

Solr高级（上）

1.Solr其他查询

1.1 facet查询

之前我们讲解Facet查询，我们说他是分为4类。

Field,Query,Range（时间范围，数字范围）,Interval（和Range类似）

1.1.2 基于Field的Facet查询

需求：对item_title中包含手机的文档，按照品牌域进行分组，并且统计数量；

```
http://localhost:8080/solr/collection1/select?q=item_title:手机
&facet=on&facet.field=item_brand&facet.mincount=1
```

```
@Test
public void testFacetFieldQuery() throws IOException, SolrServerException {
    SolrQuery params = new SolrQuery();
    //查询条件
    params.setQuery( "item_title:手机");
    //Facet相关参数
    params.setFacet(true); //facet=on
    params.addFacetField("item_brand"); //item_brand
    params.setFacetMinCount(1);
    QueryResponse response = httpSolrClient.query(params);

    //对于Facet查询来说，我们主要获取Facet相关的数据
    //根据域名获取指定分组数据
    FacetField facetField = response.getFacetField("item_brand");
    List<FacetField.Count> values = facetField.getValues();
    for (FacetField.Count value : values) {
        System.out.println(value.getName() + "--" + value.getCount());
    }
}
```

1.1.3 基于Query的Facet查询

需求：查询分类是平板电视的商品数量，品牌是华为的商品数量，品牌是三星的商品数量，价格在1000-2000的商品数量；

```
http://localhost:8080/solr/collection1/select?
q=*:*&
facet=on&
facet.query=item_category:平板电视&
facet.query=item_brand:华为&
facet.query=item_brand:三星&
facet.query=item_price:%5B1000 TO 2000%5D
```

```
public void testFacetFieldQuery() throws IOException, SolrServerException {
```

```

SolrQuery params = new SolrQuery();
//查询条件
params.setQuery( "*:*");
//Facet相关参数
/**
 * facet.query=item_category:平板电视&
 * facet.query=item_brand:华为&
 * facet.query=item_brand:三星&
 * facet.query=item_price:[1000 TO 2000]
 */
params.setFacet(true); //facet=on
params.addFacetQuery("{!key=平板电视}item_category:平板电视");
params.addFacetQuery("{!key=华为品牌}item_brand:华为");
params.addFacetQuery("{!key=三星品牌}item_brand:三星");
params.addFacetQuery("{!key=1000到2000}item_price:[1000 TO 2000]");
QueryResponse response = httpSolrClient.query(params);

//对于Fact查询来说，我们主要获取Facet相关的数据
//根据域名获取指定分组数据
/**
 * item_category:平板电视: 207,
 * item_brand:华为: 67,
 * item_brand:三星: 154,
 * item_price:[1000 TO 2000]: 217
 */
Map<String, Integer> facetQuery = response.getFacetQuery();
for (String key : facetQuery.keySet()) {
    System.out.println(key + "--" +facetQuery.get(key));
}

}

```

1.1.4 基于Range的Facet查询

需求：分组查询价格0-2000，2000-4000，4000-6000....18000-20000每个区间商品数量

```

q=*:*&
facet=on&
facet.range=item_price&
facet.range.start=0&
facet.range.end=20000&
facet.range.gap=2000

```

```

public void testFacetRange() throws IOException, SolrServerException {
    SolrQuery params = new SolrQuery();
    //查询条件
    params.setQuery( "*:*");
    //Facet相关参数
    /**
     * facet=on&
     * facet.range=item_price&
     * facet.range.start=0&
     * facet.range.end=20000&
     * facet.range.gap=2000
     */

    params.setFacet(true); //facet=on
}

```

```

        params.addNumericRangeFacet("item_price", 0, 20000, 2000);
        QueryResponse response = httpSolrClient.query(params);

        List<RangeFacet> facetRanges = response.getFacetRanges();
        for (RangeFacet facetRange : facetRanges) {
            System.out.println(facetRange.getName());
            List<RangeFacet.Count> counts = facetRange.getCounts();
            for (RangeFacet.Count count : counts) {
                System.out.println(count.getValue() + "---" + count.getCount());
            }
        }
    }
}

```

需求：统计2015年每个季度添加的商品数量

```

http://localhost:8080/solr/collection1/select?
q=*&
facet=on&
facet.range=item_createtime&
facet.range.start=2015-01-01T00:00:00Z&
facet.range.end=2016-01-01T00:00:00Z&
facet.range.gap=%2B3MONTH

```

```

@Test
public void testFacetRange() throws IOException, SolrServerException,
ParseException {
    SolrQuery params = new SolrQuery();
    //查询条件
    params.setQuery( "*:.*");
    //Facet相关参数
    /**
     * facet=on&
     * facet.range=item_price&
     * facet.range.start=0&
     * facet.range.end=20000&
     * facet.range.gap=2000
     */
    params.setFacet(true); //facet=on
    Date start = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse("2015-
01-01 00:00:00");
    Date end = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse("2016-01-01
00:00:00");
    params.addDateRangeFacet("item_createtime", start, end, "+4MONTH");
    QueryResponse response = httpSolrClient.query(params);
    List<RangeFacet> facetRanges = response.getFacetRanges();
    for (RangeFacet facetRange : facetRanges) {
        System.out.println(facetRange.getName());
        List<RangeFacet.Count> counts = facetRange.getCounts();
        for (RangeFacet.Count count : counts) {
            System.out.println(count.getValue() + "---" + count.getCount());
        }
    }
}

