

一、安装elasticSearch

1. 下载镜像

```
1 docker pull elasticSearch
```

2. 启动容器

```
1 docker run -e ES_JAVA_OPTS="-Xms256m -Xmx256m" -d  
-p 9200:9200 -p 9300:9300 --name ES01  
elasticsearch
```

- -e ES_JAVA_OPTS="-Xms256m -Xmx256m" 限制 elasticsearch所占内存为256M
- 9200为 elasticSearch对外暴露的接口
- 9300为 elasticSearch内部通信所有的端口

3. 测试安装成功

访问地址：<http://47.105.103.45:9200>

出现如下界面即为elasticSearch安装启动成功

```
← → ↻ ⓘ 47.105.103.45:9200
{
  "name" : "mff7Tgq",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "u9229Qo3R8qoCkULQQNtwA",
  "version" : {
    "number" : "5.6.10",
    "build_hash" : "b727a60",
    "build_date" : "2018-06-06T15:48:34.860Z",
    "build_snapshot" : false,
    "lucene_version" : "6.6.1"
  },
  "tagline" : "You Know, for Search"
}
```

二、elasticSearch快速入门

官方文档：<https://www.elastic.co/guide/cn/elasticsearch/guide/current/index.html>

1.基本概念

1.1文档：

在ES中，使用 JavaScript Object Notation 或者 [JSON](#) 作为文档的序列化格式。一个json文档，代表一个对象。例：

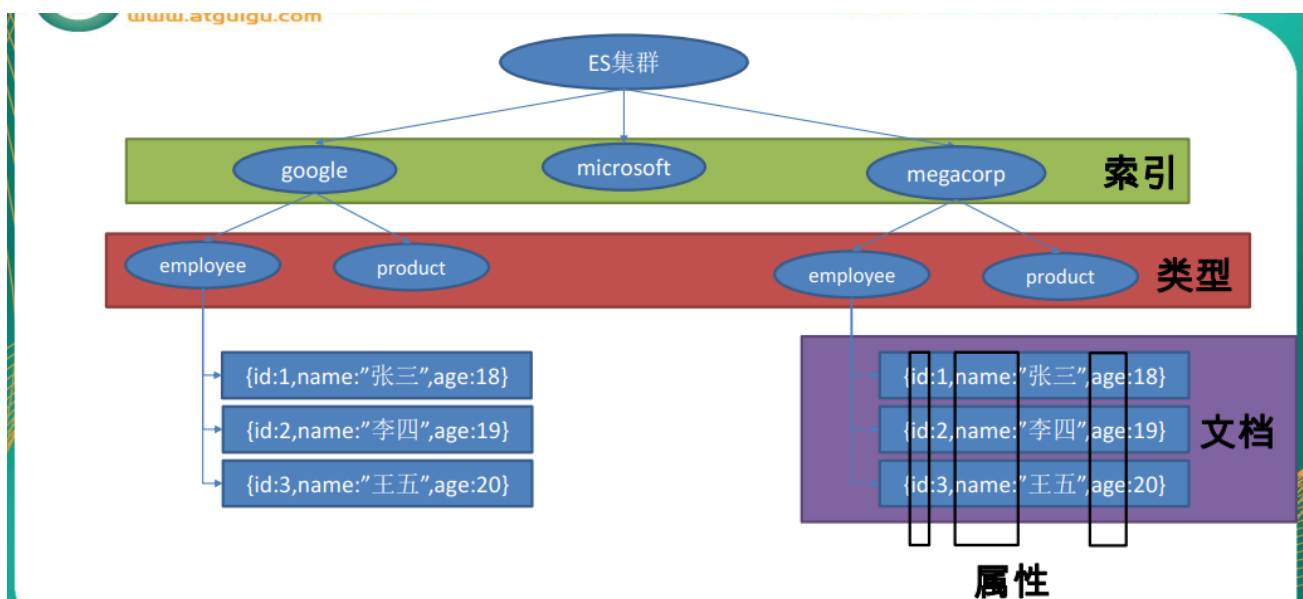
```
1 {
2     "email":      "john@smith.com",
3     "first_name": "John",
4     "last_name":  "Smith",
5     "info": {
6         "bio":      "Eco-warrior and defender
of the weak",
7         "age":      25,
8         "interests": [ "dolphins", "whales" ]
9     },
10    "join_date": "2014/05/01"
11 }
```

1.2索引：

第一个业务需求就是存储雇员数据。这将会以 *雇员文档* 的形式存储：**一个文档代表一个雇员。存储数据到 Elasticsearch 的行为叫做 索引**，但在索引一个文档之前，需要确定将文档存储在哪里。

1.2.1 ES中的层级结构

一个 Elasticsearch 集群可以包含多个 *索引*（这里索引的概念相当于mysql中的一个数据库），相应的每个索引可以包含多个 *类型*。这些不同的类型存储着多个 *文档*，每个文档又有多个 *属性*。



1.2.2 对索引的概念解释

索引这个词在 Elasticsearch 语境中包含多重意思，所以有必要做一点儿说明：

- 索引（名词）：

如前所述，一个索引类似于传统关系数据库中的一个数据库，是一个存储关系型文档的地方。索引(index)的复数词为 *indices* 或 *indexes*。

- 索引（动词）：

索引一个文档就是存储一个文档到一个索引（名词）中以便它可以被检索和查询到。这非常类似于 SQL 语句中的 `INSERT` 关键词，除了文档已存在时新文档会替换旧文档情况之外。

