File Extractor : une autre approche de l’indexation de fichier

Qui suis-je ?

- Florent Torregrosa (Grimreaper)
- Début d'utilisation avec Drupal 6 en 2011
- Tech expert Drupal chez Smile
- Contributions :
  - ex-membre du bureau de l'ADFF
  - contributions diverses pour Drupal France
  - co-organisateur d'événements
  - traductions
  - modules
  - patches

Sommaire

1)Contexte et problématique

2)Nouvelle approche et réflexions

3)Création du fork

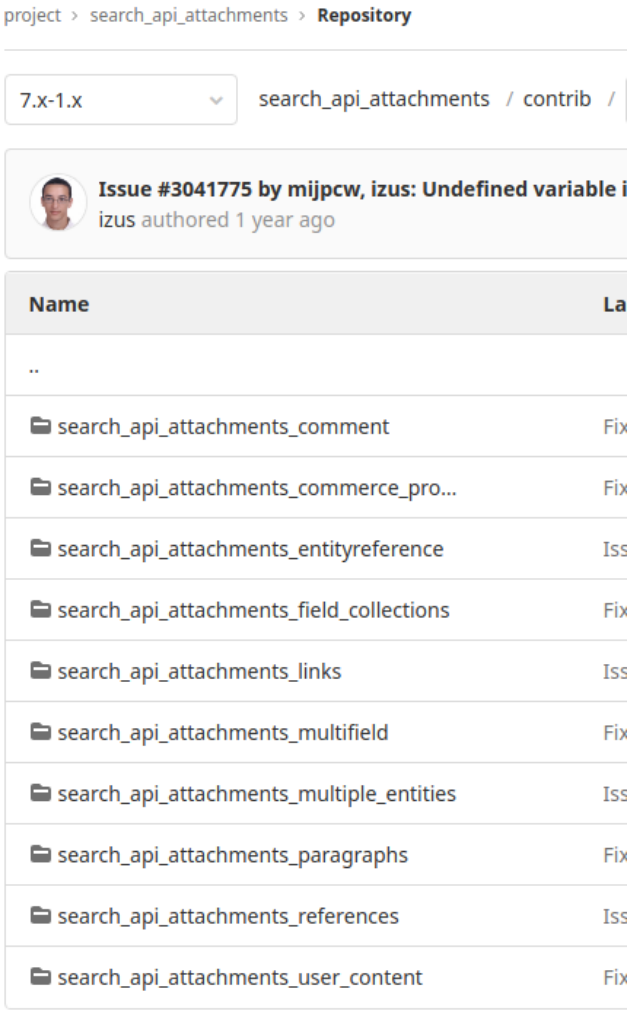
1 : Contexte et problématique

1 : Contexte et problématique

La recherche en Drupal implique dans peut-être 90 % des cas le module Search API.

Si l’on veut pouvoir effectuer des recherches sur du contenu de fichiers attachés aux contenus, la solution la plus répandue pour cela est d’utiliser le module Search API Attachments.

1 : Contexte et problématique



- Version Drupal 7
- Besoin d’un sous-module dédié à chaque variation de type de champ « entity reference »

