PROGRAMACIÓN ORIENTADA A OBJETOS

CON JAVA 17

ACADEMIA CONSULTEC

www.consultec-ti.com

VALENTÍA | AGILIDAD | EVOLUCIÓN | OPTIMIZACIÓN

Agenda **del día**

4

- JDBC API
- API REST con JAX-RS

ACADEMIA CONSULTEC

JDBC
API

ACADEMIA CONSULTEC

API JDBC

La API de JDBC (Java DataBase Connectivity) provee acceso a datos desde Java, por tanto, es un API (Application programming interface) que describe o define una librería estándar para acceso a fuentes de datos, principalmente orientado a Bases de Datos relacionales que usan SQL (Structured Query Language).

JDBC no sólo provee un interfaz para acceso a motores de bases de datos, sino que también define una arquitectura estándar, para que los fabricantes puedan crear los drivers que permitan a las aplicaciones java el acceso a los datos.

Usando esta API podemos acceder a variadas fuentes de datos:

- Bases de datos relacionales
- Hojas de cálculo (spreadsheets)
- Archivos planos.

ACADEMIA CONSULTEC

Objetivos JDBC

Al desarrollar estas API se decidió seguir una serie de normas, a seguir para la definición de la interfaz, y que han condicionado en gran manera el resultado final. Algunas de estas características son:

- API a nivel SQL: JDBC es un API de bajo nivel, es decir, que está orientado a permitir ejecutar comandos SQL directamente, y procesar los resultados obtenidos.
- Compatible con SQL: JDBC obliga a que al menos, el driver de las diferentes BD cumpla el Estándar ANSI SQL 92 permitiendo que cualquier comando SQL pueda ser pasado al driver directamente.
- JDBC debe ser utilizable sobre cualquier otro API de acceso a Bases de Datos.
- JDBC debe proveer un interfaz homogéneo al resto de APIs de Java.
- JDBC debe ser un API simple.
- JDBC debe ser fuertemente tipado, y siempre que sea posible de manera estática.
- JDBC debe mantener los casos comunes de acceso a Base de Datos lo más sencillo posible: SELECT, INSERT, DELETE y UPDATE.

Controladores JDBC (Driver)

Los controladores JDBC son adaptadores del lado del cliente (instalados en la máquina del cliente, no en el servidor) que convierten las peticiones de los programas Java a un protocolo que el SGBD pueda entender. Existen 4 tipos de controladores JDBC:

- Controlador de tipo 1 o controlador puente JDBC-ODBC
- Controlador de tipo 2 o controlador Native-API
- Controlador de tipo 3 o controlador de protocolo de red
- Controlador de tipo 4 o controlador Thin

ACADEMIA CONSULTEC

Controladores JDBC (Driver)

Controlador de tipo 1 o controlador puente JDBC-ODBC

El controlador de tipo 1 o controlador puente JDBC-ODBC utiliza el controlador ODBC para conectarse a la base de datos. El controlador puente JDBC-ODBC convierte las llamadas a métodos JDBC en llamadas a funciones ODBC. El controlador de tipo 1 también se denomina controlador universal porque puede utilizarse para conectarse a cualquiera de las bases de datos.

- Como se utiliza un controlador común para interactuar con diferentes bases de datos, los datos transferidos a través de este controlador no están tan protegidos.
- Es necesario instalar el controlador de puente ODBC en las máquinas cliente individuales.
- El controlador de tipo 1 no está escrito en Java, por lo que no es un controlador portátil.
- Este controlador está integrado en el JDK, por lo que no es necesario instalarlo por separado.
- Es un controlador independiente de la base de datos.

Controladores JDBC (Driver)

Controlador de tipo 2 o controlador Native-API

El controlador Native API utiliza las bibliotecas del lado del cliente de la base de datos. Este controlador convierte las llamadas a métodos JDBC en llamadas nativas a la API de la base de datos. Con el fin de interactuar con diferentes bases de datos, este controlador necesita su API local, es por eso que la transferencia de datos es mucho más segura en comparación con el controlador de tipo-1.

