一、安装elasticSearch

1.下载镜像

```
docker pull elasticSearch
```

2.启动容器

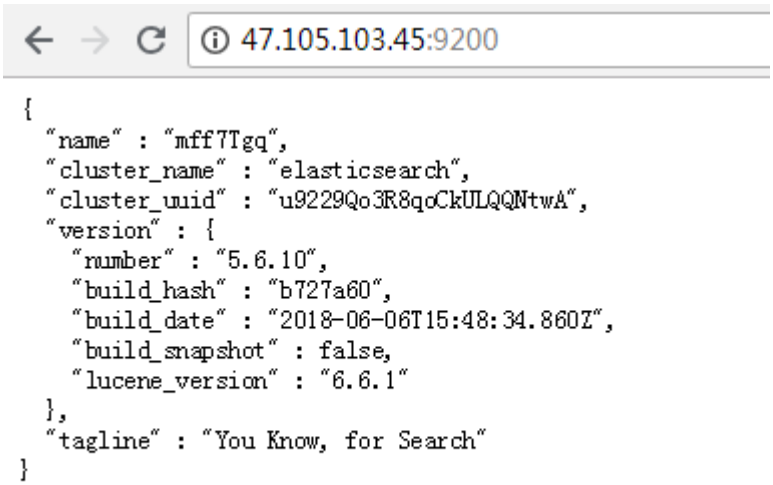
```
docker run -e ES_JAVA_OPTS="-Xms256m -Xmx256m" -d -p 9200:9200 -p 9300:9300 --name ES01 elasticsearch
```

- -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



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：

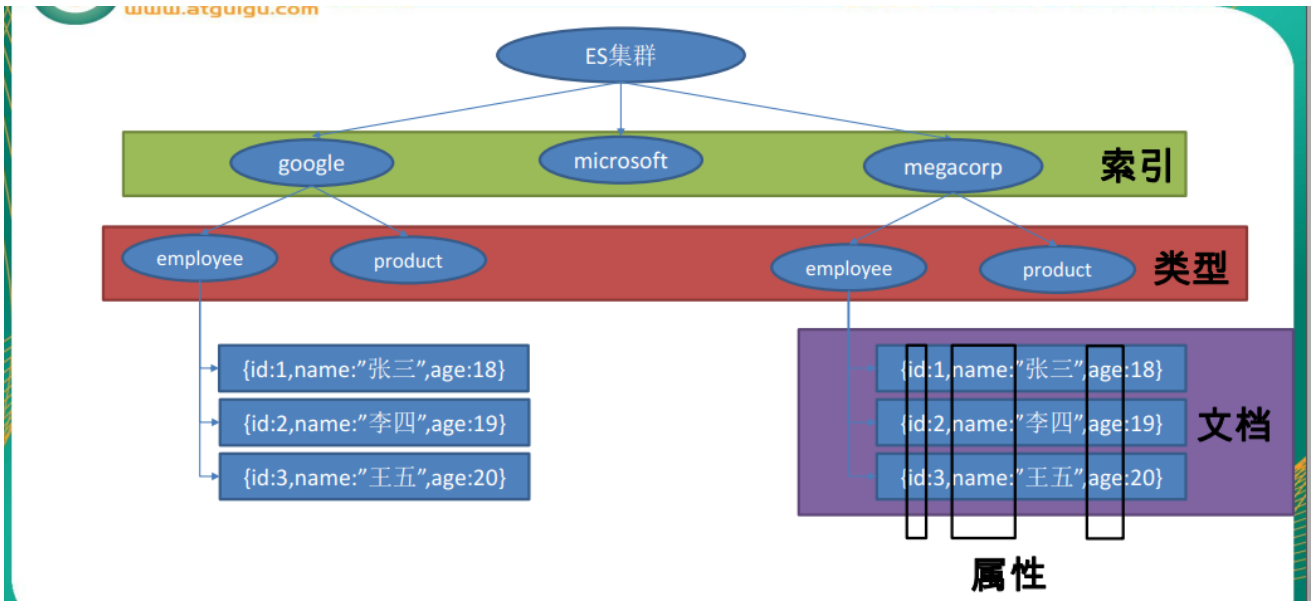
```
{
    "email":      "john@smith.com",
    "first_name": "John",
    "last_name":  "Smith",
    "info": {
        "bio":         "Eco-warrior and defender of the weak",
        "age":         25,
        "interests": [ "dolphins", "whales" ]
    },
    "join_date": "2014/05/01"
}
```

