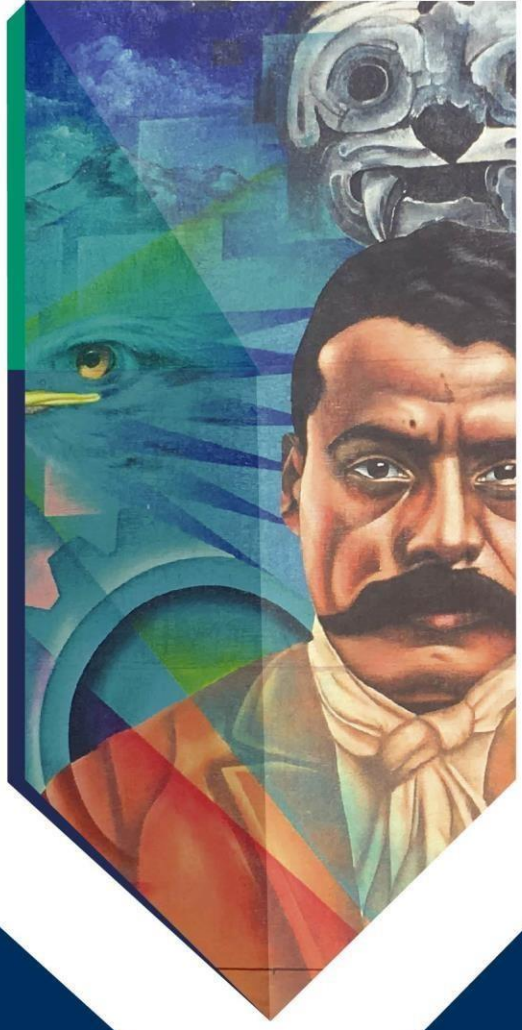
UNIVERSIDAD TECNOLÓGICA EMILIANO ZAPATA DEL ESTADO DE MORELOS



Investigación

Recuperación Unidad I

PRESENTA:

BRYAN MITCHEL MURGA ARCOS

UNIVERSIDAD TECNOLÓGICA EMILIANO ZAPATA DEL ESTADO DE MORELOS

EMILIANO ZAPATA, MOR. 26 DE SEPTIEMBRE DE 2021

División Académica de Tecnologías de la Información y Comunicación

Índice

Introducción ……. Pág. 3

Desarrollo

1. Servidor

1.1. JavaServer.java ……. Pág. 4

1.2. Handler.java (Create, Read, Update, Delete) ……. Pág. 5

2. Cliente

2.1. JavaClient.java (Create, Read, Update, Delete) …… Pág. 9

3. POM

3.1. pom.xml (dependencies) ..…… Pág. 14

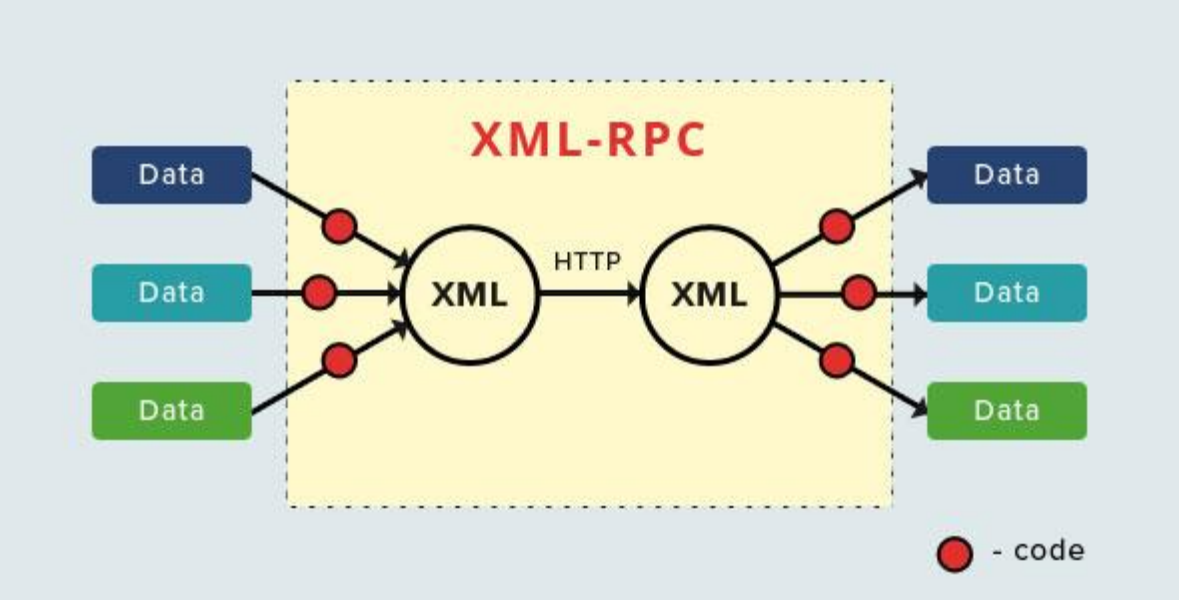
Conclusiones ……. Pág. 15

Bibliografías …… Pág. 16

Introducción

El objetivo de este reporte es dar a conocer el tipo de funcionamiento que lleva a cabo XML-RCP que es Mecanismo para llamar procedimientos o funciones disponibles en un equipo remoto.

XML-RCP utiliza el protocolo HTTP para pasar la información del cliente al servidor. Utiliza un vocabulario XML que describe la naturaleza del request y el response, haciendo que el cliente especifique el nombre del procedimiento a llamar y los parámetros que se enviaran para poder hacer distintos métodos que ayuden al funcionamiento del sistema utilizando el modelo cliente servidor.



1. Servidor

1.1. JavaServer.java

En la clase llamada JavaServer se crea el método main, dentro de este método se añaden dos posibles excepciones.

XmlRpcException: Esta lanza una excepción si falla la invocación del método remoto el cual se requiere utilizar. El error puede deberse a dos motivos: la invocación falló en el lado remoto (por ejemplo, se lanzó una excepción dentro del servidor) o la comunicación con el servidor falló. Este último se indica lanzando una instancia de XmlRpcClientException.

IOException: IOException (errores que no puede evitar el programador, generalmente relacionados con la entrada/salida del programa). Se utilizan en Java para capturar las excepciones que se hayan podido producir en el bloque de código delimitado por try y catch.

Al inicio del método se pone una simple impresión en consola para indicar que el servidor está iniciando, después se utiliza la clase PropertyHandlerMapping para iniciar una instancia llamada mapping que nos va a permitir mapear la clase Handler en la cual tendremos todos los métodos que utilizaremos.

Posteriormente se crear una instancia de la clase WebServer que nos ayudara a inicializar nuestro servidor de manera local con el puerto 1200, después se configura el server para que reciba la instancia creada anteriormente (mapping) y se inicializa el servidor.

```
public class JavaServer {
        public static void main(String[] args) throws IOException, XmlRpcException {
        System.out.println( "Iniciando servidor RPC…" );
        PropertyHandlerMapping mapping = new PropertyHandlerMapping();
        mapping.addHandler( pKey: "Handler", Handler.class );

        WebServer server = new WebServer( pPort: 1200);
        server.getXmlRpcServer().setHandlerMapping( mapping );
        server.start();
        System.out.println( "Esperando petición…" );
    }
}
```

1.2. Handler.java

Método create

El método es creado de manera pública (las demás clases pueden acceder al método) y es de tipo booleano ya que retorna un valor, dentro del método se reciben parámetros para asignar los valores que se ocuparan para hacer el registro de usuario, en este caso nuestro método se llama createUser.

```
public boolean updateUser(int id, String name, String lastname, String email, String password){
```

Posteriormente se usa la instrucción try-catch que nos permiten manejar bloques de código que puedan originar excepciones haciendo de esta manera que nuestro programa sepa que hacer en caso de un lanzamiento de un error.