- El controlador debe instalarse por separado en cada uno de los equipos cliente.
- La librería del cliente Vendor necesita ser instalada en la máquina del cliente.
- El controlador de tipo 2 no está escrito en Java, por lo que no es un controlador portátil.
- Es un controlador dependiente de la base de datos.

Controladores JDBC (Driver)

Controlador de tipo 3 o controlador de protocolo de red

El controlador de protocolo de red utiliza middleware (servidor de aplicaciones) que convierte las llamadas JDBC directa o indirectamente en el protocolo de base de datos específico del proveedor. En este caso, todos los controladores de conectividad de bases de datos están presentes en un único servidor, por lo que no es necesario instalarlos individualmente en el cliente.

- Los controladores de tipo 3 están totalmente escritos en Java, por lo que son portátiles.
- No se requiere ninguna biblioteca del lado del cliente debido a que el servidor de aplicaciones puede realizar muchas tareas como auditoría, equilibrio de carga, registro, etc.
- Se requiere soporte de red en la máquina cliente.
- El mantenimiento del controlador de protocolo de red resulta costoso porque requiere la codificación específica de la base de datos en el nivel intermedio.
- Facilidad de conmutación para cambiar de una base de datos a otra.

Controladores JDBC (Driver)

Controlador de tipo 4 o controlador Thin

El controlador de tipo 4 también se denomina controlador de protocolo nativo. Este controlador interactúa directamente con la base de datos. No requiere ninguna librería nativa de base de datos, por eso también se le conoce como Thin Driver.

No requiere ninguna biblioteca nativa ni servidor Middleware, por lo que no requiere instalación del lado del cliente ni del lado del servidor.

Está totalmente escrito en lenguaje Java, por lo que son drivers portables.

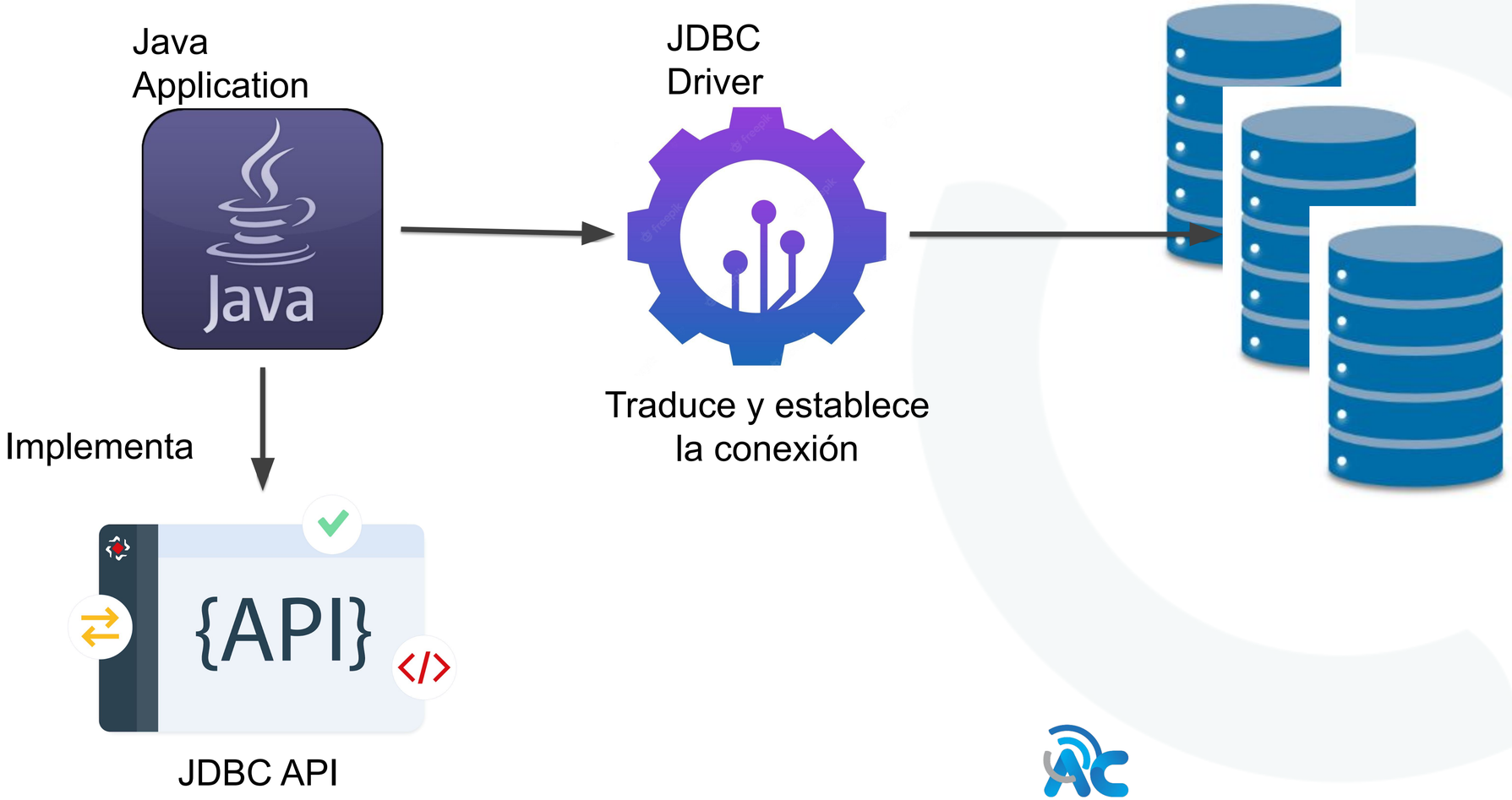
Controladores JDBC (Driver)