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

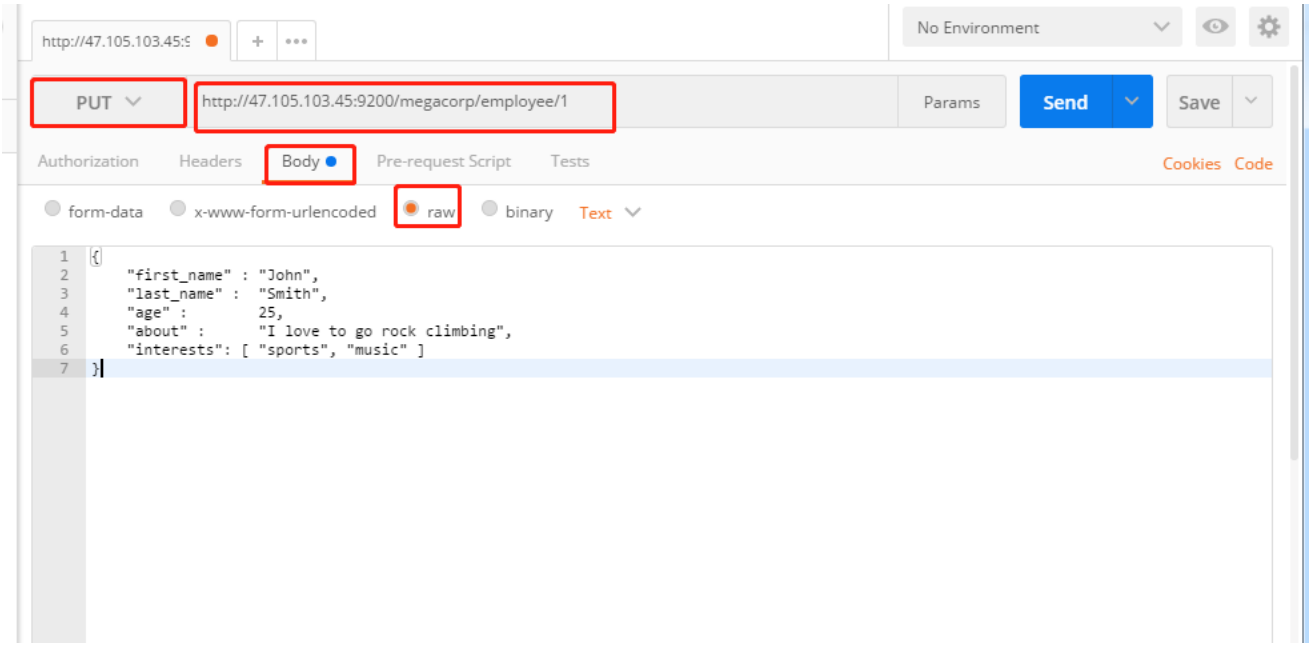
2.1轻量搜索

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。

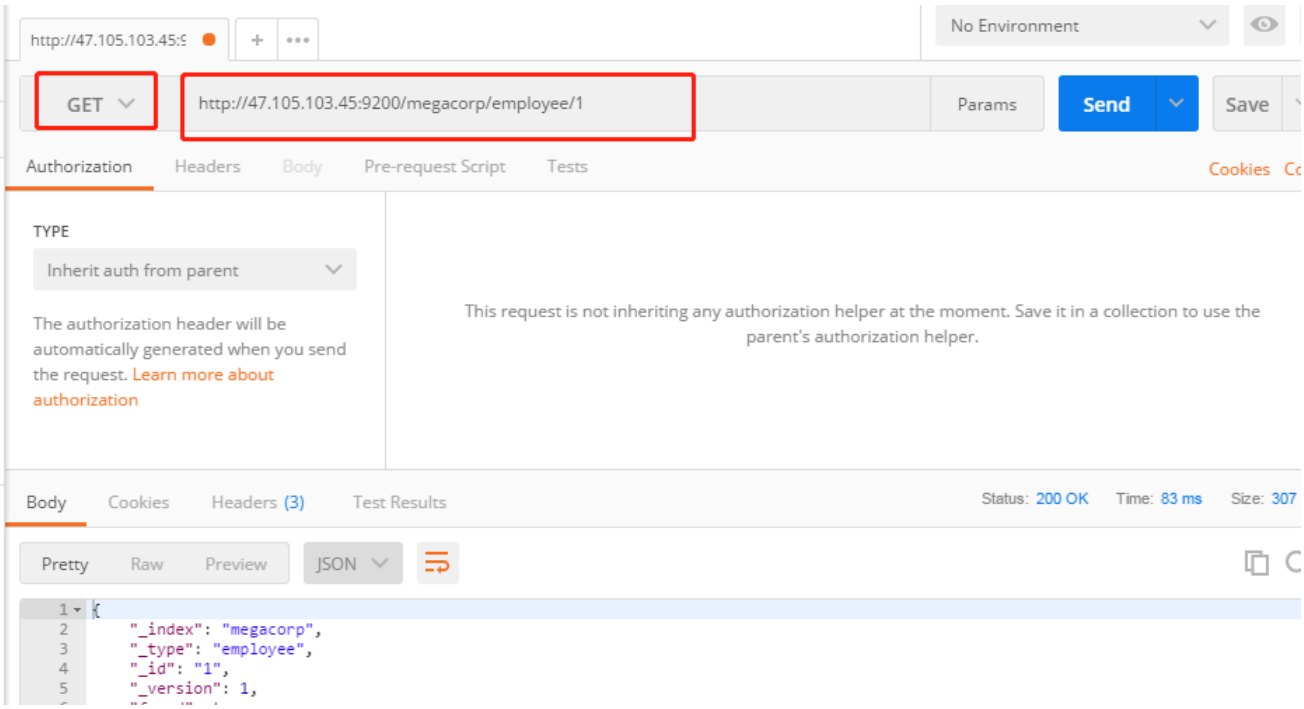
测试地址：http://47.105.103.45:9200/megacorp/employee/2 ，我们需要指定文档的地址——索引库、类型和ID 。其中

megacorp为索引库，employee为类型，2为文档ID.