```

```
}
```

1.1.5 基于Interval的Facet查询

需求：统计item_price在0-1000和0-100商品数量和item_createtime是2019年~现在添加的商品数量

```
&facet=on
&facet.interval=item_price
&f.item_price.facet.interval.set=[0,1000]
&f.item_price.facet.interval.set=[0,100]
&facet.interval=item_createtime
&f.item_createtime.facet.interval.set=[2019-01-01T0:0:0Z,NOW]
由于有特殊符号需要进行URL编码[---->%5B    ]---->%5D
http://localhost:8080/solr/collection1/select?
q=*&facet=on&facet.interval=item_price&f.item_price.facet.interval.set=%5B0,10
00%5D&f.item_price.facet.interval.set=%5B0,100%5D&facet.interval=item_createtime
&f.item_createtime.facet.interval.set=%5B2019-01-01T0:0:0Z,NOW%5D
```

```
@Test
    public void testIntervalRange() throws IOException, SolrServerException,
ParseException {
        SolrQuery params = new SolrQuery();
        //查询条件
        params.setQuery( "*:~");
        //Facet相关参数
        /**
         * &facet=on
         * &facet.interval=item_price
         * &f.item_price.facet.interval.set=[0,10]
         * &facet.interval=item_createtime
         * &f.item_createtime.facet.interval.set=[2019-01-01T0:0:0Z,NOW]
         */
        params.setFacet(true); //facet=on
        params.addIntervalFacets("item_price", new String[]{"[0,10]"});
        params.addIntervalFacets("item_createtime", new String[]{"[2019-01-
01T0:0:0Z,NOW]"});
        QueryResponse response = httpSolrClient.query(params);

        /**
         * item_price: {
         *   [0,10]: 11
         * },
         * item_createtime: {
         *   [2019-01-01T0:0:0Z,NOW]: 22
         * }
         */
        List<IntervalFacet> intervalFacets = response.getIntervalFacets();
        for (IntervalFacet intervalFacet : intervalFacets) {
            String field = intervalFacet.getField();
            System.out.println(field);
            List<IntervalFacet.Count> intervals = intervalFacet.getIntervals();
            for (IntervalFacet.Count interval : intervals) {
                System.out.println(interval.getKey());
                System.out.println(interval.getCount());
            }
        }
    }
```

```
}  
}
```

1.1.6 Facet维度查询

需求：统计每一个品牌和其不同分类商品对应的数量；

联想 手机 10

联想 电脑 2

华为 手机 10

...

```
http://localhost:8080/solr/collection1/select?q=*&  
&facet=on  
&facet.pivot=item_brand,item_category
```

```
@Test  
public void testPivotFacet() throws IOException, SolrServerException,  
ParseException {  
    SolrQuery params = new SolrQuery();  
    //查询条件  
    params.setQuery( "*:~");  
  
    /**  
     * &facet=on  
     * &facet.pivot=item_brand,item_category  
     */  
    params.addFacetPivotField("item_brand,item_category");  
    //执行查询  
    QueryResponse response = httpSolrClient.query(params);  
  
    //解析  
    NamedList<List<PivotField>> facetPivot = response.getFacetPivot();  
  
    for (Map.Entry<String, List<PivotField>> stringListEntry : facetPivot) {  
        List<PivotField> value = stringListEntry.getValue();  
        for (PivotField pivotField : value) {  
            System.out.println(pivotField.getField());  
            System.out.println(pivotField.getValue());  
            System.out.println(pivotField.getCount());  
            List<PivotField> pivot = pivotField.getPivot();  
            for (PivotField field : pivot) {  
                System.out.println(field.getField());  
                System.out.println(field.getValue());  
                System.out.println(field.getCount());  
            }  
            System.out.println("-----");  
        }  
    }  
}
```

到这关于Solr和Facet查询相关的操作就讲解完毕。

1.2 group查询

1.2.1 基础的分组

需求：查询Item_title中包含手机的文档，按照品牌对文档进行分组；同组中的文档放在一起。

```
http://localhost:8080/solr/collection1/select?
q=item_title:手机
&group=true
&group.field=item_brand
```

```
@Test
public void testGroupQuery() throws IOException, SolrServerException,
ParseException {
    SolrQuery params = new SolrQuery();
    params.setQuery("item_title:手机");

    /**
     * q=item_title:手机
     * &group=true
     * &group.field=item_brand
     */
    //注意solrJ中每没有提供分组特有API。需要使用set方法完成
    params.setGetFieldStatistics(true);
    params.set(GroupParams.GROUP, true);
    params.set(GroupParams.GROUP_FIELD, "item_brand");

    QueryResponse response = httpSolrClient.query(params);
    GroupResponse groupResponse = response.getGroupResponse();
    //由于分组的字段可以是多个。所以返回数组
    List<GroupCommand> values = groupResponse.getValues();
    //获取品牌分组结果
    GroupCommand groupCommand = values.get(0);
    //匹配到的文档数量
    int matches = groupCommand.getMatches();
    System.out.println(matches);
    //每个组合每个组中的文档信息
    List<Group> groups = groupCommand.getValues();
    for (Group group : groups) {
        //分组名称
        System.out.println(group.getGroupValue());
        //组内文档
        SolrDocumentList result = group.getResult();
        System.out.println(group.getGroupValue() + ":文档个数" +
result.getNumFound());
        for (SolrDocument entries : result) {
            System.out.println(entries);
        }
    }
}
```

1.2.2 group分页

默认情况下分组结果中只会展示前10个组，并且每组展示相关度最高的1个文档。我们可以使用start和rows可以设置组的分页，使用group.limit和group.offset设置组内文档分页。

```
q=item_title:手机&group=true&group.field=item_brand&start=0&rows=3
&group.limit=5&group.offset=0
```

展示前3个组及每组前5个文档。

```
//设置组的分页参数
params.setStart(0);
params.setRows(3);
//设置组内文档的分页参数
params.set(GroupParams.GROUP_OFFSET, 0);
params.set(GroupParams.GROUP_LIMIT, 5);
```

1.2.3 group排序

之前讲解排序的时候，group排序分为组排序，组内文档排序；对应的参数为sort和group.sort

需求：按照组内价格排序降序；

```
params.set(GroupParams.GROUP_SORT, "item_price desc");
```

当然分组还有其他的用法，都是大同小异，大家可以参考我们之前讲解分组的知识；

1.3 高亮

1.3.1 高亮查询

查询item_title中包含手机的文档，并且对item_title中的手机关键字进行高亮；

```
http://localhost:8080/solr/collection1/select?
q=item_title:手机
&hl=true
&hl.fl=item_title
&hl.simple.pre=<font>
&hl.simple.post=</font>
```

```
@Test
public void testHighlightingQuery() throws IOException, SolrServerException
{
    SolrQuery params = new SolrQuery();
    params.setQuery("item_title:三星手机");
    //开启高亮
    params.setHighlight(true);
    //设置高亮域
    //高亮的前后缀
    params.addHighlightField("item_title");
    params.setHighlightSimplePre("<font>");
    params.setHighlightSimplePost("</font>");

    QueryResponse response = httpSolrClient.query(params);

    SolrDocumentList results = response.getResults();
    for (SolrDocument result : results) {
        System.out.println(result);
    }

    //解析高亮
```

```

        Map<String, Map<String, List<String>>> highlighting =
response.getHighlighting();
//map的key是文档id,map的value包含高亮的数据
for (String id : highlighting.keySet()) {
    System.out.println(id);
    /**
     * item_title: [
     * "飞利浦 老人<em>手机</em> (X2560) 深情蓝 移动联通2G<em>手机</em> 双卡双
待"
     * ]
     */
    Map<String, List<String>> highLightData = highlighting.get(id);
    //highLightData key包含高亮域名
    //获取包含高亮的数据
    if(highLightData != null && highLightData.size() > 0) {
        //[
        //          * "飞利浦 老人<em>手机</em> (X2560) 深情蓝 移动联通
2G<em>手机</em> 双卡双待"
        //          * ]
        List<String> stringList = highLightData.get("item_title");
        if(stringList != null && stringList.size() >0) {
            String title = stringList.get(0);
            System.out.println(title);
        }
    }
}

//将高亮的数据替换到原有文档中。

}