¿Qué controlador utilizar y cuándo?

- Si está accediendo a un tipo de base de datos, como Oracle, Sybase o IBM, el tipo de controlador preferido es el tipo-4.
- Si su aplicación Java está accediendo a varios tipos de bases de datos al mismo tiempo, el controlador preferido es el de tipo 3.
- Los controladores de tipo 2 son útiles en situaciones en las que un controlador de tipo 3 o 4 aún no está disponible para su base de datos.
- El controlador de tipo 1 no se considera un controlador de nivel de despliegue, y normalmente se utiliza sólo para fines de desarrollo y pruebas.

ACADEMIA CONSULTEC

Arquitectura JDBC



ACADEMIA CONSULTEC

Principales interfaces y clases JDBC

La versión actual de JDBC es la 4.3, es la versión estable desde el 21 de septiembre de 2017. Se basa en la interfaz de nivel de llamada X/Open SQL. Para invocar métodos de JDBC se necesita poder acceder a todos los paquetes Java (o parte de ellos) donde residen estos métodos.

Se puede hacer importando los paquetes o clases específicas, o bien utilizando los nombres de clase totalmente calificados, entre ellas se encuentran los siguientes paquetes:

- **java.sql:** Contiene la API básica de JDBC.
- **javax.naming:** Contiene clases e interfaces para Java Naming and Directory Interface (JNDI), que se suele utilizar para implementar una DataSource (fuente de datos).
- **javax.sql:** Contiene métodos para producir aplicaciones de servidor mediante Java.

Paquete java.sql

Haremos una revisión de las principales interfaces y clases bajo el paquete java.sql:

Class.forName(): Aquí cargamos el fichero de clases del driver en memoria en tiempo de ejecución. No es necesario usar new o crear un objeto.

DriverManager: Esta clase se utiliza para registrar el controlador para un tipo de base de datos específico y para establecer una conexión de base de datos con el servidor a través de su método getConnection().

Connection: Esta interfaz representa una conexión de base de datos establecida (sesión) a partir de la cual podemos crear sentencias para ejecutar consultas y recuperar resultados, obtener metadatos sobre la base de datos, cerrar la conexión, etc.

