

Aplicación cliente-servidor con estado, XML y JSON

Descripción

En esta práctica se pide el diseño e implementación de una aplicación cliente-servidor con estado para proporcionar un servicio calculadora. El protocolo se implementa sobre sockets TCP y en una arquitectura multi-hilo. Se parte de la base de la práctica anterior y en esta versión se modifica el protocolo para que los mensajes vayan codificados en XML o en JSON, a elección del cliente.

Diseño

En esta aplicación, cuando un cliente se conecta al servidor tiene la opción de mandar las peticiones codificadas de dos formas distintas: bien con XML o bien con JSON. Para soportar esta funcionalidad, se ha necesitado añadir sobre el diseño anterior el soporte para estos formatos. Adicionalmente, para facilitar el desarrollo se ha creado la clase *Calculator* que representa una petición hecha por un usuario y con un resultado.

- JSON. Para tratar este tipo de mensajes nos ayudamos de la librería open source de Google llamada GSON, en este caso la versión 2.8.2. Para utilizarla, con la clase llamada *JsonParser* se transforman los mensajes JSON en objetos del tipo *Calculator*.
- XML. SAX es el analizador de XML elegido para esta aplicación, su funcionamiento es equivalente a GSON, puesto que es capaz de transformar un mensaje XML en un objeto *Calculator* haciendo uso de un handler personalizado que se encuentra en la clase *XmlSaxHandler*.

Protocolo

El protocolo consta de un único mensaje que sirve tanto para la petición como para la respuesta, esto es debido a que cada parte lo interpreta de una forma o de otra. El esquema que sigue este mensaje es el siguiente. En primer lugar, se encuentra la longitud del mensaje seguida de un final de línea (/n). A continuación, aparece el formato en el que se está mandando el mensaje (xml o json). Finalmente, y separado de lo anterior por el carácter "~", se sitúa el mensaje propiamente dicho. De esta manera, el formato del mensaje quedaría:

LongitudMensaje

TipoMensaje~Mensaje

Sabiendo esto, queda ver cómo están formados los mensajes JSON y XML, que como se puede ver en los esquemas inferiores, simplemente representan un objeto *Calculator*.

```
{
  "user": "String",
  "operand1": "String",
  "operator": "String",
  "operand2": "String",
  "result": "String"
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<calculator>
  <user>String</user>
  <operand1>String</operand1>
  <operator>String</operator>
  <operand2>String</operand2>
  <result>String</result>
</calculator>
```

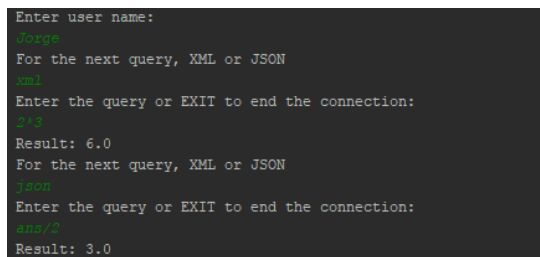
Finalmente, se ha añadido el siguiente XML Schema para validar los mensajes XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="calculator">
    <xs:sequence>
      <xs:element name="operand1" type="xs:string" minOccurs="0"/>
      <xs:element name="operand2" type="xs:string" minOccurs="0"/>
      <xs:element name="operator" type="xs:string" minOccurs="0"/>
      <xs:element name="result" type="xs:string" minOccurs="0"/>
      <xs:element name="user" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Ejemplos de uso

Como ya se mostró en la práctica anterior el funcionamiento tanto de la concurrencia como del servicio de persistencia, en este ejemplo nos centraremos en el formato de los mensajes.

A continuación, se ve como el cliente Jorge realiza dos peticiones, la primera en XML y la segunda en JSON.



```
Enter user name:
Jorge
For the next query, XML or JSON
xml
Enter the query or EXIT to end the connection:
2 * 3
Result: 6.0
For the next query, XML or JSON
json
Enter the query or EXIT to end the connection:
3.0
Result: 3.0
```

Ilustración 1: ejemplo de uso en el cliente Jorge

Como comprobamos, los resultados son correctos y dentro del directorio *files* del proyecto, podemos ver los mensajes generados por la aplicación. En primer lugar, el correspondiente a la primera consulta en XML:

```
<?xml version="1.0" encoding="utf-8"?>
<calculator>
  <user>Jorge</user>
  <operand1>2</operand1>
  <operator>*</operator>
  <operand2>3</operand2>
  <result>6.0</result>
</calculator>
```

En segundo lugar, el generado en JSON:

```
{
  "user": "Jorge",
  "operand1": "2",
  "operator": "*",
  "operand2": "3",
  "result": "6.0"
}
```

Es importante destacar que estos son los mensajes resultantes de la respuesta a la petición, cuando esta última es enviada sin conocer el resultado todavía, en el campo *result* encontramos el valor *null*.