

The Drupal Batch API

Philip Norton
DrupalCamp Scotland 2024

Philip Norton

- Developer at Code Enigma
- Write articles at `#! code` www.hashbangcode.com
- Organiser of NWDUG

Source Code

- This presentation is available at <https://github.com/hashbangcode/drupal-batch-api-talk>
- All code seen in this presentation is available at https://github.com/hashbangcode/drupal_batch_examples
- I have also written extensively about the Batch API on <https://www.hashbangcode.com/>

The Batch API

Allows data to be processed in small chunks in order to prevent timeout errors or memory problems.

What Problem Are We Solving?

Bouncing Users

- Users get bored quickly.
- Studies show that a 5 second page load has a 0.6% conversion rate.
- Reducing this to 2 seconds doubles the conversion rate.
- This still means that after 2 seconds 98% of users will assume the page will not do anything.

Server Timeouts

- Servers are designed to throw errors is something takes too long.
 - PHP (`max_execution_time`) - 30 seconds
 - PHP (`memory_limit`) - 256MB (recommended for Drupal)
 - Apache (`Timeout`) - 60 seconds
 - Nginx (`send_timeout` / `proxy_send_timeout`) - 60 seconds

What Problem Are We Solving?

- Trying to do too much in one page request.
 - Downloading lots of data from an api.
 - Create/update/delete lots of entities.
- The page times out or runs out of memory

The Batch API

- Solves these problems by splitting long tasks into smaller chunks.
- Drupal then runs them through a special interface.

Running batch process. ☆

Processing batch #5 batch size 100 for total 1,000 items.



Processing...

60%

The Batch API

The Batch Process

The batch process has the following tasks:

- **Initialise** - Set up the batch run, define callbacks.
- **Process** - The batch process operations.
- **Finish** - A finish callback.

Initialise

The BatchBuilder class is used to setup the batch.

```
use Drupal\Core\Batch\BatchBuilder;  
$batch = new BatchBuilder();
```

Initialise

A number of methods set up different parameters.

```
$batch = new BatchBuilder();  
$batch->setTitle('Running batch process.')  
->setFinishCallback([self::class, 'batchFinished'])  
->setInitMessage('Commencing')  
->setProgressMessage('Processing...')  
->setErrorMessage('An error occurred during processing.');
```

Initialise - Adding Operations

Populate the operations we want to perform.

```
// Create 10 chunks of 100 items.
$chunks = array_chunk(range(1, 1000), 100);

// Process each chunk in the array.
foreach ($chunks as $id => $chunk) {
    $args = [
        $id,
        $chunk,
    ];
    $batch->addOperation([BatchClass::class, 'batchProcess'], $args);
}
```

Process - batch_set()

Set the batch running by calling `toArray()` and passing the array to `batch_set()`.

```
batch_set($batch->toArray());
```

The whole purpose of BatchBuilder is to generate that array.

This will trigger and start up the batch process.

Process

- Callback defined in the `addOperation()` method.
- Parameters are the array of arguments you set.
- `$context` is passed by the Batch processor and is used to track progress.

```
public static function batchProcess(int $batchId, array $chunk, array &$amp;context): void {  
}
```

- This method is called multiple times (depending on the batch run).

Process - Tracking Progress

- The `$context` parameter is an array that is maintained between different batch calls.
- The `"sandbox"` element is used inside the batch process and is deleted at the end of the batch run.
- The `"results"` element is will be passed to the finished callback and is often used to track progres.

```
public static function batchProcess(int $batchId, array $chunk, array &$amp;context): void {  
    if (!isset($context['sandbox']['progress'])) { }  
    if (!isset($context['results']['updated'])) { }  
}
```

Process - Messages

- As the batch runs you can set a "message" element to print messages to the user.
- This will appear above the batch progress bar.

```
// Message above progress bar.  
$context['message'] = t('Processing batch #@batch_id batch size @batch_size for total @count items.', [  
  '@batch_id' => number_format($batchId),  
  '@batch_size' => number_format(count($chunk)),  
  '@count' => number_format($context['sandbox']['max']),  
]);
```

Finish - The Finished Callback

- When the batch finishes the finished callback is triggered.
- This has a set of parameters that detail how the batch performed.

```
public static function batchFinished(  
    bool $success,  
    array $results,  
    array $operations,  
    string $elapsed): void {  
}
```

Batch Internal Workings

- The Batch API is really an extension of the Queue system.
- When you add operations to the batch you are adding items to the queue.
- The Drupal batch runner then pulls items out of the queue and feeds them to the process method.

The Batch "finished" State

The Batch "finished" State

- So far, we have looked at pre-configured batch runs.
- A better approach is to use the `finished` property of the batch `$context` array.
- If we set this value to ≥ 1 then the batch process is considered finished.

```
if (done) {  
    $context['finished'] = 1;  
}
```

The Batch "finished" State

The setup is slightly different as we only create a single operation.

```
$array = range(1, 1000);  
$batch->addOperation([BatchClass::class, 'batchProcess'], [$array]);
```

This is run over and over until we issue the finished state.

The Batch "finished" State

It is common to divide the progress by the maximum number of items.

```
$context['finished'] = $context['sandbox']['progress'] / $context['sandbox']['max'];
```


The Batch "finished" State

This means we can just launch the batch with no arguments.

```
$batch->addOperation([BatchProcessNodes::class, 'batchProcess']);
```

The "max" is found in the batchProcess() method.

```
public static function batchProcess(array &$context): void {
    if (!isset($context['sandbox']['progress'])) {
        $query = \Drupal::entityQuery('node');
        $query->accessCheck(FALSE);
        $context['sandbox']['progress'] = 0;
        $context['sandbox']['max'] = $query->count()->execute();
    }
}
```

Running Batch With Drush

Drush

Call batch set as normal.

```
batch_set($batch->toArray());
```

Then call the Drush function.

```
drush_backend_batch_process();
```

This will run the batch on the command line.

When To Use The Batch API

- If the request processes lots of items then move it into a batch.
- Use the batch system early to save having to rework things later.

Examples Of Batch API In Action

- Some live demos!

Batch Using A Form

- Look at 1,000 items and roll a dice.

Batch Using Drush

- Look at 1,000 items and roll a dice.

Process a CSV file

- Import 1,000 nodes using a batch process.

Batch Inside Drupal

The Update Hook

- Update hooks get a \$sandbox variable. This is actually a batch \$context array.

Modules That Use Batch

View Batch Operation

- Batch process items in a view.

https://www.drupal.org/project/views_bulk_operations

Advanced Queue

- Shows a breakdown of the current queues in your system.
- Gives the option to process queues as a batch run.

<https://www.drupal.org/project/advancedqueue>

Resources

- [Drupal 11: An Introduction To Batch Processing With The Batch API](#)
- [Drupal 11: Batch Processing Using Drush](#)
- [Drupal 11: Using The Finished State In Batch Processing](#)
- [Drupal 11: Using The Batch API To Process CSV Files](#)
- [Drupal Batch Examples source code](#)

Questions?

- Slides:

<https://github.com/hashbangcodebatch-api-talk>



Thanks!

- Slides:

<https://github.com/hashbangcodebatch-api-talk>