Paquete java.sql

Statement and PreparedStatement: estas interfaces se utilizan para ejecutar consultas SQL estáticas y consultas SQL parametrizadas, respectivamente. Statement es la superinterfaz de la interfaz PreparedStatement. Sus métodos más utilizados son:

- **boolean execute(String sql):** ejecuta una sentencia SQL general. Devuelve true si la consulta devuelve un ResultSet, false si la consulta devuelve un update count o no devuelve nada. Este método sólo puede utilizarse con una sentencia.
- **int executeUpdate(String sql):** ejecuta una sentencia INSERT, UPDATE o DELETE y devuelve una cuenta de actualización indicando el número de filas afectadas (por ejemplo, 1 fila insertada, o 2 filas actualizadas, o 0 filas afectadas).
- **ResultSet executeQuery(String sql):** ejecuta una sentencia SELECT y devuelve un objeto ResultSet que contiene los resultados devueltos por la consulta.
- **ResultSet:** contiene los datos de la tabla devueltos por una consulta SELECT. Utilice este objeto para iterar sobre las filas del conjunto de resultados utilizando el método next().
- **SQLException:** esta excepción comprobada está declarada para ser lanzada por todos los métodos anteriores, por lo que tenemos que atrapar esta excepción explícitamente al llamar a los métodos de las clases anteriores.

Conectividad JDBC

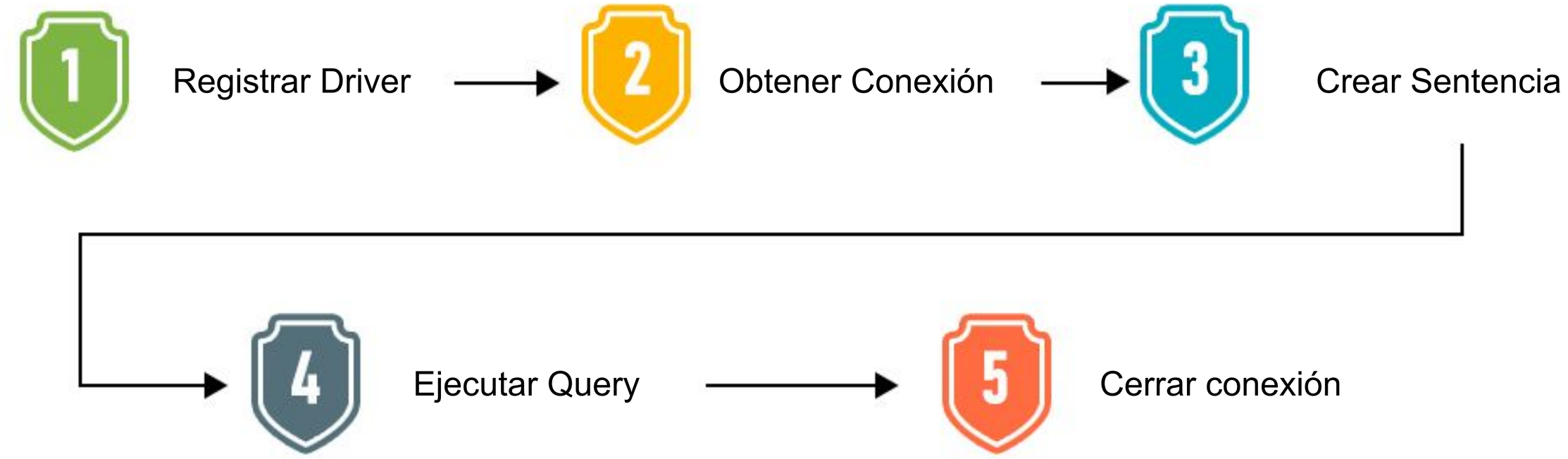
Para establecer la conexión hacia la base de datos se debe aplicar independientemente del Driver los siguientes pasos:

- Definir la clase que implementa el driver JDBC, por ejemplo: para postgresql -> *org.postgresql.Driver*;
- Definir el *string de conexión de la base de datos , básicamente es una cadena de caracteres con informaciones para conectar, entre ellas URL, HOST, PUERTO.

  Es importante mencionar que la manera de definir el string varía entre las diferentes bases de datos.
- Definir el nombre de usuario y contraseña para conectarnos a la base de datos.

