriate use of capitals. However, SmlCodes.com Publishing cannot guarantee the accuracy of this information.

If you discover any errors on our website or in this tutorial, please notify us at [support@smlcodes.com](mailto:support@smlcodes.com) or [smlcodes@gmail.com](mailto:smlcodes@gmail.com)

**First published on** Aug 2016, Published by **SmlCodes.com**

## Author Credits

Name : **Satya Kaveti**

Email : [satyakaveti@gmail.com](mailto:satyakaveti@gmail.com)

Website : smlcodes.com, satyajohnny.blogspot.com

## Digital Partners



# Table of Content

[  1](#_Toc4274686)

[- Satya Kaveti 1](file:///D:\Re-Write\Final\Advanced%20Java%20-Rewrite-Final(sql-mongo-jsp-servlets).docx#_Toc4274687)

[So called TITLE 1](#_Toc4274688)

[Author Credits 2](#_Toc4274689)

[Digital Partners 2](#_Toc4274690)

[Table of Content 3](#_Toc4274691)

[1. SQL Basics 5](#_Toc4274692)

[I.Introduction to SQL 5](#_Toc4274693)

[1.1 SQL Operations 5](#_Toc4274694)

[1.2 JOINS 8](#_Toc4274695)

[1.3 Sql Functions 13](#_Toc4274696)

[1.4 PL/SQL 13](#_Toc4274697)

[2.1 Bascis 13](#_Toc4274698)

[2. MongoDB 17](#_Toc4274699)

[1.Introduction 17](#_Toc4274700)

[2. MongoDB Operations 20](#_Toc4274701)

[3. MongoDB with Java 27](#_Toc4274702)

[References 33](#_Toc4274703)

[2. JDBC 33](#_Toc4274704)

[2.1 Introduction 33](#_Toc4274705)

[2.2 DriverManager class 37](#_Toc4274706)

[2.3 Connection interface 37](#_Toc4274707)

[2.4 Statement interface 38](#_Toc4274708)

[2.5 ResultSet interface 38](#_Toc4274709)

[2.6 PreparedStatement 44](#_Toc4274710)

[2.7 BLOB (BinaryLarge Obejects) 45](#_Toc4274711)

[2.8 CLOB 47](#_Toc4274712)

[2.9 CallableStatement 48](#_Toc4274713)

[2.10 Metadata 50](#_Toc4274714)

[2.11 Batch Processing 52](#_Toc4274715)

[2.12 Transcations 53](#_Toc4274716)

[2.13 RowSet Interfcae 53](#_Toc4274717)

[2.14 Notes 54](#_Toc4274718)

[3. Servlets 55](#_Toc4274719)

[3.1 Basics of Web Technologies 56](#_Toc4274720)

[3.2 Servlet API 60](#_Toc4274721)

[3.3 Servlet Lifecycle 62](#_Toc4274722)

[3.4 ServletRequest (interface) 🡪 getParamaters() 64](#_Toc4274723)

[3.5 ServletConfig (interface) 🡪 getInitParamaters() 69](#_Toc4274724)

[3.6 ServletContext (interface) 🡪 getInitParamaters() 70](#_Toc4274725)

[3.7 ServletChaining 73](#_Toc4274726)

[3.8 Attribbutes 77](#_Toc4274727)

[3.9 Session Tracking 80](#_Toc4274728)

[3.10 Filters 84](#_Toc4274729)

[4. JSP 88](#_Toc4274730)

[1. JSP Introduction 88](#_Toc4274731)

[2. JSP Scriptlets 89](#_Toc4274732)

[3. JSP Implicit objects 90](#_Toc4274733)

[4. JSP Directives 92](#_Toc4274734)

[5. JSP Action Tags 95](#_Toc4274735)

[6. JSP EL (JSP Expression Language) 97](#_Toc4274736)

[7. JSP - Standard Tag Library (JSTL) 98](#_Toc4274737)

[8. JSP Custom Tags 100](#_Toc4274738)

[### 5. Webservices 102](#_Toc4274739)

[1.1 SOAP [Simple Object Access Protocol 103](#_Toc4274740)

[1.2 REST [Representation State Transfer] 105](#_Toc4274741)

[1.3 Java Web Services API 107](#_Toc4274742)

[2. JAX-WS (SOAP web services) 108](#_Toc4274743)

[2.1 Diffrence between RPC-Style and Document Style 109](#_Toc4274744)

[2.2 JAX-WS Annotations 110](#_Toc4274745)

[2.3 JAX-WS RPC Style 114](#_Toc4274746)

[2.4 JAX-WS Document Style 119](#_Toc4274747)

[2.5 JAX-WS Tools 121](#_Toc4274748)

[3. JAX-RS (RESTFul web services) 128](#_Toc4274749)

[3.1 JAX-RS Annotations 129](#_Toc4274750)

[3.2 JAX-RS JERSEY 130](#_Toc4274751)

[3.3 JAX-RS RESTEasy 136](#_Toc4274752)

[3.4 JAX-RS Examples 139](#_Toc4274753)

[References 175](#_Toc4274754)

Change Table Content 🡪 Top 🡪 REFERENCES (5th Tab) 🡪update content

# 1. SQL Basics

## I.Introduction to SQL

SQL is a standard language for accessing and manipulating databases

**RDBMS**

* RDBMS stands for Relational Database Management System.
* RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
* The data in RDBMS is stored in database objects called tables.
* A table is a collection of related data entries and it consists of columns and rows

**1. Database Tables:** A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data

**2. SQL Statements** Most of the actions you need to perform on a database are done with SQL statements. SQL keywords are **NOT case sensitive**: **select** is the same as **SELECT**

## 1.1 SQL Operations

Connecting MySQL with XAMPP commndline

cd C:\xampp\mysql\bin

c:

mysql.exe -u root -p

root

MariaDB [(none)]> **show databases;**

+--------------------+

| Database |

+--------------------+

| mysql |

| student |

| test |

+--------------------+

8 rows in set (0.00 sec)

MariaDB [(none)]> use student

Database changed

MariaDB [student]> **show tables;**

+--------------------+

| Tables\_in\_student |

+--------------------+

| student |

| user |

+--------------------+

Some of the Most Important SQL Commands

1. **CREATE DATABASE** - creates a new database

**CREATE** **DATABASE** `jdbc`

1. **DROP DATABASE -** modifies a database

**DROP** **DATABASE** `jdbc`

1. **CREATE TABLE -** creates a new table

**CREATE** **TABLE** `student` (

`sno` **INT** **NOT** **NULL**,

`name` **VARCHAR(50)** **NULL**,

`address` **VARCHAR(50)** **NULL**,

**PRIMARY** **KEY** (`sno`)

);

1. **ALTER TABLE -** modifies a table

**ALTER** **TABLE** `student` **ADD** **COLUMN** `city` **INT**(11) **NOT** **;**

1. **DROP TABLE -** deletes a table

**DROP** **TABLE** `student;

1. **CREATE INDEX -** creates an index (search key)

**ALTER** **TABLE** `student` **ADD** **UNIQUE** **INDEX** `city` (`city`);

1. **DROP INDEX -** deletes an index

**ALTER** **TABLE** `student` **DELETE** **UNIQUE** **INDEX** `city` (`city`);

1. **INSERT INTO -** inserts new data into a database

**INSERT** **INTO** `student` (`name`, `address`) **VALUES** ('Ravi', 'HYD');

1. **SELECT -** extracts data from a database
2. **UPDATE -** updates data in a database
3. **DELETE -** deletes data from a database

### Select Operations

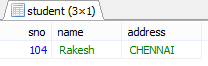
**SELECT** \* **FROM** `student`;



**SELECT** s.sno, s.name **FROM** `student` s;



**SELECT** \* **FROM** `student` s **WHERE** s.sno =104;



**SELECT** \* **FROM** `student` s **WHERE** s.name="Satya" **AND** s.address="HYD";



**SELECT** \* **FROM** `student` s **WHERE** s.name="Satya" **OR** s.address="HYD";

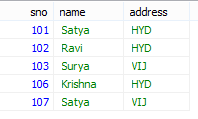


**SELECT** \* **FROM** `student` s **WHERE** s.name **like** '%a';

* **like** '%a' 🡺 **Ending with ‘a’**
* **like** 'a%' 🡺 **Starting with ‘a’**
* **like** '%a%' 🡺 **Contains with ‘a’**
* **like** '%' 🡺 **Contains any**

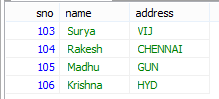
**SELECT** \* **FROM** `student` s **WHERE** s.address **IN** ('HYD', 'VIJ');

**IN operator allows you to apply multiple where conditions**



**SELECT** \* **FROM** `student` s **WHERE** s.sno **BETWEEN** 103 **AND** 106

(Mostly used in date comparision)



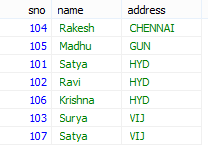
**SELECT** **DISTINCT** s.address **FROM** `student` s;



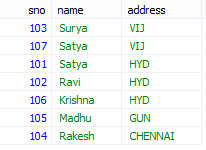
**SELECT** s.name **AS** 'Student\_Name', s.address **AS** 'CITY' **FROM** `student` s



**SELECT** \* **FROM** `student` s **ORDER** **BY** s.address **asc**; (default)



**SELECT** \* **FROM** `student` s **ORDER** **BY** s.address **desc**;



**UPDATE** student` **SET** `name`='Vishnu' **WHERE** `sno`=107 **AND** `name`='Satya' **AND** `address`='VIJ' **LIMIT** 1;

**DELETE** **FROM** `mydb`.`student` **WHERE** `sno`=107 **AND** `name`='Vishnu' **AND** `address`='VIJ' **LIMIT** 1;

## 1.2 JOINS

An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

**Rules**

1. At least on common column must be present beween two tables
2. PRIMARY\_KEY of one table as FORIEN\_KEY of another table is recommended.

**PRIMARY KEY Constraint**

* The PRIMARY KEY constraint uniquely identifies each record in a database table.
* Primary keys must contain UNIQUE values.
* A primary key column cannot contain NULL values.
* Table can have **only ONE primary key.**

**FOREIGN KEY Constraint**

* A **FOREIGN KEY in one table points to a PRIMARY KEY in another table**.

**CASCADE** will propagate the change when the parent changes. (If you delete a row, rows in constrained tables that reference that row will also be deleted, etc.)

**SET NULL** sets the column value to NULL when a parent row goes away.

**RESTRICT** causes the attempted DELETE of a parent row to fail.

**CREATE** **TABLE** `customer` (

`cid` **INT** **NOT** **NULL**,

`name` **VARCHAR**(50) **NULL**,

`addr` **VARCHAR**(50) **NULL**,

**PRIMARY** **KEY** (`cid`)

);

**CREATE** **TABLE** `order` (

`ord\_id` **INT** **NOT** **NULL**,

`ord\_item` **VARCHAR**(50) **NULL**,

`ord\_date` **VARCHAR**(50) **NULL**,

`cid` **INT** **NULL**,

**PRIMARY** **KEY** (`ord\_id`),

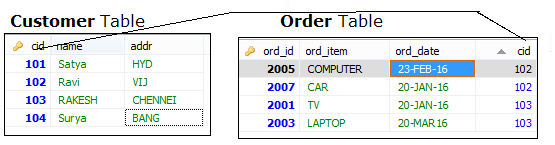
**CONSTRAINT** `FK\_\_customer` **FOREIGN** **KEY** (`cid`) **REFERENCES** `customer` (`cid`) **ON** **UPDATE** **CASCADE** **ON** **DELETE** **CASCADE**

)

**COLLATE**='utf8\_general\_ci'

**ENGINE**=**InnoDB**;

**SELECT** `DEFAULT\_COLLATION\_NAME` **FROM** `information\_schema`.`SCHEMATA` **WHERE** `SCHEMA\_NAME`='mydb';



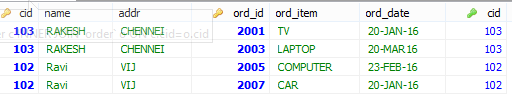
We have 4 types of Joins

1. **INNER JOIN**:

Returns all rows when there is at least one match in BOTH tables



**SELECT** \* **FROM** customer c **INNER** **JOIN** `order` o **ON** c.cid=o.cid

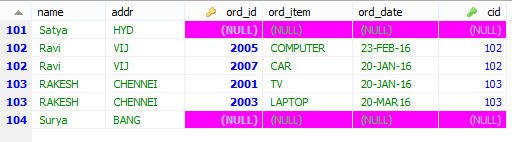


**2. LEFT JOIN:**

Return **all rows from the left table, and the matched rows from the right table.** It will fill rows with **NULL** for unmatched rows on **right**



**SELECT** \* **FROM** customer c **LEFT** **JOIN** `order` o **ON** c.cid=o.cid



**3. RIGHT JOIN**:

**Return all rows from the right table, and the matched rows from the left table.** It will fill rows with **NULL** for unmatched rows on **left**.



**SELECT** \* **FROM** customer c **RIGHT** **JOIN** `order` o **ON** c.cid=o.cid



**4. FULL JOIN:**

Return **all rows when there is a match in ONE of the tables**



**SELECT** \* **FROM** customer c **FULL** **OUTER** **JOIN** `order` o **ON** c.cid=o.cid

In MySQL we don’t have Full Outer Join, for this we have to use Unions

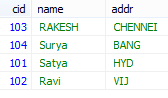
**UNION Operator**

SQL UNION operator **combines the result of two or more SELECT statements**

**SELECT** \* **FROM** customer c **WHERE** c.cid>102

**UNION**

**SELECT** \* **FROM** customer c



* The UNION operator selects only distinct values by default.
* To allow duplicate values, to get duplicates also use **UNION** **ALL**
* Column name must be equal in TWO tables

**SELECT INTO**

SELECT INTO statement **copies data from one table** and inserts it into a **new table**.

**SELECT** \* **INTO** old\_customer **FROM** customer; (mysql not supporting)

**INSERT INTO SELECT**

It selects data from one table and inserts it into an **existing table**

**Constraints**

* NOT NULL - Indicates that a column cannot store NULL value
* UNIQUE - Ensures that each row for a column must have a unique value
* PRIMARY KEY - A combination of a NOT NULL and UNIQUE
* FOREIGN KEY - Ensure the referential integrity of the data in one table to match values in another table
* CHECK - Ensures that the value in a column meets a specific condition
* DEFAULT - Specifies a default value for a column

**Views**

View is a **virtual table based on the result-set of an SQL statement**.

* A view contains rows and columns, just like a real table.
* The fields in a view are fields from one or more real tables in the database

**CREATE OR REPLACE VIEW** view\_name **AS**  
SELECT column\_name(s)  
FROM table\_name  
WHERE condition

**CREATE** **OR** **REPLACE** **VIEW** [Current Product **List**] **AS**

**SELECT** ProductID,ProductName,Category

**FROM** Products

**WHERE** Discontinued=**No**

We can perform INSERT, DELETE, UPDATE Opetations as normal as Table operations

**Date Functions**

The following table lists the most important built-in date functions in MySQL:

|  |  |
| --- | --- |
| **Function** | **Description** |
| [**NOW()**](http://www.w3schools.com/sql/func_now.asp) | **Returns the current date and time** |
| [**CURDATE()**](http://www.w3schools.com/sql/func_curdate.asp) | **Returns the current date** |
| [**CURTIME()**](http://www.w3schools.com/sql/func_curtime.asp) | **Returns the current time** |
| [**DATE()**](http://www.w3schools.com/sql/func_date.asp) | **Extracts the date part of a date or date/time expression** |
| [**EXTRACT()**](http://www.w3schools.com/sql/func_extract.asp) | **Returns a single part of a date/time** |
| [**DATE\_ADD()**](http://www.w3schools.com/sql/func_date_add.asp) | **Adds a specified time interval to a date** |
| [**DATE\_SUB()**](http://www.w3schools.com/sql/func_date_sub.asp) | **Subtracts a specified time interval from a date** |
| [**DATEDIFF()**](http://www.w3schools.com/sql/func_datediff_mysql.asp) | **Returns the number of days between two dates** |
| [**DATE\_FORMAT()**](http://www.w3schools.com/sql/func_date_format.asp) | **Displays date/time data in different formats** |

## 1.3 Sql Functions

**Number functions**

* **AVG()** - Returns the average value
* **COUNT()**  - Returns the number of rows
* **FIRST(**) - Returns the first value
* **LAST()** - Returns the last value
* **MAX()** - Returns the largest value
* **MIN()** - Returns the smallest value
* **SUM()** - Returns the sum

**String functions**

* **UCASE()**  - Converts a field to upper case
* **LCASE()** - Converts a field to lower case
* **MID()** - Extract characters from a text field
* **LEN()** - Returns the length of a text field
* **ROUND()**  - Rounds a numeric field to the number of decimals specified
* **NOW()** - Returns the current system date and time
* **FORMAT()**  - Formats how a field is to be displayed

## 1.4 PL/SQL

**PL/SQL** stands for (Procedural Language/Structure Query Language). It is extension of SQL

## 2.1 Bascis

1. **Datatpye**

Below are supported Datatypes in PL/SQL which are supported in SQL.

|  |  |  |
| --- | --- | --- |
|  | **Data type** | **Syntax** |
| **1** | **Integer** | **INTEGER** |
| **2** | **Smallint** | **SMALLINT** |
| **3** | **Numeric** | **NUMERIC(P,S)** |
| **4** | **Real** | **REAL** |
| **5** | **Decimal** | **DECIMAL(P,S)** |
| **6** | **Float** | **FLOAT(P)** |
| **7** | **Character** | **CHAR(X)** |
| **8** | **Character varying** | **VARCHAR2(X)** |
| **9** | **Date** | **Date** |
| **10** | **Time** | **TIME** |

1. **Variables**

**Variable** Declaration

**Variable\_name datatype;**

**a number;**

**Variable Initialization**

**Variable\_name datatype:=value;**

**a number=10;**

**3. Input Statement** : to provide input value at run time by using operator (&).

**4. Output Statement:**

**dbms\_output.put\_line ('Message' || variable);**

**dbms\_output.put\_line ('Sum: ' || c);**

Basic Syntax

**begin**

dbms\_output.put\_line("Hello word");

**end;**

Very simple, in the place of { }, we use **begin … end;**

### Procedure

**A pl-sql Procedure does not return any value. Procedure has two sections;**

* **Declaration of the procedure:** Declaration of procedure always start with a keyword create ends with last **variable parameters**.
* **Body of the procedure:** Body of procedure starts with a keyword called **is or as** and ends with end statement.

Example

create **or** replace procedure p1(a **in** number, b **out** number, c **out** number, d **out** number)

**is**

**begin**;

**select** ename, sal, deptno, **into** b, c, d **from** emp **where** empno=a;

**end**;

### Function

A PL/SQL Function is a self control block which is used to perform some specific task. **function must always return a value**, but a procedure may or may not return a value.

create **or** replace **function** add(a number, b number)

**return** number

**is**

number;

**begin**;

a:=10;

b:=20;

c=a+b;

**return** c;

**end**;

**Package** is a collection of sub-programs that means function or procedure

**create package package\_name** AS

procedure procedure\_name(parameter);

end package\_name;

### Cursors

A **Cursors** is a temporary work area created in the system memory when a SQL statement is executed. It is a temporary memory which is used to fetch more than one record at a time from existing table.

**Implicit cursor** cursor is perform by the system internally

**Explicit cursor** This type of cursor is performed by the user manually or programatically

### Steps to perform cursor

|  |  |
| --- | --- |
| **Steps** | **Syntax** |
| **Declare the cursor** | **open cursor\_name;** |
| **Open the cursor** | **open cursor\_name;** |
| **Fetch the record from the cursor** | **fetch cursor\_name into variables;** |
| **Close the cursor** | **close cursor\_name;** |

**Declare the cursor**

declare

a emp %rowtype;

cursor c **is** **select** \* **from** emp **where** depno=&deptno;

**begin**

open c;

loop fetch c **into** a;

**if** c % found **then**

dbms\_output.put\_line(a.empno || ' ' a.ename || ' ' || a.sal);

**else**

**exit**;

**end** **if**;

**end** loop;

close c;

**end**;

Trigger

Trigger is a pl/sql block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed

**Purpose of Triggers**

Triggers can be written for the following purposes:

* Generating some derived column values automatically
* Enforcing referential integrity
* **Event logging and storing information on table access Auditing**
* Synchronous replication of tables
* **Imposing security authorizations**
* To avoid invalid transactions
* To generate the resulting data automatically.

**Part of Trigger**

A database trigger has 5 parts.

* Trigger timing
* Trigger event or statement
* Trigger level
* Trigger restriction
* Trigger body

CREATE [OR REPLACE ] TRIGGER trigger\_name

{BEFORE | AFTER | INSTEAD OF }

{INSERT [OR] | UPDATE [OR] | DELETE}

[OF col\_name]

ON table\_name

[REFERENCING OLD AS o NEW AS n]

[FOR EACH ROW]

WHEN (condition)

**BEGIN**

--- sql statements

**END**;

**Write a trigger to stop delete operation on emp\_table**

create **or** replace trigger mytrigger

before

**delete**

on emp

**begin**

raise\_application\_error(-20000, 'sorry we can not delete any record from this table');

**end**;

***References***

* <http://www.w3schools.com/sql/>
* <http://www.sitesbay.com/plsql/>

# 2. MongoDB

MongoDB is an open-source **NoSQL, Document Database** Written in **C++** that provides high performance, high availability, and automatic scaling.

## 1.Introduction

### 1.1 NoSQL (Not Only SQL)

It provides a mechanism for storage and retrieval of data other than tabular relations model used in relational databases. NoSQL database doesn’t use tables for storing data. It is generally used to store big data and real-time web applications.

**NoSQL Database Types**

* **Document databases:** Documents can contain many different key-value pairs, or key-array pairs.
* **Graph stores:** are used to store networks of data, such as social connections.
* **Key-value stores:** simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value.
* **Wide-column** stores such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.

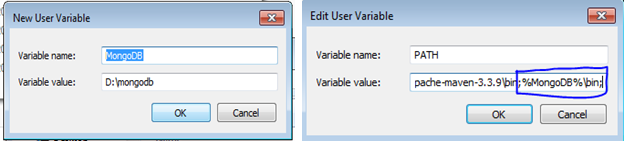


### 1.2 MongoDB Features & Advantages

* **High Performance** : Indexes support faster queries
* **Rich Query Language**: supports CRUD Operation, Data Aggregation, Text Search &Geospatial Queries.
* **High Availability**: MongoDB’s replication facility, called replica set provide atomatic failover and Data redundancy.
* **Horizontal Scalability**: Sharding (partitioning) distributes data across a cluster of machines.

### 1.3 MongDB Installation & Configuration

The MongoDB does not require installation, just download and extracts the zip file, configure the data directory and start it with command “mongod“.

* [**Download**](https://www.mongodb.org/dl/win32/x86_64-2008plus-ssl?_ga=1.58892013.1757468972.1486358241) MongoDB and extract into some folder, ex: **d:/mongodb**
* Create following directories inside **d:/mongodb**
  + **D:\mongodb\data**
  + **D:\mongodb\log**
* Configure Environmnet variables MongoDB = D:\mongodb PATH=%MongoDB%\bin
* Create a mongodb config file under : d:\mongodb\mongo.config

##store data here

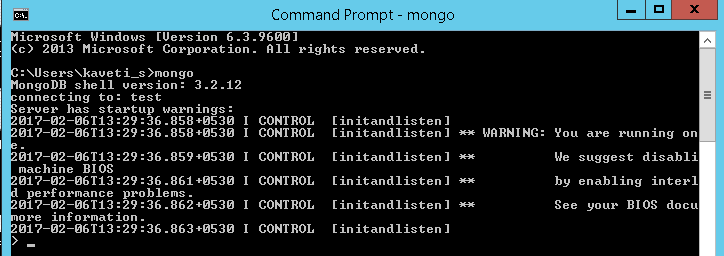
dbpath=D:\mongodb\data

##all output go here

logpath=D:\mongodb\log\mongo.log

##log read and write operations

diaglog=3

* Start MongoDB using any of below commands
  + **D:\mongodb\bin>mongod –dbpath=D:/mongodb**
  + **d:\mongodb\bin>mongod.exe --config="D:\mongodb\mongo.config"**
* Connect to MongoDB using **mongo** command

To start MongoDB Service

net start MongoDB

To stop MongoDB Service

net stop MongoDB

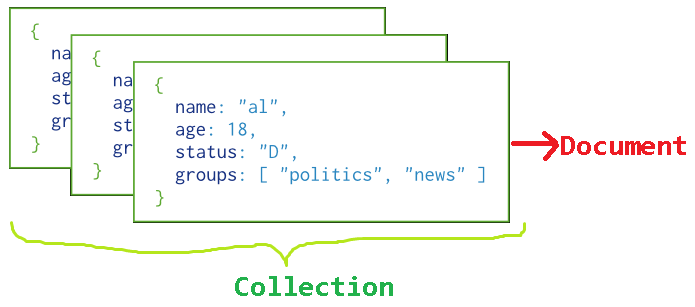
To remove MongoDB Service

c:\mongodb\bin>mongod --remove

## 2. MongoDB Operations

### 2.1 Database – Collection – Documents

* **Document**: is a single entry / record. i.e., **single row** in a table
* **Collection**: Group of Documents are known as Collection. i.e., **Table**
* **Database**: Group of Collections are known as **Database**



### 2.2 Database Operations

#### 1. Show all Databases

Syntax: >show dbs

> show dbs

local 0.000GB

smlcodes 0.000GB

#### 2. Create / Use Database

Syntax: use <database\_name>

> use smlcodes

switched to db smlcodes

Here, your created database "smlcodes" is not present in the list, insert at least one document into it to display database

#### 2. Check the currently selected database

Syntax: >db

> db

Smlcodes

#### 4. Drop Database

Syntax: >db.dropDatabase()

> db.dropDatabase()

{ "dropped" : "smlcodes", "ok" : 1 }

### 2.3 Collection Operations

Usually we don’t need to create collection. MongoDB creates collection automatically when you insert some documents.

**Example**: Insert a document named “admin” into a collection named “users”. The operation will create the collection if the collection does not currently exist

> db.users.insert({"username":"admin", "password":"Admin@123"})

WriteResult({ "nInserted" : 1 })

> show collections

users

We can also create collection by using db.createCollection(name, options)

#### 1. Create Collection

Syntax: db.createCollection(name, options)

* **Name**: is a string type, specifies the name of the collection to be created.
* **Options**: is a document type, specifies the memory size and indexing of the collection. (optional)

> db.createCollection("books")

{ "ok" : 1 }

#### 2.check the collections in the database

Syntax: show collections()

> show collections

books

users

#### 3. Drop Collection

Syntax: db.<COLLECTION\_NAME>.drop()

> db.books.drop();

true

> show collections

users

The drop command returns true if it successfully drops a collection. It returns false when there is no existing collection to drop.

### 2.4 Document Operations

|  |  |
| --- | --- |
| **Data Types** | Description |
| **String** | String is the most commonly used datatype. It is used to store data |
| **Integer** | Integer is used to store the numeric value. It can be 32 bit or 64 bit depends on server |
| **Boolean** | This datatype is used to store boolean values. It just shows YES/NO values. |
| **Double** | Double datatype stores floating point values. |
| **Min/Max Keys** | This datatype compare a value against the lowest and highest bson elements. |
| **Arrays** | This datatype is used to store a list or multiple values into a single key. |
| **Object** | Object datatype is used for embedded documents. |
| **Null** | It is used to store null values. |
| **Symbol** | It is generally used for languages that use a specific type. |
| **Date** | This datatype stores the current date or time in unix time format |

#### 2.4.1 Insert Documents

MongoDB provides the following methods for inserting documents into a collection:

1. **db.collection.insert()**
2. **db.collection.insertOne()**
3. **db.collection.insertMany()**

**If the collection does not currently exist, insert operations will create the collection.**

**\_id Field:** In MongoDB, each document stored in a collection requires a **unique \_id field that acts as a primary key**. If an inserted document omits the \_id field, the MongoDB driver automatically generates an ObjectId for the \_id field.

**1.db.collection.insert():**

**Inserts a single document or multiple documents into a collection**. To insert a single document, pass a document to the method; to insert multiple documents, pass an array of documents to the method

db.users.insert(

{

username: "Satya",

password: "Satya@134",

age:27,

status:"active"

}

)

WriteResult({ "nInserted" : 1 })

**2.db.collection.insertOne()**

Inserts a **single document** into a collection

> db.users.insertOne(

{

username: "Smlcodes",

password: "Smlcodes@134",

age:27,

status:"active"

}

)

{

"acknowledged" : true,

"insertedId" : ObjectId("58986791fb8a774546289da0")

}

**3.db.collection.insertMany()**

Inserts multiple documents into a collection.

db.users.insertMany(

[

{ username:"Surya", password:"Password@1345", age: 42, status: "inactive", },

{ username:"Ravi", password:"Password@1345", age: 22, status: "inactive", },

{ username:"Rakesh", password:"Password@1345", age: 34, status: "active", }

]

)

----

... )

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("589868c4fb8a774546289da1"),

ObjectId("589868c4fb8a774546289da2"),

ObjectId("589868c4fb8a774546289da3")

]

}

#### 2.4.2 Query Documents (find Operations)

MongoDB provides the **db.collection.find()** method to read documents from a collection. The **db.collection.find()** method returns **a cursor** to the matching documents.

Syntax > db.collection.find( <query filter>, <projection> )

* **<query filter>:** a query filter to specify which documents to return.
* **< projection>:** which fields from the matching documents to return

**1.Select All Documents in a Collection**

An empty query filter document ({}) selects all documents in the collection

Syntax > db.users.find( {} ) or db.users.find()

> db.users.find({})

{ "\_id" : ObjectId("58986156fb8a774546289d9e"), "username" : "admin", "password" : "Admin@123" }

{ "\_id" : ObjectId("58986716fb8a774546289d9f"), "username" : "Satya", "password" : "Satya@134", "age" : 27, "status" : "active" }

{ "\_id" : ObjectId("58986791fb8a774546289da0"), "username" : "Smlcodes", "password" : "Smlcodes@134", "age" : 27, "status" : "active" }

{ "\_id" : ObjectId("589868c4fb8a774546289da1"), "username" : "Surya", "password" : "Password@1345", "age" : 42, "status" : "inactive" }

**2.Select All Documents with Condition**

> db.users.find( { status: "active" } )

{ "\_id" : ObjectId("58986716fb8a774546289d9f"), "username" : "Satya", "password" : "Satya@134", "age" : 27, "status" : "active" }

{ "\_id" : ObjectId("58986791fb8a774546289da0"), "username" : "Smlcodes", "password" : "Smlcodes@134", "age" : 27, "status" : "active" }

{ "\_id" : ObjectId("589868c4fb8a774546289da3"), "username" : "Rakesh", "password" : "Password@1345", "age" : 34, "status" : "active" }

**3.Select All Documents with AND Condition**

We can specify the the no. of conditions by using **comma (,)** operator

Retrieves all documents where **status equals "active"** **and** **age is less than ($lt) 30:**

> db.users.find( { status: "active", age: { $lt: 30 } } )

{ "\_id" : ObjectId("58986716fb8a774546289d9f"), "username" : "Satya", "password" : "Satya@134", "age" : 27, "status" : "active" }

{ "\_id" : ObjectId("58986791fb8a774546289da0"), "username" : "Smlcodes", "password" : "Smlcodes@134", "age" : 27, "status" : "active" }

**4.Select All Documents with OR Condition**

Using the **$or** operator, you can specify a compound query that joins each clause with a logical OR conjunction so that the query selects the documents in the collection that match at least one condition.

Retrieves all documents where the **status equals** **"inactive" or age is less than ($lt) 30:**

db.users.find(

{

$or: [ { status: "inactive" }, { age: { $lt: 30 } } ]

}

)

-------

{ "\_id" : ObjectId("58986716fb8a774546289d9f"), "username" : "Satya", "password" : "Satya@134", "age" : 27, "status" : "active" }

{ "\_id" : ObjectId("589868c4fb8a774546289da1"), "username" : "Surya", "password" : "Password@1345", "age" : 42, "status" : "inactive" }

For more related Query Documents visit [**MongoDB Offcial website**](https://docs.mongodb.com/v3.2/tutorial/query-documents/)

#### 2.4.3 Update Documents

MongoDB provides the following methods for updating documents in a collection

1. **db.collection.update()**
2. **db.collection.updateOne()**
3. **db.collection.updateMany()**
4. **db.collection.replaceOne()**

Once set, you cannot update the value of the \_id field nor can you replace an existing document with a replacement document that has a different \_id field value.

**1.db.collection.update()**

Either updates or replaces a **single document** that match a specified filter **or updates all documents** that match a specified filter.

db.users.update(

{ "status": "inactive" },

{

$set: { "Level": 2}

}

)

------

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

**2.db.collection.updateOne()**

**Updates at most a single document** that match a specified filter even though multiple documents may match the specified filter.

db.users.updateOne(

{ "username": "Surya" },

{

$set: { "password": "901290190", age: 20 }

}

)

------

{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

**3.db.collection.updateMany()**

**Update all documents** that match a specified filter.

db.users.updateMany(

{ "status": "active" },

{

$set: { "Level": 1}

}

)

----------

{ "acknowledged" : true, "matchedCount" : 6, "modifiedCount" : 6 }

**4.db.collection.replaceOne()**

**Replaces at most a single document** that match a specified filter even though multiple documents may match the specified filter.