- 倒排索引：

关系型数据库通过增加一个 *索引*/比如一个 B树 (B-tree) 索引 到指定的列上，以便提升数据检索速度。Elasticsearch 和 Lucene 使用了一个叫做 *倒排索引*的结构来达到相同的目的。

- +默认的，一个文档中的每一个属性都是 *被索引*的（有一个倒排索引）和可搜索的。一个没有倒排索引的属性是不能被搜索到的。

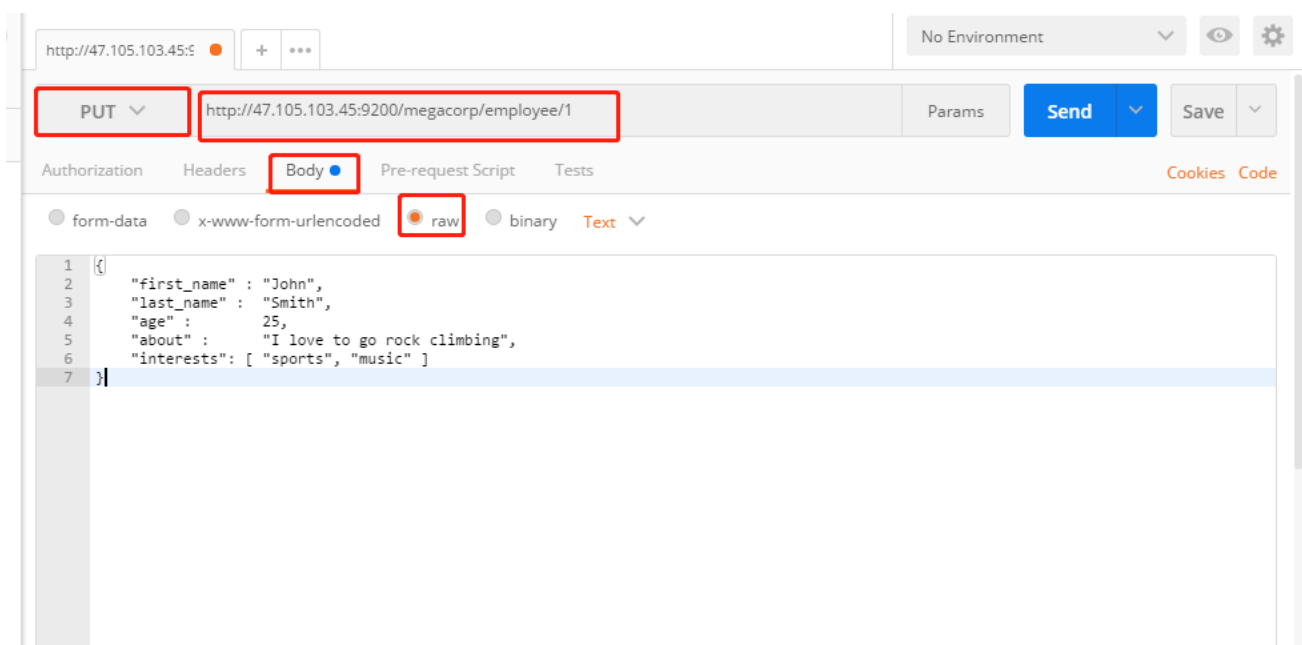
2.测试

2.1轻量搜索

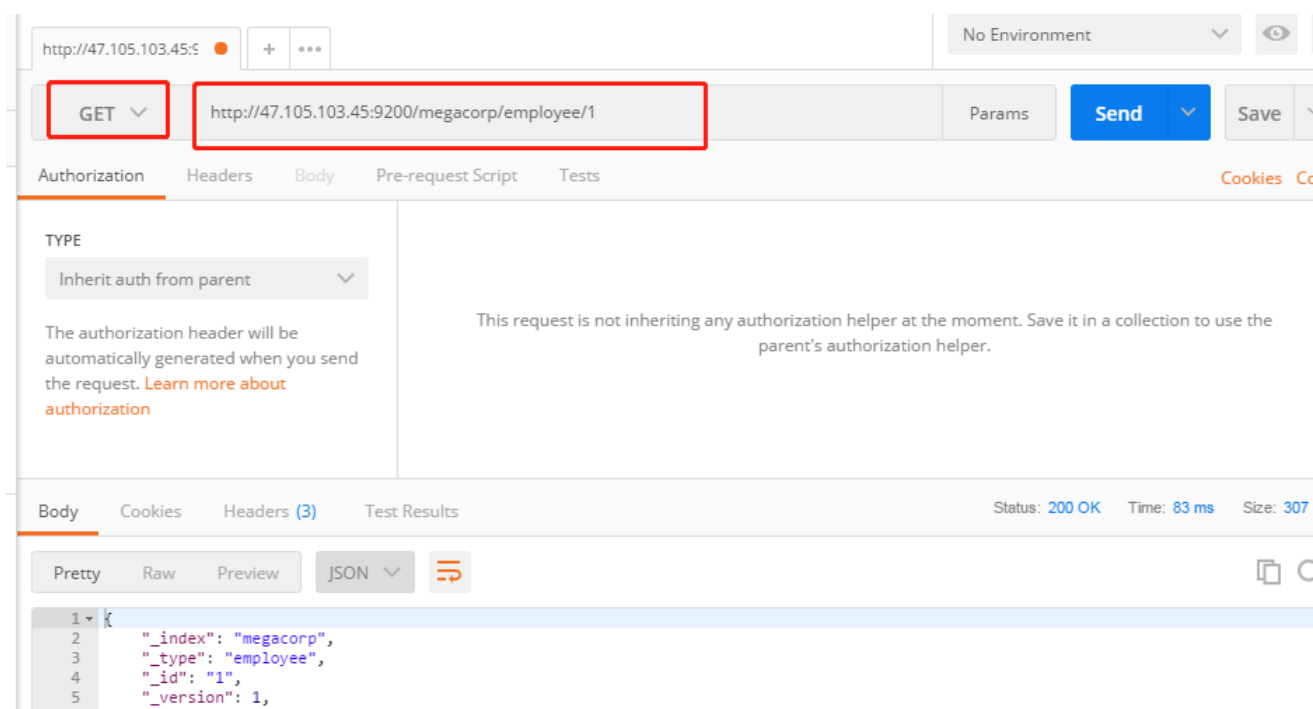
ES中分别支持PUT,GET,POST,DELETE,HEAD等请求。put请求用来插入和修改数据，get请求用来请求数据，delete请求用来删除数据，head请求用来判断请求是否存在。post请求用来支持复杂的查询。

