**Eval Functions**

**How to Use a Simple Eval Function**

Eval is the most common type of function. It can be used in FOREACH statements as shown in this script:

-- myscript.pig

REGISTER myudfs.jar;

A = LOAD 'student\_data' AS (name: chararray, age: int, gpa: float);

B = FOREACH A GENERATE myudfs.UPPER(name);

DUMP B;

The command below can be used to run the script. Note that all examples in this document run in local mode for simplicity but the examples can also run in Hadoop mode. For more information on how to run Pig, please see the PigTutorial.

java -cp pig.jar org.apache.pig.Main -x local myscript.pig

The first line of the script provides the location of the jar file that contains the UDF. (Note that there are no quotes around the jar file. Having quotes would result in a syntax error.) To locate the jar file, Pig first checks the classpath. If the jar file can't be found in the classpath, Pig assumes that the location is either an absolute path or a path relative to the location from which Pig was invoked. If the jar file can't be found, an error will be printed:java.io.IOException: Can't read jar file: myudfs.jar.

Multiple register commands can be used in the same script. If the same fully-qualified function is present in multiple jars, the first occurrence will be used consistently with Java semantics.

The name of the UDF has to be fully qualified with the package name or an error will be reported: java.io.IOException: Cannot instantiate:UPPER. Also, the function name is case sensitive (UPPER and upper are not the same). A UDF can take one or more parameters. The exact signature of the function should clear from its documentation.

The function provided in this example takes an ASCII string and produces its uppercase version. If you are familiar with column transformation functions in SQL, you will recognize that UPPER fits this concept. However, as we will see later in the document, eval functions in Pig go beyond column transformation functions and include aggregate and filter functions.

If you are just a user of UDFs, this is most of what you need to know about UDFs to use them in your code.

**How to Write a Simple Eval Function**

Let's now look at the implementation of the UPPER UDF.

package myudfs;

import java.io.IOException;

import org.apache.pig.EvalFunc;

import org.apache.pig.data.Tuple;

import org.apache.pig.impl.util.WrappedIOException;

public class UPPER extends EvalFunc<String>

{

public String exec(Tuple input) throws IOException {

if (input == null || input.size() == 0)

return null;

try{

String str = (String)input.get(0);

return str.toUpperCase();

}catch(Exception e){

throw WrappedIOException.wrap("Caught exception processing input row ", e);

}

}

}

The first line indicates that the function is part of the myudfs package. The UDF class extends the EvalFunc class which is the base class for all eval functions. It is parameterized with the return type of the UDF which is a Java String in this case. We will look into the EvalFunc class in more detail later, but for now all we need to do is to implement the exec function. This function is invoked on every input tuple. The input into the function is a tuple with input parameters in the order they are passed to the function in the Pig script. In our example, it will contain a single string field corresponding to the student name.

The first thing to decide is what to do with invalid data. This depends on the format of the data. If the data is of type bytearray it means that it has not yet been converted to its proper type. In this case, if the format of the data does not match the expected type, a NULL value should be returned. If, on the other hand, the input data is of another type, this means that the conversion has already happened and the data should be in the correct format. This is the case with our example and that's why it throws an error (line 16.) Note that WrappedIOException is a helper class to convert the actual exception to an IOException.

Also, note that lines 10-11 check if the input data is null or empty and if so returns null.

The actual function implementation is on lines 13-14 and is self-explanatory.

Now that we have the function implemented, it needs to be compiled and included in a jar. You will need to build pig.jar to compile your UDF. You can use the following set of commands to checkout the code from SVN repository and create pig.jar:

svn co http://svn.apache.org/repos/asf/pig/trunk

cd trunk

ant

You should see pig.jar in your current working directory. The set of commands below first compiles the function and then creates a jar file that contains it.

cd myudfs

javac -cp pig.jar UPPER.java

cd ..

jar -cf myudfs.jar myudfs

You should now see myudfs.jar in your current working directory. You can use this jar with the script described in the previous section.

**pig udf jar execution**

javac -cp /usr/lib/hadoop/\*:/usr/lib/hadoop-0.20-mapreduce/\*:/usr/lib/hadoop-mapreduce/\*:/usr/lib/hadoop-yarn/\*:/usr/lib/pig/pig.jar /root/Desktop/pig/UPPER.java

jar -cf myudfs.jar myudfs