```

1.3.2 高亮器的切换

当然我们也可以使用Solr完成高亮器的切换。之前我们讲解过一个高亮器fastVector,可以实现域中不同的词使用不同颜色。

查询item_title中包含三星手机的文档.item_title中三星手机中不同的词，显示不同的颜色；

```

http://localhost:8080/solr/collection1/select?
q=item_title:三星手机
&hl=true
&hl.fl=item_title
&hl.method=fastVector

```

到这使用Solr进行高亮查询就讲解完毕。

1.4 suggest查询

1.4.1 spell-checking 拼写检查。

需求：查询item_title中包含iphone的内容。要求进行拼写检查。

```

http://localhost:8080/solr/collection1/select?
q=item_title:iphonx&spellcheck=true

```



```

@Test
public void test01() throws IOException, SolrServerException {
    SolrQuery params = new SolrQuery();
    params.setQuery("item_title:iphonxx");
    params.set("spellcheck",true);
    QueryResponse response = httpSolrClient.query(params);
    /**
     * suggestions: [
     * "iphonxx",
     * {
     * numFound: 1,
     * startOffset: 11,
     * endOffset: 18,
     * suggestion: [
     * "iphone6"
     * ]
     * }
     * ]
     */
    SpellCheckResponse spellCheckResponse =
response.getSpellCheckResponse();
    Map<String, SpellCheckResponse.Suggestion> suggestionMap =
spellCheckResponse.getSuggestionMap();
    for (String s : suggestionMap.keySet()) {
        //错误的词
        System.out.println(s);
        //建议的词
        SpellCheckResponse.Suggestion suggestion = suggestionMap.get(s);

        List<String> alternatives = suggestion.getAlternatives();
        System.out.println(alternatives);
    }
}
}

```

1.4.2Auto Suggest自动建议。

上面我们讲解完毕拼写检查，下面我们讲解自动建议，自动建议也是需要在SolrConfig.xml中进行相关的配置。

需求：查询三星，要求基于item_title域进行自动建议

```

http://localhost:8080/solr/collection1/select?
q=三星&suggest=true&suggest.dictionary=mySuggester&suggest.count=5

```

```

@Test
public void test02() throws IOException, SolrServerException {
    SolrQuery params = new SolrQuery();
    //设置参数
    params.setQuery("java");
}

```

```

//开启自动建议
params.set("suggest",true);
//指定自动建议的组件
params.set("suggest.dictionary","mySuggester");

QueryResponse response = httpSolrClient.query(params);
SuggesterResponse suggesterResponse = response.getSuggesterResponse();
Map<String, List<Suggestion>> suggestions =
suggesterResponse.getSuggestions();
for (String key : suggestions.keySet()) {
    //词
    System.out.println(key);
    List<Suggestion> suggestionList = suggestions.get(key);
    for (Suggestion suggestion : suggestionList) {
        String term = suggestion.getTerm();
        System.out.println(term);
    }
}
}

```

1.5 使用SolrJ完成Core的管理

1.5.1 Core添加

1.要想完成SolrCore的添加，在solr_home必须提前创建好SolrCore的目录结构，而且必须包相关的配置文件。

新加卷 (D:) > solr_home > collection4				
名称	修改日期	类型	大小	
conf	2020/4/21 12:45	文件夹		
data	2020/4/21 12:45	文件夹		

```

修改配置文件中url:http://localhost:8080/solr
CoreAdminRequest.createCore("collection4", "D:\\solr_home\\collection4",
solrClient );

```

1.5.2 重新加载Core

从Solr中移除掉，然后在添加。

```
CoreAdminRequest.reloadCore("collection4", solrClient );
```

1.5.3 重命名Core

```
CoreAdminRequest.renameCore("collection4","newCore" , solrClient)
```

1.5.4 卸载solrCore

卸载仅仅是从Solr中将该Core移除，但是SolrCore的物理文件依然存在

```
CoreAdminRequest.unloadCore("collection4", solrClient );
```

```
CoreAdminRequest.swapCore("collection1", "collection4", solrClient);
```

2. Solr集群搭建

2.1 SolrCloud简介

2.1.1 什么是SolrCloud

Solr集群也成为SolrCloud，是Solr提供的分布式搜索的解决方案，当你需要大规模存储数据或者需要分布式索引和检索能力时使用 SolrCloud。

所有数据库集群，都是为了解决4个核心问题，单点故障，数据扩容，高并发，效率

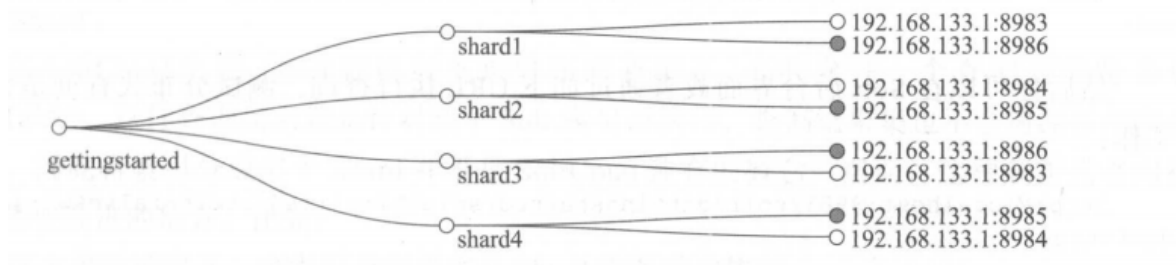
搭建SolrCloud是需要依赖一个中间件Zookeeper，它的主要思想是使用Zookeeper作为集群的配置中心。

