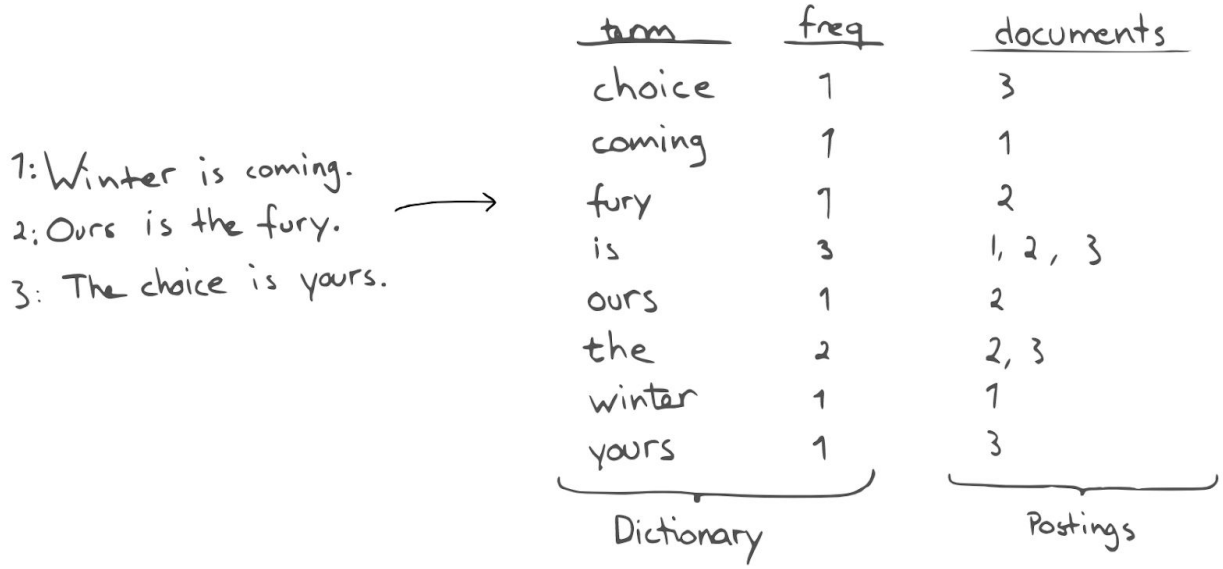
Indexing

As a search engine, indexing is an essential function of **Elasticsearch**. A typical create index request is like the following:

```
PUT /customer/_doc/1
{
  "name": "John Doe"
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started-index.html

The indices are stored in the form of inverted index as shown in the following diagram.



"Sample documents and resulting inverted index"

https://www.elastic.co/blog/found-elasticsearch-from-the-bottom-up

In Elasticsearch, client requests go through two layers, Rest layer and Transport layer. Where the Rest layer is for parsing request parameters and the Transport layer is the layer that handles the request.

Client Side

On the client side, Request are dispatched in `RestController`

```
// java/org/elasticsearch/rest/RestController.java
@Override
public void dispatchRequest(RestRequest request, RestChannel channel,
ThreadContext threadContext) {
        if (request.rawPath().equals("/favicon.ico")) {
            handleFavicon(request.method(), request.uri(), channel);
            return;
        }
        try {
            tryAllHandlers(request, channel, threadContext);
        } catch (Exception e) {
            ......
        }
    }
```

`tryAllHandlers` method will try out every possible handler and find the appropriate handler based on the request method. In the case of bulk action, the handler is `RestBulkAction`. In the constructor of `RestBulkAction`, it will register the handler to `RestController`.

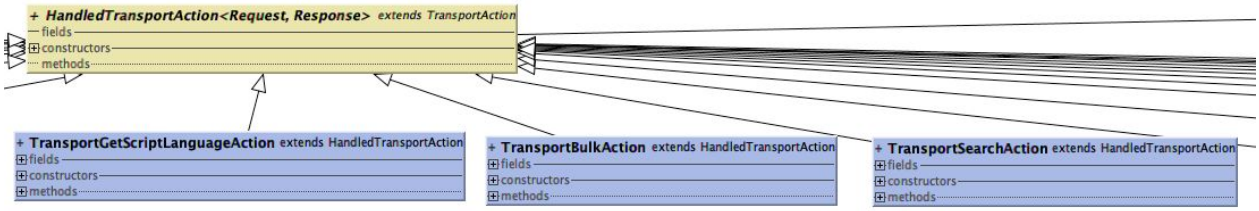
```
public RestBulkAction(Settings settings, RestController controller) {
    controller.registerHandler(POST, "/_bulk", this);
    controller.registerHandler(PUT, "/_bulk", this);
    controller.registerHandler(POST, "/{index}/_bulk", this);
    controller.registerHandler(PUT, "/{index}/_bulk", this);
```

```
    this.allowExplicitIndex = MULTI_ALLOW_EXPLICIT_INDEX.get(settings);
}
```

`RestBulkAction` will analyze the `RestRequest` and transform it to a `bulkRequest`. Then hands the bulkRequest to `NodeClient`.