2.1.1 put请求插入数据。

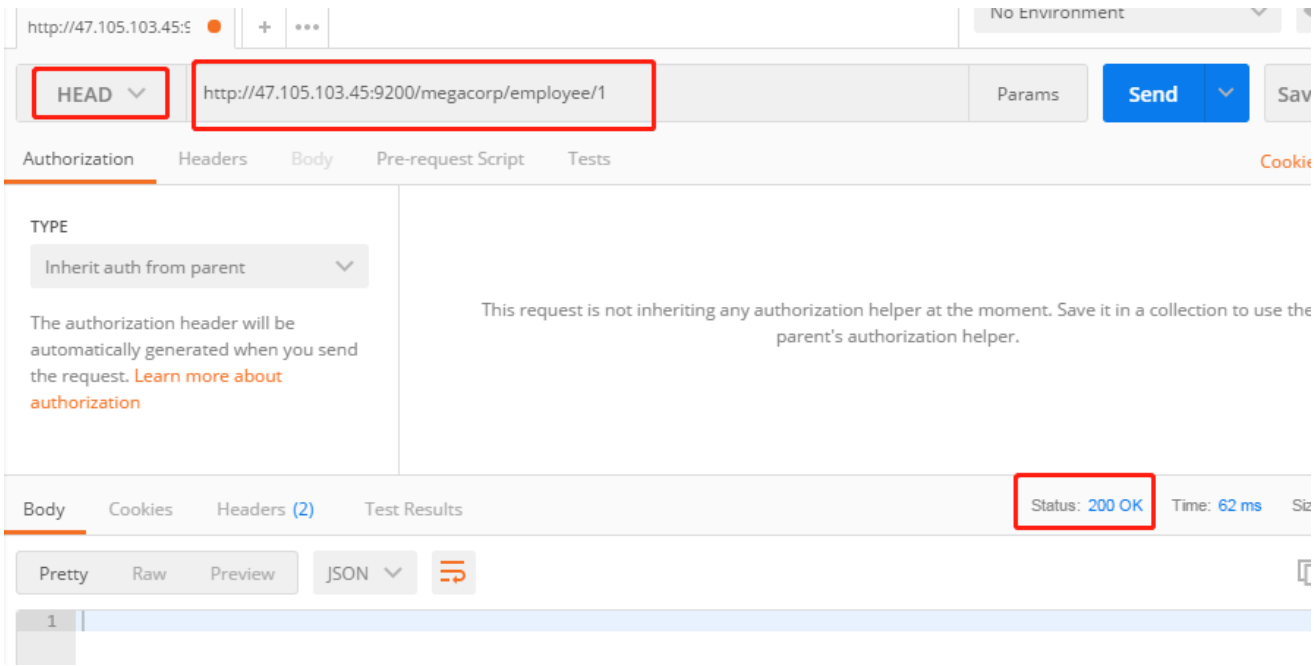


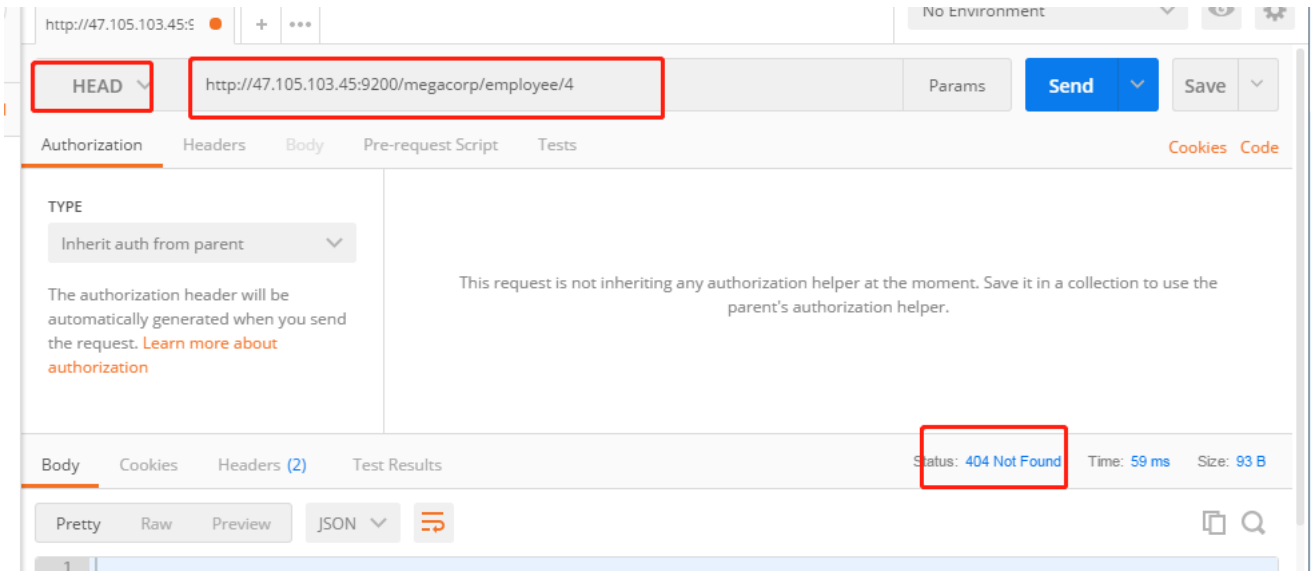
2.1.2 get请求进行查询



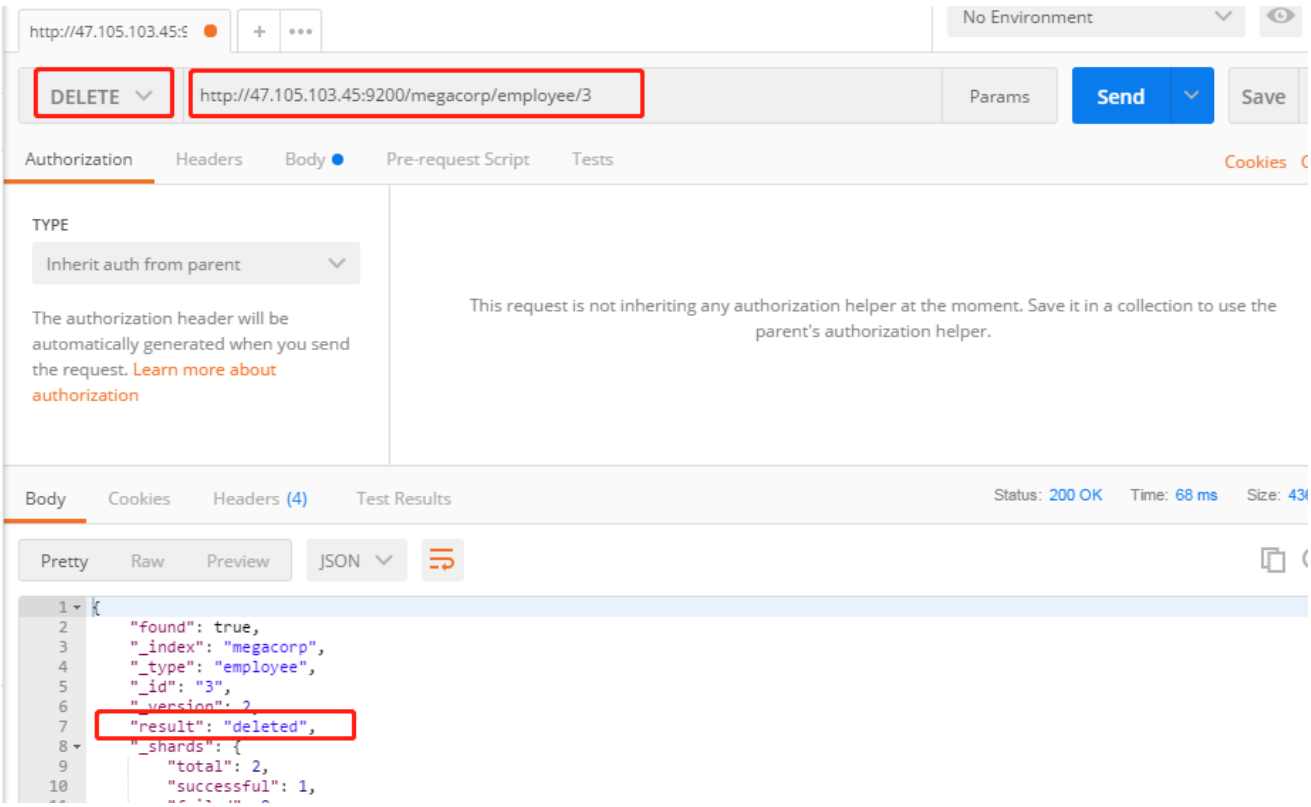
2.1.3 使用HEAD请求

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

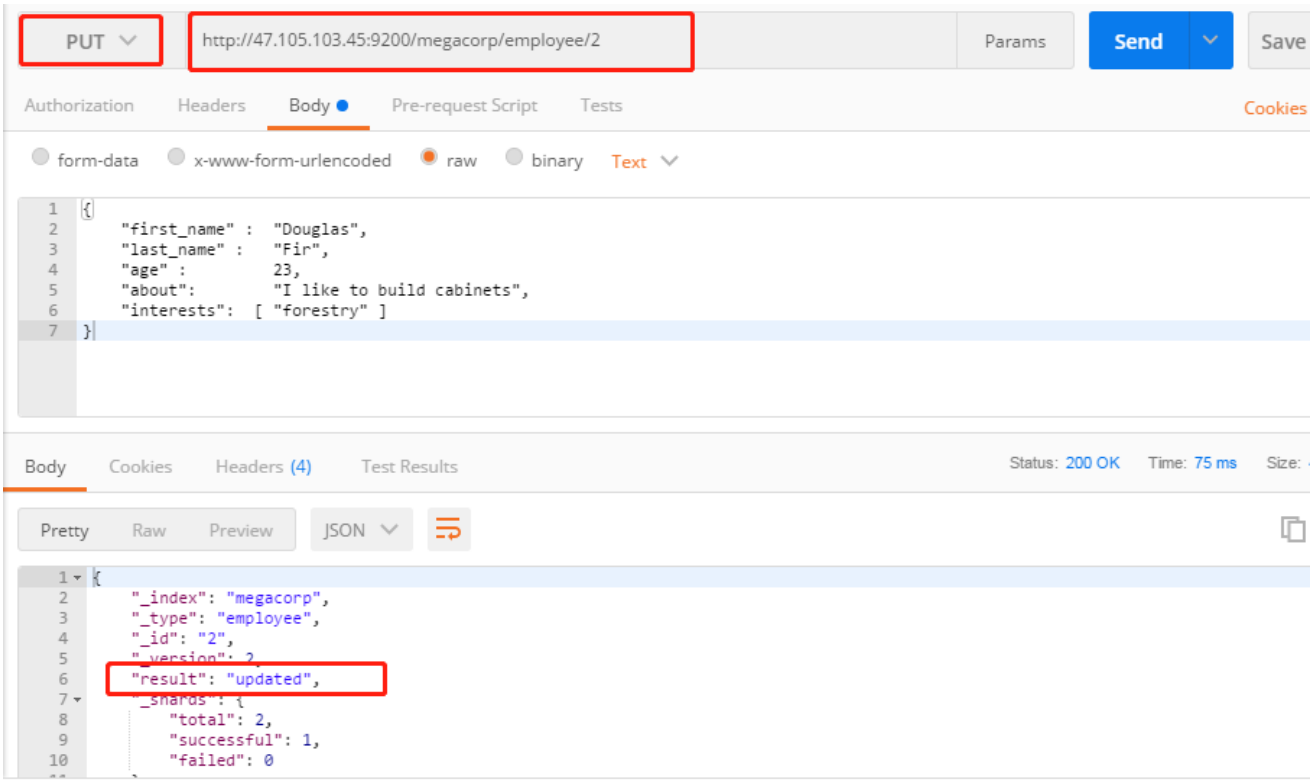




2.1.4 delete请求删除数据



2.1.5 PUT请求修改数据



2.1.6 搜索所有的雇员

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

2.1.7 条件查询

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

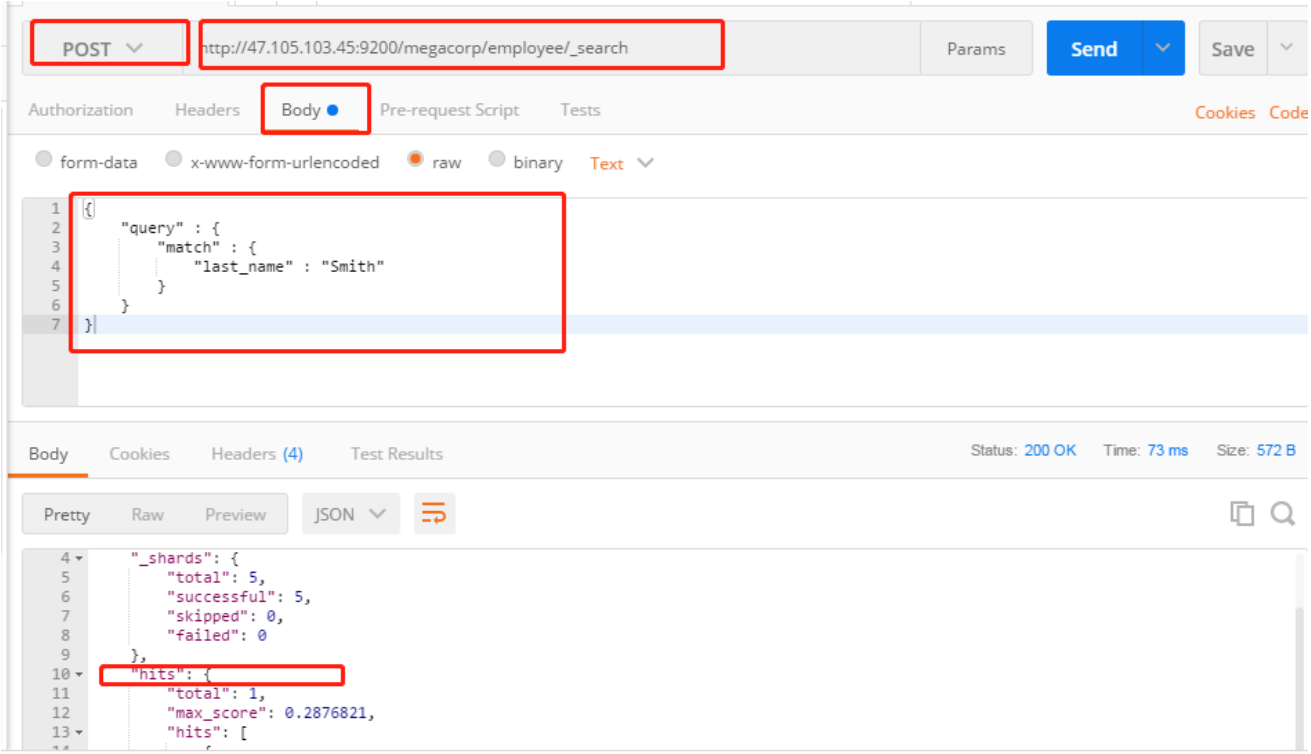
2.2 使用表达式查询

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

```
GET /megacorp/employee/_search
{
    "query" : {
        "match" : {
            "last_name" : "Smith"
        }
    }
}
```

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

```
GET /megacorp/employee/_search
{
    "query" : {
        "bool": {
            "must": {
                "match" : {
                    "last_name" : "smith"
                }
            },