测试地址：<http://47.105.103.45:9200/megacorp/employee/2>，我们需要指定文档的地址——索引库、类型和ID。其中 megacorp为索引库，employee为类型，2为文档ID。

2.1.1 put请求插入数据。

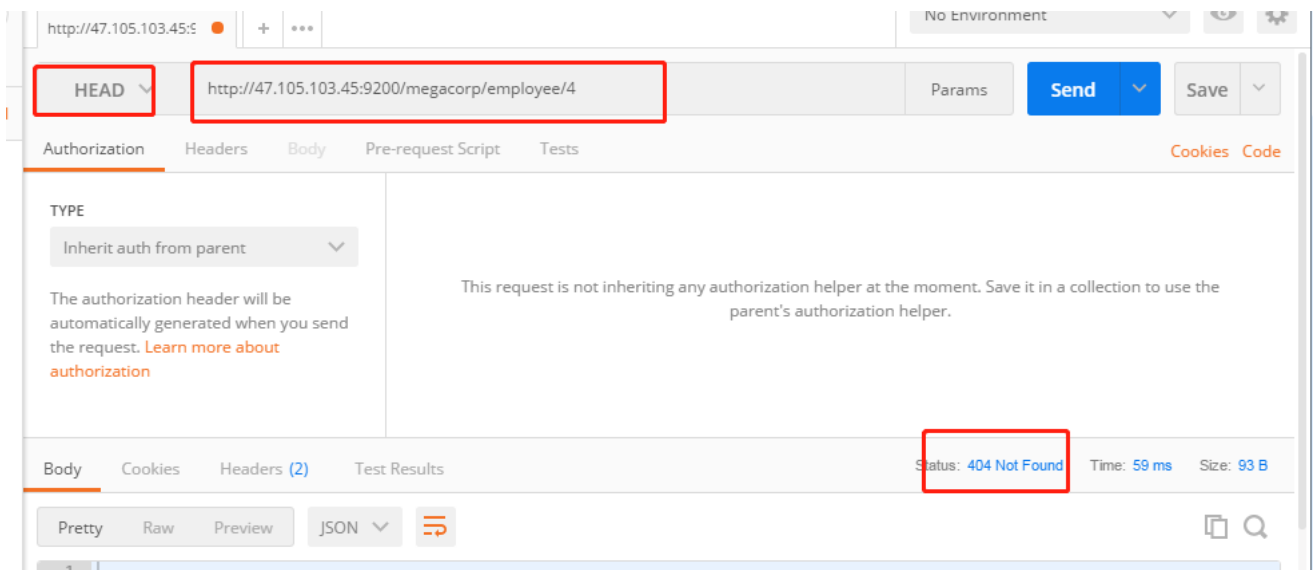
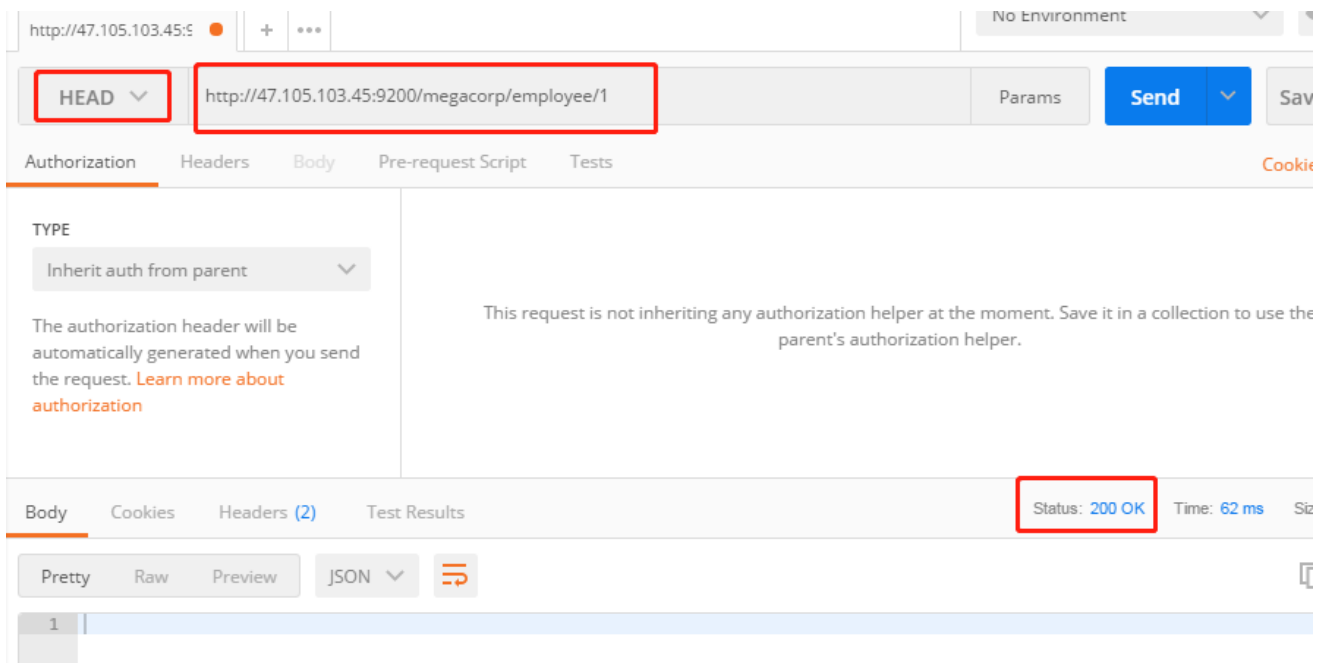