2.1.2 SolrCloud架构

SolrCloud逻辑概念：

一个Solr集群是由多个collection组成，collection不是SolrCore只是一个逻辑概念。一个collection是由多个文档组成，这些文档归属于指定的分片。下面表示的就是一个逻辑结构图。

该集群只有一个collection组成。collection中的文档数据分散在4个分片。

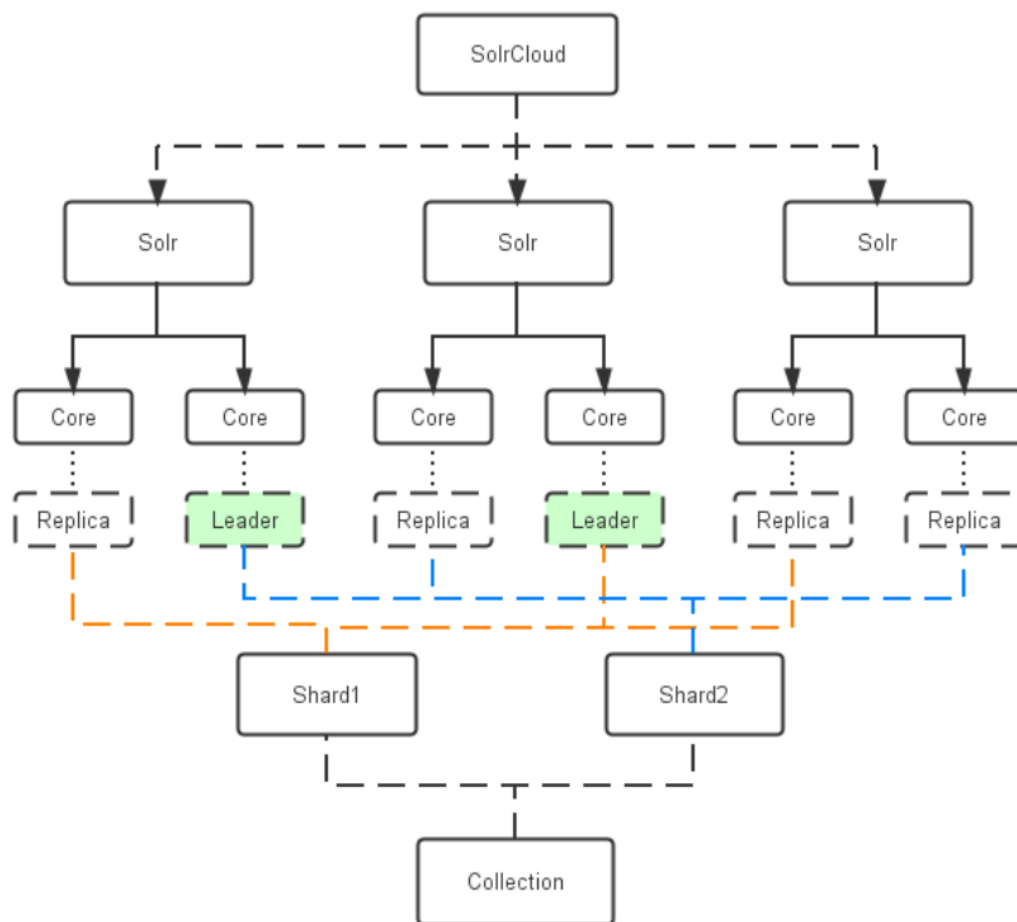


分片中的数据，到底存储在那呢？这就属于物理概念。

SolrCloud物理概念：

Solr 集群由一个或多个 Solr服务（tomcat中运行solr服务）组成，这些Solr服务可以部署一台服务器上，也可以在多台服务器。每个Solr服务可以包含一个或者多个Solr Core 。SolrCore中存储的是 Shard的数据；

下面的图是一个物理结构和逻辑结构图。



概念：

Collection是一个逻辑概念，可以认为是多个文档构成的一个集合。

Shard是一个逻辑概念，是对Collection进行逻辑上的划分。

Replica：Shard的一个副本，一个Shard有多个副本，同一个Shard副本中的数据一样；

Leader：每个Shard会以多个Replica的形式存在，其中一个Replica会被选为Leader，负责集群环境中的索引和搜索。

SolrCore：存储分片数据的基本单元。一个SolrCore只存储一个分片数据，一个分片数据可以存储到多个SolrCore中；一个SolrCore对应一个Replica

Node：集群中的一个Solr服务

2.2Linux集群搭建

2.2.1 基于tomcat的集群搭建

2.2.1.1集群架构介绍

物理上：

搭建三个Zookeeper组成集群，管理SolrCloud。

搭建四个Solr服务，每个Solr服务一个SorCore.

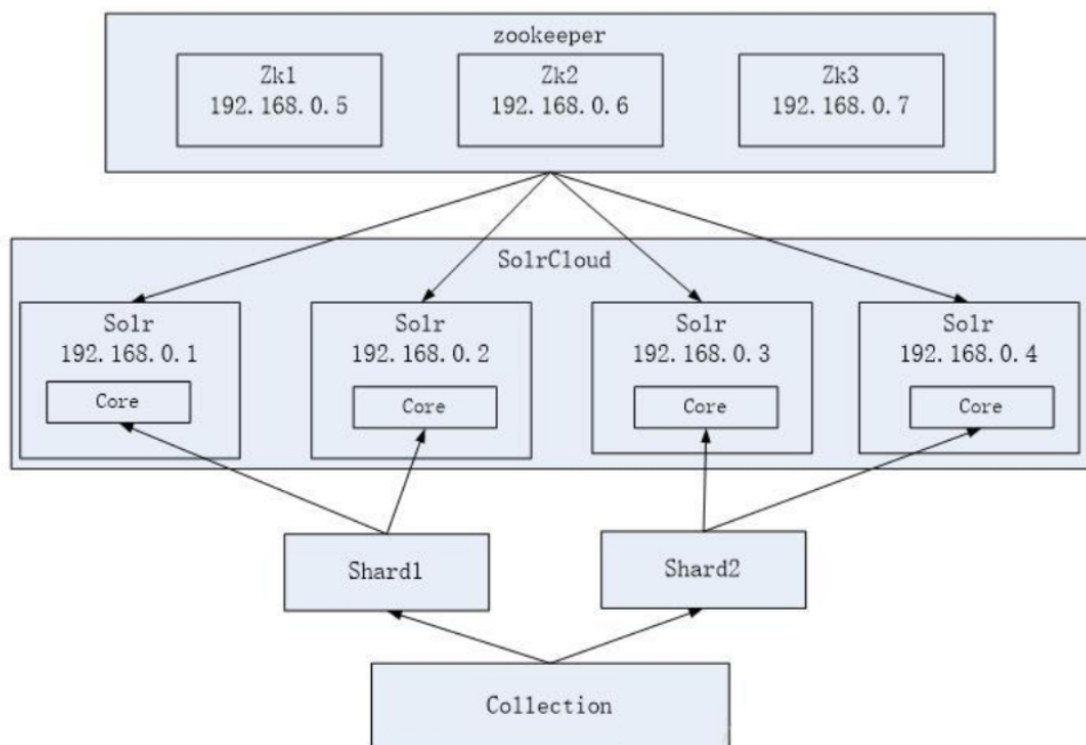
逻辑上:

整个SolrCloud一个Connection;

Connection中的数据分为2个Shard;

Shard1的数据在物理上存储到solr1和solr2的SolrCore

Shard2的数据在物理上存储到solr3和solr4的SolrCore



2.2.1.2 环境说明

环境和我们之前搭建单机版的环境相同。

系统	版本
Linux	CentOS 7
JDK	JDK8
Tomcat	tomcat 8.5
zookeeper	zookeeper-3.4.14
solr	solr 7.7.2

2.2.1.3 Zookeeper集群搭建

首先我们先要完成Zookeeper集群搭建。在搭建集群之前，首先我们先来说一下Zookeeper集群中要求节点的个数。

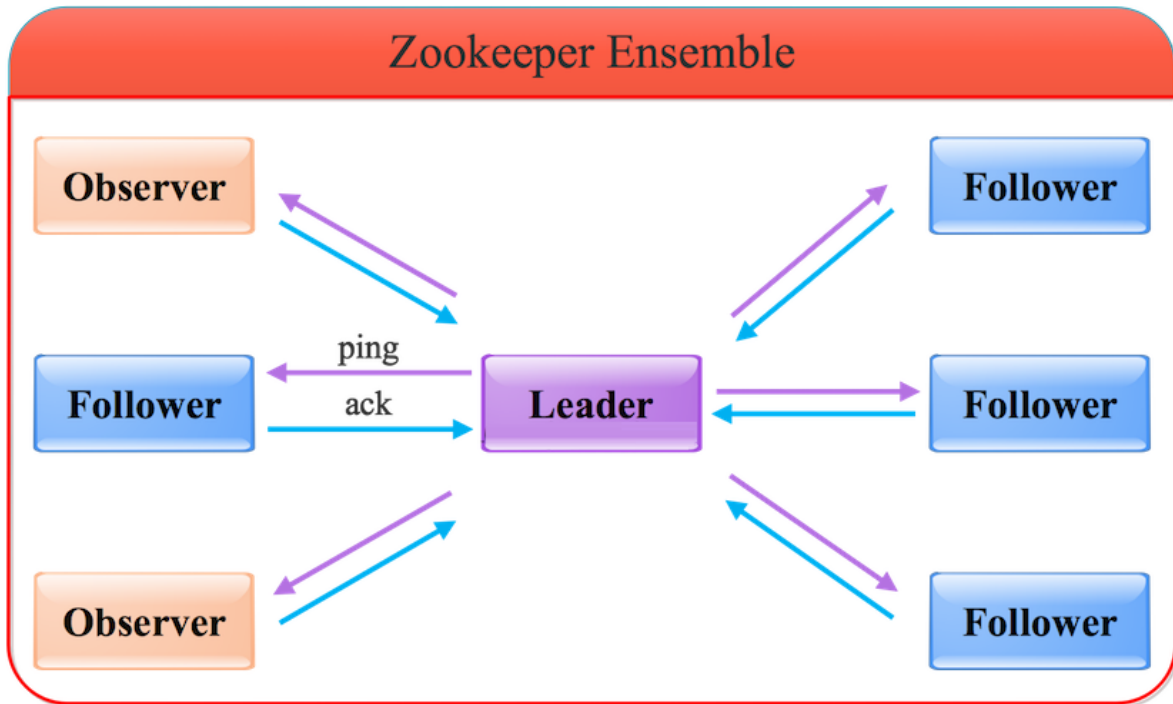
在Zookeeper集群中节点的类型主要有2类，Leader，Follower，一个集群只有一个Leader。

到底谁是Leader呢？

投票选举制度：集群中所有的节点来进行投票，半数以上获票的节点就是Leader.Zookeeper要求集群节点个数奇数。

容错:zookeeper集群一大特性是只要集群中半数以上的节点存活，集群就可以正常提供服务，而 $2n+1$ 节点和 $2n+2$ 个节点的容错能力相同，都是允许 n 台节点宕机。本着节约的宗旨，一般选择部署 $2n+1$ 台机器

本次我们采用最少的集群节点3个。



1.下载Zookeeper的安装包到linux。

下载地址：

<http://archive.apache.org/dist/zookeeper/zookeeper-3.4.14/>

上面的地址可能受每日访问量现在

wget

<https://mirrors.tuna.tsinghua.edu.cn/apache/zookeeper/zookeeper-3.4.14/zookeeper-3.4.14.tar.gz>