            "filter": {
                "range" : {
                    "age" : { "gt" : 30 }
                }
```

```
            }
        }
    }
}
```

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

```
GET /megacorp/employee/_search
{
    "query" : {
        "match_phrase" : {
            "about" : "rock climbing"
        }
    }
}
```

2.2.3 高亮搜索

在刚才的查询结果中高亮显示about字段

```
GET /megacorp/employee/_search
{
    "query" : {
        "match_phrase" : {
            "about" : "rock climbing"
        }
    },
    "highlight": {
        "fields" : {
            "about" : {}
        }
    }
}
```

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 `<em></em>` 封装：

```
    "hits": {
        "total": 1,
        "max_score": 0.53484553,
        "hits": [
            {
                "_index": "megacorp",
                "_type": "employee",
                "_id": "1",
                "_score": 0.53484553,
```

```
                "_source": {
                    "first_name": "John",
                    "last_name": "Smith",
                    "age": 25,
                    "about": "I love to go rock climbing",
                    "interests": [
                        "sports",
                        "music"
                    ]
                },
                "highlight": {
                    "about": [
                        "I love to go <em>rock</em> <em>climbing</em>"
                    ]
                }
            }
        ]
    }
```

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

```
<!--
https://mvnrepository.com/artifact/io.searchbox/jest -->
<dependency>
    <groupId>io.searchbox</groupId>
    <artifactId>jest</artifactId>
    <version>5.3.3</version>
</dependency>
```

1.2 配置文件

其中uris是一个数据

```yaml
spring:
  elasticsearch:
    jest:
      uris:
      - 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