#### 2.4.4 Delete Documents

MongoDB provides the following methods to delete documents of a collection:

1. **db.collection.remove()**
2. **db.collection.deleteOne()**
3. **db.collection.deleteMany()**

**Delete operations do not drop indexes, even if deleting all documents from a collection**

**1.db.collection.remove()**

**To remove all documents** from the collection based on condition

> db.users.remove( { age: 27 } )

WriteResult({ "nRemoved" : 2 })

**2.db.collection.deleteOne()**

Delete at most a single document that match a specified filter,even though multiple documents may match

> db.users.deleteOne( { status: "active" } )

"acknowledged" : true, "deletedCount" : 1 }

**3.db.collection.deleteMany()**

**To remove all documents** from the collection based on condition

> db.users.deleteMany({ status : "inactive" })

"acknowledged" : true, "deletedCount" : 5 }

**The following methods can also delete documents from a collection:**

* **db.collection.findOneAndDelete().**
* **findOneAndDelete()** provides a sort option. The option allows for the deletion of the first document sorted by the specified order.
* **db.collection.findOneAndModify().**
* **db.collection.findOneAndModify()** provides a sort option. The option allows for the deletion of the first document sorted by the specified order.
* **db.collection.bulkWrite().**

#### 2.4.5 Advanced Operations

**limit()** To limit the records in MongoDB, you need to use **limit()** method

>db.COLLECTION\_NAME.find().limit(NUMBER)

**Skip()** used to skip the number of documents.

>db.COLLECTION\_NAME.find().limit(NUMBER).skip(NUMBER)

**sort()** specify sorting order. 1 is used for ascending order while -1 is used for descending order.

>db.COLLECTION\_NAME.find().sort({KEY:1})

**aggregate()** aggregation in MongoDB, you should use aggregate() method.

>db.COLLECTION\_NAME.aggregate(AGGREGATE\_OPERATION)

AGGREGATE\_OPERATION = $sum, $avg, $min, $max, $push, $addToSet, $first, $last

**db.collection.save()**

Updates an existing document or inserts a new document, depending on its document parameter.

db.products.save( { item: "book", qty: 40 } )

**help()** uses to guide you how to do things in MongoDB.

db.help() help on db methods

db.mycoll.help() help on collection methods

sh.help() sharding helpers

rs.help() replica set helpers

help admin administrative help

help connect connecting to a db help

help keys key shortcuts

help misc misc things to know

help mr mapreduce

show dbs show database names

show collections show collections in current database

show users show users in current database

show profile show most recent system.profile entries time >= 1ms

show logs show the accessible logger names

show log [name] prints out the last segment of log in memory,

use <db\_name> set current database

db.foo.find() list objects in collection foo

db.foo.find( { a : 1 } ) list objects in foo where a == 1

it result of the last line evaluated; use to further iterate

DBQuery.shellBatchSize = x set default number of items to display on shell

exit quit the mongo shell

## 3. MongoDB with Java

To use MongoDB in our Java programs, we need MongoDB JDBC driver. Follow the below steps to do so.

Steps to Connect with MongoDB Using Java

**1.Create Java Project using Elipse Convet that into Maven Project**

**2.** **Download mongo-java driver from**[**github**](https://github.com/mongodb/mongo-java-driver/downloads)**. Or declare mongo-java driver in pom.xml**

<dependencies>

<dependency>

<groupId>org.mongodb</groupId>

<artifactId>mongo-java-driver</artifactId>

<version>2.10.1</version>

</dependency>

</dependencies>

**3. Write a Java class to connect with MongoDB & perform operations**

package core;

import java.net.UnknownHostException;

import com.mongodb.DB;

import com.mongodb.DBCollection;

import com.mongodb.MongoClient;

import com.mongodb.MongoException;

public class MongoDBConnect {

public static void main(String[] args) {

try {

/\*\*\*\* Connect to MongoDB \*\*\*\*/

// Since 2.10.0, uses MongoClient

MongoClient mongo = new MongoClient("localhost", 27017);

/\*\*\*\* Get database \*\*\*\*/

// if database doesn't exists, MongoDB will create it for you

DB db = mongo.getDB("smlcodes");

/\*\*\*\* Get collection / table from 'testdb' \*\*\*\*/

// if collection doesn't exists, MongoDB will create it for you

DBCollection table = db.getCollection("user");

if (mongo != null) {

System.*out*.println("============\n MongoDB Connected!!! \n===========");

System.*out*.println("Database Name : " + db.getName());

System.*out*.println("Collection : " + table.getName());

}

} catch (UnknownHostException e) {

e.printStackTrace();

} catch (MongoException e) {

e.printStackTrace();

}

}

}

Output

MongoDB Connected!!!

===========

Database Name : smlcodes

Collection : user

### 3.1 MongoDB with Java

**1.Mongo Connection**

Connect to MongoDB server. For MongoDB version >= 2.10.0, uses MongoClient.

// Old version, uses Mongo

Mongo mongo = new Mongo("localhost", 27017);

// Since 2.10.0, uses MongoClient

MongoClient mongo = new MongoClient( "localhost" , 27017 );

**2.Mongo Database**

Get database. If the database doesn’t exist, MongoDB will create it for you.

DB db = mongo.getDB("database name");

If MongoDB in secure mode, authentication is required

boolean auth = db.authenticate("username", "password".toCharArray());

Display all databases.

List<String> dbs = mongo.getDatabaseNames();

for(String db : dbs){

System.out.println(db);

}

**3.Mongo Collection**

Get collection / table.

DB db = mongo.getDB("testdb");

DBCollection table = db.getCollection("user");

Display all collections from selected database.

DB db = mongo.getDB("testdb");

Set<String> tables = db.getCollectionNames();

for(String coll : tables){

System.out.println(coll);

}

Steps to develop any MongoDB Application

1.Connect with MongoDB Server

MongoClient mongoClient = new MongoClient("localhost", 27017);

2.Connect with Database

DB db = mongoClient.getDB("smlcodes");

3.Get the Collection , on which collection you want to work

DBCollection collection = db.getCollection("users");

4.Get Document Object to perform CURD operaions on Document

BasicDBObject document = new BasicDBObject();

1. MongoDB Authenetication Example

Add user to smlcodes Collection for testing purpose

db.createUser(

{

user: "admin",

pwd: "admin",

roles: [

{role: "readWrite", db: "smlcodes"}

]

}

)

**Example**

import com.mongodb.DB;

import com.mongodb.MongoClient;

public class MongoDB\_Authentication {

public static void main(String args[]) {

try {

// To connect to mongodb server

MongoClient mongoClient = new MongoClient("localhost", 27017);

// Now connect to your databases

DB db = mongoClient.getDB("smlcodes");

System.*out*.println("Connect to database successfully");

boolean auth = db.authenticate("admin", "admin".toCharArray());

System.*out*.println("Authentication: " + auth);

} catch (Exception e) {

System.*err*.println(e.getClass().getName() + ": " + e.getMessage());

}

}

}

3.1.2 MongoDB Java Complete Example

package core;

import java.net.UnknownHostException;

import java.util.HashMap;

import java.util.Map;

import com.mongodb.BasicDBObject;

import com.mongodb.BasicDBObjectBuilder;

import com.mongodb.DB;

import com.mongodb.DBCollection;

import com.mongodb.DBCursor;

import com.mongodb.DBObject;

import com.mongodb.Mongo;

import com.mongodb.MongoException;

import com.mongodb.util.JSON;

public class MongoDB\_Insert {

public static void main(String[] args) {

try {

Mongo mongo = new Mongo("localhost", 27017);

DB db = mongo.getDB("smlcodes");

DBCollection collection = db.getCollection("users");

collection.remove(new BasicDBObject());

// 1. BasicDBObject example

System.*out*.println("1.BasicDBObject example...");

System.*out*.println("===========================");

BasicDBObject document = new BasicDBObject();

document.put("username", "satyajohnny");

document.put("password", "password254");

BasicDBObject documentDetail = new BasicDBObject();

documentDetail.put("street", "RAMALAYAM");

documentDetail.put("city", "VIJAYAWADA");

documentDetail.put("state", "ANDHRA PRADESH");

document.put("address", documentDetail);

collection.insert(document);

DBCursor cursorDoc = collection.find();

while (cursorDoc.hasNext()) {

System.*out*.println(cursorDoc.next());

}

collection.remove(new BasicDBObject());

// 2. BasicDBObjectBuilder example

System.*out*.println("\n\n 2.BasicDBObjectBuilder Insert");

System.*out*.println("===========================");

BasicDBObjectBuilder documentBuilder = BasicDBObjectBuilder.*start*()

.add("username", "Anil")

.add("password", "Anigirekula123");

BasicDBObjectBuilder documentBuilderDetail = BasicDBObjectBuilder.*start*()

.add("street", "NTR STREET")

.add("city", "HYDERABAD").add("state", "TN");

documentBuilder.add("detail", documentBuilderDetail.get());

collection.insert(documentBuilder.get());

DBCursor cursorDocBuilder = collection.find();

while (cursorDocBuilder.hasNext()) {

System.*out*.println(cursorDocBuilder.next());

}

collection.remove(new BasicDBObject());

// 3. Map example

System.*out*.println("\n\n 3.MAP Insert");

System.*out*.println("===========================");

Map<String, Object> documentMap = new HashMap<String, Object>();

documentMap.put("username", "mapuser");

documentMap.put("password", "mapassword");

Map<String, Object> documentMapDetail = new HashMap<String, Object>();

documentMapDetail.put("street", "JAMES STREET");

documentMapDetail.put("city", "GEORGIO");

documentMapDetail.put("state", "U.S");

documentMap.put("detail", documentMapDetail);

collection.insert(new BasicDBObject(documentMap));

DBCursor cursorDocMap = collection.find();

while (cursorDocMap.hasNext()) {

System.*out*.println(cursorDocMap.next());

}

collection.remove(new BasicDBObject());

// 4. JSON parse example

System.*out*.println("\n\n 4.JSON Insert");

System.*out*.println("===========================");

String json = "{'username' : 'jsonuser','password' : 'JsonPass',"

+ "'detail' : {'street' : 'FIGHTCLUB STREET', 'city' : 'MELBORN', 'state' : 'AUS'}}}";

DBObject dbObject = (DBObject) JSON.*parse*(json);

collection.insert(dbObject);

DBCursor cursorDocJSON = collection.find();

while (cursorDocJSON.hasNext()) {

System.*out*.println(cursorDocJSON.next());

}

collection.remove(new BasicDBObject());

} catch (UnknownHostException e) {

e.printStackTrace();

} catch (MongoException e) {

e.printStackTrace();

}

}

}

//Output

1.BasicDBObject example...

===========================

{ "\_id" : { "$oid" : "589b07a32989f6de61c17c09"} , "username" : "satyajohnny" , "password" : "password254" , "address" : { "street" : "RAMALAYAM" , "city" : "VIJAYAWADA" , "state" : "ANDHRA PRADESH"}}

2.BasicDBObjectBuilder Insert

===========================

{ "\_id" : { "$oid" : "589b07a32989f6de61c17c0a"} , "username" : "Anil" , "password" : "Anigirekula123" , "detail" : { "street" : "NTR STREET" , "city" : "HYDERABAD" , "state" : "TN"}}

3.MAP Insert

===========================

{ "\_id" : { "$oid" : "589b07a32989f6de61c17c0b"} , "password" : "mapassword" , "detail" : { "city" : "GEORGIO" , "street" : "JAMES STREET" , "state" : "U.S"} , "username" : "mapuser"}

4.JSON Insert

===========================

{ "\_id" : { "$oid" : "589b07a32989f6de61c17c0c"} , "username" : "jsonuser" , "password" : "JsonPass" , "detail" : { "street" : "FIGHTCLUB STREET" , "city" : "MELBORN" , "state" : "AUS"}}

In above we are removing inserted Object for display purpose only

collection.remove(new BasicDBObject());

Similarly we can perform CURD operations using below methods in the same way

**Update Operation**

Update a document where “username”=”satya” to SatyaKaveti.

DBCollection table = db.getCollection("user");

BasicDBObject query = new BasicDBObject();

query.put("username", "satya");

BasicDBObject newDocument = new BasicDBObject();

newDocument.put("username", "SatyaKaveti");

BasicDBObject updateObj = new BasicDBObject();

updateObj.put("$set", newDocument);

table.update(query, updateObj);

**Find/Query/Search Operation**

Find document where “username =satya”, and display it with DBCursor

DBCollection table = db.getCollection("user");

BasicDBObject searchQuery = new BasicDBObject();

searchQuery.put("username ", " satya ");

DBCursor cursor = table.find(searchQuery);

while (cursor.hasNext()) {

System.out.println(cursor.next());

}

**Delete Operation**

Find document where “username =satya”, and delete it.

DBCollection table = db.getCollection("user");

BasicDBObject searchQuery = new BasicDBObject();

searchQuery.put("username ", " satya ");

table.remove(searchQuery);

## References

<http://www.javatpoint.com/mongodb-tutorial>

<https://docs.mongodb.com/v3.2/tutorial/>

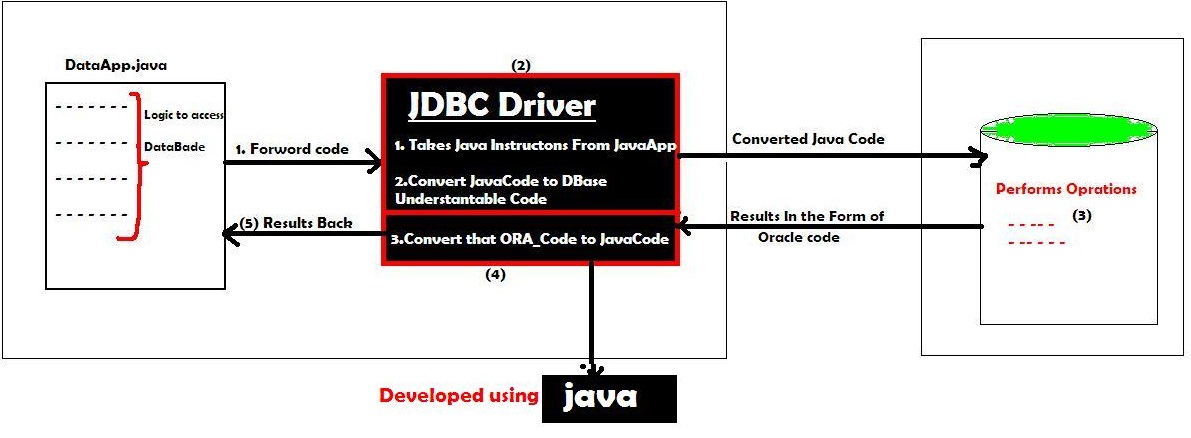
<http://www.mkyong.com/tutorials/java-mongodb-tutorials/>

<https://www.tutorialspoint.com/mongodb/mongodb_environment.htm>

# 2. JDBC

## 2.1 Introduction

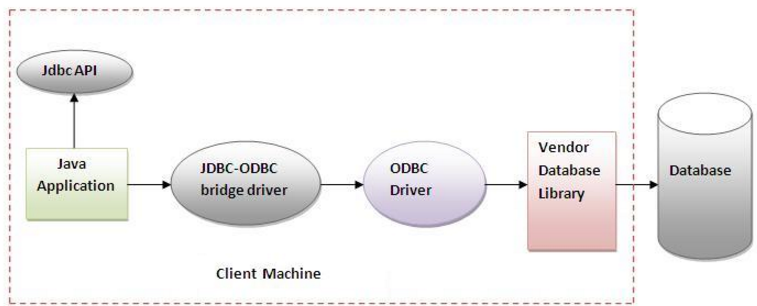
Before JDBC, ODBC API was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured).

That is why Java has defined its own API (JDBC API) that uses JDBC drivers (writtn in Java). 

There are 4 types of JDBC drivers:

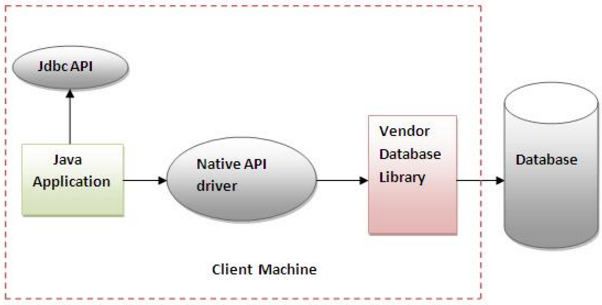
1. **JDBC-ODBC bridge driver**
2. **Native-API driver (partially java driver)**
3. **Network Protocol driver (fully java driver)**
4. **Thin driver (fully java driver)**

**1. JDBC-ODBC bridge driver (Type-1)**

* The JDBC-ODBC bridge driver uses **ODBC driver to connect to the database**.
* The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.
* Can be easily connected to **ANY database.**
* Performance degraded because **JDBC method call is converted into the ODBC calls**
* **ODBC driver needs** to be installed on the **client machine**
* Sun provided ODBC driver name : **sun.jdbc.odbc.JdbcOdbcDriver**

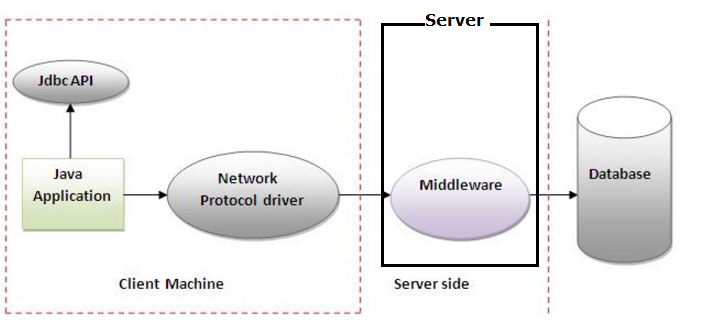
**2. Native API driver (Type-1)**

* The Native API driver uses the client-side libraries of the database.
* For **MySQL they have own Native API Driver, similarly for ORACLE, Postgres etc,**
* The driver converts JDBC (Java) method calls into native calls (MySQL, Oracle) of the database API.
* It is **NOT FULLY written entirely in java**
* The Native driver needs to be installed on the each client machine.

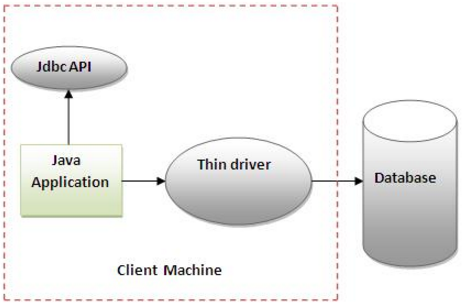


**3. Network Protocol driver (Type-3)**

* The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol.
* It is fully written in java.
* Used in **Connection Pooling**
* **Network support is required on client machine.**
* Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.



**4. Thin driver (Type -4)**

* The thin driver converts JDBC calls directly into the vendor-specific database protocol.
* It is **fully written in Java language**
* Better performance than all other drivers.
* **No software is required at client side or server side**.
* **com.mysql.jdbc.Driver** (MySQL), **oracle.jdbc.driver.OracleDriver**(ORACLE)

**Steps to connect to the database**

There are 5 steps to connect...

1. **Register the driver class**

Class.forName ("oracle.jdbc.driver.OracleDriver");

1. **Creating connection**

Connection con=DriverManager.getConnection ("url","system","password");

Connection con=DriverManager.getConnection ("url "); //2nd Way

1. **Creating statement**

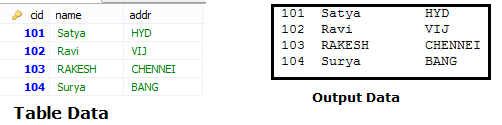
Statement stmt=con.createStatement ();

1. **Executing queries**

ResultSet rs=stmt.executeQuery("select \* from emp");

1. **Closing connection**

con.close();



|  |
| --- |
| **public** **class** JDBC {  **public** **static** **void** main(String[] args) **throws** Exception {  Class.*forName*("com.mysql.jdbc.Driver");  Connection con = DriverManager.*getConnection*  ("jdbc:mysql://localhost:3306/mydb", "root", "123456");  Statement stmt = con.createStatement();  ResultSet rs = stmt.executeQuery("SELECT \* FROM customer");  **while** (rs.next())  System.***out***.println(rs.getInt(1)+":"+rs.getString(2)+ " " + rs.getString(3));  con.close();  }  } |

In above Example we use getConnection(“URL”,”UNAME”,”PWD”) to connect with database.we can use directly without giving username/pwd also

There are two ways to connect java application with the access database.

* **Without DSN (Data Source Name) 🡪 above**
* **With DSN 🡪** jdbc:odbc:mydsn (mydsn is DSN)

**Creating DSN**

Start > Administrative Tools > Data Sources (ODBC). 🡪Add 🡪 (.mdb)

## 2.2 DriverManager class

* The DriverManager class acts as an interface between user and drivers
* It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.
* The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method **DriverManager.registerDriver().**

**Methods**

1. **public static void registerDriver(Driver driver):**
2. **public static void deregisterDriver(Driver driver):**
3. **public static Connection getConnection(String url):**
4. **public static Connection getConnection(String url,String uname,String pwd)**

## 2.3 Connection interface

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e.

* **Statement createStatement()**
* **PreparedStatement prepareStatement(String sql)**
* **CallableStatement prepareCall(String sql)**
* **Blob createBlob()**
* **Clob createClob()**
* **String getSchema()**
* **DatabaseMetaData getMetaData()**
* **void close()**
* **void commit()**
* **void rollback()**
* **void setAutoCommit(boolean autoCommit)**
* **boolean getAutoCommit()**

All above statement related methods can have

* **ResultSet.TYPE\_SCROLL\_SENSITIVE**🡪 used to move ResultSet Both Directions
* **ResultSet.CONCUR\_UPDATABLE** 🡪 used to perform DML operations on ResultSet

## 2.4 Statement interface

* The Statement interface provides methods to execute queries with the database.
* The statement interface is a factory of ResultSet
* it provides factory method to get the object of ResultSet
* **public ResultSet executeQuery(String sql):**

Is used to execute **SELECT query**. It returns the object of ResultSet.

* **public int executeUpdate(String sql):**

Is used to execute DML, **CREATE, DROP, INSERT, UPDATE, DELETE** etc.

* **public boolean execute(String sql):**

Is used to execute queries that may return multiple results. **>1** if success, **0** on fail

* **public int[] executeBatch():**

Is used to execute batch of commands.

## 2.5 ResultSet interface

* The object of ResultSet maintains a cursor pointing to a row of a table.
* Initially, cursor points to before the first row.
* By default, ResultSet object can be **moved forward only** and it **is not updatable**.
  + - **public XXX getXXX(int rowIndex):**
    - **public XXX getXXX(String rowName):**
    - public boolean next()
    - public boolean previous()
    - public boolean first()
    - public boolean last()

executeQuery (String sql) example

We use this method to execute SELECT queries

**public** **class** JDBC {

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

Class.*forName*("com.mysql.jdbc.Driver");

Connection con = DriverManager.*getConnection*

("jdbc:mysql://localhost:3306/mydb", "root", "123456");

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM customer");

**while** (rs.next()) {

System.***out***.println(rs.getInt(1)+""+rs.getString(2)+" "+ rs.getString(3));

}

con.close();

}

}

101 Satya HYD

102 Ravi VIJ

103 RAKESH CHENNEI

104 Surya BANG

executeUpdate (String sql) example

We use this method to **NON-SELECT** Queries like UPDATE, DELETE, etc

Returns 1 if 🡪 success

Returns 0 if 🡪 Failure

**public** **class** JDBC {

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

String url = "jdbc:mysql://localhost:3306/mydb";

Class.*forName*("com.mysql.jdbc.Driver");

Connection con = DriverManager.*getConnection*(url, "root", "123456");

Statement stmt = con.createStatement();

String qry = "UPDATE `customer` SET `name`='Ram' WHERE `cid`=102";

**int** res = stmt.executeUpdate(qry);

**if** (res > 0)

System.***out***.println("Success is :" + res);

**else**

System.***out***.println("Failure is :" + res);

con.close();

}

}

Success is :1

Failure is :0

Boolean execute example

We can use **execute()** method in **both SELECT & NON-SELECT queries.**

**1. SELECT**

It returns TRUE on SELECT queres we can get ResultSet by calling below method

**ResultSet rs = statement.getResultSet ()**

**2. NON-SELECT**

It returns FALSE on NON-SELECT queres. we can get **Int** value by calling below method

**int i = statement.getUpdateCount();**

**public** **class** JDBC {

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

String url = "jdbc:mysql://localhost:3306/mydb";

Class.*forName*("com.mysql.jdbc.Driver");

Connection con = DriverManager.*getConnection*(url, "root", "123456");

Statement stmt = con.createStatement();

// String qry = "SELECT \* FROM customer";

String qry = "UPDATE `customer` SET `name`='Ram' WHERE `cid`=102";

**boolean** flag = stmt.execute(qry);

**if** (flag == **true**) {

System.***out***.println("SELECT QUERY\n --------");

ResultSet rs = stmt.getResultSet();

**while** (rs.next()) {

System.***out***.println(rs.getString(1) + ":" + rs.getString(2));

}

}

**else** {

System.***out***.println("NON-SELECT QUERY\n --------");

**int** i = stmt.getUpdateCount();

System.***out***.println("Result is : " + i);

}

}

}

SELECT QUERY

--------

101:Satya

102:Ram

103:RAKESH

104:Surya

NON-SELECT QUERY

--------

Result is : 1

executeBatch(String sql) example

**public** **class** BatchDemo {

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

String url = "jdbc:mysql://localhost:3306/mydb";

Class.*forName*("com.mysql.jdbc.Driver");

Connection con = DriverManager.*getConnection*(url, "root", "123456");

Statement st = con.createStatement();

st.addBatch("insert into student values(81, 'Syam', 'mtm')");

st.addBatch("insert into student values(11, 'ram', 'mum')");

st.addBatch("insert into student values(14, 'bam', 'kuk')");

st.addBatch("insert into student values(44, 'pram', 'secu')");

**int** rs[] = st.executeBatch();

**int** sum = 0;

**for** (**int** i = 0; i < rs.length; i++) {

sum = sum + i;

}

System.***out***.println(sum + "Record are UPDATED using BATCH");

}

}

Scrollabe Resultset(String sql) example

By Default ResultSet Object is not SCROLLABLE & NOT UPDATABLE.to make ResultSet Object to move both Directions we need to configure TYPE & MODE Values

Possible TYPE Values

* **ResultSet.TYPE\_SCROLL\_SENSITIVE 🡪 (Update Possible)**
* **ResultSet.TYPE\_SCROLL\_INSENSITIVE 🡪 (Default)**

Possible MODE Values

* **ResultSet.CONCUR\_READ\_ONLY 🡪 (Update Possible)**
* **ResultSet.CONCUR\_UPDATABLE 🡪 (Default)**

Methods applicable on Scrolable ResultSet Object

* **int getRow()** 🡪 Returns ROW INDEX
* **boolean first()** 🡪 Keep CURSOR at 1st Record
* **boolean last()** 🡪 Keep CURSOR at LAST Record
* **boolean next()** 🡪 Moves Cursor to Forward
* **boolean previous()** 🡪 Moves Cursor to Backword
* **boolean absolute(int +/-**)🡪 Moves Cursor to given Index on ResultSet
* **boolean relative**(int +/-)🡪 Moves Cursor to given Index, based on current Row

**public** **class** JDBC {

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

String url = "jdbc:mysql://localhost:3306/mydb";

Class.*forName*("com.mysql.jdbc.Driver");

Connection con = DriverManager.*getConnection*(url, "root", "123456"); Statement st = con.createStatement

(ResultSet.***TYPE\_SCROLL\_SENSITIVE***, ResultSet.***CONCUR\_READ\_ONLY***);

ResultSet rs = st.executeQuery("select \* from customer");

System.***out***.println("From Using Next\n------");

**while** (rs.next()) {

System.***out***.println(rs.getString(1)+":"+rs.getString(2));

}

System.***out***.println("\nFrom Using Previous ");

**while** (rs.previous()) {

System.***out***.println(rs.getString(1)+":"+rs.getString(2));

}

System.***out***.println("randomly................... ");

rs.first();

System.***out***.println(rs.getRow()+"First:"+rs.getString(1)+":"+rs.getString(2));

rs.last();

System.***out***.println(rs.getRow()+"Last: "+rs.getString(1)+":"+rs.getString(2));

rs.absolute(4);//from starting point to 4 records

System.out.println(rs.getRow()+"Absolute:"+rs.getString(1)+":"+rs.getString(2));

rs.relative(-2); //from here to 2 points back

System.out.println(rs.getRow()+"relative:"+rs.getString(1)+":"+rs.getString(2));

}

}

From Using Next

------

101:Satya

102:Ram

103:RAKESH

104:Surya

From Using Previous

104:Surya

103:RAKESH

102:Ram

101:Satya

randomly...................

1First: 101:Satya

4Last: 104:Surya

4Absolute Record : 104:Surya

2relative Record : 102:Ram

In above example we used on for SCROLLING resultset on both ditections using ResultSet.***TYPE\_SCROLL\_SENSITIVE***, ResultSet.***CONCUR\_READ\_ONLY.***

If we want perform UPDATE operations & SCROLLING also, we have to use ResultSet.TYPE***\_SCROLL\_SENSITIVE***, ResultSet.***CONCUR\_UPDATABLE***

**Steps to Perform Insert/ UPDATE /Delete Operations on ResultSet**

**1. Select the Records**

**while** (rs.next())

{

System.***out***.println(rs.getRow()+””+rs.getString(1)+","+ rs.getString(2));

}

**2. Perform INSERT Operation**

System.***out***.println("1.INSERT OPERATION\n-----");

rs.moveToInsertRow(); // creates Empty Record

rs.updateInt(1, 200);

rs.updateString(2, "SACHIN");

rs.updateString(3, "MUMBAI");

rs.insertRow(); // Inserts Row

**3. Perform UPDATE Operation**

System.***out***.println("\n2.UPDATE OPERATION\n-----");

rs.absolute(2); // move to row to update

rs.updateString(3, "KOLKATA");

rs.updateRow();

**4. Perform DELETE Operation**

System.***out***.println("\n3.DELTE OPERATION\n-----");

rs.absolute(1); // move to row to DELETE

rs.deleteRow();

Example

**public** **class** JDBC {

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

String url = "jdbc:mysql://localhost:3306/mydb";

Class.*forName*("com.mysql.jdbc.Driver");

Connection con = DriverManager.*getConnection*(url, "root", "123456");

Statement st = con.createStatement

ResultSet.***TYPE\_SCROLL\_SENSITIVE***, ResultSet.***CONCUR\_UPDATABLE***);

ResultSet rs = st.executeQuery("select \* from customer");

**while** (rs.next()) {

System.***out***.println(rs.getRow() + "->" + rs.getString(1) + "," + rs.getString(2));

}

System.***out***.println("1.INSERT OPERATION\n-----");

rs.moveToInsertRow(); // creates Empty Record

rs.updateInt(1, 200);

rs.updateString(2, "SACHIN");

rs.updateString(3, "MUMBAI");

rs.insertRow(); // Inserts Row

System.***out***.println("\n2.UPDATE OPERATION\n-----");

rs.absolute(2); // move to row to update

rs.updateString(3, "KOLKATA");

rs.updateRow();

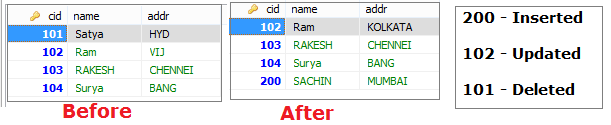
System.***out***.println("\n3.DELTE OPERATION\n-----");

rs.absolute(1); // move to row to DELETE

rs.deleteRow();

}

}



## 2.6 PreparedStatement

The PreparedStatement interface is a sub interface of Statement. It is used to execute parameterized query.The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

String sql="insert into emp values(?,?,?)";

(?) values will be set by calling the setter methods of PreparedStatement.

|  |  |
| --- | --- |
| **Method** | **Description** |
| **public void setInt(int index,int value)** | sets integer value to the given parameter index. |
| **public void setString(int index, String val)** | sets String value to the given parameter index. |
| **public void setFloat(int index, float value)** | sets float value to the given parameter index. |
| **public void setDouble(int index, double val)** | sets double value to the given parameter index. |
| **public int executeUpdate()** | Uses for create, drop, insert, update, delete queries |
| **public ResultSet executeQuery()** | Executes the select query. |
| **Boolean execute()** | It returns TRUE for SELECT queries, FALSE for update, delete Queries |
| **void addBatch()** |  |
| **ParameterMetaData getParameterMetaData()** | Retrieves the number, types and properties of this PreparedStatement object's parameters |
| **ResultSetMetaData getMetaData()** | contains information about the columns of the ResultSet object |
| **void setBlob(int index, Blob x)**  **void setBlob(int index,InputStream is)**  **void setBinaryStream(int i,InputStream is)**  **getBinarayStream(“column”);** | Inserts Binary Large Object Videos,Audios |
| **void setClob(int index, Clob x)**  **void setClob(int index, Reader reader)**  **getCharacterStream(“column”);** | Inserts character Large Object Files |

|  |  |
| --- | --- |
| **java.sql.Statement** | **java.sql.PreparedStatement** |
| Statement is used for executing a static SQL statement in java JDBC. | PreparedStatement is used for executing a precompiled SQL statement in java JDBC. |
| java.sql.Statement cannot accept parameters at runtime in java JDBC. | java.sql.PreparedStatement can be executed repeatedly, it can accept different parameters at runtime in java JDBC. |
| java.sql.Statement is slower as compared to PreparedStatement in java JDBC. | java.sql.PreparedStatement is faster because it is used for executing precompiled SQL statement in java JDBC. |

**public** **class** JDBC {

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

String url = "jdbc:mysql://localhost:3306/mydb";

String u = "root";

String p = "123456";

PreparedStatement ps=null;

Class.*forName*("com.mysql.jdbc.Driver");

Connection con = DriverManager.*getConnection*(url, u, p);

ps = con.prepareStatement("insert into customer values(?,?,?)");

ps.setInt(1, 143);

ps.setString(2, "sri");

ps.setString(3, "mum");

**int** res = ps.executeUpdate();

System.***out***.println("Result : " + res);

ps =con.prepareStatement("SELECT \* FROM customer c WHERE c.cid<? ");

ps.setInt(1, 200);

ResultSet rs = ps.executeQuery();

**while** (rs.next()) {

System.***out***.println(rs.getString(2));

}

}

}

Result : 1

Ram

RAKESH

Surya

sri

## 2.7 BLOB (BinaryLarge Obejects)

We use BLOB objects to store binary data like images, videos etc

We have following methods to Save & Retrive BLOB Objects

**To Save**

* **void setBinaryStream(int parameterIndex, InputStream x)**
* **void setBinaryStream(int parameterIndex, InputStream x, int length)**
* **void setBlob(int index, Blob x)**
* **void setBlob(int index,InputStream is)**

**To Retrive**

* **Blob getBlob(int columnIndex)**
* **Blob getBlob(String columnLabel)**
* **InputStream getBinaryStream(int columnIndex)**
* **InputStream getBinaryStream(String column)**

**Steps:**

1. **Read Image/ video data by using InputStream**

FileInputStream fis=**new** FileInputStream("d:\\g.jpg");

1. **Create PreparedStatement Object to write insert image query**

PreparedStatement ps=con.prepareStatement("insert into imgtable values(?,?)");

1. **Set parameter values**

**ps.setInt(1, 101);**

**ps.setBinaryStream(2,fis);**

1. **Execute Query**

**int** i=ps.executeUpdate();

1. To get image from table🡺 execute Select Quey , call on rs object

**FileInputSteam fs= rs.getBinarayStream(“column”);**

1. Choose Location to Store new Image

FileOutputStream fos = **new** FileOutputStream("res/newpict.gif");



Example BlobInsert Operation

**public** **class** BlobInsert {

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

String url = "jdbc:mysql://localhost:3306/mydb";

String u = "root";

String p = "123456";

Class.*forName*("com.mysql.jdbc.Driver");

Connection con = DriverManager.*getConnection*(url, u, p);

PreparedStatement ps = con.prepareStatement("insert into blobtest values(?,?,?)");

File f = **new** File("res/img.gif");

FileInputStream fis = **new** FileInputStream(f);

ps.setInt(1, 100);

ps.setString(2, "johny");

ps.setBinaryStream(3, fis, (**int**) f.length());

ps.executeUpdate();

System.***out***.println("Record is Inserted");

}

}

Example BlobInsert Operation

**public** **class** BlobRetrive {

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

String url = "jdbc:mysql://localhost:3306/mydb";

String u = "root";

String p = "123456";

Class.*forName*("com.mysql.jdbc.Driver");

Connection con = DriverManager.*getConnection*(url, u, p);

Statement st = con.createStatement();

ResultSet rs = st.executeQuery("select \* from blobtest");

**if** (rs.next()) {

InputStream in = rs.getBinaryStream("img");

FileOutputStream fos = **new** FileOutputStream("res/newpict.gif");

**int** bytesRead = 0;

**byte**[] buffer = **new** **byte**[4096];

**while** ((bytesRead = in.read(buffer)) != -1) {

fos.write(buffer, 0, bytesRead);

}

System.***out***.println("photo is stored in newpict.gif");

fos.close();

in.close();

rs.close();

st.close();

con.close();

} // if

}// main

}// class

## 2.8 CLOB

We use CLOB objects to store character data like txt files,word files etc

We have following methods to Save & Retrive CLOB Objects

**To Save**

* **void setCharacterStream(int parameterIndex, InputStream x)**
* **void setCharacterStream(int parameterIndex, InputStream x, int length)**
* **void setClob(int index, Clob x)**
* **void setClob(int index,InputStream is)**

**To Retrive**

* **Blob getClob(int columnIndex)**
* **Blob getClob(String columnLabel)**
* **InputStream getCharacterStream (int columnIndex)**
* **InputStream getCharacterStream (String column)**

**Steps:**

1. **Read File data by using InputStream**

FileInputStream fis=**new** FileInputStream("d:\\g.jpg");

1. **Create PreparedStatement Object to write insert image query**

PreparedStatement ps=con.prepareStatement("insert into filetable values(?,?)");

1. **Set parameter values**

**ps.setInt(1, 101);**

**ps.setCharacterStream (2,fis);**

1. **Execute Query**

**int** i=ps.executeUpdate();

1. To get image from table🡺 execute Select Quey , call on rs object

**FileInputSteam fs= rs. getCharacterStream (“column”);**

## 2.9 CallableStatement

CallableStatement interface is used to call the **stored procedures and functions**.