ACADEMIA CONSULTEC

Conectividad JDBC



ACADEMIA CONSULTEC

Conectividad JDBC

```
// se definie la url con nombre del esquema, puerto y host
static final String url = "jdbc:mysql://localhost:3306/db";
static String usuario = "root";
static String clave = "1234";

try {
    // usamos MySQL driver
    Class.forName("com.mysql.jdbc.Driver");

    Connection conn = DriverManager.getConnection(url, usuario, clave);
    conn.close();
} catch (SQLException e) {
    System.out.println(e.getMessage());
}
```

Connection - Representa una sesión junto al banco de datos deseado. Vimos la clase en el trecho de código del ejemplo anterior y vamos a ejecutar las instrucciones SQL dentro de la conexión establecida.

ACADEMIA CONSULTEC

PreparedStatement interface

Es una subinterfaz de Statement y se utiliza para ejecutar consultas parametrizadas. Aunque funciona similar a Statement, esta interfaz mejora el rendimiento de la aplicación por tanto será más rápido si se utiliza la interfaz PreparedStatement porque la consulta se compila una sola vez.

Metodos de la interfaz PreparedStatement

Method	Description
public void setInt(int paramIndex, int value)	Asigna el valor entero al índice del parámetro dado
public void setString(int paramIndex, String value)	Asigna el valor String al índice del parámetro.
public void setFloat(int paramIndex, float value)	Asigna un valor flotante al índice del parámetro.
public void setDouble(int paramIndex, double value)	Asigna el valor double al índice del parámetro dado.
public int executeUpdate()	Ejecuta la consulta. Se utiliza para crear, eliminar, insertar, actualizar, borrar, etc.
public ResultSet executeQuery()	Ejecuta la consulta select. Devuelve una instancia de ResultSet.

ACADEMIA CONSULTEC

PreparedStatement interface

```
try{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

    PreparedStatement stmt = con.prepareStatement("INSERT INTO Emp VALUES(?,?)");
    stmt.setInt(1,101);//1 especifica el primer valor esperado acordo al ?
    stmt.setString(2,"Ratan");

    int i=stmt.executeUpdate();
    System.out.println(i+" registro almacenados");

    con.close();
}catch(Exception e){ System.out.println(e);}
```

ACADEMIA CONSULTEC

ResultSet interface

El objeto ResultSet mantiene un cursor que apunta a una fila de una tabla. Inicialmente, el cursor apunta a antes de la primera fila. Pero podemos hacer que este objeto se mueva hacia adelante y hacia atrás pasando TYPE_SCROLL_INSENSITIVE o TYPE_SCROLL_SENSITIVE en el método createStatement(int,int) así como podemos hacer que este objeto sea actualizable mediante:

Metodos de la interfaz ResultSet

public boolean next():	Se utiliza para mover el cursor a la fila siguiente desde la posición actual.
public boolean previous():	Se utiliza para mover el cursor a la fila anterior desde la posición actual.
public boolean first():	Se utiliza para mover el cursor a la primera fila del objeto conjunto de resultados.
public boolean last():	Se utiliza para mover el cursor a la última fila del objeto conjunto de resultados.
public boolean absolute(int row):	Se utiliza para mover el cursor al número de fila especificado en el objeto ResultSet.
public boolean relative(int row):	Se utiliza para mover el cursor al número de fila relativo en el objeto ResultSet, puede ser positivo o negativo.
public int getInt(int columnIndex):	Sirve para devolver los datos del índice de columna especificado de la fila actual en forma de int.
public int getInt(String columnName):	Devuelve como int el nombre de la columna especificada en la fila actual.
public String getString(int columnIndex):	Devuelve como cadena el índice de la columna especificada en la fila actual.
public String getString(String columnName):	Permite obtener como cadena el nombre de la columna especificada en la fila actual.

ResultSet interface

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
ResultSet rs=stmt.executeQuery(“SELECT * FROM emp765");