2. 解压zookeeper，复制三份到/usr/local/solrcloud下，复制三份分别并将目录名改为zk01、在zk02、zk03

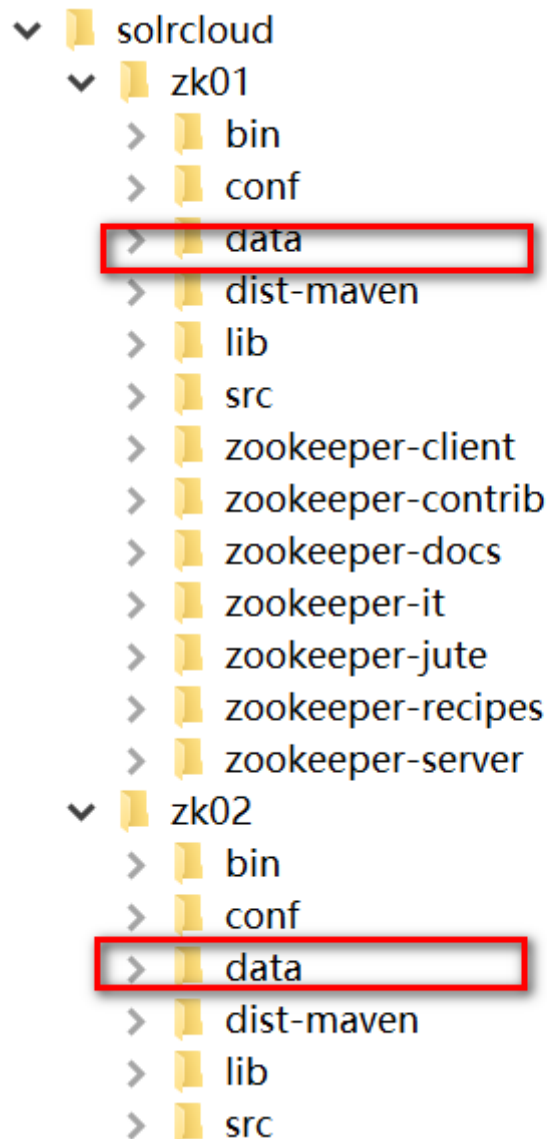
```
mkdir /usr/local/solrcloud
tar -xvzf zookeeper-3.4.14.tar.gz
cp -r zookeeper-3.4.14 /usr/local/solrcloud/zk01
cp -r zookeeper-3.4.14 /usr/local/solrcloud/zk02
cp -r zookeeper-3.4.14 /usr/local/solrcloud/zk03
```

结果

```
[root@pinyoyougou-docker solrcloud]# ll
总用量 12
drwxr-xr-x. 14 root root 4096 2月 22 09:44 zk01
drwxr-xr-x. 14 root root 4096 2月 22 09:44 zk02
drwxr-xr-x. 14 root root 4096 2月 22 09:44 zk03
```

3. 为每一个Zookeeper创建数据目录

```
mkdir zk01/data
mkdir zk02/data
mkdir zk03/data
```



4. 在data目录中为每一个Zookeeper创建myid文件,并且在文件中指定该Zookeeper的id;

使用重定向指令>>将打印在控制台上的1 2 3分别写入到data目录的myid文件中。

```
echo 1 >> zk01/data/myid
echo 2 >> zk02/data/myid
echo 3 >> zk03/data/myid
```

zk01	bin	Name	Size	Type	Date Modified
	conf	..		Directory	2020/2/22 9:47
	data	myid	2	File	2020/2/22 9:53
	dist-maven				
	lib				
	src				
	zookeeper-client				
	zookeeper-contrib				
	zookeeper-docs				
	zookeeper-it				
	zookeeper-jute				
	zookeeper-recipes				
	zookeeper-server				

5. 修改Zookeeper的配置文件名称。

5.1 需要将每个zookeeper/conf目录中的zoo_sample.cfg 文件名改名为zoo.cfg，否则该配置文件不起作用。

```
mv zk01/conf/zoo_sample.cfg zk01/conf/zoo.cfg
mv zk02/conf/zoo_sample.cfg zk02/conf/zoo.cfg
mv zk03/conf/zoo_sample.cfg zk03/conf/zoo.cfg
```

zk01	bin	Name	Size	Type	Date Modified
	conf	..		Directory	2020/2/22 9:47
	data	configuration.xml	535	XSL Stylesheet	2020/2/22 9:44
	dist-maven	log4j.properties	2161	File	2020/2/22 9:44
	lib	zoo.cfg	922	File	2020/2/22 9:44
	src				
	zookeeper-client				
	zookeeper-contrib				
	zookeeper-docs				
	zookeeper-it				
	zookeeper-jute				
	zookeeper-recipes				
	zookeeper-server				

6. 编辑zoo.cfg配置文件

5.1 修改Zookeeper客户端连接的端口，Zookeeper对外提供服务的端口。修改目的，我们需要在一台计算机上启动三个Zookeeper。

```
clientPort=2181
clientPort=2182
clientPort=2183
```

5.2 修改数据目录

```
dataDir=/usr/local/solrcloud/zk01/data
dataDir=/usr/local/solrcloud/zk02/data
dataDir=/usr/local/solrcloud/zk03/data
```

5.3 在每一个配置文件的尾部加入，指定集群中每一个Zookeeper的节点信息。


```
server.1=192.168.200.128:2881:3881
server.2=192.168.200.128:2882:3882
server.3=192.168.200.128:2883:3883
```

server.1 server.2 server.3 指定Zookeeper节点的id.之前我们myid已经指定了每个节点的id;
内部通信端口:

Zookeeper集群节点相互通信端口;

为什么会有投票端口呢?

1. 因为三台Zookeeper要构成集群, 决定谁是leader, Zookeeper中是通过投票机制决定谁是leader。
 2. 如果有节点产生宕机, 也需要通过投票机制将宕机节点从集群中剔除。
- 所以会有一个投票端口

7.启动Zookeeper集群。

在bin目录中提供了一个脚本zkServer.sh, 使用zkServer.sh start|restart|stop|status 就可以完成启动, 重启, 停止, 查看状态。

可以写一个脚本启动所有Zookeeper。也可以一个个的启动;

```
./zk01/bin/zkServer.sh start
./zk02/bin/zkServer.sh start
./zk03/bin/zkServer.sh start
```

```
Using config: /usr/local/solrcloud/zk01/bin/./conf/zoo.cfg
Starting zookeeper ... STARTED
[root@pinyoyougou-docker solrcloud]# ./zk02/bin/zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /usr/local/solrcloud/zk02/bin/./conf/zoo.cfg
Starting zookeeper ... STARTED
[root@pinyoyougou-docker solrcloud]# ./zk03/bin/zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /usr/local/solrcloud/zk03/bin/./conf/zoo.cfg
Starting zookeeper ... STARTED
```

8.查看Zookeeper集群状态。

```
./zk01/bin/zkServer.sh status
./zk02/bin/zkServer.sh status
./zk03/bin/zkServer.sh status
```

```
Using config: /usr/local/solrcloud/zk01/bin/./conf/zoo.cfg
Mode: follower
[root@pinyoyougou-docker solrcloud]# ./zk02/bin/zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /usr/local/solrcloud/zk02/bin/./conf/zoo.cfg
Mode: leader
[root@pinyoyougou-docker solrcloud]# ./zk03/bin/zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /usr/local/solrcloud/zk03/bin/./conf/zoo.cfg
Mode: follower
```

follower就是slave;