1 : Contexte et problématique

```php
public function addFieldValues(ItemInterface $item) {
  $files = [];
  $config = $this->configFactory->get(static::CONFIGNAME);
  $extractor_plugin_id = $config->get('extraction_method');
  if ($extractor_plugin_id != '') {
    $configuration = $config->get($extractor_plugin_id . '_configuration');
    $extractor_plugin = $this->textExtractorPluginManager->createInstance($extractor_plugin_id, $configuration);
    // Get the entity.
    $entity = $item->getOriginalObject()->getValue();
    $is_entity_type_file = $entity->getEntityTypeId() == 'file';
    foreach ($this->getFileFieldsAndFileEntityItems() as $field_name => $label) {
      // If the parent entity is not a file, no need to parse the
      // saa static::SAA_FILE_ENTITY item.
      if (!$is_entity_type_file && $field_name == static::SAA_FILE_ENTITY) {
        break;
      }
      if ($is_entity_type_file && $field_name == static::SAA_FILE_ENTITY) {
        $files[] = $entity;
      }

      $property_path = static::SAA_PREFIX . $field_name;

      // A way to load $field.
      foreach ($this->fieldHelper->filterForPropertyPath($item->getFields(), NULL, $property_path) as $field) {
        $all_fids = [];
        if ($entity->hasField($field_name)) {
          // Get type to manage media entity reference case.
          $type = $entity->get($field_name)->getFieldDefinition()->getType();
          if ($type == 'entity_reference') {
            /** @var \Drupal\Core\Field\BaseFieldDefinition $field_def */
            $field_def = $entity->get($field_name)->getFieldDefinition();
            if ($field_def->getItemDefinition()->getSetting('target_type') === 'media') {
              // This is a media field.
              $filefield_values = $entity->get($field_name)->filterEmptyItems()->getValue();
              foreach ($filefield_values as $media_value) {
                $media = Media::load($media_value['target_id']);
                if ($media !== NULL) {
                  $bundle_configuration = $media->getSource()->getConfiguration();
                  if (isset($bundle_configuration['source_field'])) {
                    /** @var \Drupal\Core\Field\FieldItemListInterface $field_item */
                    foreach ($media->get($bundle_configuration['source_field'])->filterEmptyItems() as $field_item) {
                      if ($field_item->getFieldDefinition()->getType() === 'file') {
                        $value = $field_item->getValue();
                        $all_fids[] = $value['target_id'];
                      }
                    }
                  }
                }
              }
            }
          }
          elseif ($type == "file") {
            $filefield_values = $entity->get($field_name)->filterEmptyItems()->getValue();
            foreach ($filefield_values as $filefield_value) {
              $all_fids[] = $filefield_value['target_id'];
            }
          }

          $fids = $this->limitToAllowedNumber($all_fids);
          // Retrieve the files.
          $files = $this->entityTypeManager
            ->getStorage('file')
            ->loadMultiple($fids);
        }
        if (!empty($files)) {
          $extraction = '';

          foreach ($files as $file) {
            if ($this->isFileIndexable($file, $item, $field_name)) {
```

- Drupal 8
- Méthode « addFieldValues » du plugin pour ajouter des données indexées
- Conditions sur :
  - Le type de champ
  - Le type d’entité
- Limite de profondeur d’indexation

2 : Nouvelle approche et réflexions

2 : Nouvelle approche et réflexions

- À l'occasion de la consultation du code du module « Content Moderation » (Core).
- Le champ de base « moderation_state » est un champ calculé.
- Hors, « moderation_state » est indexable par Search API (et est exposé par la JSON:API par exemple).

```
244   */
245   public function entityBaseFieldInfo(EntityTypeInterface $entity_type) {
246     if (!$this->moderationInfo->isModeratedEntityType($entity_type)) {
247       return [];
248     }
249
250     $fields = [];
251     $fields['moderation_state'] = BaseFieldDefinition::create('string')
252       ->setLabel(t('Moderation state'))
253       ->setDescription(t('The moderation state of this piece of content.'))
254       ->setComputed(TRUE)
255       ->setClass(ModerationStateFieldItemList::class)
256       ->setDisplayOptions('view', [
257         'label' => 'hidden',
258         'region' => 'hidden',
259         'weight' => -5,
260       ])
261       ->setDisplayOptions('form', [
262         'type' => 'moderation_state_default',
263         'weight' => 100,
264         'settings' => [],
265       ])
266       ->addConstraint('ModerationState', [])
267       ->setDisplayConfigurable('form', TRUE)
268       ->setDisplayConfigurable('view', FALSE)
269       ->setReadOnly(FALSE)
270       ->setTranslatable(TRUE);
271
272     return $fields;
273   }
274
```

2 : Nouvelle approche et réflexions

Idée :

- Créer un nouveau « base field » sur le type d’entité fichier
- Ce nouveau champ aura pour valeur le contenu du fichier indexé

2 : Nouvelle approche et réflexions

Avantages :