En el bloque de la cláusula try se inicializa la conexión a la base de datos. Después se prepara la llamada de la sentencia que va a seguir nuestra base de datos. La sentencia que se ocupo para el registro de la base de datos es que inserte dentro de la tabla usuario en los atributos name, lastname, email, password, date_registered, status los valores los cuales nosotros le asignemos. En la siguiente parte los valores se establecen en cada valor de los parámetros que anteriormente declaramos al inicio del método donde corresponden siguiendo el orden de declaración. Al final se ejecuta la sentencia que fue declarada en la variable pstm.

```
try{
    con = ConnectionMysql.getConnection();
    pstm = con.prepareCall( s: "INSERT INTO `user`(name, lastname, email, password, date_registered, status)VALUES(?,?,?,?,
    pstm.setString( i: 1, name);
    pstm.setString( i: 2, lastname);
    pstm.setString( i: 3, email);
    pstm.setString( i: 4, password);

    flag = pstm.executeUpdate() == 1;
```

En la parte del bloque catch por si se origina alguna excepción se manda a imprimir en una variable y se informa cual fue la causa de la cual se originó el error utilizando la una excepción llamada SQLException por si ocurrió algún error por parte de la base de datos por ejemplo un error en el registro o un problema de conexión con la base de datos. Por último, en el bloque de finally se hace el cierre de la conexión de la base de datos.

```
}catch(SQLException e){
    System.out.println("Error" + e.getMessage());
}finally{
    closeConnection();
}
```

Método Read

El método read lo utilizamos para hacer la impresión de todos los usuarios que han sido registrados en la base de datos.

En el método utiliza una lista llamada List<User> en la cual se encuentran los atributos de cada usuario que ha sido registrado. Al inicio del bloque de este método se crea una instancia de la lista llamada listUser la cual hará que podamos acceder a ella desde este método, En el bloque de la clausula try primero se inicializa la conexión de la base de datos, después guardamos en la variable pstm la sentencia que ocupamos para mostrar todos los usuarios de la base de datos, en este caso se hace una selección de todos los atributos que contiene cada registro e indicamos que se encuentran en la tabla user.

```
public  List<User> findAll(){
    List<User> listUser = new ArrayList<>();
    boolean flag = false;

    try{
        con = ConnectionMysql.getConnection();
        pstm = con.prepareStatement( s: "Select user.id, user.name,  user.lastname, user.email, user.password, user.date_regist
```

Posteriormente se crea una instancia de nuestro objeto usuario la cual contiene las variables en las que podremos establecer todos los datos. Después se establecen los parámetros que recibimos de la base de datos y los asignamos en las variables dependiendo del orden especifico en las que fueron declarados en los parámetros. Al establecer todos los atributos de nuestro objeto usuario se añaden a la lista que fue creada anteriormente llamada listUser y finalmente de ejecuta la sentencia para realizarla en nuestra base de datos.

En la parte del bloque catch por si se origina alguna excepción se manda a imprimir en una variable y se informa cual fue la causa de la cual se originó el error utilizando la una excepción llamada SQLException por si ocurrió algún error por parte de la base de datos por ejemplo un error en el registro o un problema de conexión con la base de datos. Por último, en el bloque de finally se hace el cierre de la conexión de la base de datos.

Por último se retorna la lista de Usuarios para poder utilizarla mas tarde.

```
    User user = new User();
    user.setId(rs.getInt( s: "id"));
    user.setLastname(rs.getString( s: "lastname"));
    user.setEmail(rs.getString( s: "email"));
    user.setPassword(rs.getString( s: "password"));
    user.setDate_registered(rs.getString( s: "date_registered"));
    user.setStatus(rs.getInt( s: "status"));
    listUser.add(user) ;
    flag = pstm.executeUpdate() == 1;
}catch(SQLException e){
    System.out.println("Error" + e.getMessage());
}finally{
    closeConnection();
}

return listUser;
```

Método Update

El método update lo utilizamos para actualizar los atributos de nuestros usuarios en la base de datos.