```
@Override
    public RestChannelConsumer prepareRequest(final RestRequest request,
final NodeClient client) throws IOException {
        // Process the RestRequest and construct bulkRequest
        ......
        // To Node client
        return channel -> client.bulk(bulkRequest, new
RestStatusToXContentListener<>(channel));
    }
```

`NodeClient` will pass the action to the transport layer by converting the action to a `TransportAction`. In this case, a `TransportBulkAction` instance.



Server Side

The `doExecute()` method in `TransportBulkAction` analyzes all indices in bulkRequest, checking if the index already exists and creates all the indices that are missing.

Create Index:

1. The masterOperation will convert `CreateIndexRequest` to `CreateIndexClusterStateUpdateRequest` and pass it to `MetaDataCreateIndexService.createIndex()`, this method creates an index in the cluster state and waits for the specified number of shard copies to become active before sending the response on the listener.

```
//java/org/elasticsearch/action/admin/indices/create/TransportCreateIndexAction.java
@Override
    protected void masterOperation(Task task, final CreateIndexRequest request, final ClusterState state, final ActionListener<CreateIndexResponse> listener) {
        String cause = request.cause();
        if (cause.length() == 0) {
            cause = "api";
        }

        final String indexName = indexNameExpressionResolver.resolveDateMathExpression(request.index());
        final CreateIndexClusterStateUpdateRequest updateRequest =
            new CreateIndexClusterStateUpdateRequest(cause, indexName, request.index())
.ackTimeout(request.timeout()).masterNodeTimeout(request.masterNodeTimeout())
                .settings(request.settings()).mappings(request.mappings())
                .aliases(request.aliases())
```

```
                .waitForActiveShards(request.waitForActiveShards());

        createIndexService.createIndex(updateRequest,
ActionListener.map(listener, response ->
            new CreateIndexResponse(response.isAcknowledged(),
response.isShardsAcknowledged(), indexName)));
    }
```

2. This will call `MetaDataCreateIndexService.onlyCreateIndex()`, it will submit a cluster state update task by calling the `execute()` method of `AckedClusterStateUpdateTask`, and start building index by calling `IndicesService.createIndex()`, and the actual create index is done by `IndexModule.newIndexService()`.

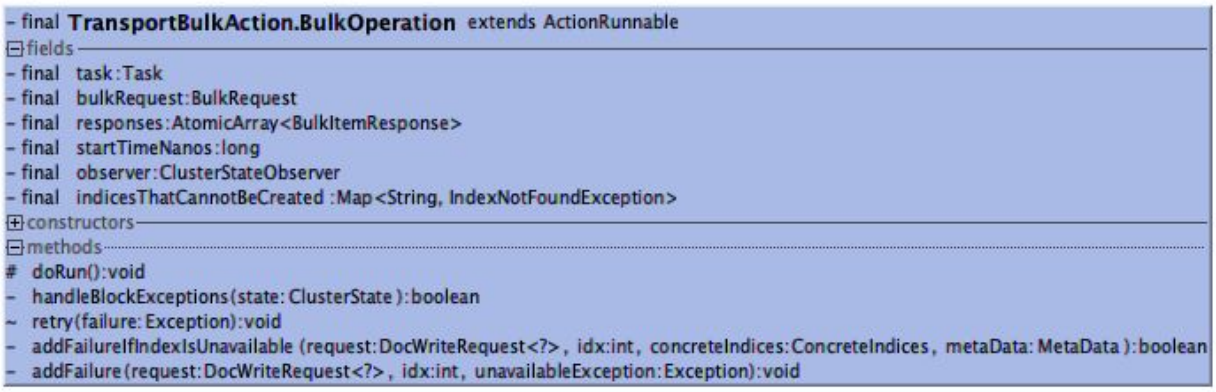
```
// MetaDataCreateIndexService
clusterService.submitStateUpdateTask("create-index [" +
request.index() + "], cause [" + request.cause() + "]",
 new AckedClusterStateUpdateTask<>(Priority.URGENT, request,
listener) {
                .......
            });
```

3. In the `MetaDataCreateIndexService.applyCreateIndexRequest()`, after we finish creating the index, it constructs `IndexMetaData` and generates the updated `ClusterState`.
4. After updating the `ClusterState`, if the current node is Master Node, it will notify other nodes and synchronize the cluster state.

We have created an index, next is the procedure of indexing documents.

Index documents:

In `BulkOperation.doRun()`:



1. Get the newest ClusterState

```
final ClusterState clusterState = observer.setAndGetObservedState();
```

2. Traverse the documents in the request and get the operation type OpType. Then preprocess the document.
3. Traverse the documents, get the shardId for every request.

```
ShardId shardId =
clusterService.operationRouting().indexShards(clusterState,
concreteIndex, request.id(), request.routing()).shardId();
```

4. Group the request in the same shard, pack the requests as BulkShardRequest, send the request to its shard's node.
5. The node will update the main shard by calling `TransportReplicationAction.doExecute()`, this will create a ReroutePhase task, at this time the index will write to the main shard by calling `InternalEngine.indexIntoLucene()`.

Relation with other parts

Indexing is the fundamental function of elasticsearch.

- it is invoked by rest api request.

- Creating indices will update the cluster state.
- Searching functionality is based on indexing