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 presentaiton 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 doubes 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

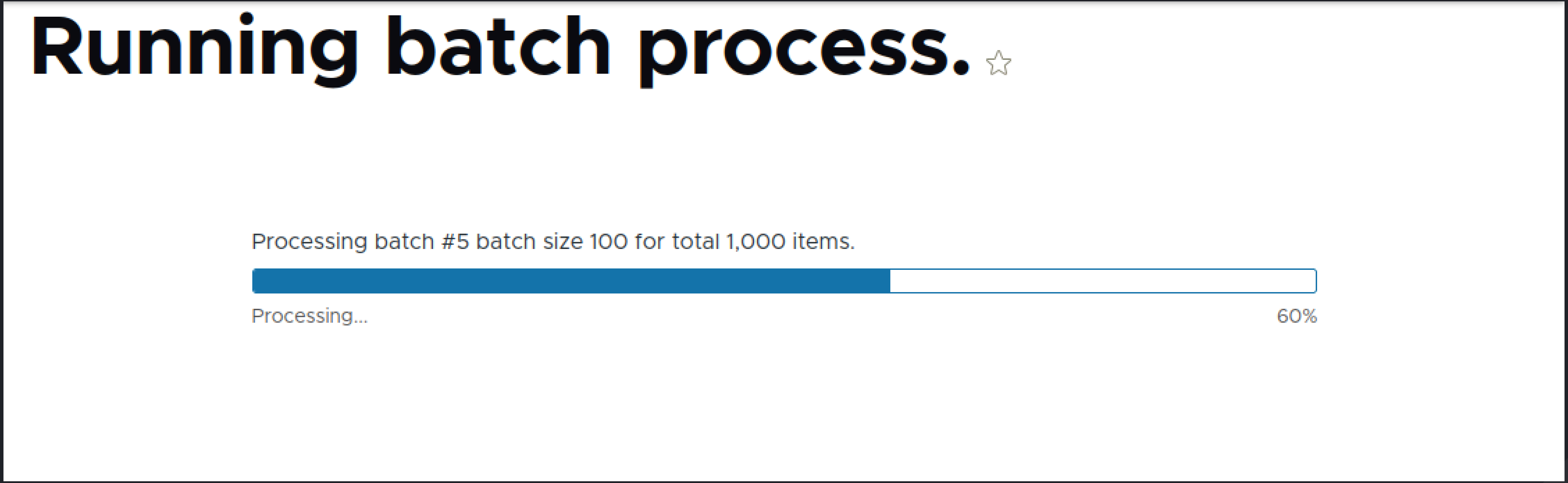
What Problem Are We Solving?

- 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:

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

BatchBuilder Class

Used to setup the batch.

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

BatchBuilder Class

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.');
```

BatchBuilder Class

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);
}
```

Running The Batch

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

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

The whole purpose of this class to generate that array. This will initiate the batch process.

Batch 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 $chunk, array &$context): void {
}
```

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

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.

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

Messages

- As the batch runs you can set a `"message"` element to print messages to the user.
- This will appaer 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']),
]);
```

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 Batch "finished" State

The Batch "finished" State

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

The Batch "finished" State

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

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 them move it into a batch.
- Use the batch system early to save having to rework things later.

Examples Of Batch API In Action

Batch Using A Form

Process a CSV file.

Modules That Use Batch

Advanced Queue

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

Example Links

- Link to website
- Link to slide

Page without a footer

- Nope, no footer.

Small Text Slide

This is small text.

Talk template with image

- Something.

#!