We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

|  |  |
| --- | --- |
| **Stored Procedure** | **Function** |
| is used to perform business logic. | is used to perform calculation. |
| must not have the return type. | must have the return type. |
| may return 0 or more values. | may return only one values. |
| We can call functions from the procedure. | Procedure cannot be called from function. |
| It supports input and output parameters. | Function supports only input parameter. |
| Exception handling using try/catch block can be used in stored procedures. | Exception handling using try/catch can't be used in user defined functions. |

We use following method on Connection object to get CallableStatement Object

**public** CallableStatement prepareCall**("{ call procedurename(?,?...?)}");**

CallableStatement cs=con.prepareCall**("{call myprocedure(?,?)}");**

Example procedeure

**create or replace function sum (n1 in number,n2 in number)**

**return** number

is

temp number(8);

begin

temp :=n1+n2;

**return** temp;

end;

/

In above example n1, n2 are Input Paramaters & temp is the Output parameter

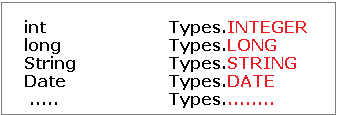
**To set Input Paramaters** we use setXXX(int index, Value) methods

**cs.setInt(1, 10);**

**cs.setInt(2, 20);**

**To set Output Paramaters** we use registerOutParameter(int index, Type.TYPE) method

**cs.registerOutParameter(1, Types.INTEGER);**



**To excute CallableStatement** we use **execute()** method

**cs.execute();**

**To Get results** we use **getXXX(int outputParamIndex)** method

**cs.getInt(1);**

**public** **class** FuncSum {

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

Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

CallableStatement stmt=con.prepareCall("{?= call sum4(?,?)}");

stmt.setInt(2,10);

stmt.setInt(3,43);

stmt.registerOutParameter(1,Types.INTEGER);

stmt.execute();

System.out.println(stmt.getInt(1));

}

}

Output: 53

## 2.10 Metadata

We have 3 types of metadata in jdbc

1. **DataBaseMetaData**
2. **ResultSetMetaData**
3. **ParameterMetaData**

### 1. DataBaseMetaData

We can get database meta like database details, driver name, name of total number of tables, no.of views etc. by using DataBaseMetaData class, we can get by using Connection Object

**DatabaseMetaData getMetaData()**

**DatabaseMetaData dm = con.getMeteData()**

* **String getDriverName()**
* **String getDriverVersion()**
* **String getURL()**
* **String getUserName()**
* **String getDatabaseProductName()**
* **String getDatabaseProductVersion()**
* **int getDatabaseMajorVersion()**
* **int getDatabaseMinorVersion()**

**public** **class** JDBC {

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

String url = "jdbc:mysql://localhost:3306/mydb";

String u = "root";

String p = "123456";

Class.*forName*("com.mysql.jdbc.Driver");

Connection con = DriverManager.*getConnection*(url, u, p);

DatabaseMetaData dm = con.getMetaData();

System.***out***.println("Driver : "+dm.getDriverName());

System.***out***.println("DriverVersion: "+dm.getDriverVersion());

System.***out***.println("URL : "+dm.getURL());

System.***out***.println("UserName : "+dm.getUserName());

System.***out***.println("DatabseName : "+dm.getDatabaseProductName());

System.***out***.println("DVersion : "+dm.getDatabaseProductVersion());

System.***out***.println("Major : "+dm.getDatabaseMinorVersion());

System.***out***.println("Minor : "+dm.getDatabaseMajorVersion());

}

}

Driver : MySQL-AB JDBC Driver

DriverVersion: mysql-connector-java-5.1.18 ( Revision: tonci.grgin@oracle.com-20110930151701-jfj14ddfq48ifkfq )

URL : jdbc:mysql://localhost:3306/mydb

UserName : root@localhost

DatabseName : MySQL

DVersion : 5.6.26

Major : 6

Minor : 5

### **2. ResultSetMetaData**

* The metadata means data about data
* We can get ReselutSet meta information like no.of columns, column data,table name etc
  + - **String getTableName(int column)**
    - **int getColumnCount()**
    - **String getColumnName(int column)**
    - **int getColumnType(int column)**

**public** **class** JDBC {

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

String url = "jdbc:mysql://localhost:3306/mydb";

String u = "root";

String p = "123456";

Class.*forName*("com.mysql.jdbc.Driver");

Connection con = DriverManager.*getConnection*(url, u, p);

Statement st = con.createStatement();

ResultSet rs = st.executeQuery("select \* from customer");

ResultSetMetaData rm = rs.getMetaData();

System.***out***.println(rm.getTableName(1)); // customer

System.***out***.println(rm.getColumnCount());// 3

System.***out***.println(rm.getColumnName(2));// name

System.***out***.println(rm.getColumnType(2));// 12

}

}

### **3. ParameterMetaData**

Used to get information about the types and properties for each parameter marker in a **PreparedStatement** object

* **int getParameterCount()**
* **int getParameterType(int param)**
* **String getParameterTypeName(int param)**

## 2.11 Batch Processing

Instead of executing a single query, we can execute a group of queries.The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing

* **void addBatch(String query)** 🡺 It adds query into batch.
* **int[] executeBatch()** 🡺 It executes the batch of queries.

Statement stmt=con.createStatement();

stmt.addBatch("insert into user420 values(190,'abhi',40000)");

stmt.addBatch("insert into user420 values(191,'umesh',50000)");

stmt.executeBatch();//executing the batch

**public** **class** JDBC {

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

String url = "jdbc:mysql://localhost:3306/mydb";

String u = "root";

String p = "123456";

PreparedStatement ps = **null**;

Class.*forName*("com.mysql.jdbc.Driver");

Connection con = DriverManager.*getConnection*(url, u, p);

Statement st = con.createStatement();

st.addBatch("insert into student values(81, 'Syam', 'mtm')");

st.addBatch("insert into student values(11, 'ram', 'mum')");

st.addBatch("insert into student values(14, 'bam', 'kuk')");

st.addBatch("insert into student values(44, 'pram', 'sec')");

**int** rs[] = st.executeBatch();

**int** sum = 0;

**for** (**int** i = 0; i < rs.length; i++) {

sum = sum + i;

}

System.***out***.println(sum + "Record are UPDATED using BATCH");

}

}

## 2.12 Transcations

In JDBC, **Connection interface** provides methods to manage transaction.

|  |  |
| --- | --- |
| **Method** | **Description** |
| **Void setAutoCommit(boolean status)** | If it is true each transaction is committed bydefault. |
| **void commit()** | Commits the transaction. |
| **void rollback()** | Cancels the transaction. |

**class** FetchRecords{

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

Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","u","p");   con.setAutoCommit(**false**);

   Statement stmt=con.createStatement();

stmt.executeUpdate("insert into user420 values(190,'abhi',40000)");

stmt.executeUpdate("insert into user420 values(191,'umesh',50000)");

con.commit();

con.close();  }

}

## 2.13 RowSet Interfcae

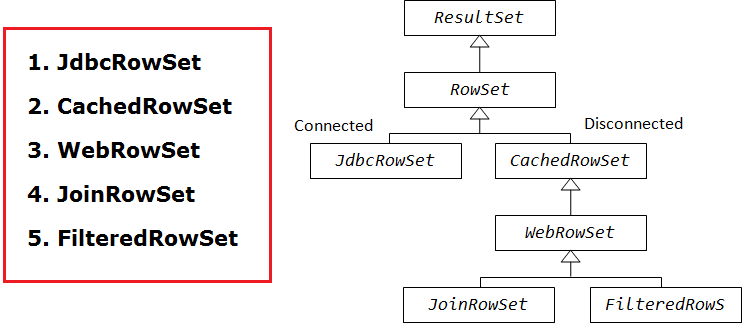
**ResultSet Disdavantages**

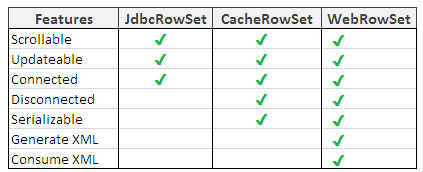
* ResultSet Object is not serilaizable , because it is always maintains a connection with DB
* We can’t pass theResultset object from one class to other class across the network.

**Rowset**

* **RowSet extends the ResultSet interface** so it has all the methods of ResultSet
* RowSet **can be serialized** because it doesn’t have a connection to any database
* RowSet **Object can be sent from one class to another across the network**.

We have following types of RowSets





JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();

rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");

rowSet.setUsername("system");

rowSet.setPassword("oracle");

rowSet.setCommand("select \* from emp400");

rowSet.execute();

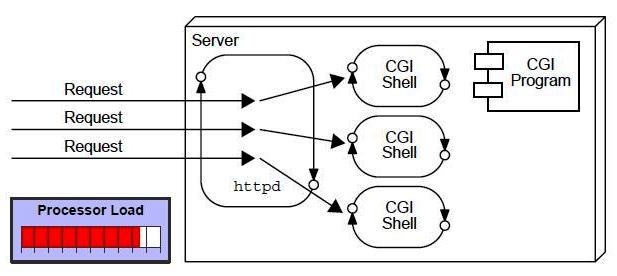
## 2.14 Notes

# Servlets

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter etc. and other components such as HTML. The web components typically execute in Web Server and respond to HTTP request.

**1. CGI**

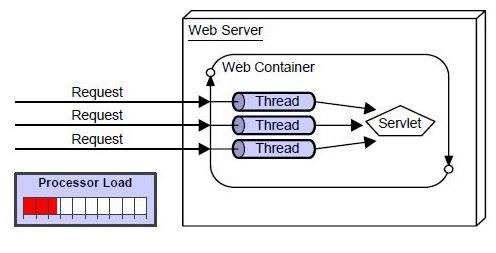
CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



***Disadvantages of CGI***

* If number of client’s increases, it takes more time for sending response.
* For each request, it starts a process and Web server is limited to start processes.
* It uses platform dependent language e.g. C, C++, perl.

**2. Servlet**



The web container creates threads for handling the multiple requests to the servlet.

* **Better performance:** because it creates a thread for each request not process.
* **Portability:** because it uses java language.
* **Robust:** Servlets are managed by JVM so we don't need to worry about memory leak, garbage collection etc.
* **Secure:** because it uses java language..

## 3.1 Basics of Web Technologies

|  |  |
| --- | --- |
| **Static Website** | **Dynamic Website** |
| Prebuilt content is same every time the page is loaded. | Content is generated quickly and changes regularly. |
| It uses the **HTML**code for developing a website. | It uses the server side languages such as **PHP,SERVLET, JSP, and ASP.NET**etc. for developing a website. |
| It sends exactly the same response for every request. | It may generate different HTML for each of the request. |
| The content is only changes when someone publishes and updates the file (sends it to the web server). | The page contains **"server-side"** code it allows the server to generate the unique content when the page is loaded. |

**1. HTTP (Hyper Text Transfer Protocol)**

HTTP is TCP/IP based communication protocol, which is used to deliver the data like image files, query results, HTML files etc on the World Wide Web (WWW) with the default port is TCP 80. It provides the standardized way for computers to communicate with each other.

There are three fundamental features that make the HTTP a simple and powerful protocol used for communication:

* **HTTP is media independent:** It refers to any type of media content can be sent by HTTP as long as both the server and the client can handle the data content.
* **HTTP is connectionless:** It is a connectionless approach in which HTTP client i.e., a browser initiates the HTTP request and after the request is sends the client disconnects from server and waits for the response.
* **HTTP is stateless:** The client and server are aware of each other during a current request only. Afterwards, both of them forget each other. Due to the stateless nature of protocol, neither the client nor the server can retain the information about different request across the web pages.

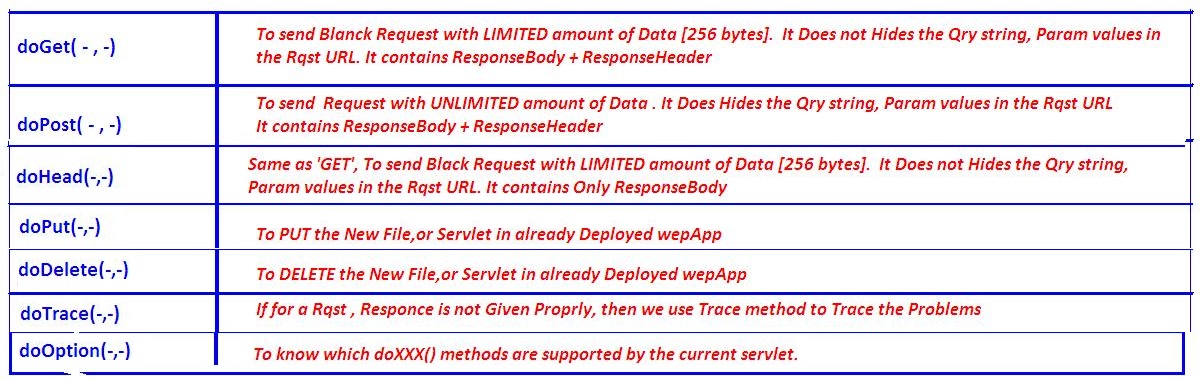
**2. HTTP Requests**

The request sends by the computer to a web server that contains all sorts of potentially interesting information is known as HTTP requests.

It will send following information to Server

* The analysis of source IP address, proxy and port
* The analysis of destination IP address, protocol, port and host
* The Requested URI (Uniform Resource Identifier)
* The Request method and Content
* The User-Agent header
* The Connection control header

We have following HTTP request methods:



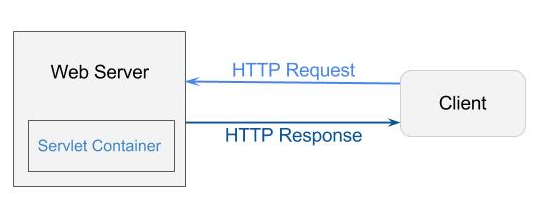
**3. GET vs POST**

|  |  |
| --- | --- |
| **GET** | **POST** |
| 1) In case of Get request, only **limited amount of data**can be sent because data is sent in header. | In case of post request, **large amount of data**can be sent because data is sent in body. |
| 2) Get request is **not secured**because data is exposed in URL bar. | Post request is **secured**because data is not exposed in URL bar. |
| 3) Get request **can be bookmarked.** | Post request **cannot be bookmarked.** |
| 4) Get request is **idempotent.** It means second request will be ignored until response of first request is delivered | Post request is **non-idempotent.** |
| 5) Get request is **more efficient**and used more than Post. | Post request is **less efficient**and used less than get. |

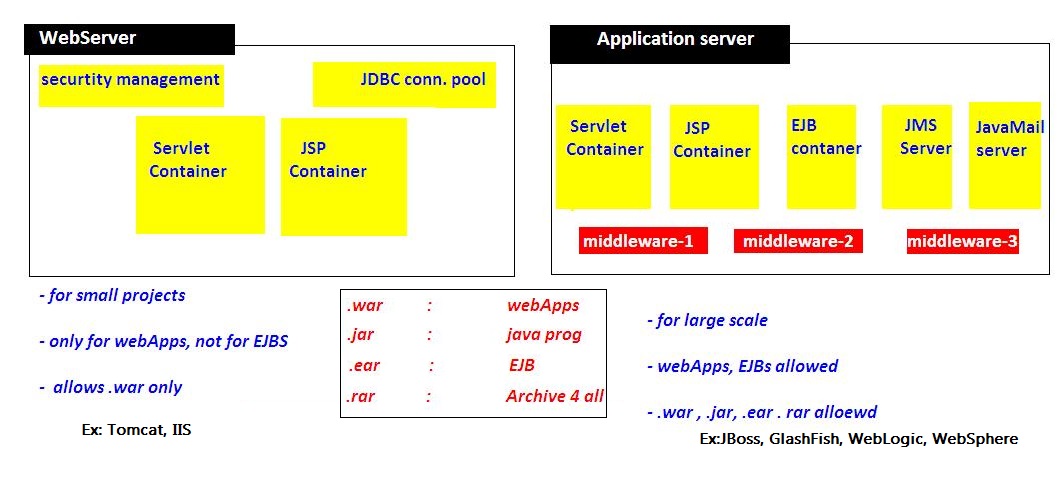
**4. Servlet Container:** **is the place where Servlet programs are executed**

The servlet container is used in java for dynamically generate the web pages on the server side. Therefore the servlet container is the part of a web server that interacts with the servlet for handling the dynamic web pages from the client.

The Servlet Container performs many operations that are given below:

1. **Life Cycle Management**
2. **Multithreaded support**
3. **Object Pooling**
4. **Security etc.**

**5. Web Server VS Application Server**

****

**6. Content Type**

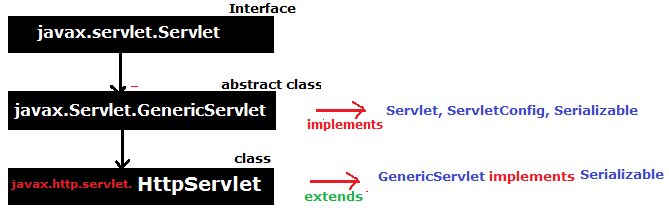
Content Type is also known as MIME (Multipurpose internet Mail Extension) Type. It is a HTTP header that provides the description about what are you sending to the browser.

* **It supports the non-ASCII characters**
* **It supports the multiple attachments in a single message**
* **It supports the attachment contains audio, images and video files etc.**
* **It supports the unlimited message length.**

Commonly used content types are given below:

* **text/html**
* **text/plain**
* **application/msword**
* **application/vnd.ms-excel**
* **application/jar**
* **application/pdf**
* **application/octet-stream**
* **application/x-zip**
* **images/jpeg**
* **images/png**
* **images/gif**
* **audio/mp3**
* **video/mp4**
* **Video/quicktime etc.**

## 3.2 Servlet API

we can create any servlet program by using below 3 ways

### **1. javax.servlet.Servlet (Interface)**

Servlet interface is the ROOT interface of Servlet API. It provides common behaviour to all the servlets.

|  |  |
| --- | --- |
| **Method** | **Description** |
| **public void init(ServletConfig config)** | Initializes the servlet. It is the life cycle method of servlet and invoked by the web container **only once.** |
| **public void service(ServletRequest req,ServletResponse response)** | Provides response for the incoming request. **It is invoked at each request by the web container.** |
| **public void destroy()** | Is invoked only once and indicates that servlet is being destroyed. |
| **public ServletConfig getServletConfig()** | Returns the object of ServletConfig. |
| **public String getServletInfo()** | returns information about servlet such as writer, copyright, version etc. |

Steps to implement Servlet program using Servlet Interface

1. Create a Class which **implements Servlet Interface**
2. **Implement** all **5** abstract **methods**
3. Write Request Processing logic in **service(req,res) method**

### **javax.servlet.GenericServlet (absraect class)**

* GenericServlet class implements Servlet, ServletConfig and Serializable interfaces.
* It provides implementation for all methods of Servlet interface **except the service().**
* **it is protocol-independent**, so it can handle any request of any protocal
* Create servlet by providing the implementation of the service() method.

**Init(),destroy(),getServletConfig(),getservletInfo() are inherted and implemented**

**1. public abstract void service(ServletRequest req, ServletResponse res)**

**2. public void init()**

it is a convenient method for the servlet programmers, now there is no need to call super.init(config)

**3. public ServletContext getServletContext()**

**4. public String getInitParameter(String name**)

**5. public Enumeration getInitParameterNames()**

**6. public String getServletName()**

Steps to write Servlet Program using GenericServlet

1. Create a Class which **extends GenericServlet Interface**
2. Implement & Write Request Processing logic in **service(req,res) method**

### **3.javax.servlet.http.HttpServlet**

HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

**We have 2 service methods**

1. **Public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req,HttpServletResponse res)**

Receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.

**7 proreddoXXX (HttpServletRequest, HttpServletResponce) service methods**

1. **protected void doGet(Htt**pServletRequest req, HttpServletResponse res)
2. **protected void doPost(Ht**tpServletRequest req, HttpServletResponse res)
3. **protected void doHead(H**ttpServletRequest req, HttpServletResponse res)
4. **protected void doOptions**(HttpServletRequest req, HttpServletResponse res)
5. **protected void doPut(Htt**pServletRequest req, HttpServletResponse res)
6. **protected void doTrace(H**ttpServletRequest req, HttpServletResponse res)
7. **protected void doDelete(**HttpServletRequest req, HttpServletResponse res)

Steps to write Servlet Program using GenericServlet

1. Create a Class which **extends HttpServlet Interface**
2. Write Request Processing logic in **service(req,res) OR 🡪 Not Recommended**
3. Write Request Processing logic in **doXXX(req,res) 🡪 doGet,doPost Recommended**

## Servlet Lifecycle

First we see the example, then we can understand the LifeCycle. For every Servlet program contains following strcuture

 Here **Servlets** is Application name

Example 1: Using Servlet Interface

**public** **class** HelloServlet **implements** Servlet{

ServletConfig config = **null**;

@Override

**public** **void** init(ServletConfig config) **throws** ServletException {

**this**.config = config;

System.***out***.println("1.Init...");

}

@Override

**public** **void** service(ServletRequest req, ServletResponse res) **throws** ServletException, IOException {

System.***out***.println("2.Service ...");

PrintWriter pw = res.getWriter();

pw.write("<h1>Hello, World!</h1>");

}

@Override

**public** **void** destroy() {

System.***out***.println("3.Destroy ..");

}

@Override

**public** ServletConfig getServletConfig() {

System.***out***.println("4.getServletConfig ..");

**return** config;

}

@Override

**public** String getServletInfo() {

**return** "getServletInfo";

}

}

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app>

<servlet>

<servlet-name>**hello**</servlet-name>

<servlet-class>demo.HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>**hello**</servlet-name>

<url-pattern>**/hello**</url-pattern>

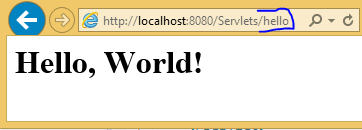
</servlet-mapping>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

</web-app>



Flow of Excecution

1. When ever we deploys the application, container loads the application & creates **ServletContext** Object & waits for the Request
2. if we give **<load-on-startup>**1**</load-on-startup>**  container will creates ServletConfig Object when the time of Deploying application
3. when we give the url : **http://localhost:8080/Servlets/hello ,** request goes to container, and it searches for **/hello** url pattern in web.xml
4. web.xml searches for **/hello** , in <servlet-mapping> and gets **Servelt-name**
5. container loads demo.HelloServlet class and creates creates **ServletConfig** Object and calls inti() method
6. for every request it will calls **service(req,res)** method, for 100 requests it will execute 100 times
7. **destroy()** method will be called before servlet is removed from the container, and finally it will be garbage collected as usual.

In above **<load-on-startup>**1**</load-on-startup>** we may give (1,2..10). based up on priority order it will creates the ServletConfig Object

**<welcome-file-list>**

* If we want to make any page/servlet as Homepage we have to specify in this tag
* If it contains more then 1 file, it will give priority by the Order

Example 2: Using GenericServlet

**public** **class** HelloServlet **extends** GenericServlet {

**public** **void** service(ServletRequest req, ServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

pw.write("Hello, Generic Servlet");

}

}

Example 3: Using HttpServlet

**public** **class** HelloServlet **extends** HttpServlet {

@Override

**public** **void** service(ServletRequest req, ServletResponse res) **throws** ServletException, IOException {

System.***out***.println("Public Service ........");

}

@Override

**protected** **void** service(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

System.***out***.println("Protecd Service ........");

}

@Override

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

System.***out***.println("doGet() ....");

}

@Override

**protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

// **TODO** Auto-generated method stub

System.***out***.println("doPost() ....");

}

}

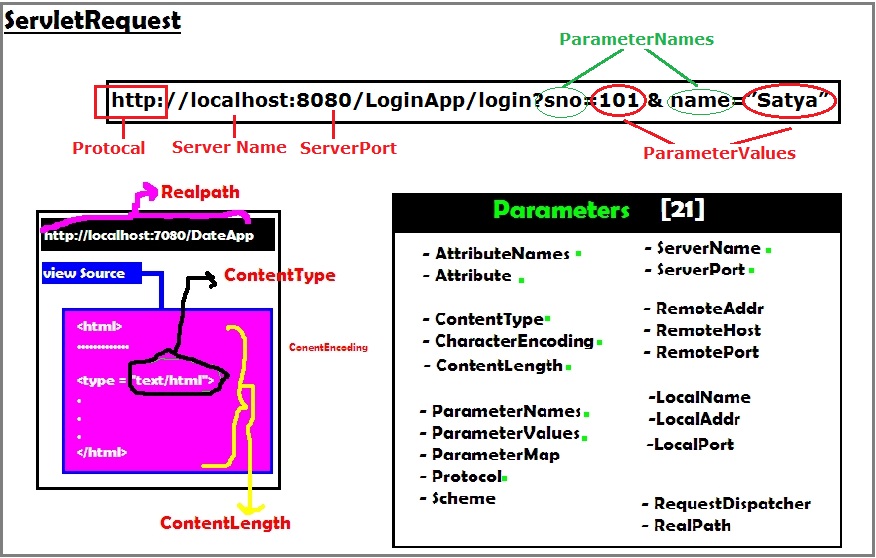
INFO: Reloading Context with name [/Servlets] is completed

Public Service........

* Container first calls **public Service(req,res)** method
* Public Service() method internally calls **protected Service(req,res)** method
* Protected Service() method will internally calling **doGet() or doPost() or doXXX()** depends on the type of http method used by the client
* If the client is **not specifying the type of Http** method then Http protocol by **default consider GET method**,
* so **finally** the client request is processed at **doGet() method**

## 3.4 ServletRequest (interface) 🡪 getParamaters()

ServletRequest is send to Server to process particular request. It can send following details to servlet by submitting FORM or by URL.we can get these details at server side



Example to getRequest details

**public** **class** ServletReq **extends** HttpServlet {

@Override

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

pw.println("<br> getProtocol \t:" + req.getProtocol());

pw.println("<br> getServerName \t:" + req.getServerName());

pw.println("<br> getServerPort \t:" + req.getServerPort());

pw.println("<br> getRemotePort \t:" + req.getRemotePort());

pw.println("<br> getLocalPort \t:" + req.getLocalPort());

pw.println("<br> getContentType \t:" + req.getContentType());

pw.println("<br> getContentLength \t:" + req.getContentLength());

pw.println("<br> CharacterEncoding\t:" + req.getCharacterEncoding());

pw.println("<br> req.getScheme \t:" + req.getScheme());

}

}

getProtocol :HTTP/1.1   
getServerName :localhost   
getServerPort :8080   
getRemotePort :63205   
getLocalPort :8080   
getContentType :null   
getContentLength :-1   
CharacterEncoding :null   
req.getScheme :http

We mainly use ServletRequest **Object to retrieve data from FORM Submission or URL**

We can get the paramaters by using following methods

1. public String **getParameter(“paramname”);**
2. public Enumeration **getParameterNames();**
3. public String[] **getParamterValues(“paramname”);**
4. public Map **getParameterMap();**

Example: getParamater ()

**Index.html**

<form action=*"login"* method=*"get"*>

SNO:<input type=*"text"* name=*"sno"*><br>

NAME:<input type=*"text"* name=*"name"*><br>

<input type=*"submit"* value=*"Submit"*>

</form>

**LoginServlet.java**

**public** **class** LoginServlet **extends** HttpServlet {

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

PrintWriter pw = res.getWriter();

res.setContentType("text/html");

String sno = req.getParameter("sno");

String name = req.getParameter("name");

pw.println("<h1>Sno : " + sno);

pw.println("<br>Name : " + name);

pw.close();

}

}

<web-app

<servlet>

<servlet-name>login</servlet-name>

<servlet-class>demo.LoginServlet</servlet-class>

</servlet>

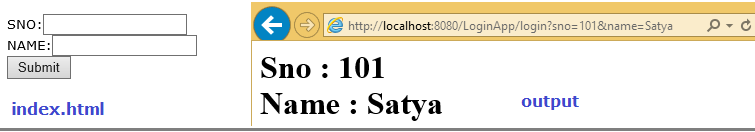
<servlet-mapping>

<servlet-name>login</servlet-name>

<url-pattern>/login</url-pattern>

</servlet-mapping>

</web-app>



Make sure if we use GET method in form we must use doGet (Req, Res) & for POST we have to use doPost(req,res). Otherwise it throws Get/Post not supported error.in this type of case write logic doGet() & call doGet() in doPost()

public Enumeration getParameterNames();

Sometimes we don’t know the request parameter names, in this case we use getParameterNames ();. See the same UI for this only Servlet code is changed

**public** **class** LoginServlet **extends** HttpServlet {

**protected** **void** doPost(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

PrintWriter pw = res.getWriter();

res.setContentType("text/html");

Enumeration e = req.getParameterNames();

**while** (e.hasMoreElements()) {

String s = (String) e.nextElement();

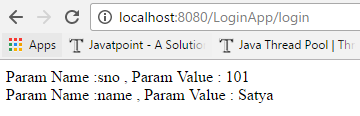
pw.write("Param Name :" +s + ",Param Value : " + req.getParameter(s)+"<br>");

}

pw.close();

}

}



**getParamterValues(“paramname”), getParameterMap();** are used in the case of Single parameter can having multiple values, like checkboxes. See below example

getParamterValues(“paramname”) Example

<form action=*"login"* method=*"post"*>

NAME:<input type=*"text"* name=*"name"*><br>

Skills : <br>

<input type=*"checkbox"* name=*"skill"* value=*"java"*>JAVA<br>

<input type=*"checkbox"* name=*"skill"* value=*"cpp"*>CPP<br>

<input type=*"checkbox"* name=*"skill"* value=*"hadoop"*>HADOOP<br>

<input type=*"checkbox"* name=*"skill"* value=*"devops"*>DevOps<br>

<input type=*"submit"* value=*"Submit"*>

</form>

**public** **class** LoginServlet **extends** HttpServlet {

**protected** **void** doPost(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

PrintWriter pw = res.getWriter();

res.setContentType("text/html");

pw.write("<h1> Name : " + req.getParameter("name"));

pw.write("<br> Skills : <br> ");

String[] skills = req.getParameterValues("skill");

**for** (**int** i = 0; i < skills.length; i++) {

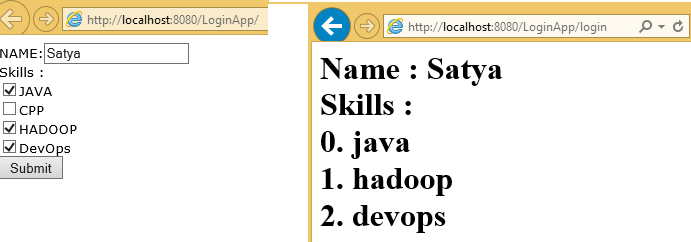
pw.write(i + ". " + skills[i] + "<br>");

}

pw.close();

}

}



getParameterMap(); Example

**public** **class** LoginServlet **extends** HttpServlet {

**protected** **void** doPost(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

PrintWriter pw = res.getWriter();

res.setContentType("text/html");

Map m = req.getParameterMap();

Set s = m.entrySet();

Iterator it = s.iterator();

**while** (it.hasNext()) {

Map.Entry entry = it.next();

String key = entry.getKey();

String[] value = entry.getValue();

pw.println("Key is " + key + "<br>");

**if** (value.length > 1) {

**for** (**int** i = 0; i < value.length; i++) {

pw.println("<li>" + value[i].toString() + "</li><br>");

}

} **else**

pw.println("Value is " + value[0].toString() + "<br>");

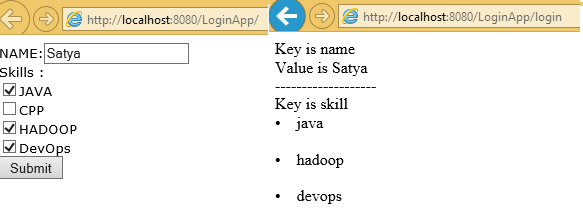
pw.println("-------------------<br>");

}

pw.close();

}

}



## 3.5 ServletConfig (interface) 🡪 getInitParamaters()

* ServletConfig is one of the **pre-defined interface**.
* ServletConfig object exists **one per servlet program**.
* An object of ServletConfig created by the container **during its initialization phase**.
* An object of ServletConfig is available to the servlet during its execution, once the servlet execution is completed, automatically ServletConfig interface object will be removed by the container.
* **It contains <init-param> details at web.xml, of a particular servlet.**
* The moment when we are using an object of ServletConfig, **we need to configure the web.xml by writing <init-param> tag under <servlet> tag of web.xml**.