En este caso nuestro metodo llamado updateUser recibe como parámetro todos los atributos de nuestro objeto User.En la parte del bloque de la cláusula try se empieza inicializando la conexión de la base de datos, después se prepara la llamada a la sentencia que ocuparemos en nuestra base de datos para poder hacer la actualización de nuestros usuarios la cual es guardada en la variable pstm. La sentencia que utilizamos es que actualice todos los atributos de nuestro usuario y le establezca los valores que nosotros queremos modificar donde la id del usuario del que queremos modificar sus atributos.

```
public boolean updateUser(int id, String name, String lastname, String email, String password){
    boolean flag = false;
    try{
        con = ConnectionMysql.getConnection();
        pstm = con.prepareCall( s: "UPDATE `user` SET name = ?, lastname = ?, email = ?, password = ? WHERE id = ?;");
```

Posteriormente se establecen los atributos de nuestro objeto usuario tomando los parámetros que fueren declarados en el método siguiendo el mismo orden de esta declaración que previamente a sido declarada. Por ultimo se hace la ejecución de la sentencia que usaremos en la base de datos para actualizar los atributos de nuestro usuario especificado por id.

En la parte del bloque catch por si se origina alguna excepción se manda a imprimir en una variable y se informa cual fue la causa de la cual se originó el error utilizando la una excepción llamada SQLException por si ocurrió algún error por parte de la base de datos por ejemplo un error en el registro o un problema de conexión con la base de datos. Por último, en el bloque de finally se hace el cierre de la conexión de la base de datos.

```
        pstm.setString( i: 1, name);
        pstm.setString( i: 2, lastname);
        pstm.setString( i: 3, email);
        pstm.setString( i: 4, password);

        flag = pstm.executeUpdate() == 1;
    }catch(SQLException e){
        System.out.println("Error" + e.getMessage());
    }finally{
        closeConnection();
    }
```

Método Delete

El método delete lo utilizamos para poder eliminar el registro de nuestra base de datos especificando la id del usuario.

En este caso nuestro método es llamada deleteUser el cual recibe el parámetro de la id del usuario el cual queremos eliminar su registro de la base de datos. En la pare del bloque de código de la cláusula del try inicializamos la conexión a la base de datos y después preparamos la sentencia la cual sirve para hacer la eliminación de nuestro registro especifico, la cual elimina de la tabla usuario cuyo registro contenga el id específico del registro que deseamos eliminar de nuestra base de datos. Posteriormente se establece el atributo de nuestra instancia usuario la cual es asignada dependiendo del parámetro que nosotros especificamos. Por último ejecuta la sentencia la cual queremos que utilice en nuestra base de datos.

```
public boolean deleteUser(int id){
    boolean flag = false;
    try{
        con = ConnectionMysql.getConnection();
        pstm = con.prepareCall( s: "DELETE FROM `user` WHERE id = ?;");
        pstm.setInt( i: 1, id);

        flag = pstm.executeUpdate() == 1;
```

En la parte del bloque catch por si se origina alguna excepción se manda a imprimir en una variable y se informa cual fue la causa de la cual se originó el error utilizando la una excepción llamada SQLException por si ocurrió algún error por parte de la base de datos por ejemplo un error en el registro o un problema de conexión con la base de datos. Por último, en el bloque de finally se hace el cierre de la conexión de la base de datos.

```
}catch(SQLException e){
    System.out.println("Error" + e.getMessage());
}finally{
    closeConnection();
}
```

