RESEARCH ARTICLE OPEN ACCESS

Filling the database with application of protocol buffers

Bulat Umurzakov*, Naila Maksutkaliyeva**
*(Department of Computer Systems, Software engineering and Telecommunications, International Information Technology University, Almaty , Kazakhstan)
** (Department of Computer Systems, Software engineering and Telecommunications, International Information Technology University, Almaty , Kazakhstan)

----------**********----------

Abstract:

The results of research, submitted in the report, are the approaches for data delivery to databases using XML and protocol buffers. Comparisons of two methods and their advantages and shortcomings are given. It is shown that data exchange between various DBMSs goes much quicker when using protocol buffers.

Keywords — **Protocol Buffers, serializing, parsing, database;**

----------**********----------

Introduction :

In this paper we would like to present an application with a relatively unknown format pioneered by Google, called Protocol Buffers[1]. Nowdays a number of data formats for submission of network messages between servers are used. This is mentioned on ru.wikipedia.org/wiki/XML [2]. Data formats can be structured. If there is an exchange of large volumes of data, how can we increase the speed of such processes? As it is noted on ru.wikipedia.org/wiki/ Protocol Buffers [2], XML is ineffective for this purpose.

The Protocol_Buffers developed by Google allows us to define simple structures of data in a special language and then brings them together to create classes to represent these structures in the language

The Protocol_Buffers developed by Google allows us to define simple structures of data in a special language and then brings them together to create classes to represent these structures in the language chosen by the developer. The optimized code is attached to these classes to sort and transform the data into a compact, consecutive form of the message. In short, Protocol Buffers is a compact way of coding data in a binary format which allows us to define the simple structure of data and then to compile them to appropriate classes for representation of these structures in the necessary language (Java, Python or C ++). According to statements by Google, in comparison with XML, Protocol Buffers ("ProtoBuf") is much more simple and much quicker. However, we do not want to say that Protocol Buffers is always a better choice than XML. For example, Protocol Buffers cannot do a good job of simulating a text-based document, HTML for example.

In this work, transfer of ProtoBuf format data to the database is carried out. The same trans-formation was carried out from XML format to the database . Postgresql and MySQL DBMSs were used on servers. Generally the C++ and Java languages were applied. The results include a comparison of the two approaches and their advantages and shortcomings are considered.

II. Application of C++ ProtoBuf classes

At the beginning we will apply the C++ programming language. We will consider library.proto file displaying structure of one table "member' of a database of library "library":

```
message Member {
required string Fname = 1;
required int32 id = 2;
optional string email = 3;
}
```

After defining our messages, we can run the protocol buffer compiler for the application's language on library.`proto` file to generate data access classes. These classes provide accessors for each field as well as methods to serialize and parse the structure. The command for compiling is :

```
protoc -I=$SRC_DIR --
cpp_out=$DST_DIR
$SRC_DIR/library.proto
```

where $SRC_DIR is a source directory for the application's source code , $DST_DIR is a destination directory for the generated code.

Each chosen language, running the compiler on the above example, will generate a class called Member. Then this class can be used for the application to populate, serialize, and retrieve Member protocol buffer messages. Then the writing code can look like this:

```
Member member;
member.set_Fname("Tom Jakson");
member.set_id(125);

member.set_email("tom@example.com");
fstream output("dbdata", ios::out |
ios::binary);
member.SerializeToOstream(&output);
```

At the next step, the code reads the file created by the previous application:

```
fstream input("dbdata", ios::in |
ios::binary);
Member member;
member.ParseFromIstream(&input);
int id= member.id();
string Fname = member.Fname();
string email = member.email();
EXEC SQL CONNECT library
IDENTIFIED BY password;
EXEC SQL SET TRANSACTION;.
EXEC SQL INSERT INTO member
(mem_id, memname, mem_email)
VALUES ('id', 'Fname', 'email');
EXEC SQL COMMIT;
EXEC SQL DISCONNECT;
```

III. Application of Java ProtoBuf classes

At application of the Java language library.proto file is:

```
package lbrary;
option java_package =
"com.libproj.library";
option java_outer_classname =
"LibraryProtos";
message Member {
required string Fname = 1;
required int32 id = 2;
optional string email = 3;
}
```

Compiler for this case is:

```
protoc -I=$SRC_DIR --
java_out=$DST_DIR
$SRC_DIR/library.proto
```

Because we use Java classes , the `java_out` option will be provided, similar to the other supported languages. This generates

`com/libproj/library/LibraryProtos.java`

in a specified destination directory.