- Fonctionnalités automatiquement héritées de Search API :
  - Détecté comme n’importe quel autre champ
  - Récursivité
  - Mise en avant (highlight)
  - Possibilité de choisir quels champs fichiers indexés
  - Autres
- Plus instinctif lors de la configuration de l’index de recherche
- Plus besoin d’implémenter un Plugin Search API

2 : Nouvelle approche et réflexions

- Même avec un refactoring complet, il resterait le nom du module « **Search API** Attachments ».
- Décision difficile de faire un fork :
  - Ce n'est pas quelque chose de courant dans la communauté Drupal, qui cherche à favoriser l'entraide plutôt que le fork.
  - Le module Search API Attachments a plus de 8000 sites (dont plus de 4000 Drupal 8) qui l'utilisent.
  - J'avais effectué de nombreuses contributions. Je connais le mainteneur, Ismaeil Abouljamal (izus), qui fait partie de mes mentors. Et Search API Attachments nous avait permis d'apprendre Drupal 8 en portant le module de D7 à D8.
- Malgré cela, impossible dans ma réflexion d'avoir le nom d'un module préfixé par un autre alors qu'il n'y a plus de lien de dépendance (et même quasiment plus de liens) entre eux => Création d'un fork.

3 : Création du fork

3 : Création du fork

- Renommage du module
- Refactoring du système de plugin :
  - Meilleure gestion des formulaires de configuration des méthodes d’extraction
  - Uniformisation des plugins
- Refactoring du formulaire de configuration du module
  - Meilleure gestion de la configuration des méthodes d’extraction
- Création d’un service d’extraction :
  - Meilleure DX
  - Éviter la duplication de code
- Retrait du plugin de Views pour exclure des champs de la recherche à la volée : remplacé par un patch sur le module Search API directement

3 : Création du fork

- Standardisation du fichier README.txt
- Création de pages de documentation :
  - Plus facilement accessibles et éditables que dans le README.txt
- Création de tests automatisés pour toutes les méthodes d’extraction :
  - sauf Search API Solr car la méthode utilisée dans le module dépendant est testée

3 : Création du fork

- Durant l’adaptation du code, l’historique Git du code du formateur de champ « file_extracted_text » a permis de comprendre qu’il avait initialement été créé pour contourner le problème de niveau arbitraire de profondeur de référence d’entité. Donc la raison principale de l’introduction du champ de base n’avait plus lieu d’être.
- Mais son utilisation via des views mode reste assez peu pratique donc la nouvelle approche a encore son sens.
- Le travail de fork était déjà commencé.

Conclusion

- Quid de Search API Attachments ?
  - Voir les issues en lien en fin de présentation.
  - La dépréciation de Search API Attachments n’est pas à l’ordre du jour.
  - Proposition au mainteneur qu’il soit également mainteneur de File Extractor : pas de réponse.
- Désormais pour indexer des fichiers, il y a 2 modules avec 2 approches différentes.
- File Extractor a une meilleure architecture et s’adaptera à plus de situations (avis non impartial), car il aura pu bénéficier de l’expérience acquise sur Search API Attachments.
- Search API Attachments reste plus utilisé (historiquement ancré).

Merci pour votre attention

Avez-vous des questions ?

Liens

- https://www.drupal.org/project/file_extractor
- https://www.drupal.org/project/search_api
- https://www.drupal.org/project/search_api_attachments
- https://git.drupalcode.org/project/search_api_attachments/-/blob/8.x-1.x/src/Plugin/search_api/processor/FilesExtractor.php
- https://git.drupalcode.org/project/search_api_attachments/-/tree/7.x-1.x/contrib
- https://git.drupalcode.org/project/drupal/-/blob/8.9.x/core/modules/content_moderation/src/EntityTypeInfo.php
- https://www.drupal.org/project/file_extractor/issues/3126845
- https://www.drupal.org/project/search_api_attachments/issues/3008580
- https://www.drupal.org/project/search_api_attachments/issues/2844979

Crédits

- https://unsplash.com/photos/05HLFQu8bFw
- https://unsplash.com/photos/Y3KEBQlB1Zk
- https://unsplash.com/photos/6AtQNsjMoJo
- https://unsplash.com/photos/u0vgcIOQG08