1. How to get ServletConfig Object

We can ServletConfig object in 2 ways

**1. By calling getServletConfig() on current servlet**

**ServletConfig conf = getServletConfig();**

Above method is available in Servlet interface, inherited in to GenericServlet & HttpServlet

**2. ServletConfig object will be available in init() method of the servlet.**

  public void init(ServletConfig config)  
{  
// …………………  
}

2. How to place <init-param> in web.xml

We have to place **<init-param>** in between **<servlet>** tags

<web-app>

<servlet>

<servlet-name>login</servlet-name>

<servlet-class>demo.LoginServlet</servlet-class>

**<init-param>**

**<param-name>s1</param-name>**

**<param-value> 100 </param-value>**

**</init-param>**

**<init-param>**

**<param-name>s2</param-name>**

**<param-value>200</param-value>**

**</init-param>**

</servlet>

<servlet-mapping>

<servlet-name>login</servlet-name>

<url-pattern>/login</url-pattern>

</servlet-mapping>

</web-app>

3.how to get Initparameters in Servlet Programe

We can retrieve <init-param> values by using following methods

* public String **getInitParameter(“param name”);**
* public Enumeration **getInitParameterNames();**

**public** **class** LoginServlet **extends** HttpServlet {

**protected** **void** doPost(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

PrintWriter pw = res.getWriter();

res.setContentType("text/html");

ServletConfig cfg = getServletConfig();

pw.write("<h3> 1. Using getInitParameter()");

pw.write("<br> s1 : " + cfg.getInitParameter("s1"));

pw.write("<br> s2 :" + cfg.getInitParameter("s2"));

pw.write("<br><br> 2. Using getInitParameterNames()");

Enumeration e = cfg.getInitParameterNames();

**while** (e.hasMoreElements()) {

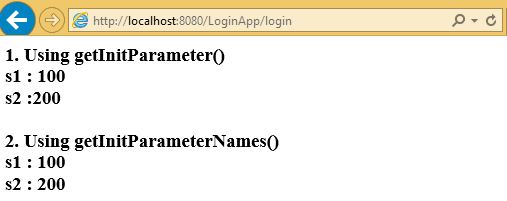
String s = (String) e.nextElement();

pw.write("<br>" +s + "\t : " + cfg.getInitParameter(s));

}

}

}



## 3.6 ServletContext (interface) 🡪 getInitParamaters()

* Object of ServletContext interface is available **one per web application.**
* ServletContext object is automatically created by the container **when the web application is deployed**.
* <context-param> is placed between <web-app> tags. Because the paramaters can be accessed by all the servlets in the Web Application

1. how to get ServletContext Object

We have 3 ways

**1. Using ServletConfig Object**

ServletConfig conf = getServletConfig();  
ServletContext context = conf.getServletContext();

**2. By calling getServletContext() on GenericServlet**

ServletContext ctx = getServletContext();

getServletContext () method is defined in GenericServlet

**3. By calling getServletContext() on HttpServlet**

ServletContext ctx = getServletContext();

getServletContext () method is defined in GenericServlet inherited to HttpServlet

2. How to place <context-param> in web.xml

<context-param> is placed between <web-app> tags. Because the paramaters can be accessed by all the servlets in the Web Application

<web-app>

**<context-param>**

**<param-name>c1 </param-name>**

**<param-value>1000</param-value>**

**</context-param>**

**<context-param>**

**<param-name>c2 </param-name>**

**<param-value>200</param-value>**

**</context-param>**

<servlet>

<servlet-name>login</servlet-name>

<servlet-class>demo.LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>login</servlet-name>

<url-pattern>/login</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

</web-app>

3. how to context-params in Servlet Programe

We have two methods

* **public String getInitParameter(“param name”);**
* **public Enumeration getInitParameterNames();**

**public** **class** LoginServlet **extends** HttpServlet {

**protected** **void** doPost(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

PrintWriter pw = res.getWriter();

res.setContentType("text/html");

ServletConfig cfg = getServletConfig();

ServletContext context = cfg.getServletContext();

pw.write("<h3> 1. Using getInitParameter()");

pw.write("<br> s1 : " + context.getInitParameter("c1"));

pw.write("<br> s2 : " + context.getInitParameter("c2"));

pw.write("<br><br> 2. Using getInitParameterNames()");

Enumeration e = context.getInitParameterNames();

**while** (e.hasMoreElements()) {

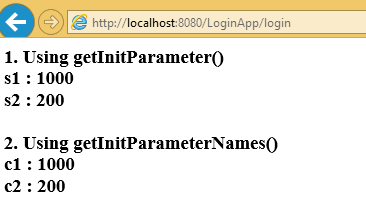
String s = (String) e.nextElement();

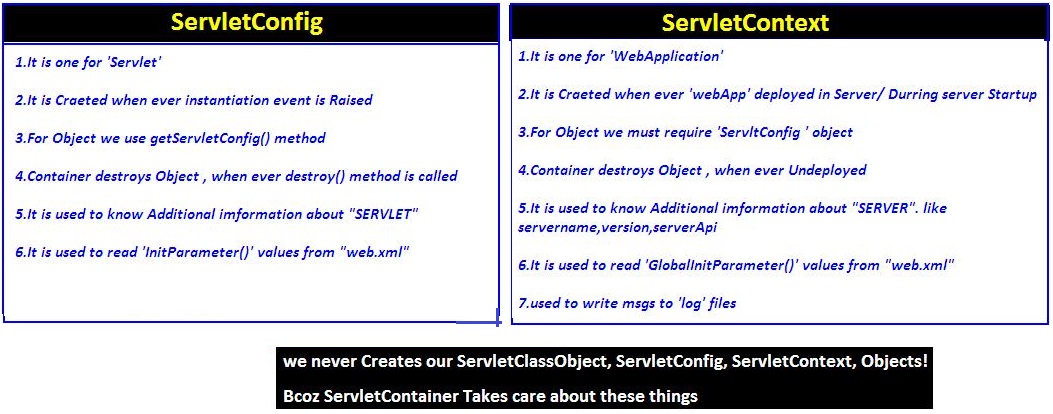
pw.write("<br>" + s + "\t : " + context.getInitParameter(s));

}

}

}





## 3.7 ServletChaining

Servlet chaining is used to achive Communication beween servlets. To perform this we have to use RequestDisptcher inferface .following are the possible ways to achive ServletChaining

1. **rd. forward(req, res)**
2. **rd. include (req, res)**
3. **res.sendRedirect(/url)**

**RequestDisptcher Interface**

The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. We have 2 main methods in this RequestDisptcher

1. **public void forward(ServletRequest req,ServletResponse res)**
2. **public void include (ServletRequest req,ServletResponse res)**

1. How to get RequestDisptcher Object

We have 3 ways to get RequestDisptcher object

1. **using Request object**

RequestDispatcher rd = request.getRequestDispatcher("/url or servletname");

rd.forward(req, res);

rd.include(req, res);

If we use **request** object, the webresource programs **are must be** in **same web application**

1. **using ServletContext object with getRequestDispatcher(“url”) method**

RequestDispatcher rd = context.getRequestDispatcher("/url or servletname");

rd.forward(req, res);

rd.include(req, res);

If we use **Context** object, the webresource programs **are may in** **same/different web applictions**

1. **using ServletContext object with getNamedDispatcher(“servletname”) method**

RequestDispatcher rd = context. getNamedDispatcher("serv1");

rd.forward(req, res);

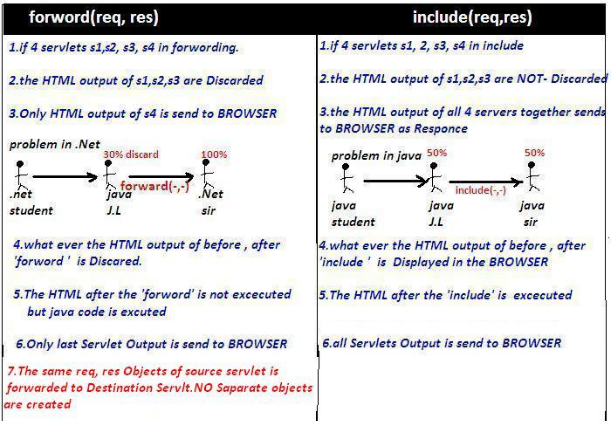
rd.include(req, res);

* **/URL** 🡪 if we are placing .html, .jsp type of files we have to add **‘/’** in path
* **Servltname** 🡪 if we are using logical names of servlet/jsp like serv1, serv2 etc, then **we must not to** add **‘/’** in path

|  |  |
| --- | --- |
| **RequestDispatcher** | **NamedDispatcher** |
| Invokable on both request & context Objects  Exptects servlet url-pattren logical name or filenames of .html, .jsp files as argument  RequestDisptacher Object can points destination servlets,jsps & .html pages | Invokable only on context Object  Exptects only servlet url-pattren logical name as argument  RequestDisptacher Object can points only destination servlets,jsps but not .html pages |

Servlet chaining in Same Server

We can use forward(), include() methods to perform chaining between two servlets which are resides in same web application or different web applications of same server



Forword() example

**Input.html**

<form action=*"s1"* method=*"GET"*>

Number1 : <input type=*"text"* name=*"n1"*><br>

<input type=*"submit"* value=*"SQURE"*>

</form>

**Web.xml**

<web-app>

<servlet>

<servlet-name>s1</servlet-name>

<servlet-class>demo.srv1</servlet-class>

</servlet>

<servlet>

<servlet-name>s2</servlet-name>

<servlet-class>demo.srv2</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>s1</servlet-name>

<url-pattern>/s1</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>s2</servlet-name>

<url-pattern>/s2</url-pattern>

</servlet-mapping>

</web-app>

**public** **class** srv1 **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

ServletConfig cg = getServletConfig();

ServletContext sc = cg.getServletContext();

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

String s1 = req.getParameter("n1");

**int** a = Integer.*parseInt*(s1);

**int** b = a \* a;

pw.println("<h1>Before forword : " + b + "</h1>");

RequestDispatcher rd = sc.getRequestDispatcher("/s2");

**rd.forward(req, res); //🡪 (1)**

pw.println("<h1> After forword</h1>");

}

**public** **void** doPost(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

doGet(req, res);

}

}

**public** **class** srv2 **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

String s1 = req.getParameter("n1");

**int** a = Integer.*parseInt*(s1);

**int** b = a \* a;

pw.println("<h1>Squre from SRV2 : " + b + "</h1>");

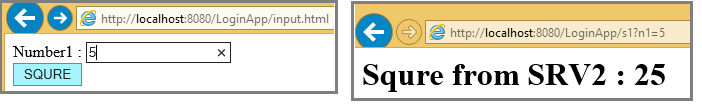
}

**public** **void** doPost(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

doGet(req, res);

}

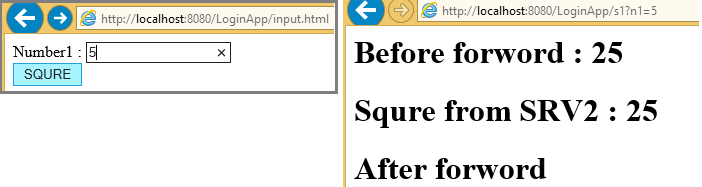
}



In above (1), if we just replace with **include(req, res)** as below it include serv1 result also

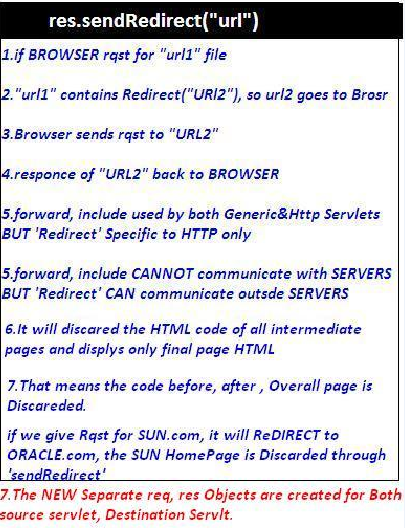
RequestDispatcher rd = sc.getRequestDispatcher("/s2");

rd.include(req, res);



Servlet chaining in Different Server 314 page in natraj

We can use res.sendRedirect(url) method to perform chaining between two servlets which are running on different servers



**public** **class** srv3 **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

pw.println("<h1>Before sendReditrect</h1>");

res.sendRedirect("http://www.google.com");

pw.println("<h1>After sendReditrect</h1>");

}

**public** **void** doPost(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

doGet(req, res);

}

}

|  |  |
| --- | --- |
| **forward() method** | **sendRedirect() method** |
| The forward() method works at server side. | The sendRedirect() method works at client side. |
| It sends the same request and response objects to another servlet. | It always sends a new request. |
| It can work within the server only.  **Example:** request.getRequestDispacher("servlet2").forward(request,response); | It can be used within and outside the server.  **Example:** response.sendRedirect("servlet2"); |

## 3.8 Attribbutes

The servlet programmer can pass informations from one servlet to another using attributes. It is just like passing object from one class to another so that we can reuse the same object again and again.

We have 3 types of scopes for attribues

1. **Request scope**
2. **Session scope**
3. **Application scope (ServletContext Scope)**

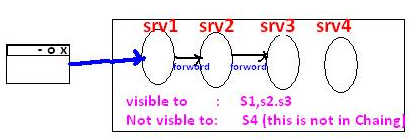
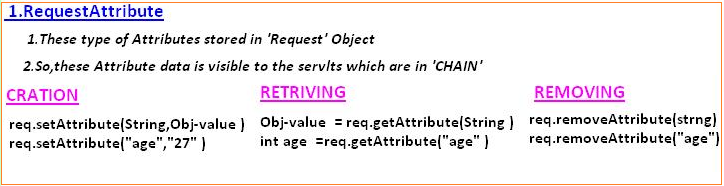
**Attributes can be apply**

* if Both Source& Destination Servlets are in same/difffrent webapplication
* if Both Source& Destination Servlets are in same Server
* It is **not** applicable if Both Source& Destination **Servlets are in different Server**

We have 3 methods to deal with attribues

1. **public void setAttribute(String name,Object object**
2. **public Object getAttribute(String name)**
3. **public void removeAttribute(String name)**

**1. Request Attribute:** It applicable only if both servlets must be in **CHAIN**



**public** **class** srv1 **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

req.setAttribute("uname", "ADMIN");

req.setAttribute("pwd", "123abc$");

RequestDispatcher rd = req.getRequestDispatcher("/s2");

rd.forward(req, res);

}

}

**public** **class** srv2 **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

pw.write("Username : "+req.getAttribute("uname"));

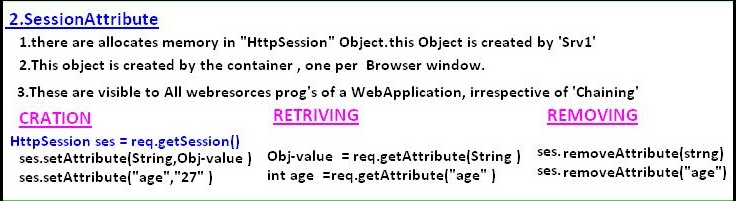
pw.write("Password : "+req.getAttribute("pwd"));

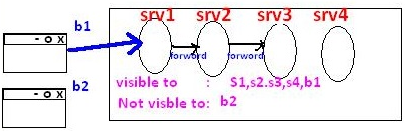
}

}

Output: Username: ADMIN Password : 123abc$

**2. Session Attribute:** It applicable per **one browser window** at a time. I.e Session is maintain in single window





**public** **class** srv1 **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

HttpSession sess = req.getSession();

sess.setAttribute("id", "10001");

sess.setAttribute("name", "Satya");

}

}

**public** **class** srv2 **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

HttpSession sess = req.getSession();

pw.write("ID : "+sess.getAttribute("id"));

pw.write("<br>Name : "+sess.getAttribute("name"));

}

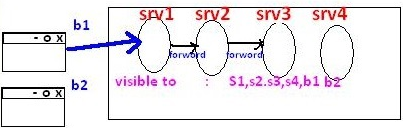
}

Output 🡪 ID : 10001 Name : Satya

**3. Application /Context Attribute**

* It can applicable both servelts **must be in Single Server**
* No need of Servlet Chaining & Session because it is per web application





**public** **class** srv1 **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

ServletContext cxt= req.getServletContext();

cxt.setAttribute("name", "Johnny");

cxt.setAttribute("age", "26");

}

}

**public** **class** srv2 **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

ServletContext cxt= req.getServletContext();

pw.write("Name: "+cxt.getAttribute("name"));

pw.write("Age : "+cxt.getAttribute("age"));

}

}

Output: Name: Johnny Age : 26

**Stateless Behaviour:** is nothing but while processing current request in any web resource program is cannot use previous request data is nothing but Statteless here.

**HTTP is a Stateless protocol that means each request is considered as the new request**

To make our HttpServlet as a Statefull resource program we use Session Tracking

## 3.9 Session Tracking

Session Tracking is a way of remembering clinet data across the multiple requests during a session.

There are 4 techniques used in Session tracking:

1. **Hidden Form Field**
2. **Cookies**
3. **HttpSession**
4. **URL Rewriting**

1. Hidden Form Field

* We store the information in the hidden field and get it from another servlet
* <input type="hidden" name="uname" value="Satya">
* It easy to write

**Disadvantages**

* Used only on TextBoxes
* If we see the view-source of html page, the hidden values can visible
* Not secure

<form action = *"s1"* method = *"get"*>

Name : <input type = *"text"* name = *"name"*><br>

Age : <input type = *"text"* name = *"age"*><br>

Marrige : <input type = *"checkbox"* name = *"mrg"* value = *"yes"*><br>

<input type = *"submit"* name = *"btn"* value = *"NEXT"*> <br>

</form>

**public** **class** srv1 **extends** HttpServlet {

**public** **void** service(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

String name = req.getParameter("name");

String age = req.getParameter("age");

String mrg = req.getParameter("mrg");

**if** (mrg == **null**) {

mrg = "single";

pw.println("<form action = 's2'>");

pw.println("Why do u want to marry :<input type='text' name = 'why'><br>");

pw.println("<input type = 'hidden' name = 'name' value = " + name + ">");

pw.println("<input type = 'hidden' name = 'age' value = " + age + ">");

pw.println("<input type = 'hidden' name = 'mrg' value = " + mrg + ">");

pw.println("<input type = 'submit' name = 'btn' value = 'OK'><br>");

pw.println("</form>");

}

**else** {

mrg = "married";

pw.println("<form action = 's2'>");

pw.println("How Many Childrens:<input type = 'text' name = 'child'><br>");

pw.println("<input type = 'hidden' name = 'name' value = " + name + ">");

pw.println("<input type = 'hidden' name = 'age' value = " + age + ">");

pw.println("<input type = 'hidden' name = 'mrg' value = " + mrg + ">");

pw.println("<input type = 'submit' name = 'btn' value = 'OK'><br>");

pw.println("</form>");

}

}

}

**public** **class** srv2 **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

pw.println("<h3> Status of yours...........</h1>");

pw.println("<h4> Name :" + req.getParameter("name"));

pw.println("<br> Age :" + req.getParameter("age"));

pw.println("<br> Status :" + req.getParameter("mrg"));

**if** (req.getParameter("mrg").equals("single")) {

pw.println("<br> Reason :" + req.getParameter("why"));

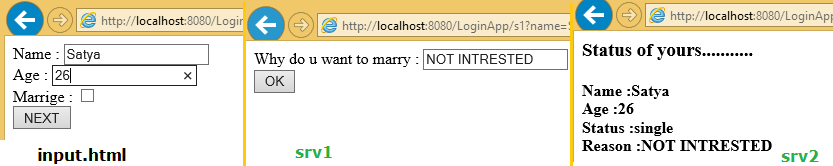
}**else**{

pw.println("<br> No.of Childrents :"+ req.getParameter("child"));

}

}

}



2. Cookies

A cookie is a small piece of information saved in the browser between the multiple client requests.

There are 2 types of cookies in servlets.

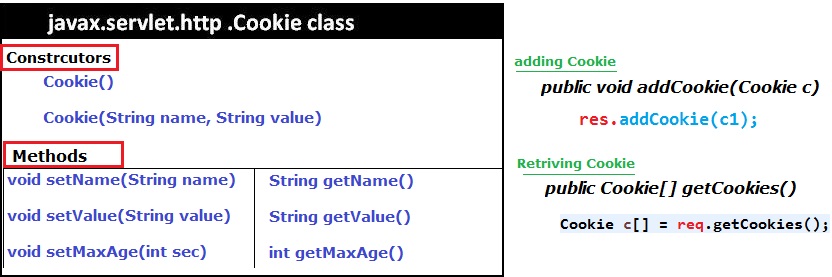
1. **Non-persistent cookie**
2. **Persistent cookie** 

**Advantage of Cookies**

* Simplest technique of maintaining the state.
* Cookies are maintained at client side.

**Disadvantage of Cookies**

* It will not work if cookie is disabled from the browser.
* Only textual information can be set in Cookie object.

****

**public** **class** srv1 **extends** HttpServlet {

**public** **void** service(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

**Cookie c1 = new Cookie("name", "Satya");**

**Cookie c2 = new Cookie("age", "28");**

**c1.setMaxAge(5000); //max 5 sec alive**

**res.addCookie(c1);**

**res.addCookie(c2);**

pw.write("<h3>Cookies Added!");

}

}

**public** **class** srv2 **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

**Cookie c[] = req.getCookies();**

pw.println("<h4> Cookie Name : Cookie Value </h4>");

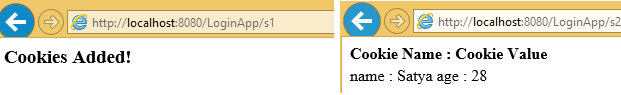
**for** (**int** i = 0; i < c.length; i++) {

pw.println(c[i].getName() + " : " + c[i].getValue());

}

}

}

****

3. Http Session

* HttpSession Object memory **allocates in Server**
* it remembers the client data across the multiple requests in the form of **Session Attribute values**
* Every Session object contains **SessionID, & stored in browser.**
* Session of a browser can be identified by SessionID.

**Constrcurors**

1. **public HttpSession getSession():**

* Returns the current session associated with this request
* If the request does not have a session, creates one.

1. **public HttpSession getSession(boolean create**

* Returns the current session associated with this request
* **True** 🡪 request does not have a session, creates new session.
* **False** 🡪 request does not have a session, it wont create new session

**Methods**

* **public String getId() :**Returns a string containing the unique identifier value.
* **public long getCreationTime() :**Returns the time when this session was created
* public long **getLastAccessedTime():**Returns the last time the client sent a
* **public void invalidate():** Invalidates this session then unbinds any objects bound to it.

**public** **class** srv1 **extends** HttpServlet {

**public** **void** service(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

HttpSession ses = req.getSession();

ses.setAttribute("name", "Ravi");

ses.setAttribute("city", "HYD");

pw.write("<h3>Session Added!");

}

}

**public** **class** srv2 **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

HttpSession sess = req.getSession();

pw.write("Name : "+sess.getAttribute("name"));

pw.write("City : "+sess.getAttribute("city"));

}

}

**Output :** Name : Ravi , City : HYD

4. URL Rewriting

In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource.

We can send parameter name/value pairs using the following format:

**url?name1=value1&name2=value2&??**

**Advantage of URL Rewriting**

* It will always work whether cookie is disabled or not (browser independent).
* Extra form submission is not required on each pages.

**Disadvantage of URL Rewriting**

* It will work only with links.
* It can send only textual information.
* Not Secure, user can read the information what we are sending

<form action="servlet1">

Name:<input type="text" name="userName"/><br/>

<input type="submit" value="go"/>

</form>

public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        String n=request.getParameter("userName");

        out.print("Welcome "+n);

        //appending the username in the query string

        out.print("<a href='servlet2?uname="+n+"'>visit</a>");

        out.close();

    }

}

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        //getting value from the query string

        String n=request.getParameter("uname");

        out.print("Hello "+n);

        out.close();

    }

}

## 3.10 Filters

A filter is invoked at the **preprocessing and postprocessing of a request.**



Filter is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet. So it will be easier to maintain the web application.

**Usage of Filter**

* recording all incoming requests
* logs the IP addresses of the computers from which the requests originate
* conversion
* data compression
* encryption and decryption
* Input validation etc.

**Advantage of Filter**

* Filter is pluggable.
* One filter don't have dependency onto another resource.
* Less Maintenance

Filter API

Like servlet filter have its own API.**The javax.servlet** package contains the **3 interfaces**

1. **Filter**
2. **FilterChain**
3. **FilterConfig**

**1) Filter interface**

For creating any filter, you must implement the Filter interface.Filter interface provides the life cycle methods for a filter.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void init(FilterConfig config) | init() method is invoked only once. It is used to initialize the filter. |
| Public void doFilter(HttpServletRequest req,HttpServletResponse res, FilterChain chain) | doFilter() method is invoked every time when user request to any resource, to which the filter is mapped.It is used to perform filtering tasks. |
| public void destroy() | This is invoked only once when filter is taken out of the service. |

**2) FilterChain interface**

The object of FilterChain is responsible to invoke the next filter or resource in the chain.This object is passed in the doFilter method of Filter interface.The FilterChain interface contains only one method:

**public void doFilter(HttpServletRequest, HttpServletResponse):**

It passes the control to the next filter or resource.

Example

**index.html**

<a href="servlet1">click here</a>

public class MyFilter implements Filter{

public void init(FilterConfig arg0) throws ServletException {}

public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws IOException, ServletException {

    PrintWriter out=resp.getWriter();

    out.print("filter is invoked before");

    chain.doFilter(req, resp);//sends request to next resource

    out.print("filter is invoked after");

    }

    public void destroy() {}

}

public class HelloServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)

            throws ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        out.print("<br>welcome to servlet<br>");

    }

}

<web-app>

<servlet>

<servlet-name>s1</servlet-name>

<servlet-class>HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>s1</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

<filter>

<filter-name>f1</filter-name>

<filter-class>MyFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>f1</filter-name>

<url-pattern>/servlet1</url-pattern>

</filter-mapping>

</web-app>

FilterConfig

An object of FilterConfig is created by the web container. This object can be used to get the configuration information from the web.xml file.

1. **public void init(FilterConfig config):** init() method is invoked only once it is used to initialize the filter.
2. **public String getInitParameter(String parameterName):** Returns the parameter value for the specified parameter name.
3. **public java.util.Enumeration getInitParameterNames():** Returns an enumeration containing all the parameter names.
4. **public ServletContext getServletContext():** Returns the ServletContext object

<filter>

  <filter-name>f1</filter-name>

  <filter-class>MyFilter</filter-class>

   <init-param>

   <param-name>age</param-name>

  <param-value>27</param-value>

   </init-param>

  </filter>

  <filter-mapping>

  <filter-name>f1</filter-name>

  <url-pattern>/servlet1</url-pattern>

  </filter-mapping>

Servlet with Annotation (feature of servlet3):

* Annotation represents the metadata.
* If you use annotation, deployment descriptor (web.xml file) is not required.
* But you should have tomcat7 as it will not run in the previous versions of tomcat.

**@WebServlet(“/url”)** annotation is used to map the servlet with the specified name.

@WebServlet("/Simple")

public class Simple extends HttpServlet {

    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)

                            throws ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter out=response.getWriter();

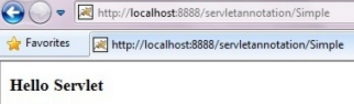
        out.print("<html><body>");

        out.print("<h3>Hello Servlet</h3>");

        out.print("</body></html>");

    }

}



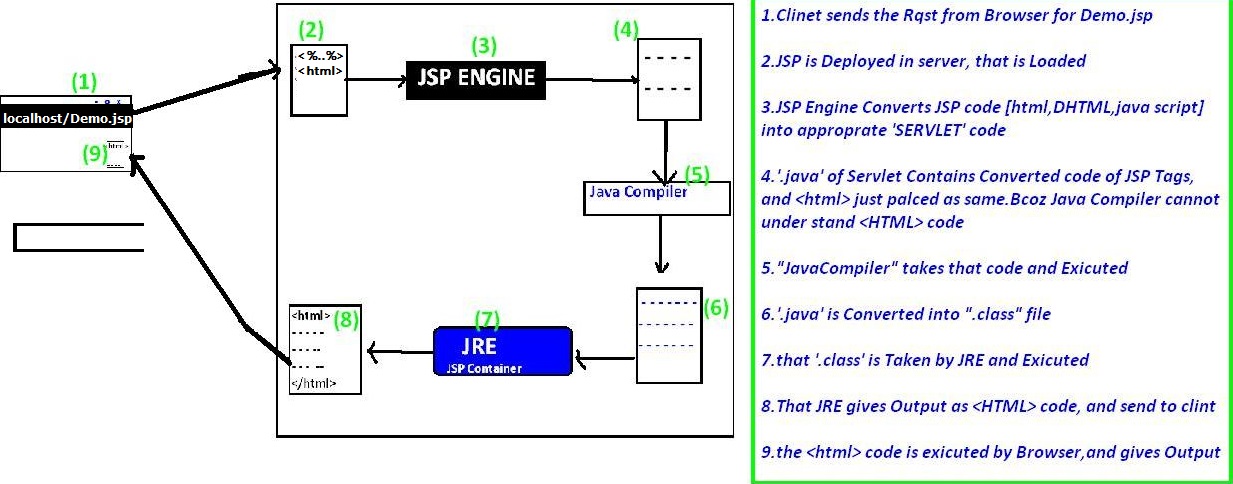
# JSP

## 1. JSP Introduction

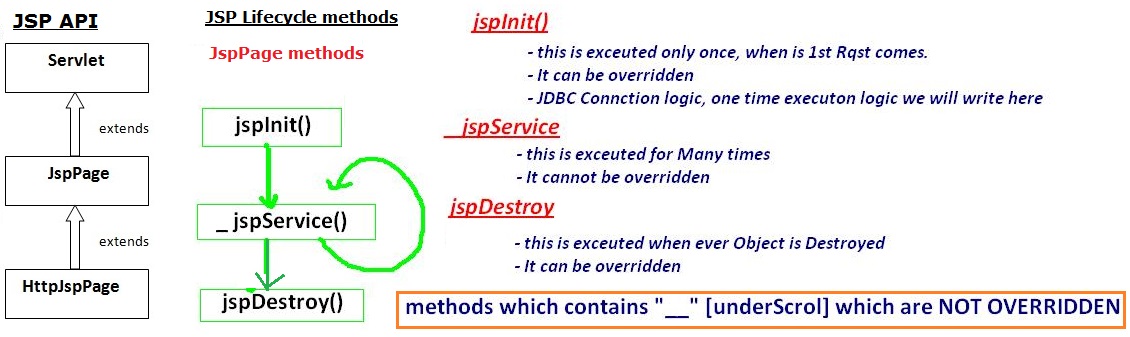
**Features of JSP**

* Extension to Servlet
* Easy to maintain
* Fast Development: No need to recompile and redeploy
* Less code than Servlet

JSP Architecure



JSP Lifecycle



## 2. JSP Scriptlets

In JSP, java code can be written inside the jsp page using the scriptlet tag

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

1. **scriptlet tag**
2. **expression tag**
3. **declaration tag**

**1. Scriptlet tag**

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

<% java source code %>

<html>

<body>

<%

out.print("welcome to jsp");

%>

</body>

</html>

Output: welcome to jsp

It is placed in **\_JspService ()** method. So method declarations not possible

**2. Expression tag**

It is mainly used for **printing calculations, print the values of variable or method**. The code placed within JSP expression tag is written to the output stream of the response. So **you need not write out.print () to write data.** Below is the syntax



<html>

<body>

Date:<%=java.util.Calendar.getInstance().getTime()%>

</body>

</html>

Date:Thu Sep 22 19:10:14 IST 2016

It is placed in **JspInit()** method.

**: Do not end your statement with semicolon in case of expression tag.**

**3. Declaration tag**

* The JSP declaration tag is used to **declare fields and methods.**
* Code written inside the jsp declaration tag is placed **outside the service()** method

Variable Declaration

<html>

<body>

<%!**int** data = 50;%>

<%="Value of the variable is:" + data%>

</body>

</html>

Method Declaration

<html>

<body>

<%!