2.1.2 get请求进行查询



2.1.3 使用HEAD请求

HEAD请求确定数据是否存在。如果存在，返回状态码为200，不存在返回404



2.1.4 delete请求删除数据

The screenshot shows a REST client interface with a DELETE request to `http://47.105.103.45:9200/megacorp/employee/3`. The response status is 200 OK. The JSON response body is as follows:

```
{
  "found": true,
  "_index": "megacorp",
  "_type": "employee",
  "_id": "3",
  "_version": 2,
  "result": "deleted",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  }
}
```

2.1.5 PUT请求修改数据

The screenshot shows a REST client interface with a PUT request to `http://47.105.103.45:9200/megacorp/employee/2`. The request body is a JSON object:

```
{
  "first_name": "Douglas",
  "last_name": "Fir",
  "age": 23,
  "about": "I like to build cabinets",
  "interests": [ "forestry" ]
}
```

The response status is 200 OK. The JSON response body is as follows:

```
{
  "_index": "megacorp",
  "_type": "employee",
  "_id": "2",
  "_version": 2,
  "result": "updated",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  }
}
```

2.1.6 搜索所有的雇员


```
1 GET /megacorp/employee/_search
```

2.1.7 条件查询

```
1 GET /megacorp/employee/_search?  
  q=last_name:Smith
```

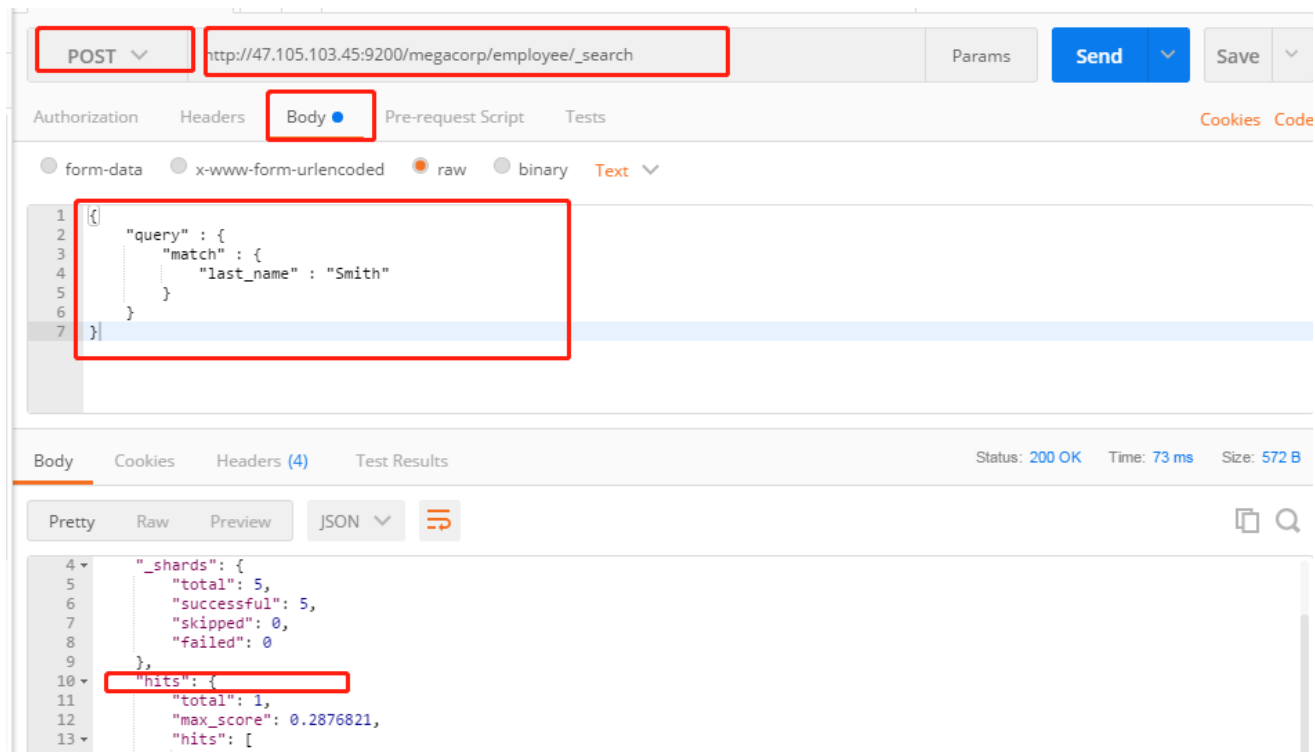
2.2 使用表达式查询

2.2.1 _search

使用查询表达式来进行搜索，同样是查询所有last_name为Smith的员工

```
1 GET /megacorp/employee/_search  
2 {  
3     "query" : {  
4         "match" : {  
5             "last_name" : "Smith"  
6         }  
7     }  
8 }
```

如图所示，因为get请求没有请求体，所以我们在这儿使用post请求，在body中放入查询表达式来查询所有last_name为Smith的员工，返回的数据结果放在hits中。



2.2.2 filter

查询last_name为smith并且年龄大于30岁的员工

```
1 GET /megacorp/employee/_search
2 {
3     "query" : {
4         "bool": {
5             "must": {
6                 "match" : {
7                     "last_name" : "smith"
8                 }
9             },
10            "filter": {
11                "range" : {
12                    "age" : { "gt" : 30 }
13                }
14            }
15        }
16    }
17 }
```

```
14         }
15     }
16 }
17 }
```

2.2.3 短语搜索

查询about中仅匹配同时包含“rock”和“climbing”，并且二者以短语“rock climbing”的形式紧挨着的雇员记录

```
