```
{
    "mappings":{
        "hello":{
            "properties":{
                "id":{
                    "type":"long",
                    "store":true
                },
                "title":{
                    "type":"text",
                    "store":true,
                    "analyzer":"ik_smart"
                },
                "content":{
                    "type":"text",
                    "store":true,
                    "analyzer":"ik_smart"
                }
            }
        }
    }
}
```

指定 ik 分词器，这里的参数还有很多，比方说 index:true 表示建立索引

第六章 ElasticSearch集群

ES集群是一个 P2P类型(使用 gossip 协议)的分布式系统，除了集群状态管理以外，其他所有的请求都可以发送到集群内任意一台节点上，这个节点可以自己找到需要转发给哪些节点，并且直接跟这些节点通信。所以，从网络架构及服务配置上来说，构建集群所需要的配置极其简单。在 Elasticsearch 2.0 之前，无阻碍的网络下，所有配置了相同 cluster.name 的节点都自动归属到一个集群中。2.0 版本之后，基于安全的考虑避免开发环境过于随便造成的麻烦，从 2.0 版本开始，默认的自动发现方式改为了单播(unicast)方式。配置里提供几台节点的地址，ES 将其视作 gossip router 角色，借以完成集群的发现。由于这只是 ES 内一个很小的功能，所以 gossip router 角色并不需要单独配置，每个 ES 节点都可以担任。所以，采用单播方式的集群，各节点都配置相同的几个节点列表作为 router 即可。

集群中节点数量没有限制，一般大于等于2个节点就可以看做是集群了。一般处于高性能及高可用方面来考虑一般集群中的节点数量都是3个及3个以上。

6.1 集群的相关概念

6.1.1 集群 cluster

一个集群就是由一个或多个节点组织在一起，它们共同持有整个的数据，并一起提供索引和搜索功能。一个集群由一个唯一的名字标识，这个名字默认就是"elasticsearch"。这个名字是重要的，因为一个节点只能通过指定某个集群的名字，来加入这个集群

6.1.2 节点 node

一个节点是集群中的一个服务器，作为集群的一部分，它存储数据，参与集群的索引和搜索功能。和集群类似，一个节点也是由一个名字来标识的，默认情况下，这个名字是一个随机的漫威漫画角色的名字，这个名字会在启动的时候赋予节点。这个名字对于管理工作来说挺重要的，因为在这个管理过程中，你会去确定网络中的哪些服务器对应于Elasticsearch集群中的哪些节点。

一个节点可以通过配置集群名称的方式来加入一个指定的集群。默认情况下，每个节点都会被安排加入到一个叫做"elasticsearch"的集群中，这意味着，如果你在你的网络中启动了若干个节点，并假定它们能够相互发现彼此，它们将会自动地形成并加入到一个叫做"elasticsearch"的集群中。

在一个集群里，只要你想，可以拥有任意多个节点。而且，如果当前你的网络中没有运行任何Elasticsearch节点，这时启动一个节点，会默认创建并加入一个叫做"elasticsearch"的集群。

