Search.Integration

Общая архитектура

Для индексации документов из различных источников данных используются несколько сервисов интеграции, по одному на каждый источник: QP, DPC, etc.

Чтобы не останавливать запросы поиска из Search.API на время переиндексации, каждый индекс имеет две версии: старая, из которой идет чтение и новая, в которую идет запись. Также каждый индекс имеет алиас — логическое имя, по которому он доступен Search.API. По завершении индексации алиас перекидывается со старой версии индекса на новую, а старая версия удаляется.

Также, чтобы сосуществовать с другими приложениями в рамках одного кластера Elastic, все индексы и алиасы имеют оперделенный префикс, который скрывается от конечного пользователя Search.API (см. `Settings.IndexPrefix`, `Settings.AliasPrefix`).

Высокоуровнево процесс индексации одного источника выглядит так:

Полная индексация

1. Создаем новые индексы вида `"{IndexPrefix}.{DocumentType}.{yyyy-MM-ddThh-mm-ss}"`
2. Для каждого индекса, пока не обработаны все документы:
   - Получаем из источника данных набор JSON-документов размера `Settings.BatchSize`
   - Проставляем вычисляемые и контекстные поля, такие как `"SearchUrl"`
   - Удаляем HTML-разметку из текстовых полей и осуществляем другой препроцессинг
   - Индексируем набор документов в Elastic с помощью `/_bulk` запроса.
3. Перекидываем алиасы вида `"{AliasPrefix}.{DocumentType}"` на новые индексы
4. Удаляем старые индексы.

Частичная индексация

1. Для каждого индекса, пока не обработаны все документы, измененные **за прошедшие N суток**:
    - Получаем из источника данных набор JSON-документов размера `Settings.BatchSize`
    - Проставляем вычисляемые и контекстные поля, такие как `"SearchUrl"`
    - Удаляем HTML-разметку из текстовых полей и осуществляем другой препроцессинг
    - Обновляем набор документов в Elastic с помощью `/_bulk` запроса.

Сервисы индексации

Каждый сервис индексации определяется набором классов, объединенных с помощью интерфейса-метки `TMarker` и состоит из:

`ScheduledService<TMarker>`

Background Worker, который запускает полную или частичную индексацию источника данных по расписанию в формате CronTab или вручную через Админку. Для обеспечения возможности остановки индексации (вручную или при выключении сервиса), все его асинхронные методы должны пробрасывать `CancellationToken`.

`IndexingController<TMarker>`

Web API контроллер для ручного запуска / остановки индексации и просмотра состояния процесса индексации. Каждому такому контроллеру соответствует своя страница в Админке.

`IndexingContext<TMarker>`

Описание состояния текущей индексации. Включает в себя один или несколько экземпляров `IndexingReport` — состояние индексации по каждому отдельному индексу Elastic в рамках одной интеграции.

`Settings<TMarker>`

Настройки процесса индексации. Включают в себя:

- `CronSchedule` — расписание запуска индексации в формате CronTab. Если сервис не успел завершить предыдущую индексацию к моменту старта следующей, то следующая индексация будет пропущена. Пример: `"0 0/6 * * *"` — каждые 6 часов.
- `IndexPrefix` — префикс для индексов Elastic, создаваемых в рамках этой интеграции. Пример: `"index.search.qp."`, `"index.search.dpc."`
- `AliasPrefix` — префикс для алиасов Elastic, создаваемых в рамках этой интеграции. Пример: `"alias.search.qp."`, `"alias.search.dpc."`

`IndexPrefix` и `AliasPrefix` служат для одномоментного переключения запросов поиска в Search.API со старой версии индекса на новую.

Каждый сервис индексации имеет свою секцию настроек в `appsettings.json`. Например, `Settings.QP` или `Settings.DPC`. Все настройки могут быть переопределены через стандартные переменные окружения .NET Core в формате `PropName__NestedName`, используя разделитель `__` для вложенных объектов.

Для добавления нового сервиса интеграции необходимо создать все вышеописанные компоненты и зарегистрировать их в `Startup.cs` с помощью метода-расширения `services.AddScheduledService()`.

Препроцессинг документов

Перед отправкой на индексацию в Elastic каждый документ проходит через цепочку interceptor-ов `DocumentProcessor<TMarker>`. Каждый такой препроцессор может модифицировать JSON-дерево документа.

Например, `HtmlStripProcessor` удаляет из каждого текстового поля JSON-документа HTML-разметку, содержимое тегов script и style, HMTL-комментарии.

Препроцессоры выполняются в порядке регистрации в DI-контейнере. Для добавления препроцессора необходимо зарегистрировать его в `Startup.cs` с помощью метода-расширения `services.AddDocumentProcessor()`.

`DocumentProcessor<TMarker>` будет обрабатывать только документы, полученные из сервиса с соответствующим `TMarker`. Поэтому, обобщенные препроцессоры (такие как `HtmlStripProcessor`) нужно регистрировать отдельно для каждого `TMarker`.