**int** cube(**int** n){

**return** n\*n\*n;

}

%>

<%="Cube of 3 is:" + cube(3)%>

</body>

</html>

## 3. JSP Implicit objects

**There are 9 jsp implicit objects**. These objects are created by the web container that are available to all the jsp pages.

|  |  |
| --- | --- |
| **Object** | **Type** |
| **out** | JspWriter |
| **request** | HttpServletRequest |
| **response** | HttpServletResponse |
| **config** | ServletConfig |
| **application** | ServletContext |
| **session** | HttpSession |
| **pageContext** | PageContext |
| **page** | Object |
| **exception** | Throwable |

**Index.jsp**

<%

out.print("1.welcome to jsp");

String name=request.getParameter("a");

out.print("<br> 2.Request :"+name);

response.sendRedirect("http://www.google.com");

out.print("3.Responce : ");

String cfg=config.getInitParameter("config");

out.print("<br> 4.Config ="+cfg);

String cxt=application.getInitParameter("context");

out.print("<br> 5.Application ="+cfg);

session.setAttribute("user","Satya");

out.print("<br> 6.Session ="+session.getAttribute("user"));

%>

<web-app>

<servlet>

<servlet-name>jsp</servlet-name>

<jsp-file>/index.jsp</jsp-file>

<init-param>

<param-name>config</param-name>

<param-value>iam Config Value</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>jsp</servlet-name>

<url-pattern>/jsp</url-pattern>

</servlet-mapping>

<context-param>

<param-name>context</param-name>

<param-value>iam Context Value</param-value>

</context-param>

</web-app>

**7) pageContext implicit object**

The pageContext object can be used to **set, get or remove attributes** from one of the following scopes:

* **Page 🡪** PageContext.PAGE\_SCOPE
* **Request 🡪** PageContext.REQUEST\_SCOPE
* **Session 🡪** PageContext.SESSION\_SCOPE
* **Application 🡪** PageContext.APPLICATION\_SCOPE

**pageContext.setAttribute("name",” value”,PageContext.SESSION\_SCOPE);**

first.jsp

<html>

<body>

<%

String name=request.getParameter("uname");

out.print("Welcome "+name);

pageContext.setAttribute("user",name,PageContext.SESSION\_SCOPE);

<a href="second.jsp">second jsp page</a>

%>

</body>

</html>

Second.jsp

<html>

<body>

<%

String name=(String)pageContext.getAttribute("user",PageContext.SESSION\_SCOPE);

out.print("Hello "+name);

%>

</body>

</html>

**8. Page implicit object**

Page is an implicit object of type **Object class**

**9. Exception**

* Exception is an implicit object of type java.lang.Throwable class.
* This object can be used to print the exception.
* it can only be used in error pages

<%@ page isErrorPage="true" %>

<html>

<body>

Sorry following exception occured:<%= exception %>

</body>

</html>

## 4. JSP Directives

The jsp directives are messages that tells the web container how to translate a JSP page into the corresponding servlet.

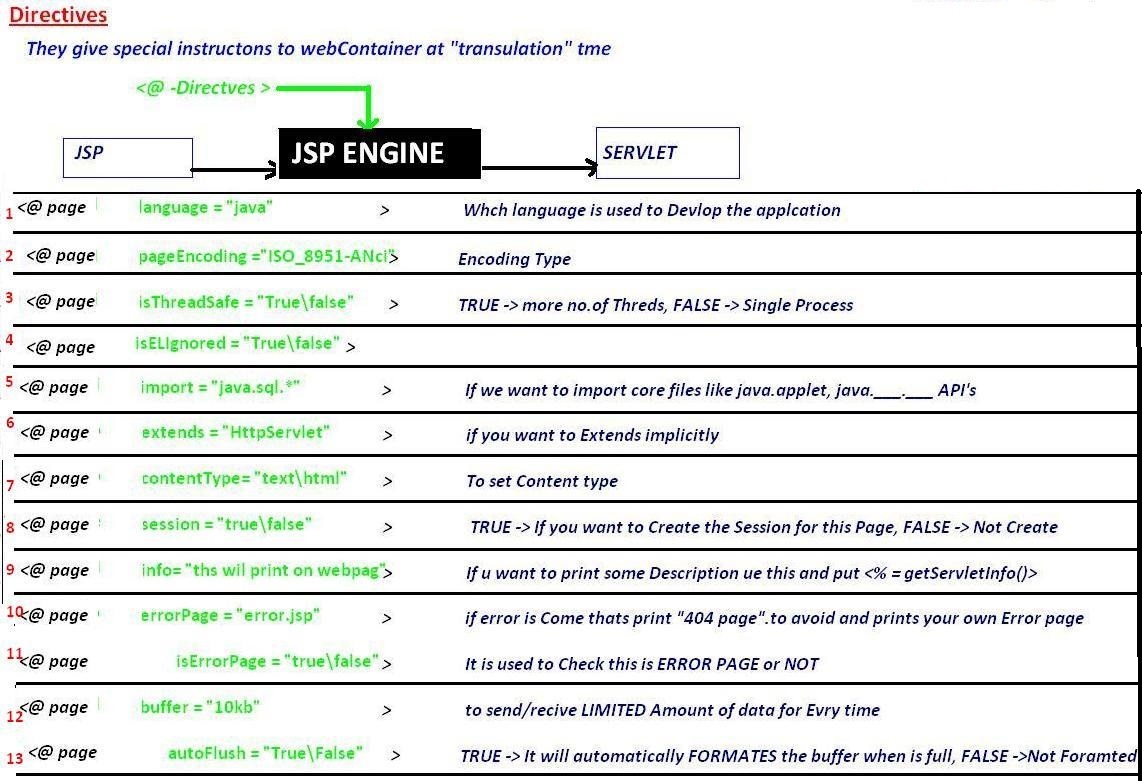
There are three types of directives:

1. **page directive**
2. **include directive**
3. **taglib directive**

****

1. Page Directive





<%@ page language=*"java"* %>

<%@ page pageEncoding=*"ISO-8859-1"*%>

<%@ page isELIgnored=*"false"*%>

<%@ page isThreadSafe=*"true"*%>

<%@ page errorPage=*"err.jsp"* isErrorPage=*"false"*%>

<%@ page import=*"java.lang.\*"*%>

<%@ page extends=*"java.lang.Object"*%>

<%@ page contentType=*"text/html"*%>

<%@ page session=*"true"*%>

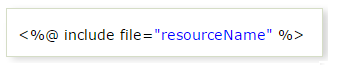
<%@ page info=*"Some Info Print on web page"*%>

<%@ page buffer=*"8kb"*%>

<%@ page autoFlush=*"true"*%>

2. include directive

The include directive is used to include the contents of any resource it may be jsp file, html file or text file. The include directive includes the original content of the included resource at page translation time (The jsp page is translated only once so it will be better to include static resource).



In this example, we are including the content of the header.html file. To run this example you must create a header.html file.

<html>

<body>

<%@ include file="header.html" %>

Today is: <%= java.util.Calendar.getInstance().getTime() %>

</body>

</html>

The include directive includes the original content, so the actual page size grows at runtime.

3. TagLib directive

* The JSP taglib directive is used to define a tag library that defines many tags.
* We use the TLD (Tag Library Descriptor) file to define the tags.
* We can insert custom tags by using this.

<html>

<body>

<%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>

<mytag:currentDate/>

</body>

</html>

## 5. JSP Action Tags

The action tags are used to control the flow between pages and to use Java Bean. The Jsp action tags are given below.

|  |  |
| --- | --- |
| JSP Action Tags | Description |
| **jsp:forward** | Forwards the request and response to another resource. |
| **jsp:include** | Includes another resource. |
| **jsp:param** | Sets the parameter value. It is used in forward and include mostly. |
| **jsp:useBean** | Creates or locates bean object. |
| **jsp:setProperty** | Sets the value of property in bean object. |
| **jsp:getProperty** | Prints the value of property of the bean. |
| **jsp:plugin** | Embeds another components such as applet. |
| **jsp:fallback** | Can be used to print the message if plugin is working. It is used in jsp: plugin. |

Forward, include, param example

Login.jsp

<h2>Login Page</h2>

<jsp:include page=*"Success.jsp"*>

<jsp:param name=*"uname"* value=*"ADMIN"* />

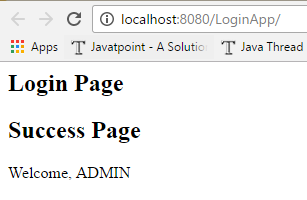
<jsp:param name=*"pwd"* value=*"admin"* />

</jsp:include>

Success.jsp

<h2>Success Page</h2>

Welcome, <%= request.getParameter("uname") %>

Similarly we can use for <jsp:forward> also

UserBean Example

1. First we have choose the Input values for the Login page

<form action=*"set.jsp"* method=*"post"*>

Email <input type=*"text"* name=*"email"*><br>

Pass <input type=*"text"* name=*"pwd"*><br>

<input type=*"submit"* value=*"Login"*><br>

</form>

2. We have to create UserBean class as per Input page parameters (email, pwd)

public class UserBean {

String email;

String pwd;

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPwd() {

return pwd;

}

public void setPwd(String pwd) {

this.pwd = pwd;

}

}

3. UserBean will set the values automatically by comparing property names

<jsp:useBean id=*"user"* class=*"demo.UserBean"*>

<jsp:setProperty name=*"user"* property=*"email"*/>

<jsp:setProperty name=*"user"* property=*"pwd"*/>

</jsp:useBean>

<h3>getProperty Details</h3>

<jsp:getProperty name=*"user"* property=*"email"* /><br>

<jsp:getProperty name=*"user"* property=*"pwd"* /><br>

Here **name** is Object of bean class. & **propery** is the userbean property names

Output

getProperty Details

satyajohnny1@gmail.com  
qw

jsp:plugin, jsp:fallbacks

The **jsp:plugin** action tag is used to embed applet in the jsp file.

**<jsp:plugin**  height="500" width="500"  type="applet"  code="MouseDrag.class"  **/>**

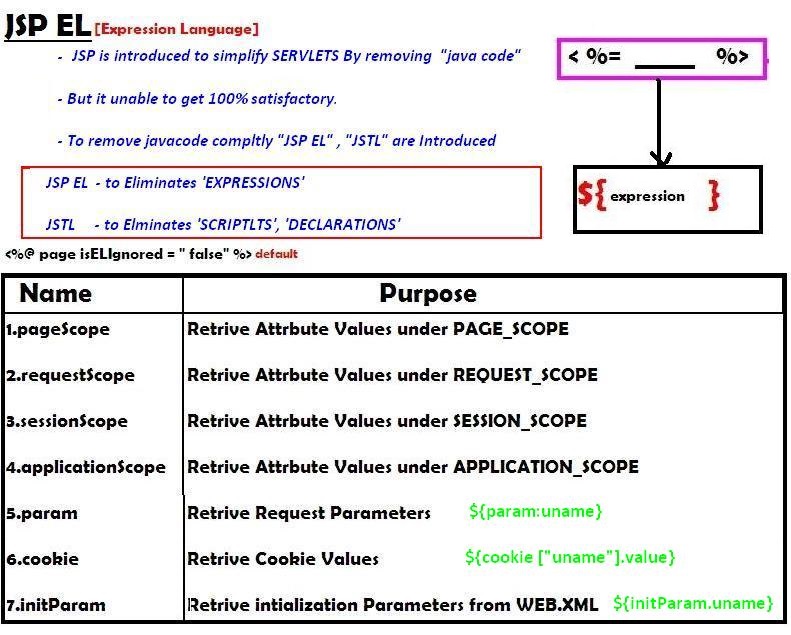
**jsp:fallback** action tag is used to display some message if Applet is not loading

**<jsp:fallback>**

**<p> Unable to start plugin </p>**

**</jsp:fallback>**

## 6. JSP EL (JSP Expression Language)



It simplifies for retrieving following types values mainly

1. **Request Paramters 🡪 req.getParamter(“”)**
2. **Init Patamter values 🡪 getInitParameter(“”)**
3. **Attribute Values 🡪 getAttribute(“”) in 4 scopes**
4. **Cookie values 🡪 getCookie(“”)**

<form action=*"jspel.jsp"* method=*"post"*>

Name <input type=*"text"* name=*"name"*><br>

<input type=*"submit"* value=*"Login"*><br>

<%

session.setAttribute("pwd", "123456");

%>

</form>

Jspel.jsp

Name : ${ param.name }

Session: ${sessionScope.pwd}

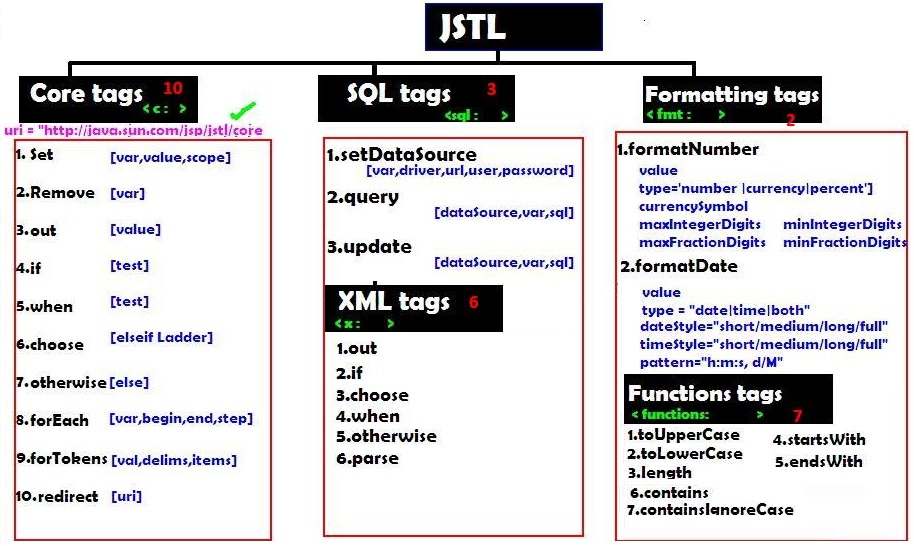


## 7. JSP - Standard Tag Library (JSTL)

* JSP Standard Tag Library (JSTL) is a standard library of readymade tags.
* The JSTL contains several tags already implemented common functionalities.
* JSTL is external tags it is not come by default with JDK.we have to download [**jstl.jar**](https://mvnrepository.com/artifact/javax.servlet/jstl/1.2) seperatly and placed in **lib/** folder

The JSTL tags can be classified, according to their functions, into following JSTL tag library groups that can be used when creating a JSP page:

1. **Core Tags 🡪 used for import,if, foreach loops**
2. **Formatting tags 🡪 used for formatting text, Date,number,URLencoding**
3. **SQL tags 🡪 Used for SQL operations like INSERT,SELECT., etc**
4. **XML tags 🡪** **provides support for XML processing**
5. **JSTL Functions 🡪** **provides support for string manipulation.**

****

1. core tags

we have to attach jstl/core url at the top of the jsp as below

<%@ taglib uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"* %>

<%@ taglib uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"* %>

<c:set var=*"salary"* scope=*"session"* value=*"*${2000\*2}*"*/>

Salary : <c:out value=*"*${salary}*"*/>

<c:if test=*"*${salary > 2000}*"*>

<p>My salary is: <c:out value=*"*${salary}*"*/><p>

</c:if>

<!-- it is like if-else loop -->

<c:choose>

<c:when test=*"*${salary <= 0}*"*>

Salary is very low to survive.

</c:when>

<c:when test=*"*${salary > 1000}*"*>

Salary is very good.

</c:when>

<c:otherwise>

No comment sir...

</c:otherwise>

</c:choose>

<!-- it is mainly used on Number prints[1,2,3,4,5]-->

<c:forEach var=*"i"* begin=*"1"* end=*"5"*>

<c:out value=*"*${i}*"*/><p>

</c:forEach>

<!-- it is mainly used on String-->

<c:forTokens items=*"Zara,nuha,roshy"* delims=*","* var=*"name"*>

<c:out value=*"*${name}*"*/><p>

</c:forTokens>

<c:remove var=*"salary"*/>

<br>Salary : <c:out value=*"*${salary}*"*/>

2. SQL Tags

<sql:setDataSource var="con" driver="com.mysql.jdbc.Driver"

url="jdbc:mysql://localhost/TEST"

user="root" password="pass123"

<sql:setDataSource/>

<sql:update dataSource="${con}" var="count">

INSERT INTO Employees VALUES (104, 2, 'Nuha', 'Ali');

</sql:update>

<sql:query dataSource="${snapshot}" var="result">

SELECT \* from Employees;

</sql:query>

<c:forEach var="row" items="${result.rows}">

<tr>

<td><c:out value="${row.id}"/></td>

<td><c:out value="${row.first}"/></td>

<td><c:out value="${row.last}"/></td>

<td><c:out value="${row.age}"/></td>

</tr>

</c:forEach>

3. Formatting Tags

<h3> Format Number:</h3>

<c:set var="balance" value="120000.2309" />

<p>Cuurency <fmt:formatNumber value="${balance}" type="currency"/></p>

<p>Integr<fmt:formatNumber type="number" maxIntegerDigits="3" value="${balance}" />

<h3> Format Date:</h3>

<c:set var="now" value="<%=new java.util.Date()%>" />

<p>Only Time <fmt:formatDate type="time" value="${now}" /></p>

<p>Only Date <fmt:formatDate type="date" value="${now}" /></p>

<p>Time+Date <fmt:formatDate type="both" value="${now}" /></p>

4. Function Tags (String Operations)

<c:set var="str" value="I am a test String"/>

<c:set var="lowStr" value="${fn:toLowerCase(string1)}" />

<c:set var="uprStr" value="${fn:toUpperCase(string1)}" />

<p>Length: ${fn:length(str)}</p>

## 8. JSP Custom Tags

For creating any custom tag, we need to follow following steps:

1. **Create the Tag handler class** (.java)
2. **Create the Tag Library Descriptor (TLD) file** and define tags**(.tld)**
3. **Create the JSP file that uses the Custom tags (.JSP)**

1.Create the Tag handler class (.java)

* To create the Tag Handler, we are inheriting the **TagSupport class**
* And override **doStartTag().**
* To write data for the jsp, we need to use the **JspWriter** class.its like res.getWriter()
* **PageContext** class provides **getOut()** method that returns **JspWriter** instance
* These classes are not Default with servlet-api.we have to download [**jsp-api.jar**](https://mvnrepository.com/artifact/javax.servlet.jsp/jsp-api/2.2)

**public** **class** MyTag **extends** TagSupport{

**public** **int** doStartTag() **throws** JspException

{

JspWriter out=pageContext.getOut();//returns the instance of JspWriter

out.print(Calendar.*getInstance*().getTime());//printing date using JspWriter

**return** ***SKIP\_BODY***;//will not evaluate the body content of the tag

}

}

2. Create the Tag Library Descriptor (TLD) file and define tags(.tld)

**Tag Library Descriptor** (TLD) file contains information of tag and Tag Hander classes. It must be contained inside the **WEB-INF** directory.

mytag.tld

<?xml version=*"1.0"* encoding=*"ISO-8859-1"* ?>

<!DOCTYPE taglib

PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"

"http://java.sun.com/j2ee/dtd/web-jsptaglibrary\_1\_2.dtd">

<taglib>

<tlib-version>1.0</tlib-version>

<jsp-version>1.2</jsp-version>

<short-name>simple</short-name>

<uri>http://tomcat.apache.org/example-taglib</uri>

<tag>

<name>today</name>

<tag-class>demo.MyTag</tag-class>

</tag>

</taglib>

3. Create the JSP file that uses the Custom tags (.JSP)

DateJsp.jsp

<%@ taglib uri=*"WEB-INF/mytag.tld"* prefix=*"m"* %>

Current Date and Time is: <m:today/>



# ### 5. Webservices

By using webservices we can communicate different applications on different platforms. For example java application in Windows platform can easily communicate with the application developed using .net/php in Linux operation system.

Web Services are mainly of 2 types,

1. **SOAP [Simple Object Access Protocol]**
2. **REST [Representational state transfer]**

**1. SOAP [Simple Object Access Protocol]**

SOAP stands for Simple Object Access Protocol. **SOAP is an XML based** industry standard protocol for designing and developing web services. **Since it’s XML based, its platform and language independent**. So our Service can be based on JAVA and client can be on .NET, PHP etc. and vice versa. **SOAP gives the output only in XML format**

**We have following API’s to implement SOAP Webservices in our java applications**

* **JAX-WS**
* **Apache Axis2**
* **URI - Uniform Resource Identitier**
* **URL - Uniform Resource Link**

**2. REST [Representational state transfer]**

* What ever the data/response we will get from the server is known as **Resource**.
* Each resource can be accessed by its URI’s.
* We can get the resource from RESTful service in different formats like, **HTML, XML, JSON, TEXT, PDF** and in the **Image formats** as well, **but in real time we mainly we will prefer JSON.**
* REST guidelines always talks about stateless communication between client and the Server. Stateless means, every single request from client to server will be considered as a fresh request. Because of this reason REST always prefers to choose HTTP as it a stateless protocol

**We have following API’s to implement RESTful Webservices in our java applications**

* **JAX-RS**

Apache CXF provides implementation for SOAP and RESTful services both.

REST is a style of software architecture. RESTful is typically used to refer to web services implementing such an architecture

|  |  |  |
| --- | --- | --- |
| **No.** | **SOAP** | **REST** |
| 1) | SOAP is a **protocol**. | REST is an **architectural style**. |
| 2) | SOAP stands for **Simple Object Access Protocol**. | REST stands for **REpresentational State Transfer**. |
| 3) | SOAP **can't use REST** because it is a protocol. | REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP. |
| 4) | SOAP **uses services interfaces to expose the business logic**. | REST **uses URI to expose business logic**. |
| 5) | **JAX-WS** is the java API for SOAP web services. | **JAX-RS** is the java API for RESTful web services. |
| 6) | SOAP **defines standards**to be strictly followed. | REST does not define too much standards like SOAP. |
| 7) | SOAP **requires more bandwidth** and resource than REST. | REST **requires less bandwidth** and resource than SOAP. |
| 8) | SOAP **defines its own security**. | RESTful web services **inherits security measures** from the underlying transport. |
| 9) | SOAP **permits XML** data format only. | REST **permits different** data format such as Plain text, HTML, XML, JSON etc. |
| 10) | SOAP is **less preferred** than REST. | REST **more preferred** than SOAP. |

## 1.1 SOAP [Simple Object Access Protocol

**Simple Object Access Protocol (SOAP)** is a standard protocol specification for message exchange based on XML. Communication between the web service and client happens using XML messages.

A simple web service architecture has two components

1. **Client**
2. **Service provider**

**To communicate clinet with service provider clinet must know about following things**

* Location of WebServices Server
* Functions available, signature and return types of function.
* Communication protocol
* Input output formats

**Service provider will create a standard XML file which will have all above information**. So if this XML file is given to client, then client will be able to access web service. **This XML file is called “WSDL”**.

**WSDL (Web Services Description Language):**

WSDL stands for **Web Service Description Language**. It is **an XML file that describes the technical details** of how to implement a web service, more specifically the **URI, port, method names, arguments, and data types**. Since WSDL is XML, it is both human-readable and machine-consumable.

Using this WSDL file we can understand things like,

* Port / Endpoint – URL of the web service
* Input message format
* Output message format
* Security protocol that needs to be followed
* Which protocol the webservice uses?

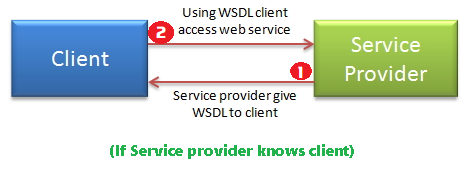
### How to access web service:

There are two ways to access web service

1. **If Service provider knows client**
2. **If Service provider register its WSDL to UDDI and client can access it from UDDI**

1. If Service provider knows client

**If Service provider knows client,** then it will provide its **wsdl** to client and client will be able to access web service.

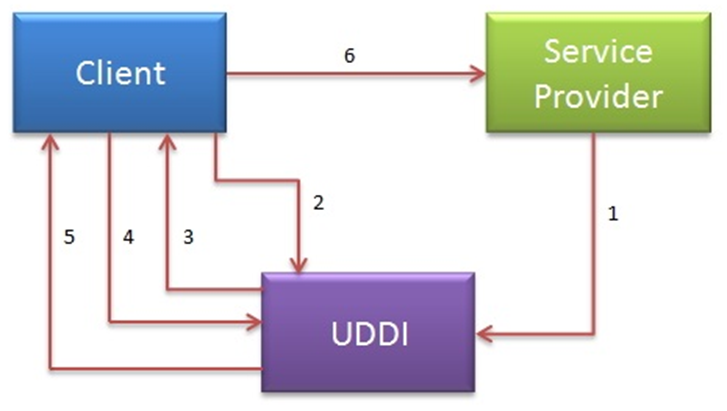


2. If Service provider register its WSDL to UDDI and client can access it from UDDI

**Service provider register its WSDL to UDDI and client can access it from UDDI**:

UDDI stands for **Universal Description, Discovery and Integration**.It is a directory service. Web services can register with a UDDI and make themselves available through it for discovery. So following steps are involved.

1. ***Service provider registers with UDDI.***
2. ***Client searches for service in UDDI.***
3. ***UDDI returns all service providers offering that service.***
4. ***Client chooses service provider***
5. ***UDDI returns WSDL of chosen service provider.***
6. ***Using WSDL of service provider, client accesses web service***

****

**UDDI:**

* UDDI is an XML-based standard for describing, publishing, and finding web services.
* UDDI is a specification for a distributed registry of web services

A business or a company can register three types of information into a UDDI registry. This information is contained in three elements of UDDI.

These three elements are:

1. **White Pages**: Basic information about the company and its business
2. **Yellow Pages**: contain more details about the company
3. **Green Pages**: contains technical information about a web service (url locations etc)

## 1.2 REST [Representation State Transfer]

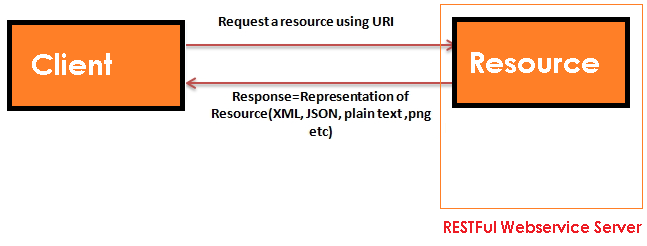
**REpresentational State Transfer (REST)** is a stateless client-server architecture in which the web services are viewed as **resources** and can **be identified by their URIs.**Web service clients that want to use these resources access via globally defined set of remote methods that describe the action to be performed on the resource.

It consists of two components

1. **REST server:** which provides access to the resources
2. **REST client** : which accesses and modify the REST resources.

In the REST architecture style, clients and servers exchange result representations of resources by using a standardized interface and protocol**.REST isn't protocol specific, but when people talk about REST they usually mean REST over HTTP.**

The response from server is considered as the result representation of the resources. This result representation can be generated from one resource or more number of resources. REST allows that resources have different result representations, **e.g.xml, json etc**. The rest client can ask for specific result representation via the HTTP protocol



### HTTP methods:

RESTful web services use HTTP protocol methods for the operations they perform.

Methods are:

* **GET**: It defines a reading access of the resource without side-effects. This operation is idempotent i.e. they can be applied multiple times without changing the result
* **PUT**:  It is generally used for updating resouce. It must also be idempotent.
* **DELETE:** It removes the resources. The operations are idempotent i.e. they can get repeated without leading to different results.
* **POST**: It is used for creating a new resource. It is not idempotent.

### Idempotent

Idempotent means result of multiple successful request will not change state of resource, after initial application

**For example:**

**GET is idempotent.** If Delete() is idempotent method because when you first time use delete, it will delete the resource (initial application) but after that, all other request will have no result(same result) because resource is already deleted.

**Post is not idempotent** method because when you use post to create resource, it will keep creating resource for each new request, so result of multiple successful request will not be same.

Some important features of Restful web services are:

**1.Resource identification through URI**:Resources are identified by their URIs (typically links on internet). So, a client can directly access a RESTful Web Services using the URIs of the resources (same as you put a website address in the browser’s address bar and get some representation as response).

**2.Uniform interface:** Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE.

**3.Client-Server:** A clear separation concerns is the reason behind this constraint. Separating concerns between the Client and Server helps improve portability in the Client and Scalability of the server components.

**4.Stateless:** each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server.

**5.Cache:** to improve network efficiency responses must be capable of being labeled as cacheable or non-cacheable.

**6.Named resources** - the system is comprised of resources which are named using a URL.

**7.Interconnected resource representations** - the representations of the resources are interconnected using URLs, thereby enabling a client to progress from one state to another.

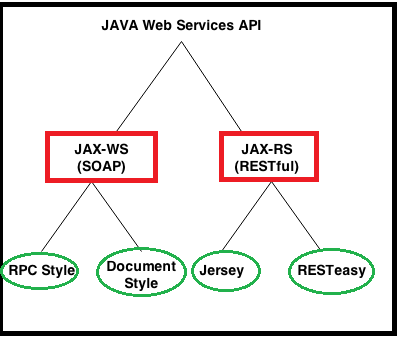
**8.Layered components** - intermediaries, such as proxy servers, cache servers, gateways, etc, can be inserted between clients and resources to support performance, security, etc.

**9.Self-descriptive messages**: Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others.

## 1.3 Java Web Services API

There are two main API's defined by Java for developing web service applications since JavaEE 6.

1. **JAX-WS:** for SOAP web services. The **are 2 ways to write JAX-WS** application code: by
   * 1. ***RPC style***
     2. ***Document style.***
2. **JAX-RS:** for RESTful web services. There are **mainly 2 implementations** currently in use for creating **JAX-RS** application:
   * 1. ***Jersey***
     2. ***RESTeasy***.



We have some other RESTFul webservices providers like

* ***Jersey***
* ***RestEasy***
* ***Restlet***
* ***CFX***
* ***Spring Rest webservices***

# 2. JAX-WS (SOAP web services)

SOAP stands for Simple Object Access Protocol. It is a XML-based protocol for accessing web services.

SOAP is a W3C recommendation for communication between two applications.

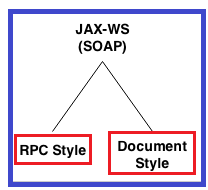
SOAP is XML based protocol. It is platform independent and language independent. By using SOAP, you will be able to interact with other programming language applications.

**Advantages of Soap Web Services**

* **WS Security**: SOAP defines its own security known as WS Security.
* **Language and Platform independent**: SOAP web services can be written in any programming language and executed in any platform

There are two ways to develop JAX-WS example.

1. **RPC style**
2. **Document style**

****

There are two encoding use models that are used to translate a WSDL binding to a SOAP message. They are: **literal, and encoded.**

The combination of the different style and use models give us four different ways to translate a WSDL binding to a SOAP message.

Document/literal

Document/encoded

RPC/literal

RPC/encoded

**When using a** **literal use model**, the body contents should conform to a user-defined **XML-schema (XSD) structure**. The advantage is two-fold. For one, you can validate the message body with the user-defined XML-schema, moreover, you can also transform the message using a transformation language like XSLT.

**With a (SOAP) encoded use model**, the message has to use XSD datatypes, but the structure of the message need not conform to any user-defined XML schema. This makes it difficult to validate the message body or use XSLT based transformations on the message body.

## 2.1 Diffrence between RPC-Style and Document Style

**The way of generating SOAP message formate is main difffrence beteween them.**

**1. RPC Stlye**:

SOAP Body must conform to a structure that indicates the **method name & Parameters name**

<soap:envelope>

<soap:body>

<myMethod>

<x xsi:type="xsd:int">5</x>

<y xsi:type="xsd:float">5.0</y>

</myMethod>

</soap:body>

</soap:envelope>

**2. Document Style**

**SOAP Body can be structurted in any way you like. their is no TYPE attribute here**

<soap:envelope>

<soap:body>

<xElement>5</xElement>

<yElement>5.0</yElement>

</soap:body>

</soap:envelope>

## 2.2 JAX-WS Annotations

We have following important annonotations in order to workwith JAX-WS webservices. They are

1. **@WebService**
2. **@SoapBinding**
3. **@WebMethod**
4. **@WebResult**
5. **@WebServiceClient**
6. **@RequestWrapper**
7. **@ResponseWrapper**
8. **@Oneway**
9. **@HandlerChain**

**1.@WebService**

This annotation can be used in 2 ways

**1. To mark the class as the implementing the Web Service**

Package webservice;

import javax.jws.WebMethod;

import javax.jws.WebService;

import javax.jws.soap.SOAPBinding;

import javax.jws.soap.SOAPBinding.Style;

@WebService

@SOAPBinding(style=Style.RPC)

public interface HelloWorld {

@WebMethod

String getHelloworldMessage(String msg);

}

**2. Defining a Web Service Interface (SEI), in other words Service Endpoint Interface**

import javax.jws.WebService;

@WebService(endpointInterface="webservice.HelloWorld ")

public class HelloWorldImpl implements HelloWorld{

@Override

public String getHelloworldMessage (String name) {

return "Hello World JAX-WS " + name;

}

}

**@Webservice with all attributes as below formate**

@WebService(portName = "SoapPort", serviceName = " HelloWorld ",

targetNamespace = "http://apache.org/hello\_world\_soap\_http",

endpointInterface="webservice.HelloWorld ")

**2.@SoapBinding**

This annotation is used to specify the SOAP messaging style which can either be **RPC** or **DOCUMENT**

/Service Endpoint Interface

@WebService

@SOAPBinding(style = Style.RPC)

//@SOAPBinding(style = Style. DOCUMENT)

public interface HelloWorld{

@WebMethod

String getHelloWorldAsString(String name);

}

**@SoapBinding with all attributes as below formate**

@SOAPBinding(style=SOAPBinding.Style.DOCUMENT,  
 use=SOAPBinding.Use.LITERAL,  
 parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)

**3.@WebMethod**

**@WebMethod** JAX-WS annotation can be applied over a method only. This specified that the method represents a web service operation.it will be used in Interface (***Service Endpoint Interface***) level method only, not in implementation method level.()

/Service Endpoint Interface

@WebService

public interface HelloWorld{

@WebMethod

String getHelloWorldAsString(String name);

}

@**WebMethod** **with all attributes as below formate**

@WebMethod(operationName="echoComplexType", action=” SOAPAction”)

**4.@WebResult**

@WebResult can be used to determine **what the generated WSDL shall look like**

@WebService

public interface HelloWorld{

@WebMethod

@WebResult(partName="Helloworld Method")

String getHelloWorldAsString(String name);

}

//Service Implementation

@WebService(endpointInterface = "com.mkyong.ws.HelloWorld")

public class HelloWorldImpl implements HelloWorld{

@Override

public String getHelloWorldAsString(String name) {

return "Hello World JAX-WS " + name;

}

}

public class WSPublisher {

public static void main(String[] args) {

Endpoint.publish("http://127.0.0.1:9999/ctf", new getHelloWorldAsString ());

}

}

On publishing the generated WSDL (at URL: http://127.0.0.1:9999/ctf?wsdl) would be like:

<definitions

xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"

xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wsp1\_2="http://schemas.xmlsoap.org/ws/2004/09/policy"

xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"

xmlns:tns="http://webresult.jaxWsAnnotations.examples.smlcodes.com/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/"

targetNamespace="http://webresult.jaxWsAnnotations.examples.smlcodes.com/"

name="WSAnnotationsWebResultImplService">

<types />

<message name=" getHelloWorldAsString ">

<part name="arg0" type="xsd: string " />

</message>

<message name=" Helloworld Method ">

<part name=" getHelloWorldAsString " type="xsd:string" />

</message>

</definitions>

[**5.@WebServiceClient**](mailto:5.@WebServiceClient)

@WebServiceClient(

name = "WsAnnotationsWebServiceImplService",

targetNamespace = "http://webservice.smlcodes.com/", wsdlLocation = "file:/Users/satyakaveti/Downloads/ctf.wsdl")

The information specified in this annotation helps in identifying a wsdl: service element inside a WSDL document. This element represents the Web service for which the generated service interface provides a client view.