2. Cliente

2.1. JavaClient.java

Esta en la clase llamada JavaClient va a ser la que va a consumir el servidor y va a hacer las peticiones correspondientes a lo que se le pidió. Dentro encontramos el metodo main el cual contiene dos excepciones,MalformedURLException, XmlRpcException por si se genera un error y se sepa el origen de este. Posteriormente se crea una instancia de la clase XmlRpcClientConfigImpl llamada config la cual nos podrá ayuda a hacer una configuración del cliente en la cual establecemos la url que queremos que utilice y tenemos que pasárselo como una instancia de URL. La cual sigue la ruta de nuestro localhost que utiliza el puerto 1200. Este puerto debe coincidir con nuestro puerto anteriormente configurado en la clase JavaClient si este no coincide entonces no será posible que este lo encuentre de una manera satisfactoria. Posteriormente se crea una instancia de la clase XmlRpcClient la cual establecerá la configuración que sea determinada por otro metodo llamado read que crearemos mas adelante. Por ultimo se imprime lo que se estableció en el método read que hará las instrucciones que nosotros le demos.

```
public class JavaClient {
    public static void main(String[] args) throws MalformedURLException, XmlRpcException {
        XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
        config.setServerURL(new URL( s: "http://localhost:1200"));
        XmlRpcClient client = new XmlRpcClient();
        client.setConfig(config);
```

Se crea un método llamado read el cual será el que determine cual de nuestros métodos utilice para el CRUD de nuestra base de datos. Al inicio del bloque de código se imprime que se está inicializando el sistema y se despliega una serie de opciones de menú la cual corresponden al Registro de un usuario, Ver todos los usuarios, Actualizar los atributos de un usuario y Eliminar del registro de la base de datos a un usuario, esta opción es guardada en una variable llamada opc e indicara a la cláusula switch el caso que corresponda.

```
public static String read(XmlRpcClient client) throws XmlRpcException{
    Scanner sc = new Scanner(System.in);
    String res = "";

    System.out.println("Iniciando mi sistema");
    System.out.println("Menú\n1.- Registrar Usuario\n2.- Ver todos\n3.- Actualizar\n4.- Eliminar");
    int opc = sc.nextInt();
```

Opción crear usuario (Create)

En la primera opción se imprime la instrucción la cual se debe introducir mediante consola los atributos que se desean asignar al usuario el cual queremos crear, estos son leídos en variables y guardados.

Posteriormente se crea una instancia de un arreglo el cual usaremos para poder enviar todos los parámetros que queremos utilizar para la creación de nuestro usuario, debido a que nuestro servidor no puede recibir los parámetros uno por uno y necesita el uso de un arreglo.

Después se crea una variable de tipo booleano llamada result, esta será ejecutada y mandara a iniciar el metodo createUser que se encuentra dentro de la clase Handler y se le serán enviados los parámetros.

Finalmente, a una variable de tipo String llamada res el cual mediante una sentencia if dará a conocer si la acción se ha completado de manera satisfactoria o no, guardando el texto en la variable y por último retornar la variable res para conocer el resultado.

```
case 1:
    // Registrar
    System.out.println("Ingresa tu nombre");
    String name = sc.next();
    System.out.println("Ingresa tu apellido");
    String lastname = sc.next();
    System.out.println("Ingresa tu correo");
    String email = sc.next();
    System.out.println("Ingresa tu password");
    String password = sc.next();

    Object[] params1 = { name, lastname, email, password };
    boolean result = (Boolean) client.execute( pMethodName: "Handler.createUser" , params1);
    res = result ? "Se ha registrado correctamente" : "No se ha registrado correctamente";
    return res;
```

Opción mostrar todos los usuarios (Read)

En la opción de Consultar todos los usuarios se crea la instancia de handler para poder acceder al método de findAll.

Se utiliza un ciclo for each que ciclara a todos los usuarios que estén registrados en nuestra base de datos. Después se mandan a llamar las variables de la instancia de usuario y así imprimiendo cada uno de los atributos de los usuarios registrados.