1 GET /megacorp/employee/_search
2 {
3     "query" : {
4         "match_phrase" : {
5             "about" : "rock climbing"
6         }
7     }
8 }
```

2.2.3 高亮搜索

在刚才的查询结果中高亮显示about字段

```
1 GET /megacorp/employee/_search
2 {
3     "query" : {
4         "match_phrase" : {
5             "about" : "rock climbing"
6         }
7     },
8     "highlight": {
9         "fields" : {
10             "about" : {}
11         }
12     }
13 }
```

当执行该查询时，返回结果与之前一样，与此同时结果中还多了一个叫做 `highlight` 的部分。这个部分包含了 `about` 属性匹配的文本片段，并以 HTML 标签 `` 封装：

```
1     "hits": {
2         "total": 1,
3         "max_score": 0.53484553,
4         "hits": [
5             {
6                 "_index": "megacorp",
7                 "_type": "employee",
8                 "_id": "1",
9                 "_score": 0.53484553,
```

```
10         "_source": {
11             "first_name": "John",
12             "last_name": "Smith",
13             "age": 25,
14             "about": "I love to go rock
15 climbing",
16             "interests": [
17                 "sports",
18                 "music"
19             ],
20             "highlight": {
21                 "about": [
22                     "I love to go
23 <em>rock</em> <em>climbing</em>"
24                 ]
25             }
26         ]
27     }
```

三、springboot整合ES

使用spring初始化器新建项目，选中web模块和elasticSearch模块

springboot 默认提供两种技术和ES进行交互，分别是Jest和springData ElasticSearch,其中Jest是默认不生效的，需要导入Jest的工具包

1.Jest

jest 给我们提供了 JestClient 与 Es 进行交互。

jest 自动配置类：JestAutoConfiguration

1.1 MAVEN依赖

```
1 <!--  
  https://mvnrepository.com/artifact/io.searchbox/j  
  est -->  
2 <dependency>  
3     <groupId>io.searchbox</groupId>  
4     <artifactId>jest</artifactId>  
5     <version>5.3.3</version>  
6 </dependency>
```

1.2 配置文件

其中uris是一个数据

```
1 spring:
2   elasticsearch:
3     jest:
4       uris:
5         - http://47.105.103.45:9200
```

启动报错：java.lang.ClassNotFoundException:
com.sun.jna.Native，添加如下依赖即可

```
1 <dependency>
2   <groupId>com.sun.jna</groupId>
3   <artifactId>jna</artifactId>
4   <version>3.0.9</version>
5 </dependency>
```