6.1.3 分片和复制 shards&replicas

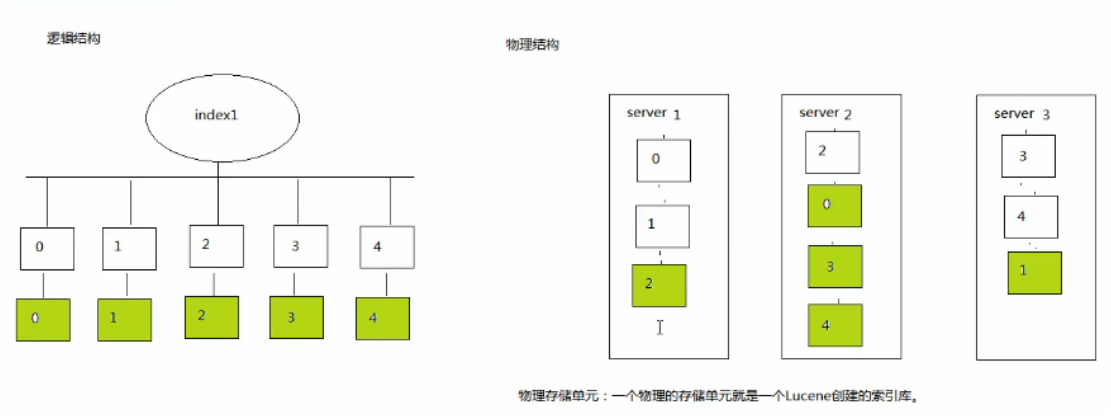
一个索引可以存储超出单个结点硬件限制的大量数据。比如，一个具有10亿文档的索引占据1TB的磁盘空间，而任一节点都没有这样大的磁盘空间；或者单个节点处理搜索请求，响应太慢。为了解决这个问题，Elasticsearch提供了将索引划分成多份的能力，这些份就叫做分片。当你创建一个索引的时候，你可以指定你想要的分片的数量。每个分片本身也是一个功能完善并且独立的"索引"，这个"索引"可以被放置到集群中的任何节点上。分片很重要，主要有两方面的原因： 1）允许你水平分割/扩展你的内容容量。 2）允许你在分片（潜在地，位于多个节点上）之上进行分布式的、并行的操作，进而提高性能/吞吐量。

至于一个分片怎样分布，它的文档怎样聚合回搜索请求，是完全由Elasticsearch管理的，对于作为用户的你来说，这些都是透明的。

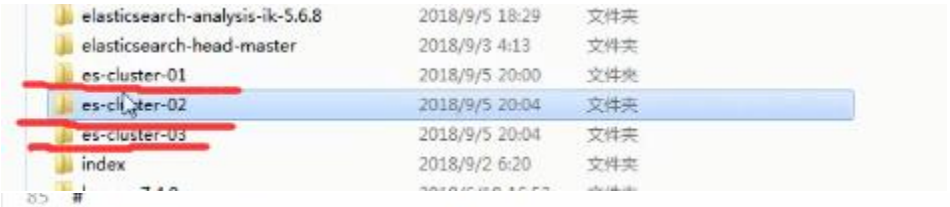
在一个网络/云的环境里，失败随时都可能发生，在某个分片/节点不知怎么的就处于离线状态，或者由于任何原因消失了，这种情况下，有一个故障转移机制是非常有用并且是强烈推荐的。为此目的，Elasticsearch允许你创建分片的一份或多份拷贝，这些拷贝叫做复制分片，或者直接叫复制。

复制之所以重要，有两个主要原因： 在分片/节点失败的情况下，提供了高可用性。因为这个原因，注意到复制分片从不与原/主要（original/primary）分片置于同一节点上是非常重要的。扩展你的搜索量/吞吐量，因为搜索可以在所有的复制上并行运行。总之，每个索引可以被分成多个分片。一个索引也可以被复制0次（意思是没有复制）或多次。一旦复制了，每个索引就有了主分片（作为复制源的原来的分片）和复制分片（主分片的拷贝）之别。分片和复制的数量可以在索引创建的时候指定。在索引创建之后，你可以在任何时候动态地改变复制的数量，但是你事后不能改变分片的数量。

默认情况下，Elasticsearch中的每个索引被分片5个主分片和1个复制，这意味着，如果你的集群中至少有两个节点，你的索引将会有5个主分片和另外5个复制分片（1个完全拷贝），这样的话每个索引总共就有10个分片。