**6.@RequestWrapper**

**@RequestWrapper** JAX-WS annotation is used to annotate methods in the Service Endpoint Interface with the request wrapper bean to be used at runtime.

It has 4 optional elements; **className** that represents the request wrapper bean name, **localName** that represents element’s local name, **partName** that represent the part name of the wrapper part in the generated WSDL file, and**targetNamespace** that represents the element’s namespace

@WebService

@SOAPBinding(style=Style.RPC)

public interface WSRequestWrapperInterface {

@WebMethod

@RequestWrapper(localName="CTF",

targetNamespace="http://smlcodes.com/tempUtil",

className="com.smlcodes.examples.jaxWsAnnotations.webservice.CTF")

float celsiusToFarhenheit(float celsius);

}

**7.@ResponseWrapper**

@ResponseWrapper JAX-WS annotation is used to annotate methods in the Service Endpoint Interface with the response wrapper bean to be used at runtime. It has 4 optional elements; className that represents the response wrapper bean name, localName that represents element’s local name, partName that represent the part name of the wrapper part in the generated WSDL file, and targetNamespace that represents the element’s namespace.

public interface WSResponseWrapperInterfaceI {

@WebMethod

@ResponseWrapper(localName="CTFResponse",

targetNamespace="http:// smlcodes.com/tempUtil",

className="com. smlcodes.examples.jaxWsAnnotations.webservice.CTFResponse")

float celsiusToFarhenheit(float celsius);

}

**8.@Oneway**

@Oneway JAX-WS annotation is applied to WebMethod which means that method will have only input and no output. When a @Oneway method is called, control is returned to calling method even before the actual operation is performed. It means that nothing will escape method neither response neither exception.

@WebService

@SOAPBinding(style = Style.RPC)

public interface WSAnnotationsOnewayI {

@WebMethod

@Oneway

void sayHello();

}

**9.@HandlerChain**

Web Services and their clients may need to access the SOAP message for additional processing of the message request or response. A SOAP message handler provides a mechanism for intercepting the SOAP message during request and response.

A handler at server side can be a validator. Let’s say we want to validate the temperature before the actual service method is called. To do this our validator class shall implement interface SOAPHandler

package handler;

public class TemperatureValidator implements SOAPHandler {

@Override

public boolean handleMessage(SOAPMessageContext context) {

// TODO Auto-generated method stub

return false;

}

@Override

public boolean handleFault(SOAPMessageContext context) {

// TODO Auto-generated method stub

return false;

}

@Override

public void close(MessageContext context) {

// TODO Auto-generated method stub

}

@Override

public Set getHeaders() {

// TODO Auto-generated method stub

return null;

}

}

// soap-handler.xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<javaee:handler-chains xmlns:javaee="http://java.sun.com/xml/ns/javaee"

xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<javaee:handler-chain>

<javaee:handler>

<javaee:handler-class>com.smlcodes.examples.jaxWsAnnotations.handler.TemperatureValidator

</javaee:handler-class>

</javaee:handler>

</javaee:handler-chain>

</javaee:handler-chains>

package handler;

@WebService

@SOAPBinding(style = Style.RPC)

public interface WSAnnotationsHandlerChainI {

@HandlerChain(file = "soap-handler.xml")

@WebMethod

float celsiusToFarhenheit(float celsius);

}

## 2.3 JAX-WS RPC Style

1. RPC style web services use **method name and parameters to generate XML structure**.
2. The generated **WSDL is difficult to be validated against schema**.
3. In RPC style, **SOAP message is sent as many elements**.
4. RPC **style message is tightly coupled.**
5. In RPC style, **SOAP message keeps the operation name**.
6. In RPC style, **parameters are sent as discrete values**.

Steps to create JAX-WS RPC Style Example

**1. JAX-WS Web Service End Point files**

1. Create a Web Service Endpoint Interface with **@SOAPBinding(style = Style.RPC)**

2. Create a Web Service Endpoint Implementation

3. Create an Endpoint Publisher

4. Test generated WSDL. Ex: **http://localhost:8080/ws/hello?wsdl**

**2. Web Service Client files**

1. Java Web Service Client

* In general words, “web service endpoint” is a service which published outside for user to access;
* where “web service client” is the party who access the published service.

### Example : Hello World using JAX-WS RPC Style



**1. JAX-WS Web Service End Point files**

**1. Create a Web Service Endpoint Interface**

package endpoint;

import javax.jws.WebMethod;

import javax.jws.WebService;

import javax.jws.soap.SOAPBinding;

import javax.jws.soap.SOAPBinding.Style;

//Service Endpoint Interface

@WebService

@SOAPBinding(style = Style.*RPC*)

public interface HelloWorld{

@WebMethod

String getHelloWorldMsg(String msg);

}

**2. Create a Web Service Endpoint Implementation**

package endpoint;

import javax.jws.WebService;

//Service Implementation

@WebService(endpointInterface = "endpoint.HelloWorld")

public class HelloWorldImpl implements HelloWorld{

@Override

public String getHelloWorldMsg(String msg) {

// TODO Auto-generated method stub

return "Your Message from WebService is : "+msg;

}

}

**3. Create an Endpoint Publisher**

package endpoint;

import javax.xml.ws.Endpoint;

//Endpoint publisher

public class HelloWorldPublisher{

public static void main(String[] args) {

Endpoint.*publish*("http://localhost:7777/ws/hello", new HelloWorldImpl());

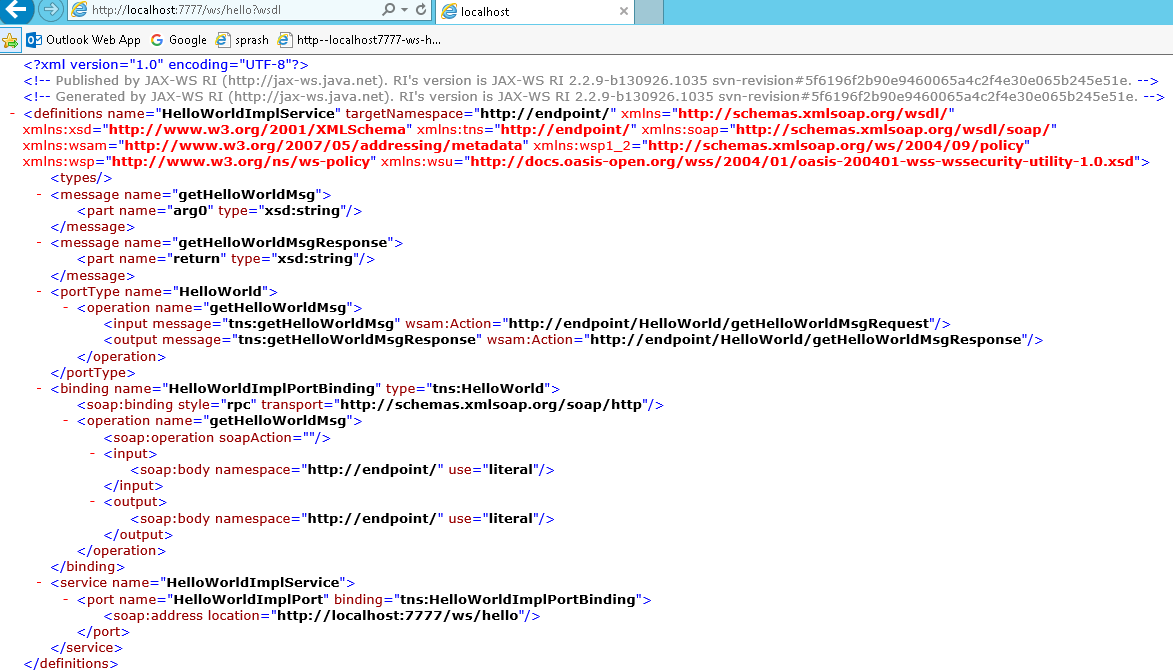
System.*out*.println("WSDL Published !!");

}

}

**4. Test generated WSDL**

Run HelloWorldPublisher as Java Application & access url: [***http://localhost:7777/ws/hello?wsdl***](http://localhost:7777/ws/hello?wsdl)

****

<http://endpoint/>" uses package name of Service endpoint publisher

the main components of WSDL documents are as below. 



WSDL Explanation

1. first Meaage part contains service method name & parameter list

[**<message name="getHelloWorldMsg">**](http://localhost:7777/ws/hello?wsdl)

**<part name="arg0" type="xsd:string"/>**

**</message>**

2. Second Meaage part contains autogenerated Response method & return type

[**<message name="getHelloWorldMsgResponse">**](http://localhost:7777/ws/hello?wsdl)

**<part name="return" type="xsd:string"/>**

**</message>**

3. PortType information is about ServiceEndpoint interface & input,output action urls

[<portType name="**HelloWorld**">](http://localhost:7777/ws/hello?wsdl)

[<operation name="**getHelloWorldMsg**">](http://localhost:7777/ws/hello?wsdl)

<input message="**tns:getHelloWorldMsg**" wsam:Action="**http://endpoint/HelloWorld/getHelloWorldMsgRequest**"/>

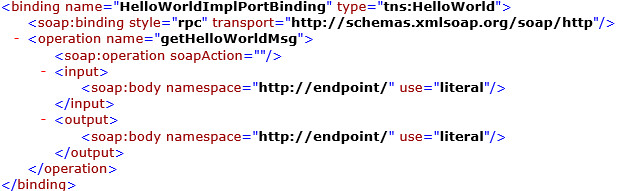
<output message="**tns:getHelloWorldMsgResponse**" wsam:Action="**http://endpoint/HelloWorld/getHelloWorldMsgResponse**"/>

</operation>

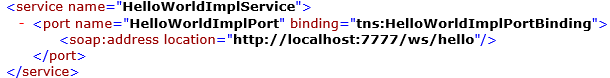
</portType>

Here [http://**endpoint**](http://endpoint) **it will take package name as automatically if we won’t provide anything**

4. Binding will generate automatically by taking RPC Style/ Document Style



5. Service tag contains service details & WSDL document location



**2. Web Service Client file**

Follow below steps to write Webservice client

1. Create **URL** object by passing WSDL document location

URL url = **new** URL("http://localhost:7777/ws/hello?wsdl");

1. Create **QName** by passing service URI, Service name as arguments

QName qname = **new** QName("http://endpoint/", "HelloWorldImplService");

1. Create Service Object by **calling create (-,-)** by passing URL,QName as arguments. Service objects provide the client view of a Web service. ports available on a service can be enumerated using the getPorts method

Service service = Service.*create*(url, qname);

HelloWorld hello = service.getPort(HelloWorld.**class**);

package client;

import java.net.URL;

import javax.xml.namespace.QName;

import javax.xml.ws.Service;

import endpoint.HelloWorld;

public class HelloWorldClient{

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

URL url = new URL("http://localhost:7777/ws/hello?wsdl");

//1st argument service URI, refer to wsdl document above

//2nd argument is service name, refer to wsdl document above

QName qname = new QName("http://endpoint/", "HelloWorldImplService");

Service service = Service.*create*(url, qname);

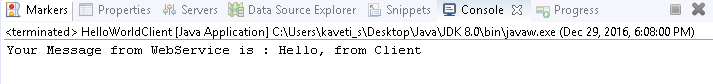
HelloWorld hello = service.getPort(HelloWorld.class);

System.*out*.println(hello.getHelloWorldMsg("Hello, from Client"));

}

}

By running Clinet application we will get output as below



## 2.4 JAX-WS Document Style

1. **SOAP Body can be structurted in any way you like**
2. Document style web services can be **validated against predefined schema**.
3. In document style, **SOAP message is sent as a single document**.
4. Document **style message is loosely coupled**.
5. In Document style, SOAP message loses the operation name.
6. In Document style, parameters are sent in XML format.

In JAX-WS development, convert from “*RPC style*” to “*Document style*” is very easy, just change the @SOAPBinding style option

### Example

**1. JAX-WS Web Service End Point files**

1. Create a Web Service Endpoint Interface with **@SOAPBinding(style = Style.Document)**

package endpoint;

import javax.jws.WebMethod;

import javax.jws.WebService;

import javax.jws.soap.SOAPBinding;

import javax.jws.soap.SOAPBinding.Style;

//Service Endpoint Interface

@WebService

@SOAPBinding(style = Style.*DOCUMENT*)

public interface HelloWorld{

@WebMethod

String getHelloWorldMsg(String msg);

}

2. Create a Web Service Endpoint Implementation

package endpoint;

import javax.jws.WebService;

//Service Implementation

@WebService(endpointInterface = "endpoint.HelloWorld")

public class HelloWorldImpl implements HelloWorld{

@Override

public String getHelloWorldMsg(String msg) {

// TODO Auto-generated method stub

return "Your Message from WebService is : "+msg;

}

}

3. Create an Endpoint Publisher & Run as Java Application

package endpoint;

import javax.xml.ws.Endpoint;

//Endpoint publisher

public class HelloWorldPublisher{

public static void main(String[] args) {

Endpoint.*publish*("http://localhost:7771/ws/hellodoc", new HelloWorldImpl());

System.*out*.println("WSDL Published !!");

}

}

4. Test generated WSDL. Ex: http://localhost:7771/ws/hellodoc**?wsdl**

****



**2. Web Service Client files**

Create Java Web Service Client & Run as Java Application

package client;

import java.net.URL;

import javax.xml.namespace.QName;

import javax.xml.ws.Service;

import endpoint.HelloWorld;

public class HelloWorldClient{

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

URL url = new URL("http://localhost:7771/ws/hellodoc?wsdl");

//1st argument service URI, refer to wsdl document above

//2nd argument is service name, refer to wsdl document above

QName qname = new QName("http://endpoint/", "HelloWorldImplService");

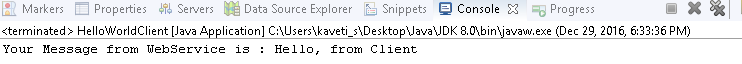
Service service = Service.*create*(url, qname);

HelloWorld hello = service.getPort(HelloWorld.class);

System.*out*.println(hello.getHelloWorldMsg("Hello, from Client"));

}

}



## 2.5 JAX-WS Tools

So far we created WebService applications manually. We have some tools to generate web service classes. let’s start understanding them

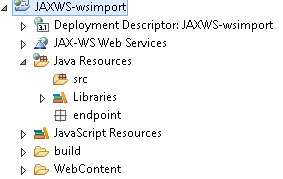
### 2.5.1 wsimport tool(WSDL import 🡪 Generate Java Files)

The **wsimport** tool is used to **parse an** **existing Web Services Description Language (WSDL)** file and generate required files (JAX-WS portable artifacts & JAX-WS Web Service End Point files).

We have to write web service client to access the published web services. This wsimport tool is available in the **$JDK/bin(**C:\Users\kaveti\_s\Desktop\Java\JDK 8.0\bin\wsimport.exe) folder. We no need add these tools to PATH, because they are built in tools

In this example we are using JAXWS-Doc-HelloWorld published WSDL to generate JAX-WS portable artifacts. The WSDL URL is

To generate JAX-WS portable artifacts using wsimport tool follow below steps

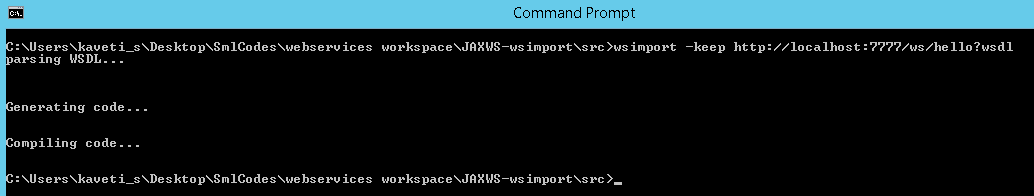
1. Create an Empty Project & create endpoint package for saving generated artifacts****

2. **Open command promt🡪 go to project location run wsimport with wsdl doc location as below**

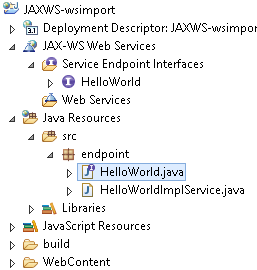
wsimport wsdl-location-path -d -keep

* **wsdl-location-path** : Is the location of wsdl file existence.
* **-d** : specify the directory where all the generated classes should be placed.
* **-keep** : It will keep the java source code of generated classes in the respective directory mentioned.

>wsimport -keep http://localhost:7777/ws/hello?wsdl



By running this it is generated Service Endpoint files as below



// HelloWorld.java

package endpoint;

import javax.jws.WebMethod;

import javax.jws.WebParam;

import javax.jws.WebResult;

import javax.jws.WebService;

import javax.jws.soap.SOAPBinding;

import javax.xml.ws.Action;

/\*\*

\* This class was generated by the JAX-WS RI. JAX-WS RI 2.2.9-b130926.1035 Generated source version: 2.2\*/

@WebService(name = "HelloWorld", targetNamespace = "http://endpoint/")

@SOAPBinding(style = SOAPBinding.Style.*RPC*)

public interface HelloWorld {

/\*\*

\* @param arg0

\* @return returns java.lang.String

\*/

@WebMethod

@WebResult(partName = "return")

@Action(input = "http://endpoint/HelloWorld/getHelloWorldMsgRequest", output = "http://endpoint/HelloWorld/getHelloWorldMsgResponse")

public String getHelloWorldMsg(@WebParam(name = "arg0", partName = "arg0") String arg0);

}

package endpoint;

import java.net.MalformedURLException;

import java.net.URL;

import javax.xml.\*;

@WebServiceClient(name = "HelloWorldImplService", targetNamespace = "http://endpoint/", wsdlLocation = "http://localhost:7777/ws/hello?wsdl")

public class HelloWorldImplService extends Service {

private final static URL *HELLOWORLDIMPLSERVICE\_WSDL\_LOCATION*;

private final static WebServiceException *HELLOWORLDIMPLSERVICE\_EXCEPTION*;

private final static QName *HELLOWORLDIMPLSERVICE\_QNAME* = new QName("http://endpoint/", "HelloWorldImplService");

static {

URL url = null;

WebServiceException e = null;

try {

url = new URL("http://localhost:7777/ws/hello?wsdl");

} catch (MalformedURLException ex) {

e = new WebServiceException(ex);

}

*HELLOWORLDIMPLSERVICE\_WSDL\_LOCATION* = url;

*HELLOWORLDIMPLSERVICE\_EXCEPTION* = e;

}

public HelloWorldImplService() {

super(*\_\_getWsdlLocation*(), *HELLOWORLDIMPLSERVICE\_QNAME*);

}

public HelloWorldImplService(WebServiceFeature... features) {

super(*\_\_getWsdlLocation*(), *HELLOWORLDIMPLSERVICE\_QNAME*, features);

}

public HelloWorldImplService(URL wsdlLocation) {

super(wsdlLocation, *HELLOWORLDIMPLSERVICE\_QNAME*);

}

public HelloWorldImplService(URL wsdlLocation, WebServiceFeature... features) {

super(wsdlLocation, *HELLOWORLDIMPLSERVICE\_QNAME*, features);

}

public HelloWorldImplService(URL wsdlLocation, QName serviceName) {

super(wsdlLocation, serviceName);

}

public HelloWorldImplService(URL wsdlLocation, QName serviceName, WebServiceFeature... features) {

super(wsdlLocation, serviceName, features);

}

/\*\*

\* @return returns HelloWorld

\*/

@WebEndpoint(name = "HelloWorldImplPort")

public HelloWorld getHelloWorldImplPort() {

return super.getPort(new QName("http://endpoint/", "HelloWorldImplPort"), HelloWorld.class);

}

@WebEndpoint(name = "HelloWorldImplPort")

public HelloWorld getHelloWorldImplPort(WebServiceFeature... features) {

return super.getPort(new QName("http://endpoint/", "HelloWorldImplPort"), HelloWorld.class, features);

}

private static URL \_\_getWsdlLocation() {

if (*HELLOWORLDIMPLSERVICE\_EXCEPTION* != null) {

throw *HELLOWORLDIMPLSERVICE\_EXCEPTION*;

}

return *HELLOWORLDIMPLSERVICE\_WSDL\_LOCATION*;

}

}

Now, create a Java web service client which depends on the above generated files

package client;

import endpoint.HelloWorld;

import endpoint.HelloWorldImplService;

public class WSImportClinet {

public static void main(String[] args) {

HelloWorldImplService service = new HelloWorldImplService();

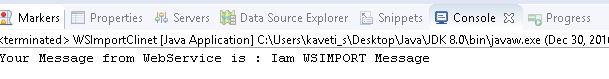
HelloWorld helloWorld = service.getHelloWorldImplPort();

String output =helloWorld.getHelloWorldMsg("Iam WSIMPORT Message");

System.out.println(output); }

}

Run this application we will get following Output



### 2.5.2 wsgen tool(WSDL Generator - Read Java Files🡪 Generate WSDL)

The wsgen tool is used to parse an existing web service implementation class and generates required files (JAX-WS portable artifacts) for web service deployment. This wsgen tool is available in $JDK/bin folder.

2 common use cases for wsgen tool:

1. ***Generates JAX-WS portable artifacts (Java files) for web service deployment.***
2. ***Generates WSDL and xsd files***
3. ***Create web service client for testing***

We need to create web service implementation class, remaing files will be generated by wsgen tool

package endpoint;

import javax.jws.WebMethod;

import javax.jws.WebService;

@WebService

public class RandomNumber {

@WebMethod

public String getRandomNumber() {

return "Random Number Is : " + Math.*random*();

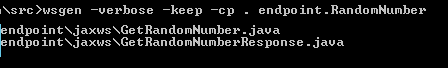
}

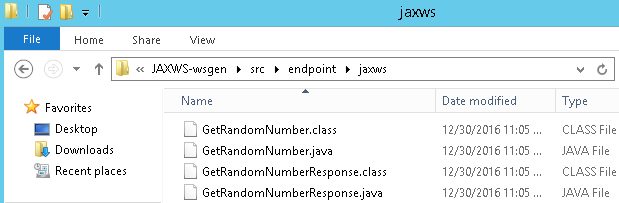
}

1. Generates JAX-WS portable artifacts (Java files) for web service deployment.

To generate all the JAX-WS portable artifacts for above web service implementation class (RandomNumber.java), use following command by going src folder from command prompt

**>wsgen -verbose -keep -cp . endpoint.RandomNumber**



It will generate 2 .java files & 2 .class files

// GetRandomNumber.java

package endpoint.jaxws;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlRootElement;

import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "getRandomNumber", namespace = "http://endpoint/")

@XmlAccessorType(XmlAccessType.*FIELD*)

@XmlType(name = "getRandomNumber", namespace = "http://endpoint/")

public class GetRandomNumber {

}

// GetRandomNumberResponse.java

package endpoint.jaxws;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "getRandomNumberResponse", namespace = "http://endpoint/")

@XmlAccessorType(XmlAccessType.*FIELD*)

@XmlType(name = "getRandomNumberResponse", namespace = "http://endpoint/")

public class GetRandomNumberResponse {

@XmlElement(name = "return", namespace = "")

private String \_return;

/\*\*

\*

\* @return returns String

\*/

public String getReturn() {

return this.\_return;

}

/\*\*

\*

\* @param \_return

\* the value for the \_return property

\*/

public void setReturn(String \_return) {

this.\_return = \_return;

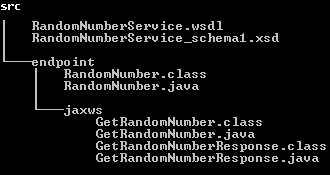
}

}

2. Genarates WSDL and xsd

To generate WSDL and xsd files for above web service implementation class (RandomNumber.java), add an extra **-wsdl** in the wsgen command



In this case it will generate 6 files (**2 java +2 class + 1 WSDL + 1 schema.xsd**). Files under src/ folder are 

RandomNumberService\_schema1.xsd

<?xml version=*"1.0"* encoding=*"UTF-8"* standalone=*"yes"*?>

<xs:schema version=*"1.0"* targetNamespace=*"http://endpoint/"* xmlns:tns=*"http://endpoint/"* xmlns:xs=*"http://www.w3.org/2001/XMLSchema"*>

<xs:element name=*"getRandomNumber"* type=*"tns:getRandomNumber"*/>

<xs:element name=*"getRandomNumberResponse"* type=*"tns:getRandomNumberResponse"*/>

<xs:complexType name=*"getRandomNumber"*>

<xs:sequence/>

</xs:complexType>

<xs:complexType name=*"getRandomNumberResponse"*>

<xs:sequence>

<xs:element name=*"return"* type=*"xs:string"* minOccurs=*"0"*/>

</xs:sequence>

</xs:complexType>

</xs:schema>

RandomNumberService.wsdl

<?xml version=*"1.0"* encoding=*"UTF-8"* standalone=*"yes"*?>

<definitions targetNamespace=*"http://endpoint/"* name=*"RandomNumberService"* xmlns=*"http://schemas.xmlsoap.org/wsdl/"* xmlns:wsp=*"http://www.w3.org/ns/ws-policy"* xmlns:wsp1\_2=*"http://schemas.xmlsoap.org/ws/2004/09/policy"* xmlns:tns=*"http://endpoint/"* xmlns:xsd=*"http://www.w3.org/2001/XMLSchema"* xmlns:soap=*"http://schemas.xmlsoap.org/wsdl/soap/"* xmlns:wsam=*"http://www.w3.org/2007/05/addressing/metadata"*>

<types>

<xsd:schema>

<xsd:import namespace=*"http://endpoint/"* schemaLocation=*"RandomNumberService\_schema1.xsd"*/>

</xsd:schema>

</types>

<message name=*"getRandomNumber"*>

<part name=*"parameters"* element=*"tns:getRandomNumber"*/>

</message>

<message name=*"getRandomNumberResponse"*>

<part name=*"parameters"* element=*"tns:getRandomNumberResponse"*/>

</message>

<portType name=*"RandomNumber"*>

<operation name=*"getRandomNumber"*>

<input wsam:Action=*"http://endpoint/RandomNumber/getRandomNumberRequest"* message=*"tns:getRandomNumber"*/>

<output wsam:Action=*"http://endpoint/RandomNumber/getRandomNumberResponse"* message=*"tns:getRandomNumberResponse"*/>

</operation>

</portType>

<binding name=*"RandomNumberPortBinding"* type=*"tns:RandomNumber"*>

<soap:binding transport=*"http://schemas.xmlsoap.org/soap/http"* style=*"document"*/>

<operation name=*"getRandomNumber"*>

<soap:operation soapAction=*""*/>

<input>

<soap:body use=*"literal"*/>

</input>

<output>

<soap:body use=*"literal"*/>

</output>

</operation>

</binding>

<service name=*"RandomNumberService"*>

<port name=*"RandomNumberPort"* binding=*"tns:RandomNumberPortBinding"*>

<soap:address location=*"REPLACE\_WITH\_ACTUAL\_URL"*/>

</port>

</service>

</definitions>

All files are ready.we have to write Publisher class to publish the WSDL document

package endpoint;

import javax.xml.ws.Endpoint;

public class RandomNumberPublisher {

public static void main(String[] args) {

Endpoint.*publish*("http://localhost:8888/ws/wsgen", new RandomNumber());

System.*out*.println("Service is published!");

}

}

Run as Java Application. It will shows output as Service is published!

[**http://localhost:8888/ws/wsgen?wsdl**](http://localhost:8888/ws/wsgen?wsdl) **it is as same as generated WSDL document**

Finally Note these

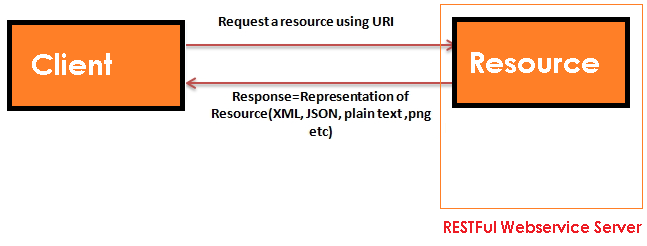
Wsimport 🡪 Uses the WSDL, generates java code for the service/client implementation.

Wsgen 🡪 Uses compiled code, generates WSDL (and artifacts)

# 3. JAX-RS (RESTFul web services)

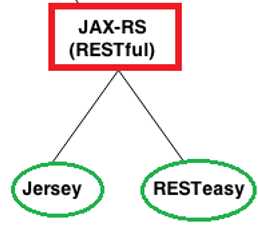
* **JAX-RS** is a specification for RESTful Web Services with Java and it is given by Sun.
* **Since JAX-RS** it is a specification, other frameworks can be written to implement these specifications.
* Few implemetations are **Jersey** from Oracle, **Resteasy** from Jboss, **CXF** from Apache, etc.

We can get the resource from RESTful service in different formats like, **HTML, XML, JSON, TEXT, PDF** and in the Image formats as well, but in real time we mainly we will prefer JSON. REST guidelines always talks about stateless communication between client and the Server. Stateless means, every single request from client to server will be considered as a fresh request. Because of this reason REST always prefers to choose HTTP as it a stateless protocol.



There are two main implementation of JAX-RS API.

1. **Jersey**
2. **RESTeasy**

****

## 3.1 JAX-RS Annotations

We have many annotations. But below are the majorly used annotations in RESTFul webservices

* **@Path(‘Path‘)**
* **@GET**
* **@POST**
* **@PUT**
* **@DELETE**
* **@Produces(MediaType.TEXT\_PLAIN [, more-types])**
* **@Consumes(type[, more-types])**
* **@PathParam()**
* **@QueryParam()**
* **@MatrixParam()**
* **@FormParam()**

[**1.@Path()**](mailto:1.@Path())

* Its a Class & Method level of annotation
* This will check the path next to the base URL

Syntax: http**://localhost:(port)/<YourApplicationName>/<UrlPattern In Web.xml>/<path>**

Here <path> is the part of URI, and this will be identified by @path annotation at class/method level.

**2.@GET**

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP GET request only, if we annotate our method with @GET, the execution flow will enter that following method if we send GET request from the client

**3.@POST**

It’s a method level of annotation, this annotation indicates that the following method should respond to the HTTP POST request only.

**4.@PUT**

It’s a method level of annotation, this annotation indicates that the following method should respond to the HTTP PUT request only.

**5.@DELETE**

It’s a method level of annotation, this annotation indicates that the following method should respond to the HTTP DELETE request only.

**6.@Produces**

It’s a method or field level annotation, this tells which **MIME** type is delivered by the method annotated with **@GET**. Whenever we send a HTTP GET request to our RESTful service, it will invokes particular method and produces the output in different formats. There you can specifies in what are all formats (MIME) your method can produce the output, by using @produces annotation.

**Remember: We will use @Produces annotation for GET requests only.**

**7.@Consumes**

This is a class and method level annotation; this will define which MIME type is consumed by the particular method. It means in which format the **method can accept the input from the client**.

*@PathParam, @QueryParam, @MatrixParam* annotations will come into picture in case if we are passing the input values to the restful service through the URL

**8.@PathParam**

**http://localhost:8001/<Rest Service Name>/rest/customers/100/Satya**

Here the two parameters appear in the end of the above URL [100 & Satya], which are separated by forward slash (/) are called as path parameters

**9.@QueryParam**

**http://localhost:8001/…/rest/customers?custNo=100&custName=Satya**

If the client sends an input in the form of query string in the URL, then those parameters are called as Query Parameters. If you observe the above syntax, client passing **custNo=100&custName=Satya** started after question mark (?) symbol and each parameter is separated by & symbol, those parameters are called as query parameters.

**10.@MatrixParam**

**http://localhost:8001/…/rest/customers;custNo=100;custName=Satya**

Matrix parameters are another way defining the parameters to be added to URL. If you observe the above syntax, client is passing two parameters each are separated by semicolon (;), these parameters are called as matrix parameters. **Remember these parameters may appear any where in the path**.

**11.@** **FormParam**

If we have a HTML form having two input fields and submit button. Lets client enter those details and submit to the RESTful web service. Then the rest service will extract those details by using this **@FormParam** annotation.

## 3.2 JAX-RS JERSEY

[**Jersey**](http://jersey.java.net/), reference implementation to develope RESTful web service based on the [JAX-RS (JSR 311)](http://jsr311.java.net/nonav/releases/1.1/index.html) specification.

If we want to implement Webservices using Jersey we need to download Jersey jar files from [**Jersey**](http://jersey.java.net/) **website**

**The major change between JERSEY & RESTEASY is just changing the configuration in web.xml**

Download & install Maven, configure maven in Eclipse

Steps to Create Jersey Web Service Application

1. Create Dynamic web project in eclipse, convert that into Maven Project

2. Add Jersey jar files manually / through Maven by writing repo details in pom.xml

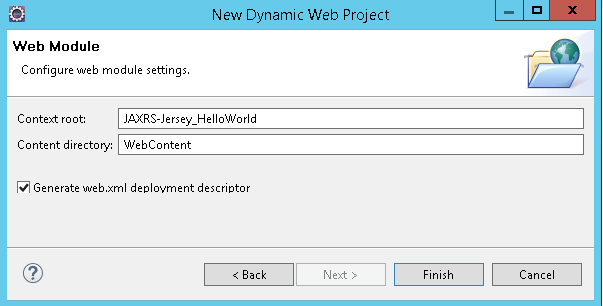
3. Create RESTFul webservice

4. Configure web.xml

5. Test Webservice directly by using URL / writing webservice client

### Example: JAXRS-Jersey-HelloWorld

**1. Create Dynamic web project in eclipse, convert that into Maven Project**

New → Dynamic web project → Provide project details 🡪 finish

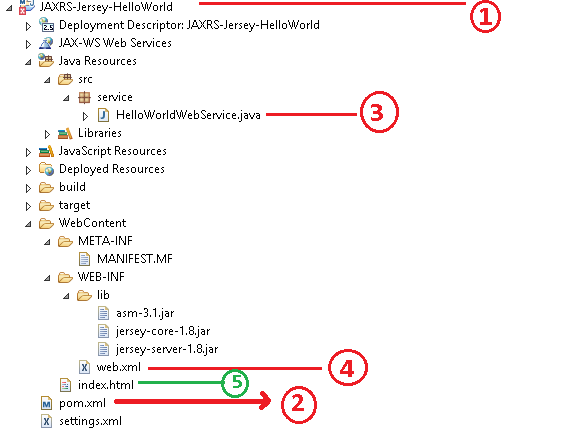
**Right-click on Project →Configure → Convert to Maven Project .** 

Figure Directory Structure after adding all files

**2. Add Jersey jar files manually / through Maven by writing repo details in pom.xml**

Jersey is published in Java.net Maven repository. To develop Jersey REST application, just declares “jersey-server” in Maven **pom.xml**.