```xml
<dependency>
    <groupId>com.sun.jna</groupId>
    <artifactId>jna</artifactId>
    <version>3.0.9</version>
</dependency>
```

1.3 测试程序

```java
//实体类
public class Article {
    //标注这个字段为主键
    @JestId
    private Integer id;
    private String author;
    private String title;
    private String content;
}
```

```
```

```java
//测试程序
import io.searchbox.core.Index;
@RunWith(SpringRunner.class)
@SpringBootTest
public class SpringbootElasticSearchApplicationTests {
    //
    @Autowired
    JestClient jestClient;

    @Test
    public void contextLoads() {
        Article article = new Article();
        article.setAuthor("lidonghao");
        article.setContent("hello world");
        article.setId(1);
        article.setTitle("first");
        //构建一个索引功能（即向ES中索引一个文档），将article放在索引haoge,类型news之下
        Index build = new Index.Builder(article).index("haoge").type("news").build();
        try {
            DocumentResult execute = jestClient.execute(build);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
```

```
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\" : \"hello\"\n" +
35                  "        }\n" +
36                  "    }\n" +
37                  "}";
38          //在haoge索引下，查询类型为news，字段中content有hello的文档
39          Search build = new Search.Builder(json).addIndex("haoge").addType("news").build();
40          try {
41              SearchResult execute = jestClient.execute(build);
42
                System.out.println(execute.getJsonString());
43          } 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 }
```

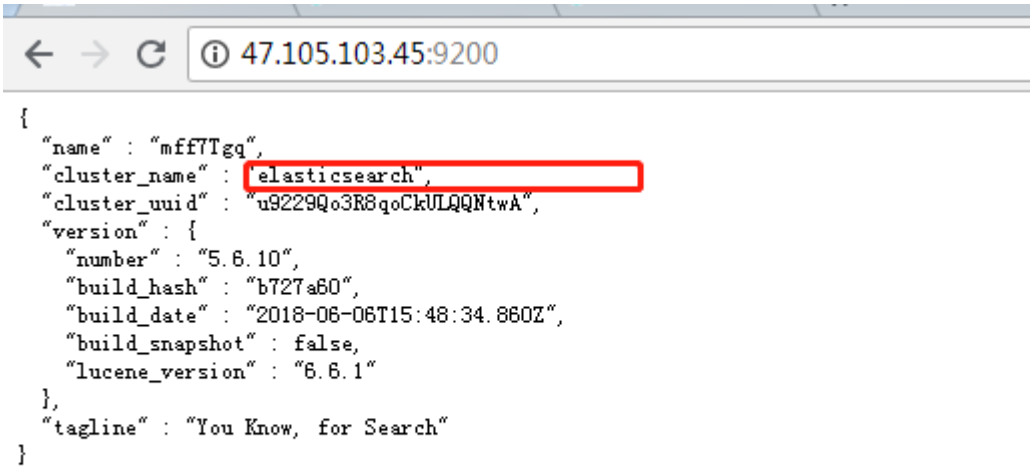
2.SpringData ElasticSearch

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

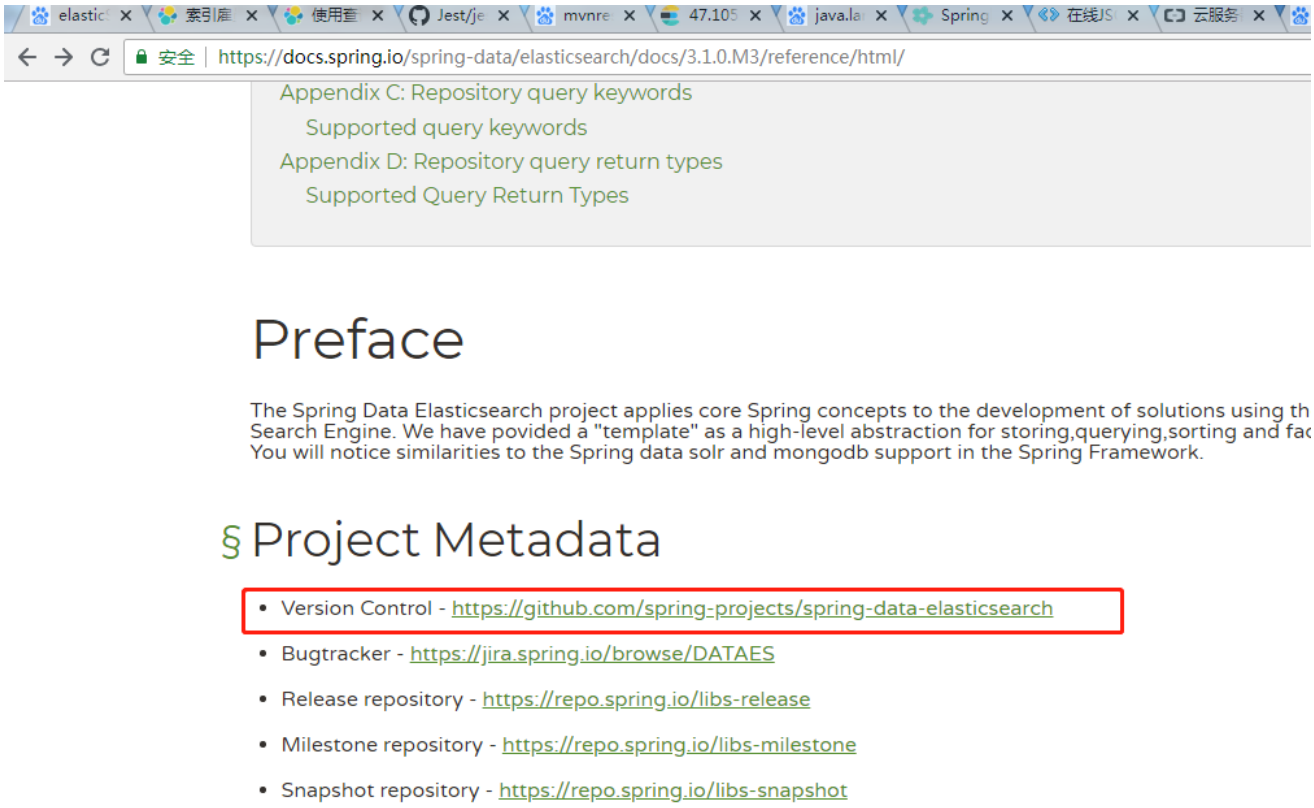
2.1 XML配置文件