Posteriormente en la parte de el atributo del estatus se verifica en que estado se encuentra el usuario haciendo uso de valores numéricos, en caso de que el usuario tenga un status activo imprime un mensaje de que el usuario se encuentra en este estado y en caso de que no también se informara. Finalmente se asigna un valor a la variable res para que haga su impresión en pantalla de que el proceso ha sido satisfactorio.

```
case 2:
    // Ver todos
    System.out.println("Consultar todos");
    Handler handler =  new Handler();
    for(User user : handler.findAll() ){
        System.out.println("================================");
        System.out.println("Id: "+user.getId());
        System.out.println("Nombre: "+user.getName());
        System.out.println("Apellido: "+user.getLastname());
        System.out.println("Email: "+user.getEmail());
        System.out.println("Contraseña: "+user.getPassword());
        System.out.println("Fecha de registro: "+user.getDate_registered());
        if(user.getStatus()==1){
            System.out.println("Status: Activo");
        }else{
            System.out.println("Status: Inactivo");
        }
        System.out.println("================================");
    }
    res = "Se ha listado correctamente";
    break;
```

Opción actualizar los usuarios (Update)

Es la opción que se utiliza para actualizar los atributos de los registros de un usuario especifico de nuestra base de datos.

Al principio del bloque de la opción comienza pidiendo todos los atributos los cuales deseas que se modifiquen de tu usuario y guardando estos en una variable por separado para poder ser enviados y que se efectué la actualización.

Después se crea una variable de tipo booleano llamada result, esta será ejecutada y mandara a iniciar el metodo updateUser que se encuentra dentro de la clase Handler y se le serán enviados los parámetros.

Finalmente, a una variable de tipo String llamada res el cual mediante una sentencia if dará a conocer si la acción se ha completado de manera satisfactoria o no, guardando el texto en la variable y por último retornar la variable res para conocer el resultado.

```
case 3:
    // Actualizar
    System.out.println("Ingresa tu nombre");
    String name2 = sc.next();
    System.out.println("Ingresa tu apellido");
    String lastname2 = sc.next();
    System.out.println("Ingresa tu correo");
    String email2 = sc.next();
    System.out.println("Ingresa tu password");
    String password2 = sc.next();
    System.out.println("Ingresa el ID");
    int id = sc.nextInt();

    Object[] params2 = { id, name2, lastname2, email2, password2 };
    boolean result2 = (Boolean) client.execute( pMethodName: "Handler.updateUser" , params2);
    res = result2 ? "Se ha actualizado correctamente" : "No se ha actualizado correctamente";
    return res;
```

Opción borrar un usuario (Delete)

Es la opción en la cual sirve para poder eliminar un registro de usuario de nuestra base de datos de manera específica.

Primero imprime la instrucción de que id de usuario especifica deseamos eliminar de nuestro registro, después esta id es guardada en una variable.

Después se crea una variable de tipo booleano llamada result, esta será ejecutada y mandara a iniciar el metodo deleteUser que se encuentra dentro de la clase Handler y se le serán enviados los parámetros.

Finalmente, a una variable de tipo String llamada res el cual mediante una sentencia if dará a conocer si la acción se ha completado de manera satisfactoria o no, guardando el texto en la variable y por último retornar la variable res para conocer el resultado.

```
case 4:
    // Eliminar
    System.out.println("Ingresa el ID");
    int id2 = sc.nextInt();

    Object[] params3 = { id2 };
    boolean result3 = (Boolean) client.execute( pMethodName: "Handler.deleteUser" , params3);
    res = result3 ? "Se ha eliminado correctamente" : "No se ha eliminado correctamente";
    return res;
```

3. POM

3.1. pom.xml (dependencies)

MYSQL

Dependencia de Mysql la dependencia de mysql se utiliza como el conector de nuestra base de datos con java la cual se utiliza un tipo de versión y se encarga de compila, empaqueta y ejecuta los test. Mediante plugins para el buen funcionamiento de nuestra conexión a una base de datos. Esta se agrega en un apartado de tipo xml llamado pom de manejar las dependencias del proyecto.