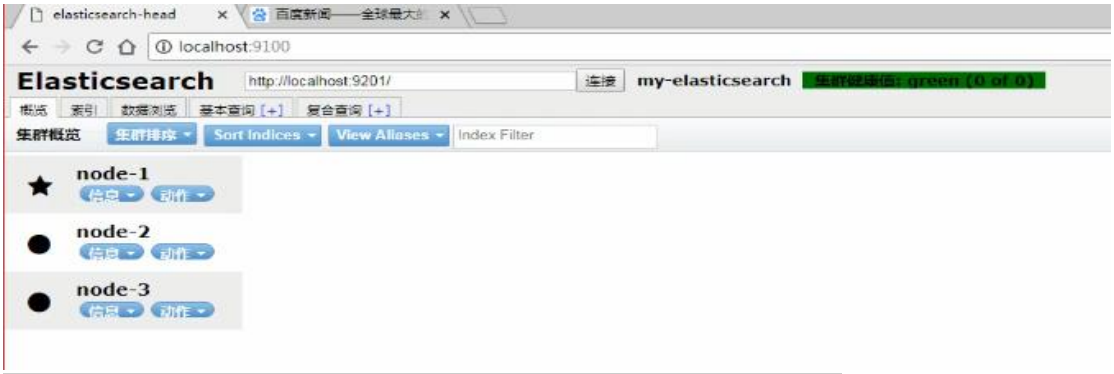
搭建集群的时候如果采用复制现有节点的话要保证把 data 目录删除（data 是我们把数据放入这个节点的时候它自动创建的，里面有多个 lucence 索引库）。




```
#
# Require explicit names when deleting indices:
#
#action.destructive_requires_name: true
http.cors.enabled: true
http.cors.allow-origin: "*"

#节点1的配置信息:
#集群名称，保证唯一
cluster.name: my-elasticsearch
#节点名称，必须不一样
node.name: node-1
#必须为本机的ip地址
network.host: 127.0.0.1
#服务端口号，在同一机器下必须不一样
http.port: 9201
#集群间通信端口号，在同一机器下必须不一样
transport.tcp.port: 9301
#设置集群自动发现机器ip集合
discovery.zen.ping.unicast.hosts: ["127.0.0.1:9301","127.0.0.1:9302","127.0.0.1:9303"]
```

之后在 config 中的 yml 文件内配置集群



这里分片 5 片，副本 1，表示每个分片有一个副本，因此总共有 10 片

```
一、使用Java客户端管理ES
1、创建索引库
步骤：
        1）创建一个Java工程
        2）添加jar包，添加maven的坐标
        3）编写测试方法实现创建索引库
                1、创建一个Settings对象，相当于是一个配置信息。主要配置集群的名称。
                2、创建一个客户端Client对象
                3、使用client对象创建一个索引库
                4、关闭client对象
2、使用Java客户端设置Mappings
        步骤：
        1）创建一个Settings对象
        2）创建一个Client对象
        3）创建一个mapping信息，应该是一个json数据，可以是字符串，也可以是XContextBuilder对象
        4）使用client向es服务器发送mapping信息
        5）关闭client对象
3、添加文档
        步骤：
        1）创建一个Settings对象
        2）创建一个Client对象
        3）创建一个文档对象，创建一个json格式的字符串，或者使用XContentBuilder
        4）使用Client对象吧文档添加到索引库中
        5）关闭client
4、添加文档第二种方式
        创建一个pojo类
        使用工具类把pojo转换成json字符串
        把文档写入索引库
```

```
Test
public void createIndex() throws Exception {
    //1、创建一个Settings对象，相当于是一个配置信息。主要配置集群的名称。
    Settings settings = Settings.builder()
            .put("cluster.name", "my-elasticsearch")
            .build();
    //2、创建一个客户端Client对象
    TransportClient client = new PreBuiltTransportClient(settings);
    client.addTransportAddress(new InetSocketTransportAddress(InetAddress.getByName("127.0.0.1"), port: 9301));
    client.addTransportAddress(new InetSocketTransportAddress(InetAddress.getByName("127.0.0.1"), port: 9302));
    client.addTransportAddress(new InetSocketTransportAddress(InetAddress.getByName("127.0.0.1"), port: 9303));
    //3、使用client对象创建一个索引库
    client.admin().indices().prepareCreate( s: "index_hello")
            //执行操作
            .get();
    //4、关闭client对象
    client.close();
```