1.3 测试程序

```
1 //实体类
2 public class Article {
3   //标注这个字段为主键
4   @JestId
5   private Integer id;
6   private String author;
7   private String title;
8   private String content;
9 }
```

```
1 //测试程序
2 import io.searchbox.core.Index;
3 @RunWith(SpringRunner.class)
4 @SpringBootTest
5 public class
6     SpringbootElasticSearchApplicationTests {
7     //
8     @Autowired
9     JestClient jestClient;
10
11     @Test
12     public void contextLoads() {
13         Article article = new Article();
14         article.setAuthor("lidonghao");
15         article.setContent("hello world");
16         article.setId(1);
17         article.setTitle("first");
18         //构建一个索引功能（即向ES中索引一个文档），
19         //将article放在索引haoge,类型news之下
20         Index build = new
21             Index.Builder(article).index("haoge").type("news
22             ").build();
23         try {
24             DocumentResult execute =
25             jestClient.execute(build);
26         } catch (IOException e) {
27             // TODO Auto-generated catch block
28             e.printStackTrace();
29         }
30     }
31 }
```



```

24         }
25         //执行完成之后，我们就向ES中索引了一个文档，
测试地址http://47.105.103.45:9200/haoge/news/1
26         //执行之后，我们可以看到我们刚才索引的文档数
据
27     }
28     //测试搜索
29     @Test
30     public void testSearch() {
31         String json="{\n" +
32             "    \"query\" : {\n" +
33             "        \"match\" : {\n" +
34             "            \"content\" :
35             \"hello\"\n" +
36             "        }\n" +
37             "    }\n" +
38             "    }";
39         //在haoge索引下，查询类型为news，字段中
content有hello的文档
40         Search build = new
Search.Builder(json).addIndex("haoge").addType("
news").build();
41         try {
42             SearchResult execute =
jestClient.execute(build);
43             System.out.println(execute.getJsonString());
44         } catch (IOException e) {

```

```
44          // TODO Auto-generated catch block
45          e.printStackTrace();
46      }
47  }
48 }
```

访问<http://47.105.103.45:9200/haoge/news/1>之后得到的数据，说明我们索引成功。

```
1 {
2     "_index": "haoge",
3     "_type": "news",
4     "_id": "1",
5     "_version": 1,
6     "found": true,
7     "_source": {
8         "id": 1,
9         "author": "lidonghao",
10        "title": "first",
11        "content": "hello world"
12    }
13 }
```

2.SpringData Elasticsearch

官方文档地址：<https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/>

- SpringData ElasticSearch需要配置节点信息：cluster-name和 cluster-nodes
- SpringData ElasticSearch分别为我们提供了ElasticsearchRepository接口和ElasticsearchTemplate来和ES进行交互

2.1 XML配置文件

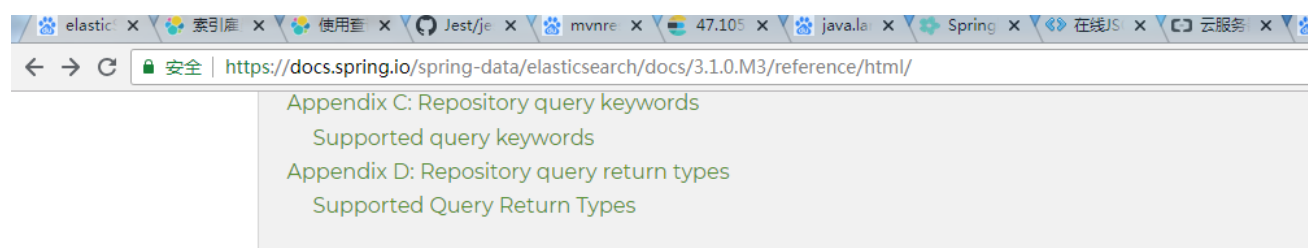
```
1  spring:
2      elasticsearch:
3          jest:
4              uris:
5                  - http://47.105.103.45:9200
6  #SpringData ElasticSearch相关的配置
7  #name取下图中标注的name
8      data:
9          elasticsearch:
10              cluster-name: elasticsearch
11              cluster-nodes: 47.105.103.45:9300
```

```
← → ↻ ⓘ 47.105.103.45:9200

{
  "name" : "mff7Tgq",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "u9229Qo3R8qoCkULQQNtwA",
  "version" : {
    "number" : "5.6.10",
    "build_hash" : "b727a60",
    "build_date" : "2018-06-06T15:48:34.860Z",
    "build_snapshot" : false,
    "lucene_version" : "6.6.1"
  },
  "tagline" : "You Know, for Search"
}
```

2.2 版本控制

查看版本对应<https://docs.spring.io/spring-data/elasticsearch/docs/3.1.0.M3/reference/html/>



Preface

The Spring Data Elasticsearch project applies core Spring concepts to the development of solutions using the Search Engine. We have provided a "template" as a high-level abstraction for storing, querying, sorting and faceting. You will notice similarities to the Spring data solr and mongodb support in the Spring Framework.

§ Project Metadata

- Version Control - <https://github.com/spring-projects/spring-data-elasticsearch>
- Bugtracker - <https://jira.spring.io/browse/DATAES>
- Release repository - <https://repo.spring.io/libs-release>
- Milestone repository - <https://repo.spring.io/libs-milestone>
- Snapshot repository - <https://repo.spring.io/libs-snapshot>

spring data elasticsearch	elasticsearch
3.1.x	6.2.2
3.0.x	5.5.0
2.1.x	2.4.0
2.0.x	2.2.0
1.3.x	1.5.2

The screenshot shows an IDE window with two panes. The left pane displays the 'Maven Dependencies' tree, where the entry 'spring-data-elasticsearch-2.1.13.RELEASE.jar' is highlighted with a red rectangular box. The right pane shows a stack trace for an exception. The top of the stack trace reads 'springboot-elasticSearch - SpringbootElast'. Below this, several lines of the stack trace are visible, including 'at org.elasti', 'at org.elasti', 'at java.util.', 'at java.util.', 'at java.lang.', and 'Caused by: java.net.C'. The stack trace continues with 'at sun.nio.ch', 'at sun.nio.ch', 'at org.jboss.', 'at org.jboss.', 'at org.jboss.', 'at org.jboss.', 'at org.jboss.', and 'at org.jboss.'.

```
← → ↻ ⓘ 47.105.103.45:9200

{
  "name" : "mff7Tgq",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "u9229Qo3R8qoCkULQQNtwA",
  "version" : {
    "number" : "5.6.10",
    "build_hash" : "b727a60",
    "build_date" : "2018-06-06T15:48:34.860Z",
    "build_snapshot" : false,
    "lucene_version" : "6.6.1"
  },
  "tagline" : "You Know, for Search"
}
```