<project …>

<repositories>

<repository>

<id>maven2-repository.java.net</id>

<name>Java.net Repository for Maven</name>

<url>http://download.java.net/maven/2/</url>

<layout>default</layout>

</repository>

</repositories>

<dependencies>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-server</artifactId>

<version>1.8</version>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-client</artifactId>

<version>1.19.3</version>

</dependency>

</dependencies>

</project>

**3. Create RESTFul webservice at Server End**

package service;

@Path("/hellojersey")

public class HelloWorldWebService {

// This method is called if HTML and XML is not requested

@GET

@Produces(MediaType.*TEXT\_PLAIN*)

public String sayPlainTextHello() {

return "Hello Jersey Plain";

}

// This method is called if HTML is requested

@GET

@Produces(MediaType.*TEXT\_HTML*)

public String sayHtmlHello() {

return "<h1>" + "Hello Jersey HTML" + "</h1>";

}

}

**4.Configure web.xml**

In web.xml,

* register “com.sun.jersey.spi.container.servlet.ServletContainer“,
* In “com.sun.jersey.config.property.packages“., provide package name, where WebService classes are implemented

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app>

<display-name>Restful Web Application</display-name>

<servlet>

<servlet-name>jersey-serlvet</servlet-name>

<servlet-class>

com.sun.jersey.spi.container.servlet.ServletContainer

</servlet-class>

<init-param>

<param-name>com.sun.jersey.config.property.packages</param-name>

<param-value>service</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>jersey-serlvet</servlet-name>

<url-pattern>/rest/\*</url-pattern>

</servlet-mapping>

</web-app>

**5. Test Webservice directly by using URL / writing webservice client**

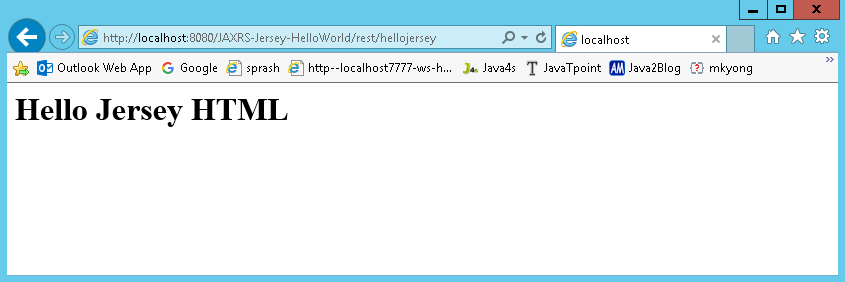
In this example, web request from “**projectURL/rest/hellojersy/**” will match to **“HelloWorldWebService”,** via @Path("/hellojersey") So we are created a test index.html conatining following url for testing purpose

Index.html

<h1>Test JERSEY WEBSERVICE </h1>

<h3><a href=*"rest/hellojersey"*>Default</a></h3>

Direct Testing URL :[**http://localhost:8080/JAXRS-Jersey-HelloWorld/rest/hellojersey**](http://localhost:8080/JAXRS-Jersey-HelloWorld/rest/hellojersey)

****

We can write The HelloworldClientTest.java file is created inside the server application. But you can run client code by other application also by having service interface and jersey jar file.

package client;

import java.net.URI;

import javax.ws.rs.client.Client;

import javax.ws.rs.client.ClientBuilder;

import javax.ws.rs.client.WebTarget;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.UriBuilder;

import org.glassfish.jersey.client.ClientConfig;

public class HelloworldClientTest {

public static void main(String[] args) {

ClientConfig config = new ClientConfig();

Client client = ClientBuilder.newClient(config);

WebTarget target = client.target(*getBaseURI*());

// Now printing the server code of different media type

System.*out*.println(target.path("rest").path("hellojersey").request().accept(MediaType.*TEXT\_PLAIN*).get(String.class));

System.*out*.println(target.path("rest").path("hellojersey").request().accept(MediaType.*TEXT\_HTML*).get(String.class));

}

private static URI getBaseURI() {

// here server is running on 4444 port number and project name is

// restfuljersey

return UriBuilder.*fromUri*("http://localhost:8080/JAXRS-Jersey-HelloWorld/rest/hellojersey").build();

}

}

**If we got 404 error , follow below steps** java.lang.ClassNotFoundException: com.sun.jersey.spi.container.servlet.ServletContainer

Right click on Project 🡪 Properties 🡪 Deployment Assembly : add : Java BuildPath Entities 🡪 Maven Dependencies 🡪 Finish



## 3.3 JAX-RS RESTEasy

[**RESTEasy**](http://www.jboss.org/resteasy)**,** JBoss project, implementation of the [**JAX-RS**](http://jsr311.java.net/) specification. In this article, we show you how to use RESTEasy framework to create a simple REST style web application

Downlaod [**RESTEasy**](http://www.jboss.org/resteasy) jars from here or add RESTEasy dependencies in POM.xml

Steps to Create RESTEasy Web Service Application

1. Create Dynamic web project in eclipse, convert that into Maven Project

2. Add RESTEasy jar files manually / through Maven by writing repo details in pom.xml

3. Create RESTFul webservice using RESTEasy

4. Configure web.xml, Register RESTEasy dependency class

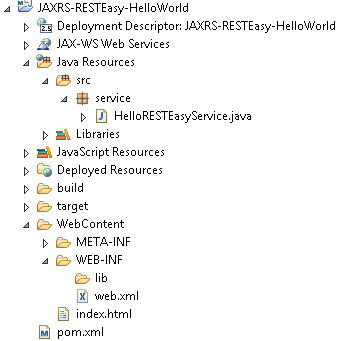
5. Test Webservice directly by using URL / writing webservice client

### Example : JAXRS- RESTEasy –HelloWorld

**1.Create Dynamic web project in eclipse, convert that into Maven Project**

**Create Dynamic Web Project : New → Dynamic web project → Provide project details 🡪 finish**

**Convert into Maven Project** : Right-click on Project →Configure → Convert to Maven Project.



**2. Add RESTEasy jar files manually / through Maven by writing repo details in pom.xml**

Declares JBoss public Maven repository and “**resteasy-jaxrs**” in your Maven pom.xml file. That’s all you need to use **RESTEasy**.

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>JAXRS-RESTEasy-HelloWorld</groupId>

<artifactId>JAXRS-RESTEasy-HelloWorld</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<plugin>

<artifactId>maven-war-plugin</artifactId>

<version>3.0.0</version>

<configuration>

<warSourceDirectory>WebContent</warSourceDirectory>

</configuration>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>JBoss repository</id>

<url>https://repository.jboss.org/nexus/content/groups/public-jboss/</url>

</repository>

</repositories>

<dependencies>

<dependency>

<groupId>org.jboss.resteasy</groupId>

<artifactId>resteasy-jaxrs</artifactId>

<version>2.2.1.GA</version>

</dependency>

</dependencies>

</project>

**3.Create RESTFul webservice using RESTEasy**

package service;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

@Path("/helloresteasy")

public class HelloRESTEasyService {

@GET

@Produces(MediaType.*TEXT\_HTML*)

public String sayHtmlHello() {

return "<h1>" + "Hello RESTEasy Service" + "</h1>";

}

}

**4. Configure** web.xml, Register **RESTEasy dependency class**

Now, configure listener and servlet to support RESTEasy. Read this [JBoss documentation](http://docs.jboss.org/resteasy/docs/2.2.1.GA/userguide/html/Installation_Configuration.html) for details

<web-app id=*"WebApp\_ID"* version=*"2.4"*

xmlns=*"http://java.sun.com/xml/ns/j2ee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/j2ee*

*http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"*>

<display-name>Restful Web Application</display-name>

<!-- Auto scan REST service -->

<context-param>

<param-name>resteasy.scan</param-name>

<param-value>true</param-value>

</context-param>

<!-- this need same with resteasy servlet url-pattern -->

<context-param>

<param-name>resteasy.servlet.mapping.prefix</param-name>

<param-value>/rest</param-value>

</context-param>

<listener>

<listener-class>

org.jboss.resteasy.plugins.server.servlet.ResteasyBootstrap

</listener-class>

</listener>

<servlet>

<servlet-name>resteasy-servlet</servlet-name>

<servlet-class>

org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher

</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>resteasy-servlet</servlet-name>

<url-pattern>/rest/\*</url-pattern>

</servlet-mapping>

</web-app>

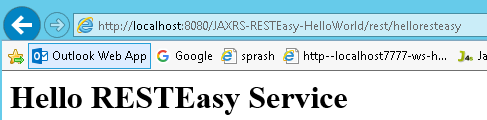
You need to set the “resteasy.servlet.mapping.prefix” if your servlet-mapping for the resteasy servlet has a url-pattern other than “/\*“.

In above example, the resteasy servlet url-pattern is “/rest/\*“, so you have to set the “resteasy.servlet.mapping.prefix” to “/rest” as well, otherwise, you will hit resource not found error message.

Remember to set “resteasy.scan” to true, so that RESTEasy will find and register your REST service automatically.

**5.Test Webservice directly by using URL / writing webservice client**

[**http://localhost:8080/JAXRS-RESTEasy-HelloWorld/rest/helloresteasy**](http://localhost:8080/JAXRS-RESTEasy-HelloWorld/rest/helloresteasy)



## 3.4 JAX-RS Examples

### 1: JAX-RS @Path annotation at Method Level Example

We can use @Path to bind URI pattern to a Java method

1. Create Dynamic web project in eclipse, convert that into Maven Project

**2. Add Jersey jar files manually / through Maven by writing repo details in pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<repositories>

<repository>

<id>maven2-repository.java.net</id>

<name>Java.net Repository for Maven</name>

<url>http://download.java.net/maven/2/</url>

<layout>default</layout>

</repository>

</repositories>

<dependencies>

<!-- https://mvnrepository.com/artifact/com.sun.jersey/jersey-server -->

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-server</artifactId>

<version>1.19.3</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.glassfish.jersey.core/jersey-client -->

<dependency>

<groupId>org.glassfish.jersey.core</groupId>

<artifactId>jersey-client</artifactId>

<version>2.25</version>

</dependency>

<!-- https://mvnrepository.com/artifact/javax.ws.rs/javax.ws.rs-api -->

<dependency>

<groupId>javax.ws.rs</groupId>

<artifactId>javax.ws.rs-api</artifactId>

<version>2.0</version>

</dependency>

</dependencies>

</project>

**3. Create RESTFul webservice**

package service;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.Response;

@Path("/country")

public class PathMethodLevelService {

@GET

@Produces("text/html")

public Response selectCountry() {

String output = " Default Country : <h1>INDIA</h1>";

return Response.*status*(200).entity(output).build();

}

@GET

@Path("/usa")

@Produces("text/html")

public Response selectUSA() {

String output = "Selected Country : <h1>United States of America(USA)</h1>";

return Response.*status*(200).entity(output).build();

}

@GET

@Path("/uk")

@Produces("text/html")

public Response selectUK() {

String output = "Selected Country : <h1>UNITED KINGDOM(UK)</h1>";

return Response.*status*(200).entity(output).build();

}

}

4. Configure web.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/j2ee"* xmlns:web=*"http://xmlns.jcp.org/xml/ns/javaee"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"* id=*"WebApp\_ID"* version=*"2.4"*>

<display-name>JAXRS-PathMethodLevel-Example</display-name>

<servlet>

<servlet-name>jersey-serlvet</servlet-name>

<servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>com.sun.jersey.config.property.packages</param-name>

<param-value>service</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>jersey-serlvet</servlet-name>

<url-pattern>/rest/\*</url-pattern>

</servlet-mapping>

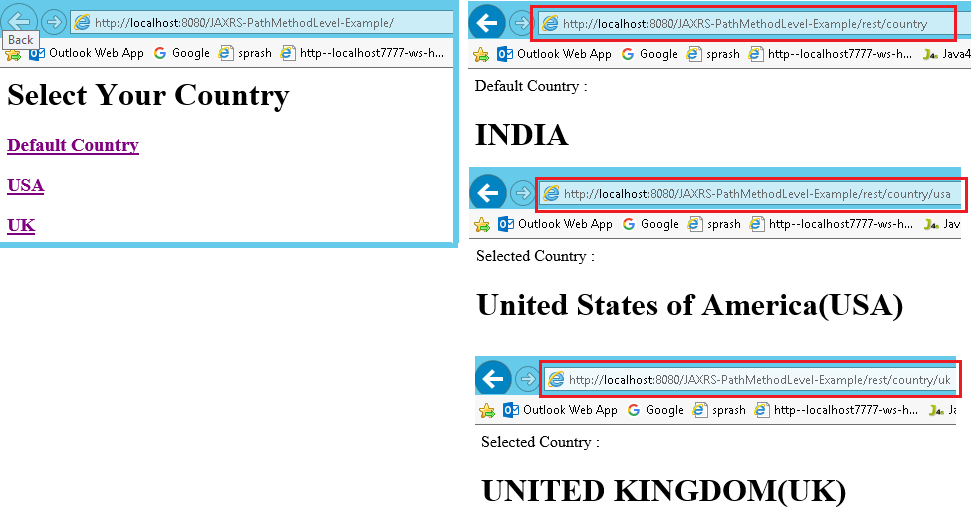
</web-app>

**5. Test Webservice directly by using URL / writing webservice client**

[**http://localhost:8080/JAXRS-PathMethodLevel-Example/**](http://localhost:8080/JAXRS-PathMethodLevel-Example/) **for Default country request**

[**http://localhost:8080/JAXRS-PathMethodLevel-Example/rest/country/usa**](http://localhost:8080/JAXRS-PathMethodLevel-Example/rest/country/usa) **for usa**

[**http://localhost:8080/JAXRS-PathMethodLevel-Example/rest/country/uk**](http://localhost:8080/JAXRS-PathMethodLevel-Example/rest/country/uk) **for UK**



Response Class in JAX-RS

**javax.ws.rs.core.Response** class is reserved for an extension by a JAX-RS implementation providers. An application should use one of the static methods to create a Response instance using a **ResponseBuilder**. An application class should not extend this class directly

We have following methods in Responnse classs which are used majorly

1. public abstract int **getStatus**()
2. public abstract [MultivaluedMap](http://docs.oracle.com/javaee/6/api/javax/ws/rs/core/MultivaluedMap.html)<String,Object> **getMetadata**()
3. public static [ResponseBuilder](http://docs.oracle.com/javaee/6/api/javax/ws/rs/core/Response.ResponseBuilder.html) **status**([Response.StatusType](http://docs.oracle.com/javaee/6/api/javax/ws/rs/core/Response.StatusType.html) status)
4. public static [Response.ResponseBuilder](http://docs.oracle.com/javaee/6/api/javax/ws/rs/core/Response.ResponseBuilder.html) **ok**()

### 2: JAX-RS @PathParam annotation Example

In RESTful (JAX-RS) web services **@PathParam** annotation will be used to bind RESTful URL parameter values with the method arguments

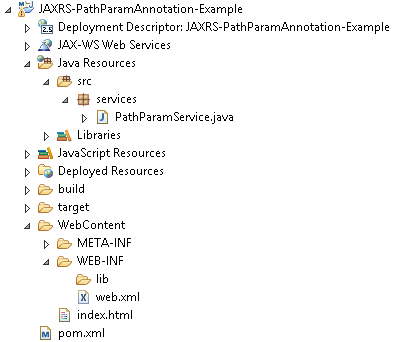
http://localhost:8001/<Rest Service Name>/rest/customers/100/Satya

Here the two parameters appear in the end of the above URL [100 & Satya], which are separated by forward slash (/) are called as path parameters

We will read those URL paramters in our webservice method using

@PathParam("paramname") String variablename

**1. Create Dynamic web project in eclipse, convert that into Maven Project**



2. **Add RESTEasy jar files manually / through Maven by writing repo details in pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>JAXRS-PathParamAnnotation-Example</groupId>

<artifactId>JAXRS-PathParamAnnotation-Example</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<repositories>

<repository>

<id>maven2-repository.java.net</id>

<name>Java.net Repository for Maven</name>

<url>http://download.java.net/maven/2/</url>

<layout>default</layout>

</repository>

</repositories>

<dependencies>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-server</artifactId>

<version>1.8</version>

</dependency>

</dependencies>

<build>

<finalName>JAXRS-PathParamAnnotation-Example</finalName>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<compilerVersion>1.5</compilerVersion>

<source>1.5</source>

<target>1.5</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

3. Create RESTFul webservice using RESTEasy

package services;

@Path("/students")

public class PathParamService {

@GET

@Path("{rollno}/{name}/{address}")

@Produces("text/html")

public Response getResultByPassingValue(

@PathParam("rollno") String rollno,

@PathParam("name") String name,

@PathParam("address") String address) {

String output = "<h1>PathParamService Example</h1>";

output = output+"<br>Roll No : "+rollno;

output = output+"<br>Name : "+name;

output = output+"<br>Address : "+address;

return Response.*status*(200).entity(output).build();

}

}

4. Configure web.xml, Register Jersey dependency class

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/j2ee"* xmlns:web=*"http://xmlns.jcp.org/xml/ns/javaee"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"* id=*"WebApp\_ID"* version=*"2.4"*>

<display-name>JAXRS-PathParamAnnotation-Example</display-name>

<servlet>

<servlet-name>jersey-serlvet</servlet-name>

<servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>com.sun.jersey.config.property.packages</param-name>

<param-value>services</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>jersey-serlvet</servlet-name>

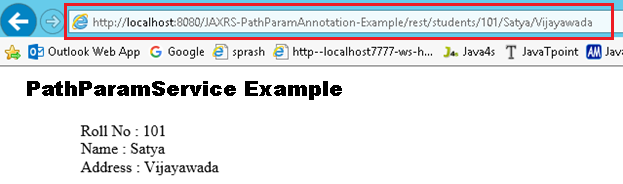
<url-pattern>/rest/\*</url-pattern>

</servlet-mapping>

</web-app>

**5. Test Webservice directly by using URL / writing webservice client**

[**http://localhost:8080/JAXRS-PathParamAnnotation-Example/rest/students/101/Satya/Vijayawada**](http://localhost:8080/JAXRS-PathParamAnnotation-Example/rest/students/101/Satya/Vijayawada)



Note : In Upcomming Examples POM.XML , Web.xml are same for all applications. So iam skipping those. If any changes in those files I will mention don’t worry ☺

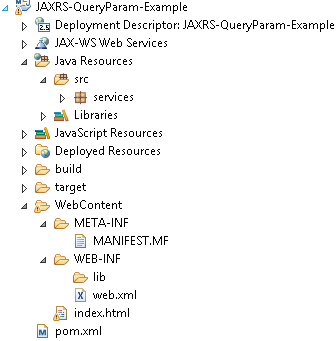
### 3: JAX-RS @QueryParam annotation Example

http://localhost:8001/…/rest/customers?custNo=100&custName=Satya

If the client sends an input in the form of query string in the URL, then those parameters are called as Query Parameters. If you observe the above syntax, client passing **custNo=100&custName=Satya** started after question mark (?) symbol and each parameter is separated by & symbol, those parameters are called as query parameters.

Steps to Implement this Web Service Application

**1. Create Dynamic web project in eclipse, convert that into Maven Project**



2. Add RESTEasy jar files manually / through Maven by writing repo details in **pom.xml(skip)**

**3. Create RESTFul webservice using Jersy**

package services;

@Path("/students")

public class QueryParamService {

@GET

@Produces("text/html")

public Response getResultByPassingValue(

@QueryParam("rollno") String rollno,

@QueryParam("name") String name,

@QueryParam("address") String address) {

String output = "<h1>QueryParamService Example</h1>";

output = output+"<br>Roll No : "+rollno;

output = output+"<br>Name : "+name;

output = output+"<br>Address : "+address;

return Response.*status*(200).entity(output).build();

}

}

4. Configure web.xml **(SKIPING)**

5. Test Webservice directly by using URL / writing webservice client

[**http://localhost:8080/JAXRS-QueryParam-Example/rest/students?rollno=1218&name=SATYA KAVETI&address=VIJAYAWADA**](http://localhost:8080/JAXRS-QueryParam-Example/rest/students?rollno=1218&name=SATYA%20KAVETI&address=VIJAYAWADA)



### 4. JAX-RS @DefaultValue Annotation

Sometimes URL doesn’t contain the values which are expected the methods. In that situation we can use @DefaultValue for passing default values to method parameters. @DefaultValue is good for optional parameter.

package services;

import javax.ws.rs.DefaultValue;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

@Path("/students")

public class QueryParamwithDefaultvalueService {

@GET

@Produces("text/html")

public Response getResultByPassingValue(@DefaultValue("1000") @QueryParam("rollno") String rollno,

@DefaultValue("XXXX") @QueryParam("name") String name,

@DefaultValue("XXXX") @QueryParam("address") String address) {

String output = "<h1>QueryParamwithDefaultvalueService Example</h1>";

output = output + "<br>Roll No : " + rollno;

output = output + "<br>Name : " + name;

output = output + "<br>Address : " + address;

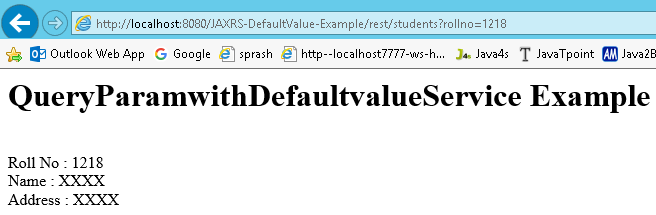
return Response.*status*(200).entity(output).build();

}

}

[**http://localhost:8080/JAXRS-DefaultValue-Example/rest/students?rollno=1218**](http://localhost:8080/JAXRS-DefaultValue-Example/rest/students?rollno=1218)

**in Above URL we are not passing Name, Address paramaeter values. So it will take Default values passed in** @DefaultValue("XXXX") annotation

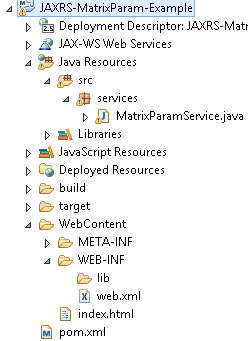


### 5: JAX-RS @MatrixParam annotation Example

http://localhost:8001/…/rest/customers;custNo=100;custName=Satya

Matrix parameters are another way defining the parameters to be added to URL. If you observe the above syntax, client is passing two parameters each are separated by semicolon(;), these parameters are called as matrix parameters. **Remember these parameters may appear any where in the path**

Steps to Implement this Web Service Application

1. Create Dynamic web project in eclipse, convert that into Maven Project

2. Add RESTEasy jar files manually / through Maven by writing repo details in **pom.xml(Skipping)**

**3. Create RESTFul webservice using Jersey**

package services;

import javax.ws.rs.\*;

import javax.ws.rs.core.Response;

@Path("/students")

public class MatrixParamService{

@GET

@Produces("text/html")

public Response getResultByPassingValue(

@MatrixParam("rollno") String rollno,

@MatrixParam("name") String name,

@MatrixParam("address") String address) {

String output = "<h1>@MatrixParam Example</h1>";

output = output+"<br>Roll No : "+rollno;

output = output+"<br>Name : "+name;

output = output+"<br>Address : "+address;

return Response.*status*(200).entity(output).build();

}

}

4. Configure **web.xml**, Register RESTEasy dependency class **(Skipping)**

5. Test Webservice directly by using URL / writing webservice client

[**http://localhost:8080/JAXRS-MatrixParam-Example/rest/students;rollno=1118;name=SATYA%20KAVETI;address=VIJAYAWADA**](http://localhost:8080/JAXRS-MatrixParam-Example/rest/students;rollno=1118;name=SATYA%20KAVETI;address=VIJAYAWADA)

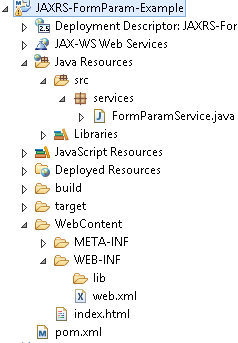


### 6: JAX-RS @FormParam annotation Example

If we have a HTML form having two input fields and submit button. Lets client enter those details and submit to the RESTful web service. Then the rest service will extract those details by using this **@FormParam** annotation.we can use @FormParam annotation to bind HTML form parameters value to a Java method

Steps to Implement this Web Service Application

1. Create Dynamic web project in eclipse, convert that into Maven Project



2. Configure **pom.xml, web.xml (Skipping)**

**3. Create RESTFul webservice Jersey**

package services;

import javax.ws.rs.FormParam;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.Response;

@Path("/students")

public class FormParamService {

@POST

@Path("/registerStudent")

@Produces("text/html")

public Response getResultByPassingValue(

@FormParam("rollno") String rollno,

@FormParam("name") String name,

@FormParam("address") String address) {

String output = "<h1>@FormParam Example - REGISTRATION COMPLETED!!!</h1>";

output = output+"<br>Roll No : "+rollno;

output = output+"<br>Name : "+name;

output = output+"<br>Address : "+address;

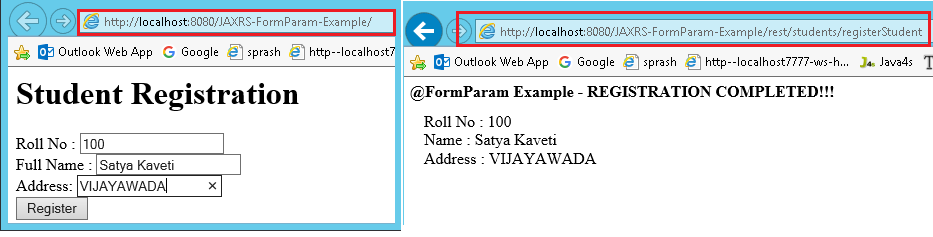
return Response.*status*(200).entity(output).build();

}

}

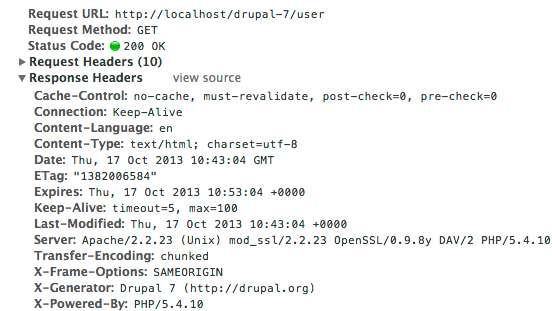
**5. Test Webservice directly by using URL / writing webservice client**

1. <http://localhost:8080/JAXRS-FormParam-Example/>
2. <http://localhost:8080/JAXRS-FormParam-Example/rest/students/registerStudent>



### 7: JAX-RS @HeaderParam Example

We can get the HTTP header details using JAXRS. HTTP headers like below formate

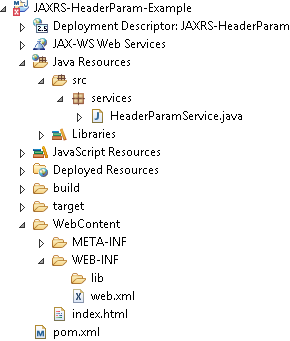


We have two ways to get HTTP request header in JAX-RS :

1. Inject directly with @HeaderParam

2. Pragmatically via @Context

JAX-RS @HeaderParam Example

1. Create Dynamic web project in eclipse, convert that into Maven Project 

2. Configure **pom.xml, web.xml (Skipping)**

3. Create RESTFul webservice

package services;

import javax.ws.rs.GET;

import javax.ws.rs.HeaderParam;

import javax.ws.rs.Path;

import javax.ws.rs.core.Response;

@Path("/rs")

public class HeaderParamService {

@GET

@Path("/headerparam")

public Response getHeader(

@HeaderParam("user-agent") String userAgent,

@HeaderParam("Accept") String accept,

@HeaderParam("Accept-Encoding") String encoding,

@HeaderParam("Accept-Language") String lang) {

String output = "<h1>@Headerparam Example</h1>";

output = output+"<br>user-agent : "+userAgent;

output = output+"<br>Accept : "+accept;

output = output+"<br>Accept-Encoding : "+encoding;

output = output+"<br>Accept-Language: "+lang;

return Response.*status*(200)

.entity(output)

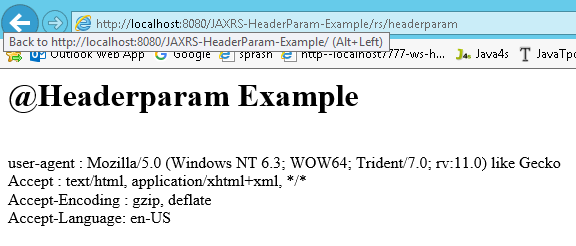
.build();

}

}

5. Test Webservice directly by using URL / writing webservice client

<http://localhost:8080/JAXRS-HeaderParam-Example/rs/headerparam>



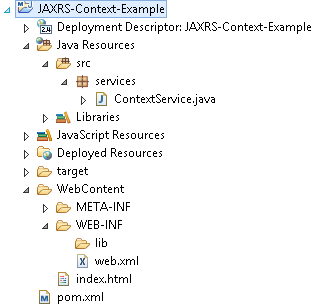
### 8: JAX-RS @Context Example

JAX-RS provides the **@Context** annotation to inject a variety of resources in your RESTful services. Some of the most commonly injected components are HTTP headers, HTTP URI related information

* ***HTTP headers***
* ***HTTP URI details***
* ***Security Context***
* ***Resource Context***
* ***Request***
* ***Configuration***
* ***Application***
* ***Providers***

JAX-RS @Context Example

1. Create Dynamic web project in eclipse, convert that into Maven Project



2. Configure **pom.xml, web.xml (Skipping)**

3. Create RESTFul webservice

package services;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.core.Context;

import javax.ws.rs.core.HttpHeaders;

import javax.ws.rs.core.Response;

import javax.ws.rs.core.SecurityContext;

import javax.ws.rs.core.UriInfo;

@Path("/rst")

public class ContextService {

@GET

@Path("httpheaders")

public Response getHttpheaders(@Context HttpHeaders headers){

String output = "<h1>@@Context Example - HTTP headers</h1>";

output = output+"<br>ALL headers -- "+ headers.getRequestHeaders().toString();

output = output+"<br>All Cookies -- "+ headers.getCookies().values();

return Response.*status*(200)

.entity(output)

.build();

}

@GET

@Path("uriinfo")

public Response test(@Context UriInfo uriDetails){

String output = "<h1>@@Context Example - HTTP URI details</h1>";

output = output+"<br>ALL query parameters -- "+ uriDetails.getQueryParameters().toString();

output = output+"<br>'id' query parameter -- "+ uriDetails.getQueryParameters().get("id");

output = output+"<br>Complete URI -- "+ uriDetails.getRequestUri();

return Response.*status*(200)

.entity(output)

.build();

}

@GET

@Path("securitycontext")

public Response test(@Context SecurityContext secContext){

String output = "<h1>@@Context Example - Security Context</h1>";

//output = output+"<br>Caller -- "+ secContext.getUserPrincipal().getName();

output = output+"<br>Authentication Scheme -- "+ secContext.getAuthenticationScheme();

output = output+"<br>Over HTTPS ? -- "+ secContext.isSecure();

output = output+"<br>Belongs to 'admin' role? -- "+ secContext.isUserInRole("admin");

return Response.*status*(200)

.entity(output)

.build();

}

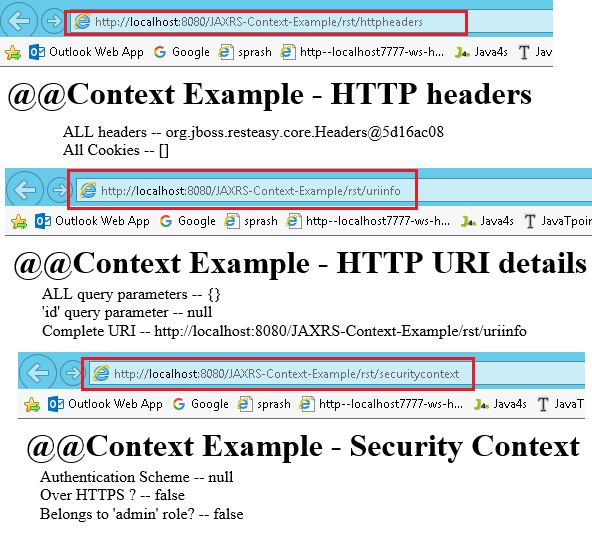
}

5. Test Webservice directly by using URL / writing webservice client

[**http://localhost:8080/JAXRS-Context-Example/rst/httpheaders**](http://localhost:8080/JAXRS-Context-Example/rst/httpheaders)

[**http://localhost:8080/JAXRS-Context-Example/rst/uriinfo**](http://localhost:8080/JAXRS-Context-Example/rst/uriinfo)

[**http://localhost:8080/JAXRS-Context-Example/rst/securitycontext**](http://localhost:8080/JAXRS-Context-Example/rst/securitycontext)



### 9: JAX-RS Download files (text/image/pdf/execel) Example

We can download any type of files from the RESTful web services, **@produces** annotation

We should annotate our method with

* **@Produces(“text/plain“)** If you are expecting Text file as response
* **@Produces(“image/your image type[.jpg/.png/.gif]”)** for downloading any Image files
* **@Produces(“application/pdf“)** for downloading PDF files

Steps to Implement this Web Service Application

1. Create **Dynamic web project in eclipse**, **pom.xml, web.xml (Skipping)**

2. Create RESTFul webservice Jersey

package service;

import java.io.File;

import javax.ws.rs\*;

@Path("/download")

public class FileDownloadService {

private static final String *TEXT\_FILE\_PATH* = "C:\\Users\\kaveti\_s\\textfile.txt";

private static final String *IMG\_FILE\_PATH* = "C:\\Users\\kaveti\_s\\img.jpg";

private static final String *PDF\_FILE\_PATH* = "C:\\Users\\kaveti\_s\\pdffile.pdf";

private static final String *XLS\_FILE\_PATH* = "C:\\Users\\kaveti\_s\\excel.xlsx";

//TEXTFILE DOWNLOAD

@GET

@Path("/textfile")

@Produces("text/plain")

public Response downloadTextFile() {

File file = new File(*TEXT\_FILE\_PATH*);

ResponseBuilder response = Response.*ok*((Object) file);

response.header("Content-Disposition",

"attachment; filename=\"smlcodes.log\"");

return response.build();

}

//IMAGE DOWNLOAD

@GET

@Path("/image")

@Produces("image/jpg")

public Response downloadImage() {

File file = new File(*IMG\_FILE\_PATH*);

ResponseBuilder response = Response.*ok*((Object) file);

response.header("Content-Disposition",

"attachment; filename=smlcodes.jpg");

return response.build();

}

//PDF DOWNLOAD

@GET

@Path("/pdf")

@Produces("application/pdf")

public Response downloadPDF() {

File file = new File(*PDF\_FILE\_PATH*);

ResponseBuilder response = Response.*ok*((Object) file);

response.header("Content-Disposition",

"attachment; filename=smlcodes.pdf");

return response.build();

}

//XLS DOWNLOAD

@GET

@Path("/xls")

@Produces("application/vnd.ms-excel")

public Response downloadXLS() {

File file = new File(*XLS\_FILE\_PATH*);

ResponseBuilder response = Response.*ok*((Object) file);

response.header("Content-Disposition",

"attachment; filename=new-smlcodes.xls");

return response.build();

}

}

//index.html

<h1>JAXRS-FileDownloads-Example</h1>

<h3><a href=*"download/textfile"*>TEXT FILE DOWNLOAD</a></h3>

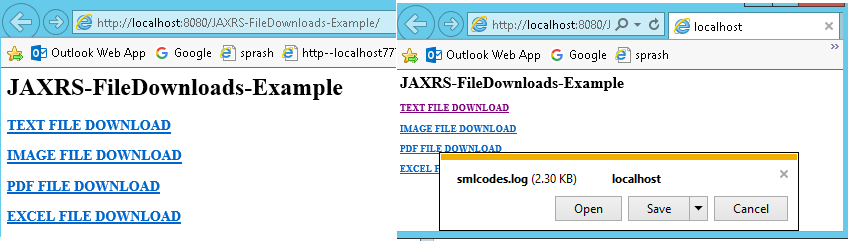
<h3><a href=*"download/image"*>IMAGE FILE DOWNLOAD</a></h3>

<h3><a href=*"download/pdf"*>PDF FILE DOWNLOAD</a></h3>

<h3><a href=*"download/xls"*>EXCEL FILE DOWNLOAD</a></h3>

5. Test Webservice directly by using URL / writing webservice client

[**http://localhost:8080/JAXRS-FileDownloads-Example/**](http://localhost:8080/JAXRS-FileDownloads-Example/)

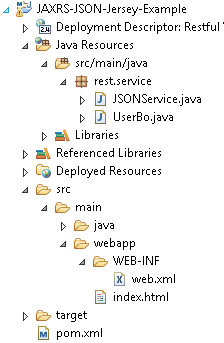


### 10. JAX-RS JSON Example Using Jersey

Jersey uses Jackson to convert **object to / form JSON**. In this example, we show you how to convert a “user” object into JSON format, and return it back to user

Steps to Implement this Web Service Application

**1. Create Dynamic web project in eclipse, convert that into Maven Project**



**2. Configure pom.xml**

To make Jersey support JSON mapping, declares “jersey-json.jar” in Maven pom.xml file.

