How to bypass the 6MB synchronous heap size limit to display data in a Visualforce page?

Two possible approaches are possible if you simply can't get past the limits.

First, you can do client-side rendering and calculation with RemoteAction methods or Remote Object invocations to get the data you need. This is generally acceptable if you're looking at a significant amount of data, and it gets really good performance, as JavaScript is far faster than Apex Code. RemoteAction methods can return 15MB of data, and you are not likely to hit the heap limit since the execution time would be short.

Second, you can do asynchronous processing, but you will need some Queueable code. Use System.enqueueJob to start the processing, and get the Job ID. Then, poll the server until the job reports as completed, then finally query the data back from the database. Note that this means you'll need to arrange a mechanism to determine where the results were stored. It's certainly complicated, but possible.

Note: There used to be a strategy for dealing with heap intensive scripts by moving them asynchronous, @Future methods. However, since the heap limits were increased in Summer '10 this is no longer a reason to simply use @Future method as the limits are the same.

Limit Modifiers

Some limit modifiers that are supported to @future annotation methods are:

@future(limits='2xHeap') - Heap size limit is doubled (24 MB).

@future(limits='3xHeap') - Heap size limit is tripled (36 MB).

@future(limits='2xCPU') - CPU timeout is doubled (120,000 miliseconds) .

@future(limits='3xCPU') - CPU timeout is tripled (180,000 milliseconds).

@future(limits='2xSOQL') - Number of SOQL queries limit is doubled (400).

@future(limits='3xSOQL') - Number of SOQL queries limit is tripled (600).

@future(limits='2xDML') - Number of DML statements limit is doubled (300).

@future(limits='3xDML') - Number of DML statements limit is tripled (450).

@future(limits='2xDMLRows') - Number of records that were processed because of DML operations is doubled (20,000). Includes Approval.process and Database.emptyRecycleBin operations.

@future(limits='3xDMLRows') - Number of records that were processed because of DML operations is tripled (30,000). Includes Approval.process and Database.emptyRecycleBin operations.

Note: You can specify only one higher limit per future method.

Reduce Heap Size:

- Refactor code if heap size is approaching limit
- Use Limits.getHeapSize() and Limits.getLimitHeapSize() to manage heap size during execution.
- Use transient keyword in controllers
- Do not store large amount of data in class variables

Other:

- Specify with sharing keyword so that records accessible to the user are retrieved

keypoints

1. If you find the heap size error, it is always caused by SOQL statement returning more than 30,000 records or more OR collection objects holding too many records in memory.
2. To fix it, remove the SOQL statement and put a “where condition” to return a limited record (1 to 5) and fine tune any collection object.
3. If you want to prevent this from happening, look out for custom objects or currency records where you are continuously loading records weekly to maintain them. Any apex code or trigger looking at this object needs to revisit.

Another Example

Use SOQL For Loops rather than standard SQOL queries

Always use a SOQL for loop to process query results that return many records, to avoid the limit on heap size.

So rather than

```
List accs = [SELECT Id, Name FROM Account];

for(Account a : accs){

  System.debug(a.Name);

}
```

You would have:

```
for(Account a : [SELECT Id, Name FROM Account]){

  System.debug(a.Name);

}
```

Use a Batch Apex or a future method

I recently had a task to upload the binary/Blob for every attachment on an opportunity to a web service as a base64 encoded string. Even after switching to a SOQL for loop it was possible to reach the standard heap size of 6000000 bytes. The attachments could be up to 5 MB in size, so just by loading the attachment Blob out of the database for one record there wasn't much space left over to base64 encode it using EncodingUtil.base64Encode.

Switching to Batch Apex increased the heap limit to 12000000 bytes. Also, by setting the scope to 1 only one attachment was processed at a time.