//obtengo el registro numero 3, por tanto el puntero se mueve a esta posicion
rs.absolute(3);
System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));

con.close();
```

Almuerzo

45 min

API REST con
JAX-RS

ACADEMIA
CONSULTEC

REST

Representational State Transfer (REST) es una interfaz basada en el estándar HTTP, que se emplee tanto para obtener datos como para realizar operaciones con ellos en un gran número de formatos, ya sean JSON, XML u otros.

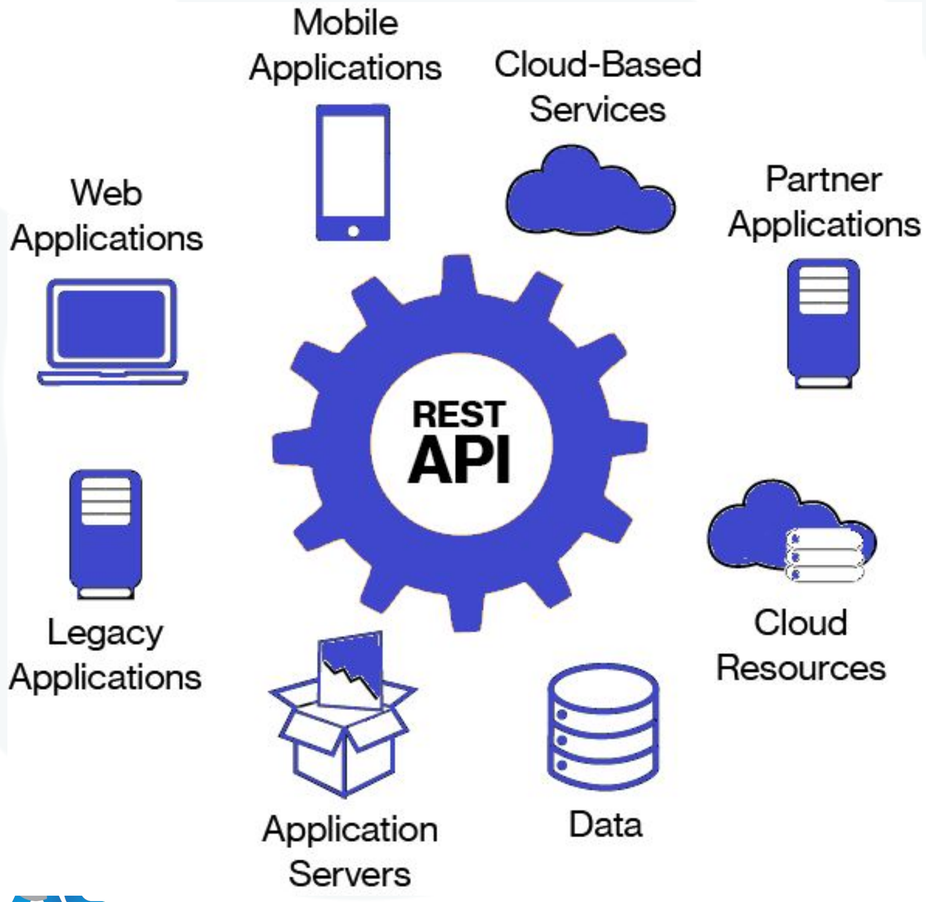
Las APIs REST son plantillas o firmas que aplican el estándar REST para representar el acceso a recursos u operaciones por medio de la red HTTP, definiendo su forma de acceder, forma de enviar como la forma de recibir la informacion que se estaria enviando a traves de ella.

ACADEMIA CONSULTEC

Principios de REST

Una implementación concreta de un servicio web REST sigue cuatro principios de diseño fundamentales:

- Utiliza los métodos HTTP de manera explícita
- No mantiene estado
- Expone URIs con forma de directorios
- Transfiere XML, JavaScript Object Notation (JSON), o ambos



Niveles de calidad

Cuando se desarrolla una aplicación web, y en especial una API, a la hora de utilizar REST se trabaja con dos niveles de calidad.

- **Uso correcto de URIs (Uniform Resource Identifier).** Las URL (Uniform Resource Locator) son las encargadas de que se pueda identificar el recurso, además de ayudar a su acceso y permitir que podamos compartir su ubicación. Por recurso entendemos una información (página, archivo, información en una sección) a la que queremos acceder, modificar o borrar. Los URIs deben mantener una jerarquía lógica, ser independientes del formato y ser únicos para cada recurso. A la hora de nombrarlos conviene evitar verbos. Hay que recordar que los URIs no son indicados para efectuar filtrados de información de un recurso.
- **Uso correcto de HTTP.** Las tres claves para desarrollar APIs REST son los métodos http, los códigos de estado y la aceptación de tipos de contenido. HTTP brinda los siguientes métodos para manipular los recursos: **GET** (consultar y leer recursos), **POST** (crear recursos), **PUT** (editar recursos), **DELETE** (eliminar recursos) y **PATCH** (editar partes concretas de un recurso).

Request / Response

Request