以前的 Java 调用都是 TransportClient

```
XContentBuilder builder = XContentFactory.jsonBuilder()
        .startObject()
            .startObject("article")
                .startObject("properties")
                    .startObject("id")
                        .field( name: "type", value: "long")
                        .field( name: "store",  value: true)
                    .endObject()
                    .startObject("title")
                        .field( name: "type",  value: "text")
                        .field( name: "store",  value: true)
                        .field( name: "analyzer",  value: "ik_smart")
                    .endObject()
                    .startObject("content")
                        .field( name: "type",  value: "text")
                        .field( name: "store",  value: true)
                        .field( name: "analyzer", value: "ik_smart")
                    .endObject()
                .endObject()
            .endObject()
        .endObject();
```

创建 es 中的 mappings（在命令行模式下它是 json 形式的，因此用 java 构建 json 就要用上述写法）

```
//使用client把mapping信息设置到索引库中
client.admin().indices()
        //设置要做映射的索引
        .preparePutMapping( ...strings: "index_hello")
        //设置要做映射的type
        .setType("article")
        //mapping信息，可以是XContentBuilder对象可以是json格式的字符串
        .setSource(builder)
        //执行操作
        .get();
//关闭链接
client.close();
```

```
@Test
public void testAddDocument() throws Exception {
    //创建一个client对象
    //创建一个文档对象
    XContentBuilder builder = XContentFactory.jsonBuilder()
            .startObject()
                .field( name: "id", value: 11)
                .field( name: "title", value: "北方入秋速度明显加快 多地降温幅度最多可达10度")
                .field( name: "content",  value: "阿联酋一架客机在纽约机场被隔离 10名乘客病倒")
            .endObject();
    //把文档对象添加到索引库
    client.prepareIndex()
            //设置索引名称
            .setIndex("index_hello")
            //设置type
            .setType("article")
            //设置文档的id，如果不设置的话自动的生成一个id
            .setId("1")
            //设置文档信息
            .setSource(builder)
            //执行操作
            .get();
```

```
@Test
public void testAddDocument2() throws Exception {
    //创建一个Article对象
    Article article = new Article();
    //设置对象的属性
    article.setId(31);
    article.setTitle("MH370坠毁在柬埔寨密林?中国一公司调十颗卫星去拍摄");
    article.setContent("警惕荒唐的死亡游戏!俄15岁少年输掉游戏后用电锯自杀");
    //把article对象转换成json格式的字符串。
    ObjectMapper objectMapper = new ObjectMapper();
    String jsonDocument = objectMapper.writeValueAsString(article);
    System.out.println(jsonDocument);
    //使用client对象把文档写入索引库
    client.prepareIndex( index: "index_hello", type: "article",  id: "3")
            .setSource(jsonDocument, XContentType.JSON)
            .get();
    //关闭客户端
    client.close();
}
}
```

使用 Jackson 将对象转化成 json

二、使用es客户端实现搜索

1、根据id搜索

```
QueryBuilder queryBuilder = QueryBuilders.idsQuery().addIds("1", "2");
```

2、根据Term查询（关键词）

```
QueryBuilder queryBuilder = QueryBuilders.termQuery("title", "北方");
```

3、QueryString查询方式（带分析的查询）

```
QueryBuilder queryBuilder = QueryBuilders.queryStringQuery("速度与激情")
        .defaultField("title");
```

查询步骤：

1）创建一个Client对象
2）创建一个查询对象，可以使用QueryBuilders工具类创建QueryBuilder对象。
3）使用client执行查询
4）得到查询的结果。
5）取查询结果的总记录数
6）取查询结果列表
7）关闭client

4、分页的处理

在client对象执行查询之前，设置分页信息。
然后再执行查询

```
//执行查询
SearchResponse searchResponse = client.prepareSearch("index_hello")
        .setTypes("article")
        .setQuery(queryBuilder)
        //设置分页信息
        .setFrom(0)
        //每页显示的行数
        .setSize(5)
        .get();
```