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>JAX-RS-JSON\_Example-Jersey</groupId>

<artifactId>JAX-RS-JSON\_Example-Jersey</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<plugin>

<artifactId>maven-war-plugin</artifactId>

<version>3.0.0</version>

<configuration>

<warSourceDirectory>WebContent</warSourceDirectory>

</configuration>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>maven2-repository.java.net</id>

<name>Java.net Repository for Maven</name>

<url>http://download.java.net/maven/2/</url>

<layout>default</layout>

</repository>

</repositories>

<dependencies>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-server</artifactId>

<version>1.8</version>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-json</artifactId>

<version>1.8</version>

</dependency>

</dependencies>

</project>

3. Configure **web.xml**

In web.xml, declares “com.sun.jersey.api.json.POJOMappingFeature” as “init-param” in Jersey mapped servlet. It will make Jersey support JSON/object mapping.

<web-app id=*"WebApp\_ID"* version=*"2.4"*

xmlns=*"http://java.sun.com/xml/ns/j2ee"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/j2ee*

*http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"*>

<display-name>Restful Web Application</display-name>

<servlet>

<servlet-name>jersey-serlvet</servlet-name>

<servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>com.sun.jersey.config.property.packages</param-name>

<param-value>rest.service</param-value>

</init-param>

<init-param>

<param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>

<param-value>true</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>jersey-serlvet</servlet-name>

<url-pattern>/rest/\*</url-pattern>

</servlet-mapping>

</web-app>

**4. Write “UserBo” class**

Write “UserBo” class object, Jersey will convert this object into JSON format.

package services;

public class UserBo {

String username;

String password;

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

@Override

public String toString() {

// TODO Auto-generated method stub

return "User [username=" + username + ", password=" + password + "]";

};

}

**5. Create RESTFul webservice Jersey**

Annotate the method with @Produces(MediaType.APPLICATION\_JSON). Jersey will use Jackson to handle the JSON conversion automatically.

package rest.service;

@Path("/json")

public class JSONService {

@GET

@Path("/getjson")

@Produces(MediaType.*APPLICATION\_JSON*)

public UserBo getboInJSON() {

UserBo bo = new UserBo();

bo.setUsername("satyakaveti@gmail.com");

bo.setPassword("XCersxg34CXeWER341DS@#we");

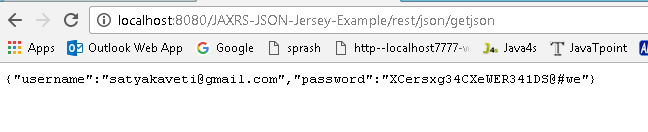
return bo;

}

}

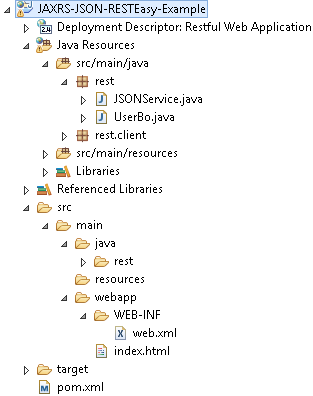
**6. Test Webservice directly by using URL / writing webservice client**

<http://localhost:8080/JAX-RS-JSON_Example-Jersey/json/getjson>



### 11. JAX-RS JSON Example Using RESTEasy

To integrate Jackson with RESTEasy, you just need to include **“resteasy-jackson-provider.jar“.**

**1. Create Dynamic web project in eclipse, convert that into Maven Project**

**2. Configure pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>JAXRS-JSON-RESTEasy-Example</groupId>

<artifactId>JAXRS-JSON-RESTEasy-Example</artifactId>

<packaging>war</packaging>

<version>1.0-SNAPSHOT</version>

<name>JAXRS-JSON-RESTEasy-Example</name>

<url>http://maven.apache.org</url>

<repositories>

<repository>

<id>JBoss repository</id>

<url>https://repository.jboss.org/nexus/content/groups/public-jboss/</url>

</repository>

</repositories>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.8.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.jboss.resteasy</groupId>

<artifactId>resteasy-jaxrs</artifactId>

<version>2.2.1.GA</version>

</dependency>

<dependency>

<groupId>org.jboss.resteasy</groupId>

<artifactId>resteasy-jackson-provider</artifactId>

<version>2.2.1.GA</version>

</dependency>

</dependencies>

<build>

<finalName>JAXRS-JSON-RESTEasy-Example</finalName>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.6</source>

<target>1.6</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**3. Configure web.xml**

In web.xml Disable RESTEasy auto scanning and register your REST service manually, otherwise, you will get ***Illegal to inject a message body into a singleton into public.JacksonJsonProvider Error***

<web-app>

<display-name>JAXRS-JSON-RESTEasy-Example</display-name>

<context-param>

<param-name>resteasy.resources</param-name>

<param-value>rest.JSONService</param-value>

</context-param>

<listener>

<listener-class>

org.jboss.resteasy.plugins.server.servlet.ResteasyBootstrap

</listener-class>

</listener>

<servlet>

<servlet-name>resteasy-servlet</servlet-name>

<servlet-class>

org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher

</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>resteasy-servlet</servlet-name>

<url-pattern>/rest/\*</url-pattern>

</servlet-mapping>

</web-app>

**4. Write “UserBo” class**

Write “UserBo” class object, Jersey will convert this object into JSON format.

package rest;

public class UserBo {

String username;

String password;

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

@Override

public String toString() {

return "USER [username=" + username + ", password=" + password + "]";

}

}

**5. Create RESTFul webservice**

package rest;

import javax.ws.rs.Consumes;

import javax.ws.rs.GET;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.Response;

@Path("/rest/json")

public class JSONService {

@GET

@Path("/get")

@Produces("application/json")

public UserBo getUserBoInJSON() {

UserBo bo = new UserBo();

bo.setUsername("satyakaveti@gmail.com");

bo.setPassword("XCersxg34CXeWER341DS@#we");

return bo;

}

@POST

@Path("/post")

@Consumes("application/json")

public Response createUserBoInJSON(UserBo UserBo) {

String result = "UserBo created : " + UserBo;

return Response.*status*(201).entity(result).build();

}

}

**6. Test Webservice directly by using URL / writing webservice client**

[**http://localhost:8080/JAXRS-JSON-RESTEasy-Example/rest/json/get**](http://localhost:8080/JAXRS-JSON-RESTEasy-Example/rest/json/get)



### 12. JAX-RS XML Example Using Jersey

JAX-RS supports conversion of java objects into XML with the help of JAXB. As Jersey it self contains JAXB libraries we no need to worry about JAXB-Jersey integration. Just include **“jersey-server.jar”**

Steps to Implement this Web Service Application

**1. Create Dynamic web project in eclipse, convert that into Maven Project**

2. Configure **pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>JAXRS-XML-Jersey-Example</groupId>

<artifactId>JAXRS-XML-Jersey-Example</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<repositories>

<repository>

<id>maven2-repository.java.net</id>

<name>Java.net Repository for Maven</name>

<url>http://download.java.net/maven/2/</url>

<layout>default</layout>

</repository>

</repositories>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.8.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-server</artifactId>

<version>1.8</version>

</dependency>

</dependencies>

<build>

<finalName>JAXRS-XML-Jersey-Example</finalName>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<compilerVersion>1.5</compilerVersion>

<source>1.5</source>

<target>1.5</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

3. Configure **web.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app>

<display-name>RestPathAnnotationExample</display-name>

<servlet>

<servlet-name>jersey-serlvet</servlet-name>

<servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>com.sun.jersey.config.property.packages</param-name>

<param-value>rest.service</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>jersey-serlvet</servlet-name>

<url-pattern>/rest/\*</url-pattern>

</servlet-mapping>

</web-app>

**4. write Customer POJO class**

Write **Customer POJO class &** Annotate object with JAXB annotation, for conversion later.

package rest.service;

import javax.xml.bind.annotation.XmlAttribute;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "customer")

public class Customer {

String custName;

String custCountry;

int custId;

@XmlElement

public String getCustName() {

return custName;

}

public void setCustName(String custName) {

this.custName = custName;

}

@XmlElement

public String getCustCountry() {

return custCountry;

}

public void setCustCountry(String custCountry) {

this.custCountry = custCountry;

}

@XmlAttribute

public int getCustId() {

return custId;

}

public void setCustId(int custId) {

this.custId = custId;

}

}

**5. Create RESTFul webservice Jersey**

To return a XML file, annotate the method with @Produces(MediaType.APPLICATION\_XML). Jersey will convert the JAXB annotated object into XML file automatically.

package rest.service;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

@Path("/customers")

public class RestfulXMLExample {

@GET

@Path("/{id}")

@Produces(MediaType.*APPLICATION\_XML*)

public Customer getCustomerDetails(@PathParam("id") int custId) {

// WRITE DATABASE LOGIC TO RETRIEVE THE CUSTOMER RECORD WITH 'custID'

Customer cust = new Customer();

cust.setCustName("satya");

cust.setCustCountry("india");

cust.setCustId(custId);

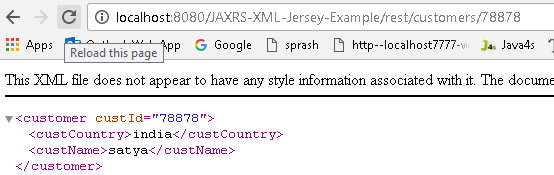
return cust;

}

}

**6. Test Webservice directly by using URL / writing webservice client**

[**http://localhost:8080/JAXRS-XML-Jersey-Example/rest/customers/78878**](http://localhost:8080/JAXRS-XML-Jersey-Example/rest/customers/78878)



### 13. JAX-RS RESTFul Java Clinets

So far we used URL directly to Test our RESTful service. But in the real time we will call the services by writing some client application logic. We have different ways to write a RESTful client

They are

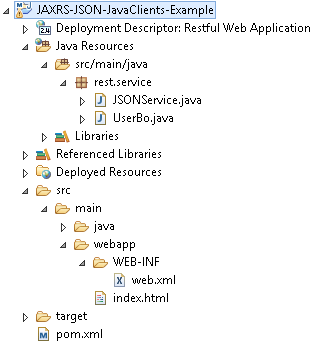
* **java.net.URL**
* **Apache HttpClient**
* **RESTEasy client**
* **Jersey client**

**Every Java Clinet can send two types of requests**

1. **GET**
2. **POST**

**We will see one by one by Example. Here we are Taking JAXRS-JSON-Jersey-Example for writing clinets. For all webservices are same. Only difference in Java Clinets**

JAXRS-JSON-Jersey-Example

1. Create Dynamic web project in eclipse, convert that into Maven Project 

2. Configure **pom.xml, web.xml (May change for Every Java Clinet, please Observe)**

3. Craete UserBo for Converting Java Object to JSON data

package rest.service;

public class UserBo {

String username;

String password;

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

@Override

public String toString() {

return "USER [username=" + username + ", password=" + password + "]";

}

}

4. Create Web Service having both @GET @POST for testing with Java Clinets

package rest.service;

import javax.ws.rs.core.Response;

@Path("/json")

public class JSONService {

@GET

@Path("/getjson")

@Produces(MediaType.*APPLICATION\_JSON*)

public UserBo getboInJSON() {

UserBo bo = new UserBo();

bo.setUsername("satyakaveti@gmail.com");

bo.setPassword("XCersxg34CXeWER341DS@#we");

return bo;

}

@POST

@Path("/postjson")

@Consumes(MediaType.*APPLICATION\_JSON*)

public Response createTrackInJSON(UserBo bo) {

String result = "USER DATA SAVED!! " + bo;

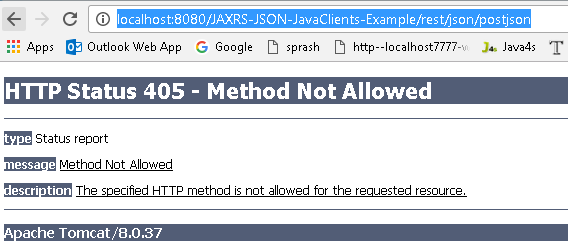
return Response.*status*(201).entity(result).build();

}

}

5. Test Webservice directly by using URL / writing webservice client

GET: [**http://localhost:8080/JAXRS-JSON-JavaClients-Example/rest/json/getjson**](http://localhost:8080/JAXRS-JSON-JavaClients-Example/rest/json/getjson)

POST: [**http://localhost:8080/JAXRS-JSON-JavaClients-Example/rest/json/postjson**](http://localhost:8080/JAXRS-JSON-JavaClients-Example/rest/json/postjson)

**So far we are used above process to Test the Web Services. Now lets see how to test webservices with Java clinets.**

#### java.net.URL

Here we will use “java.net.URL” and “java.net.HttpURLConnection” to create a simple Java client to send **“GET” and “POST”** request.

**GET Request Example**

package rest.clients;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.HttpURLConnection;

import java.net.MalformedURLException;

import java.net.URL;

public class NetUrlGETClient {

public static void main(String[] args) {

try {

URL url = new URL("http://localhost:8080/JAXRS-JSON-JavaClients-Example/rest/json/getjson");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("GET");

conn.setRequestProperty("Accept", "application/json");

if (conn.getResponseCode() != 200) {

throw new RuntimeException("Failed : HTTP error code : " + conn.getResponseCode());

}

BufferedReader br = new BufferedReader(new InputStreamReader((conn.getInputStream())));

String output;

System.*out*.println("Output from Server .... \n");

while ((output = br.readLine()) != null) {

System.*out*.println(output);

}

conn.disconnect();

} catch (MalformedURLException e) {

e.printStackTrace();

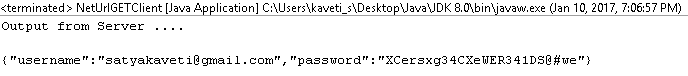
} catch (IOException e) {

e.printStackTrace();

}

}

}

****

**POST Request Example**

package rest.clients;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.net.HttpURLConnection;

import java.net.MalformedURLException;

import java.net.URL;

public class NetUrlPOSTClient {

public static void main(String[] args) {

try {

URL url = new URL("http://localhost:8080/JAXRS-JSON-JavaClients-Example/rest/json/postjson");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setDoOutput(true);

conn.setRequestMethod("POST");

conn.setRequestProperty("Content-Type", "application/json");

String input = "{\"username\":\"satyakaveti@gmail.com\",\"password\":\"XCersxg34CXeWER341DS@#we\"}";

OutputStream os = conn.getOutputStream();

os.write(input.getBytes());

os.flush();

if (conn.getResponseCode() != HttpURLConnection.*HTTP\_CREATED*) {

throw new RuntimeException("Failed : HTTP error code : " + conn.getResponseCode());

}

BufferedReader br = new BufferedReader(new InputStreamReader((conn.getInputStream())));

String output;

System.*out*.println("Output from Server .... \n");

while ((output = br.readLine()) != null) {

System.*out*.println(output);

}

conn.disconnect();

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

}



#### Apache HttpClient

Apache HttpClient is available in Maven central repository, just declares it in your Maven pom.xml file.

**1. Configure POM.xml with Apache HTTPClinet**

<dependency>

<groupId>org.apache.httpcomponents</groupId>

<artifactId>httpclient</artifactId>

<version>4.1.1</version>

</dependency

**2.GET Request Example**

package rest.clients;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import org.apache.http.HttpResponse;

import org.apache.http.client.ClientProtocolException;

import org.apache.http.client.methods.HttpGet;

import org.apache.http.impl.client.DefaultHttpClient;

public class ApacheHttpClientGet {

public static void main(String[] args) {

try {

DefaultHttpClient httpClient = new DefaultHttpClient();

HttpGet getRequest = new HttpGet("http://localhost:8080/JAXRS-JSON-JavaClients-Example/rest/json/getjson");

getRequest.addHeader("accept", "application/json");

HttpResponse response = httpClient.execute(getRequest);

if (response.getStatusLine().getStatusCode() != 200) {

throw new RuntimeException("Failed : HTTP error code : " + response.getStatusLine().getStatusCode());

}

BufferedReader br = new BufferedReader(new InputStreamReader((response.getEntity().getContent())));

String output;

System.*out*.println("Output from Server .... \n");

while ((output = br.readLine()) != null) {

System.*out*.println(output);

}

httpClient.getConnectionManager().shutdown();

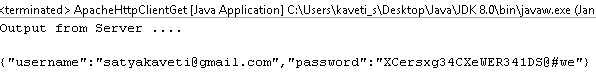
} catch (Exception e) {

e.printStackTrace();

}

}

}



**3.POST Request Example**

package rest.clients;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.MalformedURLException;

import org.apache.http.HttpResponse;

import org.apache.http.client.methods.HttpPost;

import org.apache.http.entity.StringEntity;

import org.apache.http.impl.client.DefaultHttpClient;

public class ApacheHttpClientPost {

public static void main(String[] args) {

try {

DefaultHttpClient httpClient = new DefaultHttpClient();

HttpPost postRequest = new HttpPost("http://localhost:8080/JAXRS-JSON-JavaClients-Example/rest/json/postjson");

StringEntity input = new StringEntity("{\"username\":\"satyakaveti@gmail.com\",\"password\":\"XCersxg34CXeWER341DS@#we\"}");

input.setContentType("application/json");

postRequest.setEntity(input);

HttpResponse response = httpClient.execute(postRequest);

if (response.getStatusLine().getStatusCode() != 201) {

throw new RuntimeException("Failed : HTTP error code : " + response.getStatusLine().getStatusCode());

}

BufferedReader br = new BufferedReader(new InputStreamReader((response.getEntity().getContent())));

String output;

System.*out*.println("Output from Server .... \n");

while ((output = br.readLine()) != null) {

System.*out*.println(output);

}

httpClient.getConnectionManager().shutdown();

} catch (MalformedURLException e) {

e.printStackTrace();

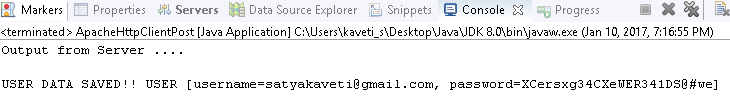
} catch (IOException e) {

e.printStackTrace();

}

}

}



#### 3.RESTEasy client

**1. Configure POM.xml with**

RESTEasy client framework is included in RESTEasy core module, so, you just need to declares the **“resteasy-jaxrs.jar”** in your pom.xml file

<dependency>

<groupId>org.jboss.resteasy</groupId>

<artifactId>resteasy-jaxrs</artifactId>

<version>2.2.1.GA</version>

</dependency>

**2.GET Request Example**

package rest.clients;

import java.io.BufferedReader;

import java.io.ByteArrayInputStream;

import java.io.IOException;

import java.io.InputStreamReader;

public class RESTEasyClientGet {

public static void main(String[] args) {

try {

ClientRequest request = new ClientRequest(

"http://localhost:8080/JAXRS-JSON-JavaClients-Example/rest/json/getjson");

request.accept("application/json");

ClientResponse<String> response = request.get(String.class);

if (response.getStatus() != 200) {

throw new RuntimeException("Failed : HTTP error code : "

+ response.getStatus());

}

BufferedReader br = new BufferedReader(new InputStreamReader(

new ByteArrayInputStream(response.getEntity().getBytes())));

String output;

System.*out*.println("Output from Server .... \n");

while ((output = br.readLine()) != null) {

System.*out*.println(output);

}

} catch (ClientProtocolException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (Exception e) {

e.printStackTrace();

}

}

}

**3.POST Request Example**

package rest.clients;

import java.io.InputStreamReader;

import java.net.MalformedURLException;

import org.jboss.resteasy.client.ClientRequest;

import org.jboss.resteasy.client.ClientResponse;

public class RESTEasyClientPost {

public static void main(String[] args) {

try {

ClientRequest request = new ClientRequest(

"http://localhost:8080/JAXRS-JSON-JavaClients-Example/rest/json/postjson");

request.accept("application/json");

String input = "{\"username\":\"satyakaveti@gmail.com\",\"password\":\"XCersxg34CXeWER341DS@#we\"}";

request.body("application/json", input);

ClientResponse<String> response = request.post(String.class);

if (response.getStatus() != 201) {

throw new RuntimeException("Failed : HTTP error code : "

+ response.getStatus());

}

BufferedReader br = new BufferedReader(new InputStreamReader(

new ByteArrayInputStream(response.getEntity().getBytes())));

String output;

System.*out*.println("Output from Server .... \n");

while ((output = br.readLine()) != null) {

System.*out*.println(output);

}

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (Exception e) {

e.printStackTrace();

}

}

}

#### 4. Jersey client

**1. Configure POM.xml with**

To use Jersey client APIs, declares **“jersey-client.jar”** in your pom.xml file.

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-client</artifactId>

<version>1.8</version>

</dependency>

**2.GET Request Example**

package rest.clients;

import com.sun.jersey.api.client.Client;

import com.sun.jersey.api.client.ClientResponse;

import com.sun.jersey.api.client.WebResource;

public class JerseyClientGet {

public static void main(String[] args) {

try {

Client client = Client.*create*();

WebResource webResource = client

.resource("http://localhost:8080/JAXRS-JSON-JavaClients-Example/rest/json/getjson");

ClientResponse response = webResource.accept("application/json").get(ClientResponse.class);

if (response.getStatus() != 200) {

throw new RuntimeException("Failed : HTTP error code : " + response.getStatus());

}

String output = response.getEntity(String.class);

System.*out*.println("Output from Server .... \n");

System.*out*.println(output);

} catch (Exception e) {

e.printStackTrace();

}

}

}

Output

Output from Server ....

{"username":"satyakaveti@gmail.com","password":"XCersxg34CXeWER341DS@#we"}

**3.POST Request Example**

package rest.clients;

import com.sun.jersey.api.client.Client;

import com.sun.jersey.api.client.ClientResponse;

import com.sun.jersey.api.client.WebResource;

public class JerseyClientPost {

public static void main(String[] args) {

try {

Client client = Client.*create*();

WebResource webResource = client

.resource("http://localhost:8080/JAXRS-JSON-JavaClients-Example/rest/json/postjson");

String input = "{\"username\":\"satyakaveti@gmail.com\",\"password\":\"XCersxg34CXeWER341DS@#we\"}";

ClientResponse response = webResource.type("application/json").post(ClientResponse.class, input);

if (response.getStatus() != 201) {

throw new RuntimeException("Failed : HTTP error code : " + response.getStatus());

}

System.*out*.println("Output from Server .... \n");

String output = response.getEntity(String.class);

System.*out*.println(output);

} catch (Exception e) {

e.printStackTrace();

}

}

}

Output

Output from Server ....

USER DATA SAVED!! USER [username=satyakaveti@gmail.com, password=XCersxg34CXeWER341DS@#we]

### 14.How to Test (JAX-RS) RESTful Web Services

in real time projects we will use different tools to test RESTful web services

**1.Browser Addons**

* Postman [ Chrome Extension ]
* REST Client [ Chrome Extension ]
* Advanced REST Client [ Chrome Extension ]
* Rest Client [ Firefox Add-On ]

**2.JAX-RS Local System Tools**

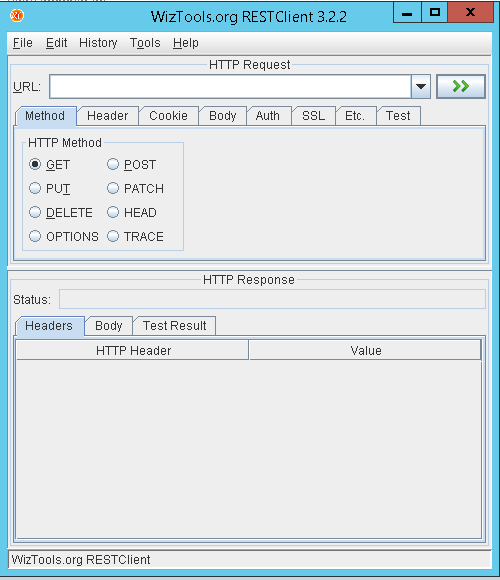
* [RESTClient UI](https://code.google.com/archive/p/rest-client/downloads)
* [SoupUi](https://www.soapui.org/)

RESTClient UI

1. Download .jar file from here <https://code.google.com/archive/p/rest-client/downloads>

2.Run jar by giving >java -jar restclient-ui-3.2.2-jar-with-dependencies.jar

3.It will Opens the window as follows

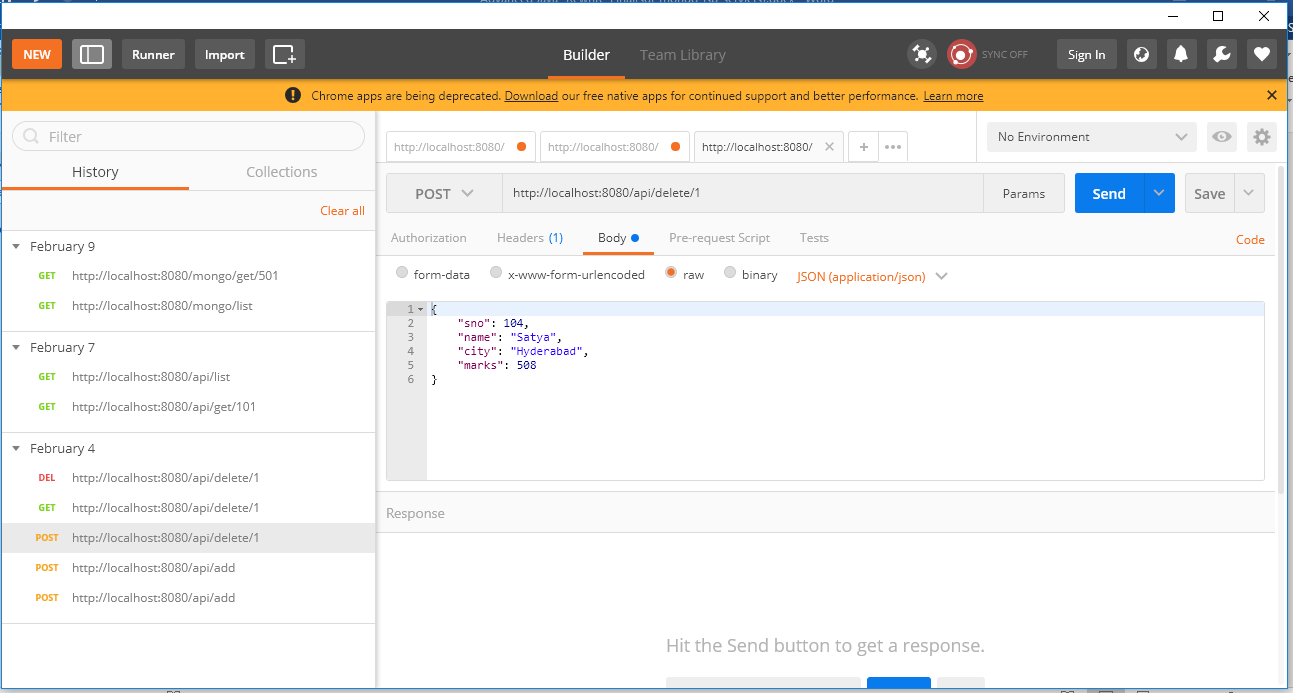


4.Test Your application by proving any running web service URL

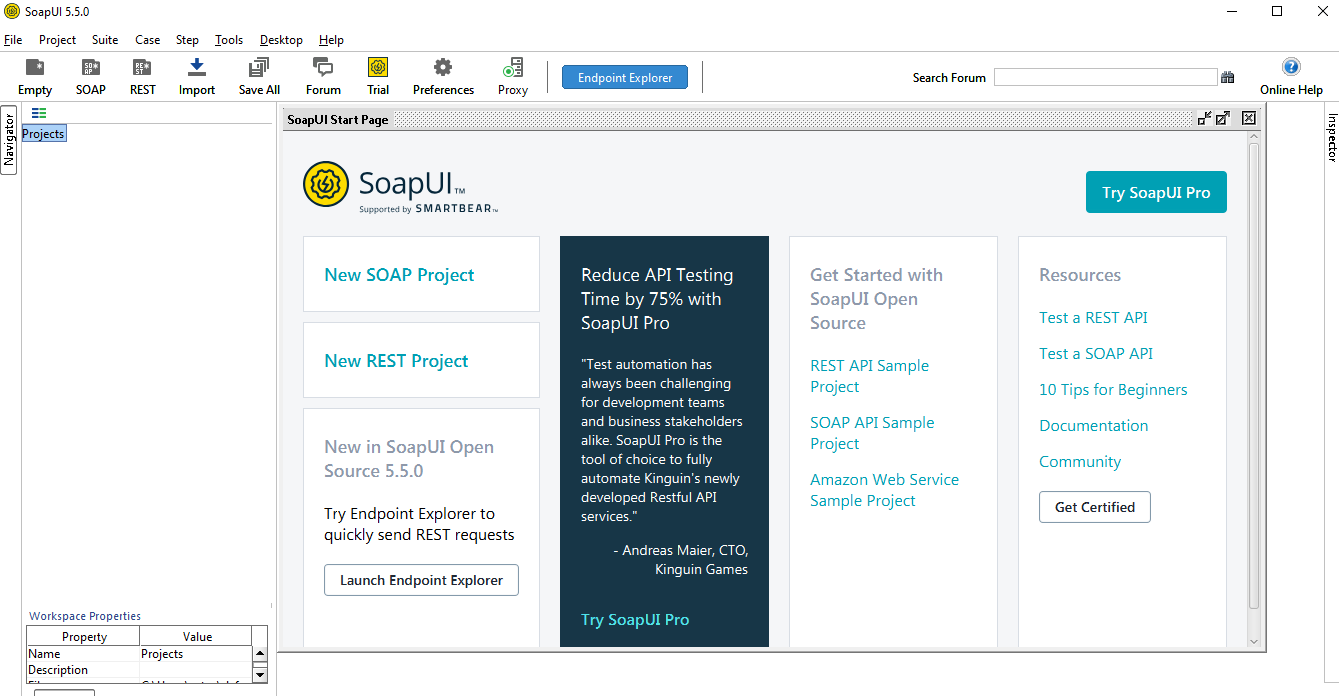
Ex: <http://localhost:8080/JAXRS-JSON-JavaClients-Example/rest/json/getjson>



Postman



Similarly we can work with SoapUI also



# References

JAX-WS

<http://www.java2blog.com/2013/03/soap-web-service-tutorial.html>

[https://examples.smlcodes.com/enterprise-java/jws/jax-ws-annotations-example/](https://examples.javacodegeeks.com/enterprise-java/jws/jax-ws-annotations-example/)

<http://cxf.apache.org/docs/developing-a-service.html>

<https://docs.oracle.com/cd/E13222_01/wls/docs92/webserv/annotations.html>

<http://docs.oracle.com/javaee/5/api/javax/jws/WebService.html>

<https://jax-ws.java.net/jax-ws-ea3/docs/annotations.html>

JAX-RS

<http://www.java4s.com/web-services/restful-web-services-jax-rs-annotations/>

<http://www.mkyong.com/tutorials/jax-rs-tutorials/>

<http://www.mkyong.com/tutorials/jax-rs-tutorials/>

<http://www.java4s.com/web-services/>