```
GET     /usuarios?nombre=Zim     HTTP/1.1
```

Response

```
GET     /usuarios?nombre=Zim     HTTP/1.1
Host:                            miservidor
Content-type:                    application/json

<usuario>
                    <nombre>Zim</nombre>
</usuario>
```

ACADEMIA CONSULTEC

JAX-RS

Se define como Java API for RESTful Web Services y es un API de Java que permite la creación de servicios web con REST (Representational State Transfer) Este API dispone de anotaciones dentro de Java SE 5 que sirve para simplificar el desarrollo y despliegue de los servicios clientes y servicios consumidores en la web.

Desde la versión 1.1 JAX-RS forma parte de manera oficial de Java EE 6. Lo interesante de ser parte de Java EE 6 es que no requiere configuraciones adicionales ya que pasa a ser nativo dentro del lenguaje de programación.

A la fecha se cuenta con la versión JAX-RS 2.0 que fue publicado en mayo de 2013 y que tiene como principales mejoras el uso de Hypermedia y ser una API de cliente común.

ACADEMIA CONSULTEC

Principales Anotaciones

JAX-RS tiene como principales anotaciones las ya conocidas etiquetas que sirven como el mecanismo para realizar las operaciones CRUD en las aplicaciones. Los llamados POJOs o clase java son mapeados como recursos a través de las siguientes anotaciones:

- **@Path** para la ruta de acceso relativa de una clase recurso o método.
- **@GET**, **@PUT**, **@POST**, **@DELETE** y **@HEAD** tipo de petición HTTP de un recurso a ejecutarse.
- **@Produces** para los tipos de medios MIME de respuesta.
- **@Consumes** para los tipos de medios de petición aceptados.

ACADEMIA CONSULTEC

Anotaciones para Cabeceras

Adicionalmente existen anotaciones para extraer información en la petición para buscar un valor requerido:

- **@QueryParam** enlaza el parámetro al valor de un parámetro de consulta HTTP.
- **@MatrixParam** enlaza el parámetro al valor de un parámetro de matriz de HTTP.
- **@HeaderParam** enlaza el parámetro a un valor de cabecera HTTP.
- **@CookieParam** enlaza el parámetro a un valor de cookie.
- **@FormParam** enlaza el parámetro a un valor de formulario.
- **@DefaultValue** especifica un valor por defecto para los enlaces anteriores cuando la clave no es encontrada.
- **@Context** devuelve todo el contexto del objeto. (Por ejemplo: @Context HttpServletRequest request)

Principales Implementaciones

Entre las implementaciones de JAX-RS se incluyen:

- **Apache CXF**, un framework de servicios web de código abierto.
- **Jersey**, la implementación de referencia de Sun (ahora Oracle).
- **RESTeasy**, implementación de JBoss.
- **Restlet**, creado por Jerome Louvel.
- **Apache Wink**, proyecto de Apache Software Foundation Incubator, el módulo del servidor implementa JAX-RS.
- **Jakarta RESTful Web Services**, es la nueva plataforma de Java Enterprise Edition o Java EE, por lo tanto de Java EE como extensión natural de este.

Implementación

```
@Path("/serviciosPersonas")
@Produces("application/json")
public class ServicioPersonas {
    private static List<Persona> listaPersonas =
                        new ArrayList<Persona>();

    public  ServicioPersonas(){
        super();
        Persona yo = new Persona("pedro", "perez");
        listaPersonas.add(yo);
    }

    @GET
    @Path("/personas")
    public List<Persona> getPersonas(){
        return listaPersonas;
    }
    @POST
    @Path("/personas")
    public void addPersonas(MultivaluedMap<String,String>  parametros){
        Persona p = new Persona(parametros.getFirst("nombre"),
parametros.getFirst("apellidos"));
        listaPersonas.add(p);
    }
}
```

ACADEMIA CONSULTEC

Gracias

¡Nos vemos pronto!