如图所示，我们的ES版本和SpringData ES版本不对应。所以启动报错。这个时候我们可以更换springboot的版本或者重新安装适配的ES版本

重新安装2.4.0的ES。

- 1 `docker run -e ES_JAVA_OPTS="-Xms256m -Xmx256m" -d -p 9201:9200 -p 9301:9300 --name ES02 bc337c8e4f39`
- 2 #将服务器的9201映射到docker容器中的9200

```
1 //标注这个文档存储的索引位置和类型
2 @Document(indexName="haoge",type="book")
3 public class Book {
4     //标注这个字段为主键
5     private Integer id;
6     private String bookName;
7     private String author;
8 }
```

2.3 ElasticsearchRepository测试

```
1 //ElasticsearchRepository为spring data
  elasticsearch提供的接口，其中有增删改查的基本方法
2 public interface BookRepository extends
  ElasticsearchRepository<Book, Integer>{
3
4     public List<Book> findByBookNameLike(String
  bookName);
5 }
```

测试：

```
1 @Autowired
2 BookRepository bookRepository;
3
4     //测试搜索
5     @Test
6     public void testRepository() {
```

```
7 //      Book book = new Book();
8 //      book.setId(1);
9 //      book.setBookName("西游记");
10 //      book.setAuthor("李东浩");
11 //      //向ES中索引一个文档
12 //      bookRepository.index(book);
13
14      List<Book> findByBookNameLike =
bookRepository.findByBookNameLike("游");
15      for (Book book : findByBookNameLike) {
16          System.out.println(book);
17      }
18 }
```

2.4 elasticsearchTemplate测试


```
1 @Autowired
2 ElasticsearchTemplate elasticsearchTemplate;
3     //向ES中索引一个文档
4     @Test
5     public void testTemplate() {
6         Book book = new Book();
7         book.setId(2);
8         book.setBookName("东游记");
9         book.setAuthor("李西浩");
10
11         IndexQuery indexQuery = new
12         IndexQueryBuilder().withId(book.getId().toString
13         ()).withObject(book).build();
14         elasticsearchTemplate.index(indexQuery);
15     }
```

四、springboot和任务

1.异步任务

1.1@EnableAsync

使用springboot的异步任务，即springboot自动使用多线程执行程序，我们需要在主程序上加@EnableAsync开启异步任务

```
1 @SpringBootApplication
2 @EnableAsync
3 public class SpringbootElasticSearchApplication {
4
5     public static void main(String[] args) {
6
7         SpringApplication.run(SpringbootElasticSearchApplic
8             ication.class, args);
9     }
10 }
```

1.2 @Async

在相应的方法上加@Async注解

```
1 @Service
2 public class AsyncService {
3     @Async
4     public void hello() {
5         try {
6             Thread.sleep(3000);
7         } catch (InterruptedException e) {
8             // TODO Auto-generated catch block
9             e.printStackTrace();
10        }
11        System.out.println("调用成功。。。");
12    }
13 }
```

测试地址：<http://localhost:8080/hello>

2.定时任务

2.1@EnableScheduling

使用@EnableScheduling开启springboot的基于注解的定时任务

```
1 @SpringBootApplication
2 @EnableAsync//允许开启基于注解的异步任务
3 @EnableScheduling//允许开启基于注解的定时任务
4 public class SpringbootElasticSearchApplication {
5
6     public static void main(String[] args) {
7
8         SpringApplication.run(SpringbootElasticSearchApplication.class, args);
9     }
10 }
```

2.2@Scheduled

使用@Scheduled标注方法并且指定cron属性规定方法定时执行的时间

```

1 @Service
2 public class ScheduledService {
3     /**
4      * second(秒), minute(分), hour(时), day of
      month(日), month(月), day of week(周)
5      */
6     @Scheduled(cron="0 * * * * MON-FRI")//周一到
      周五的每秒都执行
7     public void hello() {
8         System.out.println("hello...");
9     }
10 }

```

2.3 cron表达式规则

cron中的六个位置分别代表：秒，分，时，日，月，周

字段	允许值	允许的特殊字符
秒	0-59	, - * /
分	0-59	, - * /
小时	0-23	, - * /
日期	1-31	, - * ? / L W C
月份	1-12	, - * /
星期	0-7或SUN-SAT 0,7是SUN	, - * ? / L C #

星期位置上：1-6分别代表周一至周六。0和7都可以代表周日

特殊字符	代表含义
,	枚举。
-	区间
*	任意
/	步长
?	日/星期冲突匹配的时候，如果指定日，则星期位置用？，反之一样。
L	最后
W	工作日
C	和calendar联系后计算过的值
#	第几，4#2，表示第2个星期四