Полнотекстовая индексация полей

По умолчанию все строковые поля индексируются как `keyword` и недоступны для полнотекстового поиска, автокомплита, etc. Чтобы поле индексировалось как `text` нужно:

1. Найти соотв. mapping template) вида `index.search.{name}.json` в каталоге `QA.Search.Admin/ElasticSearch/_template`

2. Добавить название поля в одно из регулярных выражений:

```
{
  "mappings": {
    "_doc": {
      "dynamic_templates": [
        {
          "text": {
            // влияет на поля во всех вложенных объектах
            "match": "Name|Title|Description|{YourField}",
            // влияет на поля по конкретному пути
            "path_match": "Region.Title|{YourObject.YourField}"
          }
        }
      ]
    }
  }
}
```

3. Обновить измененный mapping template по следующему URL:

```
PUT /_template/index.search.{name}
{
  // JSON ...
}
```

Индексация DPC

Realtime-индексация продуктов DPC происходит не по таймеру, а с помощью Web Hooks витрины DPC, которые реализует `ProductsController`. Его адрес необходимо зарегистрировать как одну из витрин DPC. `ScheduledServiceDPC` обеспечивает же только полную переиндексацию по кнопке в Админке.

Продукты разного типа сохраняются в разных индексах Elastic. Типы продуктов для индексации определяются в настройках, в массиве `Settings.DPC.ProductTypes`.

Препроцессинг продуктов DPC

Перед индексацией у продуктов проставляются значения региональных тегов. Также происходит преобразование массива `"Parameters"` в массив `"ParameterAliases"` и словарь массивов `"ParametersByAlias"`, где в качестве ключа параметра берется поле `"BaseParameter.Alias"`. Это необходимо для организации фасетного поиска по параметрам и корректного вывода типов полей параметров в динамическом маппинге Elastic За это отвечают `RegionTagsProcessor` и `ParameterAliasProcessor`.

После этого происходит сопоставление регионов DPC и маркетинговых регионов QP в `DpcSiteRegionsProcessor`. Регионы DPC перемещаются в поле `"DpcRegions"` а на их место в поле `"Regions"` помещаются соответствующие регионы QP.

Индексация QP

Индексация QP проходит в два этапа:

1. Полная переиндексация выбранных контентов. Запускается по расписанию CronTab раз в несколько часов или дней. За неё отвечают сервисы `ScheduledServiceQP` — для БД `databaseName` и `ScheduledServiceMedia` — для БД `mediaDatabaseName`.
2. Частичная переиндексация документов, **корневые** статьи которых были обновлены за прошедшие двое суток. Запускается по расписанию CronTab раз в 10 минут. За неё отвечают сервисы `ScheduledServiceQPUpdate` — для БД `databaseName` и `ScheduledServiceMediaUpdate` — для БД `mediaDatabaseName`.

Каждому сервису соответствует своя страница в Админке.

Настройка индексации контента

Для индексации выбранного контента необходимо реализовать т.н. **View** — класс, унаследованный от `ElasticView<TContext>`, где `TContext` — соответствующий `DbContext`. И зарегистрировать его в DI-контейнере с помощью метода-расширения `.AddElasticView<TView>()`. Либо зарегистрировать сразу все View для заданного `DbContext` с помощью `.AddElasticViews<TContext>()`.

Чтобы описать какие поля нужно выбрать, какие связанные контенты объединить через JOIN, задать фильтры на загружаемые статьи — нужно переопределить свойство `IQueryable ElasticView.Query`. Воспользовавшись при этом DSL методами- расширениями из класса `QueryableExtensions`:

- `.Filter(Func<T, bool> predicate)` — фильтровать статьи по простым полям,
- `.Pick(Func<T, P> selector)` — выбрать поля для индексации,
- `.Omit(Func<T, P> selector)` — исключить поля из индексации,
- `.JoinOne(Func<T, P> selector)` — загрузить связь ManyToOne,
- `.JoinMany(Func<T, ICollection<E>> selector)` — загрузить связь OneToMany,
- `.JoinMany(Func<T, ICollection<L>> collectionSelector, Func<L, E> elementSelector)` — загрузить связь ManyToMany.

Такого рода DSL в итоге преобразуется в синтаксис `FOR JSON PATH` для SQL Server 2016+. позволяющий собрать древовидный JSON-документ за один SQL-запрос без дублирования полей в ResultSet.

Пример:

Новости за 2019 год без полей `SmallImage` и `TransliterationTitle`, включая:

- опубликовавшее их СМИ с полем `Title`
- список регионов с полями `Title`, `DpcAlias` и `Parent`

```
protected override IQueryable Query => Db.News
    .Filter(n => n.PublishDate > DateTime.Parse("2019-01-01"))
    .Omit(n => new { n.SmallImage, n.TransliterationTitle })
    .JoinOne(n => n.PublishSMI
        .Pick(s => s.Title))
    .JoinMany(n => n.Regions, l => l.MarketingRegionLinkedItem
        .Filter(r => r.DpcAlias != null)
        .Pick(r => new { r.Title, r.DpcAlias })
        .JoinOne(r => r.Parent)
            .Pick(r => new { r.Title, r.DpcAlias }));
```

Также можно переопределить следующие методы `ElasticView`:

- `InitAsync()` — инициализация View перед началом каждой индексации,
- `CountAsync()` — подсчет примерного количества документов для индексации,
- `LoadAsync()` — выгрузка из БД очередной партии JSON-документов.

Например, для загрузки дерева страниц сайта и вычисления URL для каждого JSON- документа перед отправкой на индексацию в Elastic.

Препроцессинг контентов QP

Помимо удаления HTML-разметки также удаляется микроразметка вида `||region_tag_name||` и

```
[[CamelCaseName {
  "Foo": "test test [test]",
```

```
    "Bar": 123,
  }]]
```

За это отвечает `SpecialMarkupStripProcessor`.