Returning BLOB From Embedded Java

- Waiter! There's a BLOB in my Java!

by Przemysław Kruglej
(11-2013)

przemyslawkruglej.com
przemyslaw.kruglej@gmail.com

Table of Contents

1 Returning a BLOB from a Java Method Embedded in a Database

You're about to learn your future

Oracle supports embedding Java classes in its database. Different SQL types are mapped to corresponding Java classes to allow us to make the most of this feature. Author of the following question on StackOverflow:

http://stackoverflow.com/questions/20072410/create-java-sql-blob-instance-in-java-stored-procedure

had an issue with returning a `BLOB` object from Java method back to PL/SQL context. In the beginning, I didn't even think you could return a new `BLOB` object from an embedded class. Fortunately, there was a `BLOB` in my Java, too.

1.1 Around the fence approach

Before I show you an example of how you can achieve the goal of returning new `BLOB` from Java, let me present another solution first.

Instead of creating the `BLOB` itself in Java method, we could pass a temporary `OUT BLOB` as an argument to the method and operate on it there. Any changes made to the `BLOB` in the Java method will be reflected in the temporary `BLOB` created in PL/SQL context and passed from there to Java.

A simple Java class doing just that could look like this:

```
import oracle.sql.BLOB;
import java.io.OutputStream;

public class FortuneBLOB {
  public static void getFortuneBLOB(String secret, BLOB[] fortuneBLOB)
                                                    throws Exception {
    OutputStream fortuneScribe = fortuneBLOB[0].setBinaryStream(0);
    fortuneScribe.write(secret.getBytes());
    fortuneScribe.flush();
  }
}
```

We also need a corresponding *wrapper* procedure in PL/SQL:

```
CREATE OR REPLACE PROCEDURE getOutFortuneBLOB(
                               in_secret IN VARCHAR2,
                               out_fortune_blob IN OUT NOCOPY BLOB)
  AS LANGUAGE JAVA NAME
    'FortuneBLOB.getFortuneBLOB(java.lang.String, oracle.sql.BLOB[])';
```

There are a few things to point out here:

- If we want to pass an `OUT` argument to Java method, it must be declared as an `IN OUT` parameter. I've also added the `NOCOPY` *hint* so that the parameter is passed by reference, not by value.
- Furthermore, if this is going to work, the Java method must take an **array** parameter of the type of object passed as an `OUT` parameter.
- Java argument is an array, as mentioned above, but the corresponding PL/SQL parameter is not.
- To modify passed `BLOB`, we reference the first element of `fortuneBLOB` array parameter.

We can test if this works with the following code:

```
DECLARE
  l_blob BLOB;
BEGIN
  DBMS_LOB.CreateTemporary(l_blob, TRUE);
  getOutFortuneBLOB('You will crash another Fortune BLOB today!', l_blob);

  crash_fortune_blob(l_blob);
END;
```

We can see the following output, proving that the a fortune was inserted into the `BLOB`:

```
You will crash another Fortune BLOB today!
```

If you wonder what the `crash_fortune_blob` is, it's just a simple procedure for tests which converts a `BLOB` parameter to a string and prints it to the standard output:

```
CREATE OR REPLACE PROCEDURE crash_fortune_blob(in_blob IN BLOB)
AS
BEGIN
  dbms_output.put_line(UTL_RAW.CAST_TO_VARCHAR2(in_blob));
END crash_fortune_blob;
```

OK, this works. What is the problem with this approach? First of all, if we changed it to also return the `BLOB` it is changing, we still wouldn't be able to use it in SQL context, because it has an `OUT` parameter (not that functionality to be able to run it in SQL is required, but could be). Secondly, we need to create the `BLOB` on our own each time before calling the procedure. We could, on the other hand, write a wrapper function which would create the `BLOB`, call the procedure, and then, return the `BLOB`. That doesn't seem like a clean solution when we have another way, however.

1.2 Through the fence, where the future awaits!

Instead of passing an `OUT` parameter to Java, we could actually create the `BLOB` inside our method, fill it with content and pass it back.

To do that, we need to:

- Obtain an `OracleConnection`.
- Use that object to create a temporary `BLOB`.
- Fill the `BLOB` and return it.

Here is an example of returning a new `BLOB` with the given content back to PL/SQL context:

```
import oracle.sql.BLOB;
import oracle.jdbc.OracleConnection;
import oracle.jdbc.driver.OracleDriver;
import java.io.OutputStream;

public class FortuneBLOB {
  public static BLOB getFortuneBLOB(String secret) throws Exception {
    OracleConnection conn =
            (OracleConnection)new OracleDriver().defaultConnection();

    BLOB fortuneBLOB = BLOB.createTemporary(conn,
                                            true,
                                            BLOB.DURATION_SESSION);
```

```
    OutputStream fortuneScribe = fortuneBLOB.setBinaryStream(0);
    fortuneScribe.write(secret.getBytes());
    fortuneScribe.flush();

    return fortuneBLOB;
  }
}
```

As you can see, we obtain the default connection and cast it to `OracleConnection`. Then we are ready to create our `BLOB` using the static `createTemporary` method of the `BLOB` class. We write the *secret* to the `BLOB` and return it.

To be able to use our method, we also have to create a wrapper function:

```
CREATE OR REPLACE FUNCTION getFortuneBLOB(secret IN VARCHAR2) RETURN BLOB
  AS LANGUAGE JAVA NAME
    'FortuneBLOB.getFortuneBLOB(java.lang.String) return oracle.sql.BLOB';
```

No need for an `IN OUT` parameter any more, and now we use a function instead of a procedure.

Finally, we can test our *BLOB Fortune Factory*:

```
DECLARE
  l_blob BLOB;
BEGIN
  l_blob := getFortuneBLOB('You will have a lucky day!');

  crash_fortune_blob(l_blob);
END;
```

We don't have to create the `BLOB` on our own, it is enough to call the function and it will be returned as a result.

Running the code makes the following string appear in the standard output:

```
You will have a lucky day!
```

Is that a fortune you were hoping for? No? Sorry, I'm hardly a seer, and I have a `BLOB` in my Java!

I hope you enjoyed my article. If you have found any errors in it (even typos), you think that I haven't explained anything clearly enough or you have an idea how I could make the article better – please, do not hesitate to contact me, or leave a comment.