例：

cron="1,2,3 * * * * MON-FRI" 表示1,2,3秒都执行（枚举）

cron="1-3 * * * * MON-FRI" 表示1,2,3秒都执行（区间）

cron="0/4 * * * * MON-FRI" 表示0秒开始，每4秒执行一次（步长）

cron="0 15 10 ? * 1-6" 表示每个月周一到周六每天10:15分执行一次

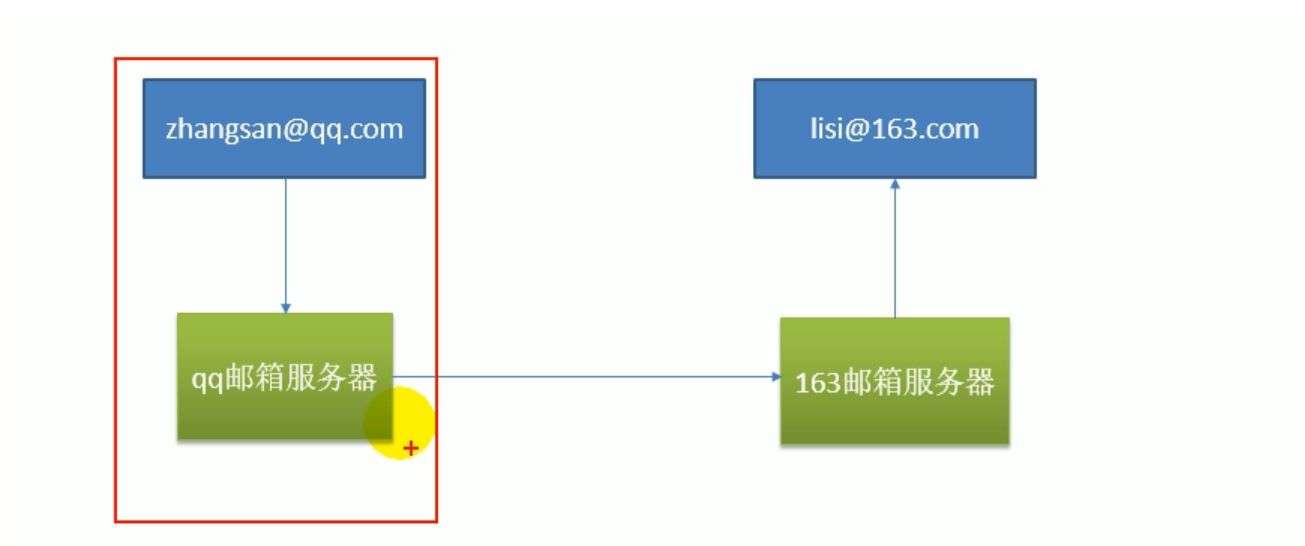
cron="0 0 2 ? * 6L" 表示每个月最后一个周六凌晨2点执行一次

cron="0 0 2 LW * ?" 表示每个月最后一个工作日凌晨2点执行一次

cron="0 0 2-4 ? * 1#1" 表示每个月第一个周一凌晨2点到4点整点各执行一次

3.邮件任务

不同邮箱之间发邮件的过程



3.1maven依赖

```
1 <dependency>
2
3   <groupId>org.springframework.boot</groupId>
4     <artifactId>spring-boot-starter-
mail</artifactId>
5 </dependency>
```

自动配置类：MailSenderAutoConfiguration

3.2 yaml配置

```
1 spring:
2   mail:
3     host: smtp.qq.com           #服务器地址
4     username: 861914994@qq.com  #QQ邮箱账号
5     password: lvkqnygbgns1bchh #QQ邮箱授权码
6     properties:
7       mail.smtp.ssl.enable: true #开启SSL
```

3.3 测试代码

```
1 @RunWith(SpringRunner.class)
2 @SpringBootTest
3 public class SpringbootMailTests {
4     @Autowired
5     JavaMailSenderImpl mailSender;
6
7     @Test
```

```
8      public void test01() {
9          //测试发送简单邮件
10         SimpleMailMessage message = new
SimpleMailMessage();
11         message.setSubject("测试邮件");//设置主题
12         message.setText("这是一份测试邮件。。");//
设置邮件内容
13         message.setTo("15810673938@163.com");//
设置发送目标
14         message.setFrom("861914994@qq.com");//设
置发送账号
15         mailSender.send(message);
16     }
17
18     @Test
19     public void test02() throws
MessagingException {
20         //测试发送复杂邮件
21         MimeMessage mineMessage =
mailSender.createMimeMessage();
22         MimeMessageHelper helper = new
MimeMessageHelper(mineMessage,true);//true表示允
许上传文件
23
24         helper.setSubject("测试邮件");//设置主题
25         helper.setText("<b style='color:red'>这是
一份测试邮件。。</b>",true);//设置邮件内容,允许使用
html
```



```
26         helper.setTo("15810673938@163.com");//设置发送目标
27         helper.setFrom("861914994@qq.com");//设置发送账号
28
29         helper.addAttachment("Chrysanthemum.jpg", new
File("C:\\\\Users\\\\Public\\\\Pictures\\\\Sample Pictures\\\\Chrysanthemum.jpg"));
30         helper.addAttachment("Desert.jpg", new
File("C:\\\\Users\\\\Public\\\\Pictures\\\\Sample Pictures\\\\Desert.jpg"));
31         mailSender.send(mineMessage);
32     }
```