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.25</version>
</dependency>
```

XMLRCP

Es un mecanismo para llamar procedimientos o funciones disponibles en un equipo remoto. Permite definir interfaces que se llaman a través de la red y pueden ser una función o una API compleja.

Características:

• Utiliza el protocolo HTTP para pasar la información del cliente al servidor.

• Utiliza un vocabulario XML que describe la naturaleza del request y el response.

• El cliente especifica el nombre del procedimiento a llamar y los parámetros que se enviarán.

• El servidor regresa una respuesta XML o un error.

• Los parámetros son listas de tipos y contenidos.

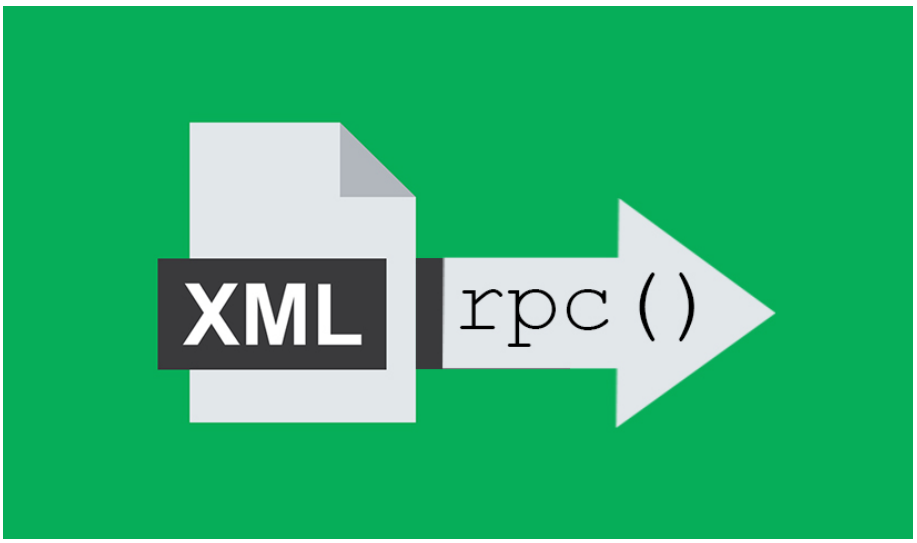
• No conoce objetos fuera del vocabulario XML.

Conclusión

Los clientes que quieren acceder a XML-RPC utilizan el protocolo de transferencia HTTP o, más concretamente, el método POST de petición HTTP. Después de recibir la petición HTTP.

El servidor evalúa el documento XML que se encuentra en el cuerpo de la solicitud. A partir de su contenido, genera, por ejemplo, los parámetros para la función deseada y la ejecuta. Como resultado, el servidor lo vuelve a empaquetar en un documento XML que se devolverá al cliente en forma de respuesta HTTP. XML-RPC.

Creando así una buena y fácil estructura de intercambio de información para llamar procedimientos o funciones disponibles en un equipo remoto.



Bibliografía

https://elvex.ugr.es/decsai/java/pdf/B2-excepciones.pdf

https://www.google.com/search?q=MalformedURLException+que+es&oq=MalformedURLException+que+es&aqs=chrome..69i57.1775j0j7&sourceid=chrome&ie=UTF-8

https://ws.apache.org/xmlrpc/apidocs/org/apache/xmlrpc/XmlRpcException.html

https://www.genbeta.com/desarrollo/introduccion-a-maven#:~:text=Maven%20maneja%20las%20dependencias%20del,el%20manejo%20de%20las%20dependencias.

https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/que-es-xml-rpc/