```
spring:
  elasticsearch:
    jest:
      uris:
      - http://47.105.103.45:9200
#SpringData ElasticSearch相关的配置
#name取下图中标注的name
  data:
    elasticsearch:
      cluster-name: elasticsearch
      cluster-nodes: 47.105.103.45:9300
```

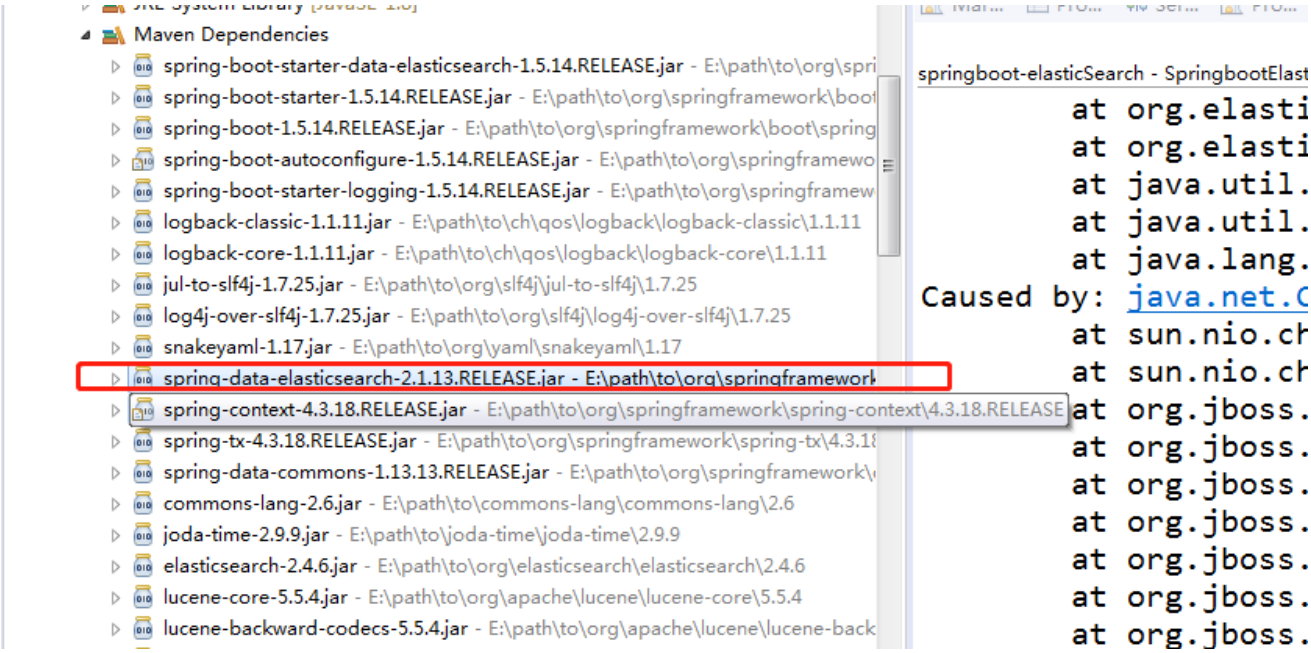


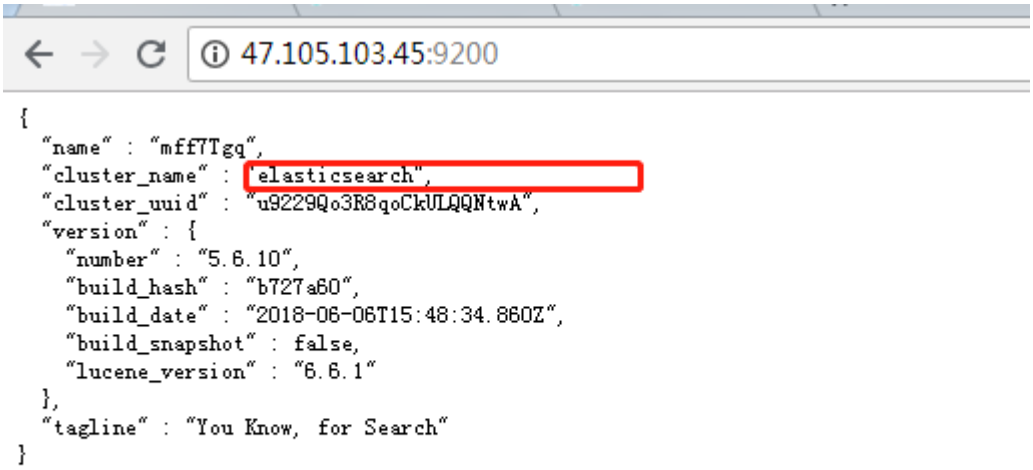
2.2 版本控制

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



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





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
```

