Making Callouts with Batch Apex for Data of over 12 MB

December 11, 2017 adityamoro

Those who often work on integrations of Salesforce with other platforms must be familiar with the restrictions imposed on callout requests or responses. For those who are not so familiar, the maximum heap size of callouts (either HTTP or WebService calls) is 6 MB for synchronous apex and 12 MB for asynchronous apex. Not so often, this comes across as a serious handicap to developers.

This hack involves a batch class to making a callout to another class that makes the actual HTTP callouts. Let us consider a scenario. Suppose you have to import a large number of attachments into Salesforce on a daily basis. These attachments are to go under Accounts created on a daily basis. In order to get the attachments, we have to make the callouts from the finish() method. So, first we write the batch class. Assuming it looks something like below:

```
global class mybatchclass Implements Database.Batchable <sObject>,database.stateful,database.allowcallouts {

    //variable to be used in SOQL
     public Date today = system.today();
     public set<ID> finalaccounts = new set<Id>();
     public String SOQL = 'SELECT Id, Name, Type from Account where CreatedDate =: today'  ;

    // Start Method of Batch class
    global Database.queryLocator start(Database.BatchableContext bc){
        // Query the records
         return Database.getQueryLocator(SOQL);

    }

    // Execute Method of Batch class
    global void execute(Database.BatchableContext bc, List<Account> accountlist) {

    // Iterate over the list of contacts and get their Account ids
      for(Account cc:accountlist){
         finalaccounts.add(cc.Id);
      }

   }

    // Finish Method of Batch class. This is where you make the callout
    global void finish(Database.BatchableContext bc) {

       attachfiles.attachfilestoaccount(finalaccounts);
     //make the callout here for getting the attachments data of size greater than 12 MB

    }

}
```

Now, lets clear a few things off here before we proceed any further. We are going for Database.Stateful as we need to maintain the state of the Accounts set variable over multiple batches. This is a particular advantage of using batch classes even though the class will become serialized and result in longer execution time. More importantly, we are implementing the Database.AllowsCallouts interface. This is more important in the current scenario where, once we get the list of Accounts, we are making the callout from the finish () method of the batch class.

The batch class can be called either from another class or from the anonymous debug window. To keep things simple, lets initiate the batch class from the anonymous debug window. The syntax will look like the code below:

```
// You can invoke the batch class from the anonymous debug window with the below syntax
   mybatchclass mb = new mybatchclass();
   Database.executebatch(mb);
```

Now, lets start the callout class. You will be passing the set of AccountIDs to the method in this class.

```
public class attachfiles(){

 public static void attachfilestoaccount(set<id> listaccount)
     {
       // First set the callout parameters such as the authToken, Endpoint and the accessToken
        String authToken = 'test authorization token';
        String authEndpoint = 'test authEndpoint';
        String calloutEndpoint = 'test calloutEndpoint';
        String accessToken = 'test_act';

       // Now make the HTTP callout
        Http h1 = new Http();
        HttpRequest hreq2 = new HttpRequest();
        String dealId = '';
        hreq2.setEndPoint(calloutEndpoint);
        hreq2.setMethod('POST');
        hreq2.setHeader('Content-type','application/xml');
        hreq2.setHeader('Authorization','Bearer'+' '+accessToken);
        hreq2.setTimeout(30000);

        hreq2.setBodyAsBlob(Blob.valueOf('testClass'));

       // Get the response which contains the attachment
        HttpResponse res = h1.send(hreq2);

       List<Attachment> atLst = new List<Attachment>();

       for(Account acc:listaccount){
        Attachment att1 = new Attachment();
        att1.Body = Blob.valueOf(res.getbody());
        att1.Name = String.valueOf('Response.txt');
        att1.ParentId = acc.Id;
        atLst.add(att1);
      }

     // Finally, insert the list
      if(atLst.size()> 0)
        insert atLst;

    }
}
```

The attachfilestoaccount() method provide simple steps to integrate with another platform. Note that we are getting the Attachments in the form of HTTP responses. Finally, we are iterating over the Account set and inserting the attachments for them.