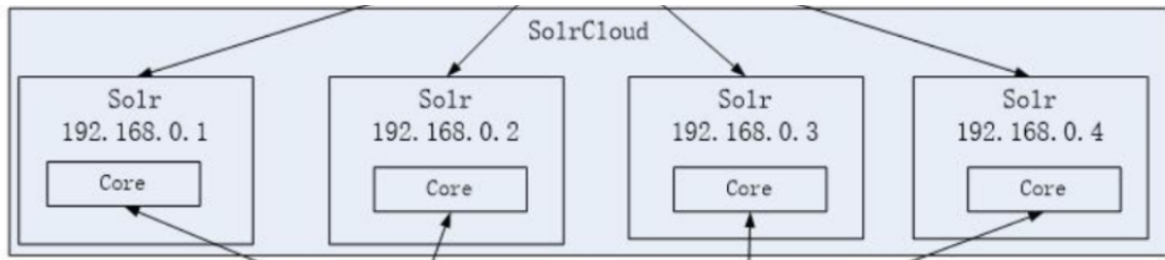
leader就是master;

如果是单机版的Zookeeper, standalone

到这关于Zookeeper集群的搭建, 我们就讲解完毕。

2.2.1.4 SolrCloud集群部署

上一节我们已经完成了Zookeeper集群的搭建，下面我们来搭建solr集群。



复制4个tomcat，并且要在4个tomcat中部署solr，单机版solr的搭建，我们之前已经讲解过了。

下面是我们已经搭建好的单机版的solr，他的tomcat和solrhome

```
drwxr-xr-x. 9 root root 220 2月 4 12:48 apache-tomcat-8.5.50
drwxr-xr-x. 2 root root 6 11月 5 2016 bin
drwxr-xr-x. 2 root root 6 11月 5 2016 etc
drwxr-xr-x. 2 root root 6 11月 5 2016 games
drwxr-xr-x. 2 root root 6 11月 5 2016 include
drwxr-xr-x. 8 10 143 255 3月 29 2018 jdk1.8.0_171
drwxr-xr-x. 2 root root 6 11月 5 2016 lib
drwxr-xr-x. 2 root root 6 11月 5 2016 lib64
drwxr-xr-x. 2 root root 6 11月 5 2016 libexec
drwxr-xr-x. 4 root root 29 8月 14 2019 myhtml
drwxr-xr-x. 2 root root 6 11月 5 2016 sbin
drwxr-xr-x. 5 root root 49 9月 3 2017 share
drwxr-xr-x. 5 root root 63 2月 22 10:27 solrcloud
drwxr-xr-x. 5 root root 109 2月 5 18:59 solr_home
drwxr-xr-x. 2 root root 6 11月 5 2016 src
drwxr-xr-x. 2 root root 20 8月 13 2019 test1
```

1.1 将tomcat复制4份到solrcloud中，便于集中管理。这4个tomcat中都部署了solr；

```
cp -r apache-tomcat-8.5.50 solrcloud/tomcat1
cp -r apache-tomcat-8.5.50 solrcloud/tomcat2
cp -r apache-tomcat-8.5.50 solrcloud/tomcat3
cp -r apache-tomcat-8.5.50 solrcloud/tomcat4
```

> include	Name	Size	Type	Date Modified
> jdk1.8.0_171	..		Directory	2020/2/22 9:28
> lib	tomcat1		Directory	2020/2/22 10:43
> lib64	tomcat2		Directory	2020/2/22 10:43
> libexec	tomcat3		Directory	2020/2/22 10:43
> myhtml	tomcat4		Directory	2020/2/22 10:43
> sbin	zk01		Directory	2020/2/22 9:47
> share	zk02		Directory	2020/2/22 9:47
> solr_home	zk03		Directory	2020/2/22 9:47
> solrcloud	zookeeper.out	22894	File	2020/2/22 10:29
> tomcat1				
> tomcat2				
> tomcat3				
> tomcat4				
> zk01				
> zk02				
> zk03				

1.2 复制4个solrhome到solrcloud中。分别对应每一个solr的solrhome。

```
cp -r solr_home solrcloud/solrhome1
cp -r solr_home solrcloud/solrhome2
cp -r solr_home solrcloud/solrhome3
cp -r solr_home solrcloud/solrhome4
```

	Name	Size	Type	Date Modified
> libexec	..		Directory	2020/2/22 9:28
> myhtml				
> sbin				
> share				
> solr_home				
▼ solrcloud				
> solrhome1	solrhome1		Directory	2020/2/22 10:48
> solrhome2	solrhome2		Directory	2020/2/22 10:48
> solrhome3	solrhome3		Directory	2020/2/22 10:48
> solrhome4	solrhome4		Directory	2020/2/22 10:48
> tomcat1	tomcat1		Directory	2020/2/22 10:43
> tomcat2	tomcat2		Directory	2020/2/22 10:43
> tomcat3	tomcat3		Directory	2020/2/22 10:43
> tomcat4	tomcat4		Directory	2020/2/22 10:43
> zk01	zk01		Directory	2020/2/22 9:47
> zk02	zk02		Directory	2020/2/22 9:47
> zk03	zk03		Directory	2020/2/22 9:47
	zookeeper.out	22894	File	2020/2/22 10:29

1.3 修改tomcat的配置文件，修改tomcat的端口，保证在一台计算机上，可以启动4个tomcat；

编辑server.xml

```

修改停止端口
对外提供服务端口
AJP端口
<Server port="8005" shutdown="SHUTDOWN">
<Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirect
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
tomcat1 8105 8180 8109
tomcat1 8205 8280 8209
tomcat1 8305 8380 8309
tomcat1 8405 8480 8409

```

1.4 为tomcat中每一个solr指定正确的solrhome，目前是单机版solrhome的位置。

编辑solr/web.xml指定对应的solrhome

```

<env-entry>
  <env-entry-name>solr/home</env-entry-name>
  <env-entry-value>/usr/local/solrcloud/solrhomeX</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>

```

1.5 修改每个solrhome中solr.xml的集群配置,对应指定tomcat的ip和端口

编辑solrhome中solr.xml

```

<solrcloud>
  <str name="host">192.168.200.1288</str>
  <int name="hostPort">8X80</int>
</solrcloud>