In order to show classes we present the part of `LibraryProtos.java`:

```
// Generated by the protobuf compiler.
Source: library.proto
package com.libproj.book;
public final class LibraryProtos {
  private LibraryProtos() {}
  public static void registerAllExtensions(

com.google.protobuf.ExtensionRegistry
registry) {
  }
  public interface MemberOrBuilder
     extends
com.google.protobuf.MessageOrBuilder {

    // required string name = 1;
    /**
     * <code>required string name =
1;</code>
     */
    boolean hasName();
    /**
     * <code>required string name =
1;</code>
     */
    java.lang.String getName();
    /**
     * <code>required string name =
1;</code>
     */
    com.google.protobuf.ByteString
       getNameBytes();

    // required int32 id = 2;
    /**
     * <code>required int32 id = 2;</code>
     */
    boolean hasId();
    /**
     * <code>required int32 id = 2;</code>
     */
    int getId();

    // optional string email = 3;
    /**
     * <code>optional string email =
3;</code>
     */
    boolean hasEmail();
    /**
     * <code>optional string email =
3;</code>
     */
    java.lang.String getEmail();
    /**
     * <code>optional string email =
3;</code>
     */
    com.google.protobuf.ByteString
       getEmailBytes();
  } …….
```

Next the application will use these classes to populate, serialize and retrieve Member protocol buffer messages.

For writing a message it is necessary to use these packages:

```
import
com.libproj.book.LibraryProtos.Library;
import
com.libproj.book.LibraryProtos.Member;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.PrintStream;
```

Then the application contains code for input variables id , name and email and serialization. At the next step another application reads the file created by the above example and saves all the information to the database.

So, connection was successfully established. Further, the task of getting data from stream and sending it to the database in the program code was decided.Java class that implements getting of connection with database and sending data to it called “ConnectDB”. ConnectDB had taken as child class of AddPerson class,that realizes serialization of data. In the main method

constructor for AddPerson class was created. In try-catch statement we call forName() method that establishes PostgreSQL driver for JAVA.

Then , connection called myConnection was created. To send database, information from fields of messages new variables were created for each field and equate it with values taken from getter methods. With help of statement.executeQuery() sql-requests were written and sent to the database. A continuance of database connection code was created:

```
import
com.libproj.book.LibraryProtos.Library;
import
com.libproj.book.LibraryProtos.Member;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.PrintStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
class ListMember {
  // Iterates though all people in the Library and
presents info about them.
  static void Print(Library library) {
    for (Member member:
library.getMemberList()) {
      Int
id=AddPerson.PromptForAddress(null,nul
l).member.getId();
      String name=
AddPerson.PromptForAddress(null,null).
member.getName();
      if (member.hasEmail()) {
        String email=
AddPerson.PromptForAddress(null,null).
member.getEmail();
      }
      }
  }
  public static void main(String[] args) throws
Exception {
    if (args.length != 1) {
      System.err.println("Usage:  ListPeople
LIBRARY_FILE");
      System.exit(-1);
    }
      Library library =
      Library.parseFrom(new
FileInputStream(args[0]));
    Print(library);
   try
      {
       String URL =
"jdbc:postgresql://ipaddress:5433/library";

Class.forName("org.postgresql.Driver");
          Connection conn =
DriverManager.getConnection( "URL",
"username", "password" );
          Statement stmt =
conn.createStatement();
                int nrows =
stmt.executeUpdate("INSERT INTO
member VALUES ( 'id', 'name','email'");
// TODO code application logic here
      }
      catch( Exception e)
      {
        System.err.println( e.toString() );
      }
  }
}
```

Applying XML to the same databases showed that the processes of converting XML data to a database was several times slower. Thus it is possible to draw the conclusion use of the protobuf protocol is considerably more efficient for data interchange between various DBMSs.

ACKNOWLEDGMENT

We express our sincere gratitude to the management of International Information Technology University, for providing us opportunities and their whole hearted support for such activities.

REFERENCES

[1] The Google website. [Online]. Available: https://developers.google.com/protocol-buffers/docs/downloads http://www.ieee.org/

[2] [Online]. Available: ru.wikipedia.org/wiki http

[3] The Google website. [Online]. Available: https://developers.google.com/protocol-buffers/docs/cpptutorial

[4] The Google website. [Online]. Available: https://developers.google.com/protocol-buffers/docs/javatutorial