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

2.3 ElasticsearchRepository测试

```
//ElasticsearchRepository为spring data elasticsearch提供的接口，其中有增删改查的基本方法
public interface BookRepository extends ElasticsearchRepository<Book, Integer>{

    public List<Book> findByBookNameLike(String bookName);
}
```

测试：

```
@Autowired
BookRepository bookRepository;

    //测试搜索
    @Test
    public void testRepository() {
```

```
//        Book book = new Book();
//        book.setId(1);
//        book.setBookName("西游记");
//        book.setAuthor("李东浩");
          //向ES中索引一个文档
//        bookRepository.index(book);

          List<Book> findByBookNameLike =
bookRepository.findByBookNameLike("游");
          for (Book book : findByBookNameLike) {
              System.out.println(book);
          }
      }
```

2.4 elasticsearchTemplate测试

```
@Autowired
ElasticsearchTemplate elasticsearchTemplate;
    //向ES中索引一个文档
    @Test
    public void testTemplate() {
        Book book = new Book();
        book.setId(2);
        book.setBookName("东游记");
        book.setAuthor("李西浩");

        IndexQuery indexQuery = new
IndexQueryBuilder().withId(book.getId().toString
()).withObject(book).build();
        elasticsearchTemplate.index(indexQuery);
    }
```

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

```
@SpringBootApplication
@EnableAsync
public class SpringbootElasticSearchApplication {

    public static void main(String[] args) {

SpringApplication.run(SpringbootElasticSearchAppl
ication.class, args);
    }
}
```

1.2 @Async

在相应的方法上加@Async注解

```
@Service
public class AsyncService {
    @Async
    public void hello() {
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        System.out.println("调用成功。。。");
    }
}
```

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

```
@SpringBootApplication
@EnableAsync//允许开启基于注解的异步任务
@EnableScheduling//允许开启基于注解的定时任务
public class SpringbootElasticSearchApplication {

    public static void main(String[] args) {

SpringApplication.run(SpringbootElasticSearchAppl
ication.class, args);
    }
}
```

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

```java
@Service
public class ScheduledService {
    /**
     * second(秒), minute(分), hour(时), day of month(日), month(月),day of week(周)
     */
    @Scheduled(cron="0 * * * * MON-FRI")//周一到周五的每秒都执行
    public void hello() {
        System.out.println("hello...");
    }
}
```

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

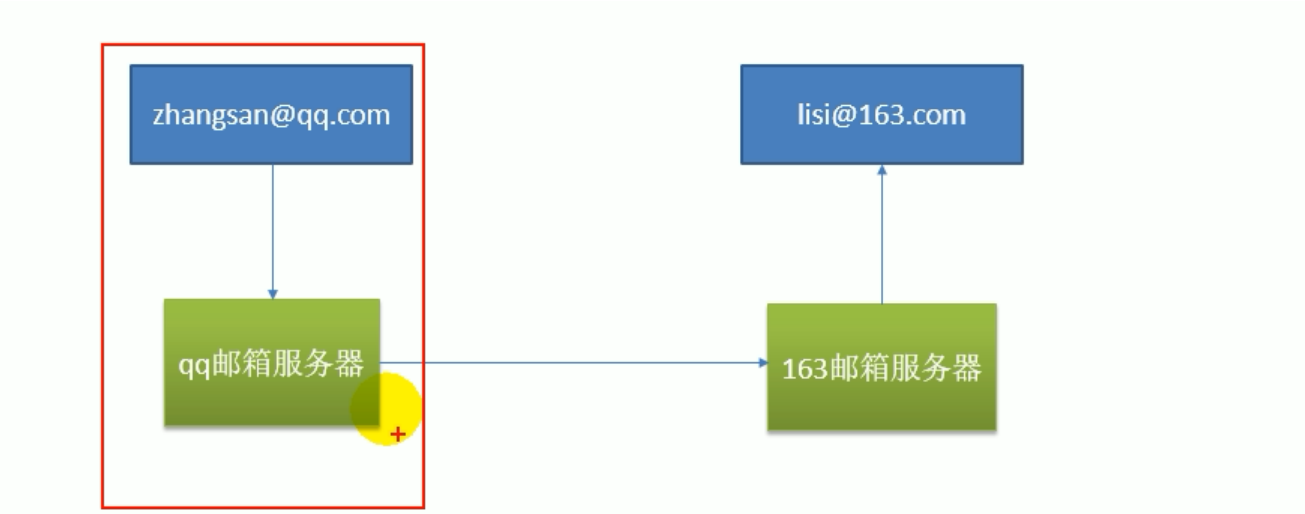
cron="0 0 2 ？ * 6L" 表示每个月最后一个周六凌晨2点执行一次

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

```xml
<dependency>

<groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-
mail</artifactId>
</dependency>
```

自动配置类：MailSenderAutoConfiguration

3.2 yaml配置

```yaml
spring:
  mail:
    host: smtp.qq.com                #服务器地址
    username: 861914994@qq.com       #QQ邮箱账号
    password: lvkqnygbgnslbchh       #QQ邮箱授权码
    properties:
      mail.smtp.ssl.enable: true     #开启SSL
```

3.3 测试代码

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class SpringbootMailTests {
    @Autowired
    JavaMailSenderImpl mailSender;

    @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
helper.addAttachment("Chrysanthemum.jpg", new File("C:\\\\Users\\\\Public\\\\Pictures\\\\Sample Pictures\\\\Chrysanthemum.jpg"));
29          helper.addAttachment("Desert.jpg", new File("C:\\\\Users\\\\Public\\\\Pictures\\\\Sample Pictures\\\\Desert.jpg"));
30          mailSender.send(mineMessage);
31      }
32  }
```