分页需要设置两个值，一个from、size
from：起始的行号，从0开始。
size：每页显示的记录数

5、查询结果高亮显示

（1）高亮的配置

1）设置高亮显示的字段
2）设置高亮显示的前缀
3）设置高亮显示的后缀

（2）在client对象执行查询之前，设置高亮显示的信息。
（3）遍历结果列表时可以从结果中取高亮结果。

ElasticSearchClientTest.java | SearchIndex.java | Article.java

```
    //创建一个client对象
    //创建一个查询对象
    QueryBuilder queryBuilder = QueryBuilders.idsQuery().addIds("1", "2");
    //执行查询
    SearchResponse searchResponse = client.prepareSearch( ...indices: "index_hello")
            .setTypes("article")
            .setQuery(queryBuilder)
            .get();
    //取查询结果
    SearchHits searchHits = searchResponse.getHits();
    //取查询结果的总记录数
    System.out.println("查询结果总记录数：" + searchHits.getTotalHits());
    //查询结果列表
    Iterator<SearchHit> iterator = searchHits.iterator();
    while(iterator.hasNext()) {
        SearchHit searchHit = iterator.next();
        System.out.println(searchHit.getSourceAsString());

    }
    //关闭client
}

}
```

```
    Iterator<SearchHit> iterator = searchHits.iterator();
    while(iterator.hasNext()) {
        SearchHit searchHit = iterator.next();
        //打印文档对象，以json格式输出
        System.out.println(searchHit.getSourceAsString());
        //取文档的属性
        System.out.println("------------文档的属性");
        Map<String, Object> document = searchHit.getSource();
        System.out.println(document.get("id"));
        System.out.println(document.get("title"));
        System.out.println(document.get("content"));

    }
    //关闭client
    client.close();
}
```

```
    @Test
    public void testQueryStringQuery() throws Exception {
        //创建一个QueryBuilder对象
        QueryBuilder queryBuilder = QueryBuilders.queryStringQuery("速度与激情")
                .defaultField("title");
        //执行查询
        search(queryBuilder);
    }
```

```
private void search(QueryBuilder queryBuilder, String highlightField)
    HighlightBuilder highlightBuilder = new HighlightBuilder();
    //高亮显示的字段
    highlightBuilder.field("title");
    highlightBuilder.preTags("<em>");
    highlightBuilder.postTags("</em>");
    //执行查询
```

```
    System.out.println("************高亮结果");
    Map<String, HighlightField> highlightFields = searchHit.getHighlightFields();
    System.out.println(highlightFields);
    //取title高亮显示的结果
    HighlightField field = highlightFields.get(highlightField);
    Text[] fragments = field.getFragments();
    if (fragments != null) {
        String title = fragments[0].toString();
        System.out.println(title);
    }
```

第三章 Spring Data ElasticSearch 使用

3.1 Spring Data ElasticSearch简介

3.1.1 什么是Spring Data

Spring Data是一个用于简化数据库访问，并支持云服务的开源框架。其主要目标是使得对数据的访问变得方便快捷，并支持map-reduce框架和云计算数据服务。 Spring Data可以极大的简化JPA的写法，可以在几乎不用写实现的情况下，实现对数据的访问和操作。除了CRUD外，还包括如分页、排序等一些常用的功能。

Spring Data的官网：http://projects.spring.io/spring-data/

Spring Data常用的功能模块如下：

- Spring Data Hazelcast - Provides Spring Data repository support for Hazelcast.
- Spring Data Jest - Spring Data for Elasticsearch based on the Jest REST client.
- Spring Data Neo4j - Spring based, object-graph support and repositories for Neo4j.
- Spring Data Spanner - Google Spanner support via Spring Cloud GCP.
- Spring Data Vault - Vault repositories built on top of Spring Data KeyValue.

3.1.2 什么是Spring Data ElasticSearch

Spring Data ElasticSearch 基于 spring data API 简化 elasticSearch操作，将原始操作elasticSearch的客户端API进行封装 。Spring Data为Elasticsearch项目提供集成搜索引擎。Spring Data Elasticsearch POJO的关键功能区域为中心的模型与Elastichsearch交互文档和轻松地编写一个存储库数据访问层。

官方网站：http://projects.spring.io/spring-data-elasticsearch/