```

2.2.1.5 Zookeeper管理的Solr集群

上一节课我们已经搭建好了Zookeeper集群，和Solr的集群。但是现在Zookeeper集群和Solr集群没有关系。

需要让Zookeeper管理Solr集群。

1.启动所有的tomcat

```
./tomcat1/bin/startup.sh
./tomcat2/bin/startup.sh
./tomcat3/bin/startup.sh
./tomcat4/bin/startup.sh
```

2.让每一个solr节点和zookeeper集群关联

编辑每一个tomcat中bin/catalina.sh文件，指定Zookeeper集群地址。

```
JAVA_OPTS="-
DzkHost=192.168.200.128:2181,192.168.200.128:2182,192.168.200.128:2183"
需要指定客户端端口即Zookeeper对外提供服务的端口
```

3.让zookeeper统一管理solr集群的配置文件。

因为现在我们已经是一个集群，集群中的每个solr节点配置文件需要一致。所以我们就需要让zookeeper管理solr集群的配置文件。主要管理的就是solrconfig.xml / schema.xml文件

下面的命令就是将conf目录中的配置文件，上传到Zookeeper，以后集群中的配置文件就以Zookeeper中的为准；

搭建好集群后，solrCore中是没有配置文件的。

```
./zkcli.sh -zkhost
192.168.200.128:2181,192.168.200.128:2182,192.168.200.128:2183 -cmd upconfig -
confdir /usr/local/solrcloud/solrhome1/collection1/conf -confname myconf
```

-zkhost: 指定zookeeper的地址列表；

upconfig : 上传配置文件；

-confdir : 指定配置文件所在目录；

-confname: 指定上传到zookeeper后的目录名；

进入到solr安装包中

```
/root/solr-7.7.2/server/scripts/cloud-scripts
```

```
./zkcli.sh -zkhost
```

```
192.168.200.128:2181,192.168.200.128:2182,192.168.200.128:2183 -cmd upconfig -
confdir /usr/local/solrcloud/solrhome1/collection1/conf -confname myconf
```

```
./zkcli.sh -zkhost
```

```
192.168.200.128:2181,192.168.200.128:2182,192.168.200.128:2183 -cmd upconfig -
confdir /root/solr7.7.2/server/solr/configsets/sample_techproducts_configs/conf
-confname myconf
```

```
[root@pinyoyougou-docker cloud-scripts]# ./zkcli.sh -zkhost 192.168.200.131:2181,192.168.200.131:
2182,192.168.200.131:2183 -cmd upconfig -confdir /usr/local/solrcloud/solrhome1/collection1/conf
-confname myconf
INFO - 2020-02-22 11:39:56.271; org.apache.solr.common.cloud.ConnectionManager; Waiting for client to connect to ZooKeeper
INFO - 2020-02-22 11:39:56.451; org.apache.solr.common.cloud.ConnectionManager; zkClient has connected
INFO - 2020-02-22 11:39:56.452; org.apache.solr.common.cloud.ConnectionManager; Client is connected to ZooKeeper
```

4.查看Zookeeper中的配置文件。

登录到任意一个Zookeeper上，看文件是否上传成功。

```
进入到任意一个Zookeeper，查看文件的上传状态。
/zkcli.sh -server 192.168.200.128:2182
```

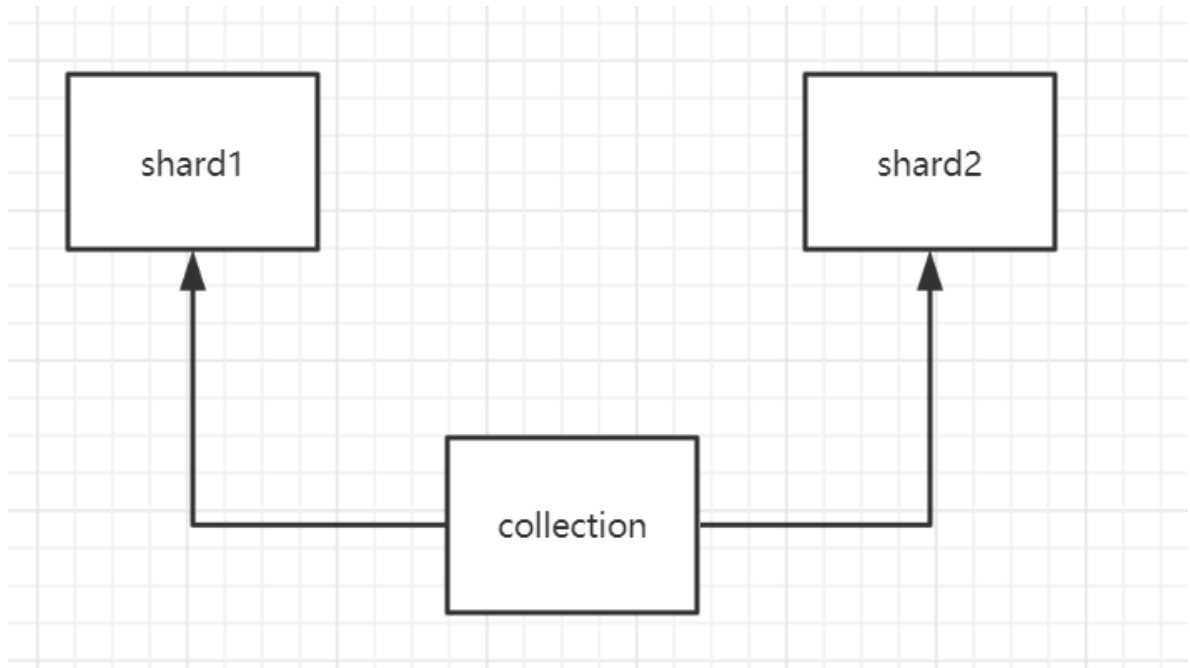
通过Zookeeper客户端工具查看。idea的Zookeeper插件，ZooInspector等。

5. 由于solr的配置文件，已经交由Zookeeper管理。SolrHome中的SolrCore就可以删除。

到这Zookeeper集群和Solr集群整合就讲解完毕。

2.2.1.6 创建Solr集群的逻辑结构

之前我们已经搭建好了Zookeeper集群及Solr集群。接下来我们要完成的是逻辑结构的创建。



我们本次搭建的逻辑结构整个SolrCloud是由一个Collection构成。Collection分为2个分片。每个分片有2个副本。分别存储在不同的SolrCore中。

启动所有的tomcat

```
./tomcat1/bin/startup.sh  
./tomcat2/bin/startup.sh  
./tomcat3/bin/startup.sh  
./tomcat4/bin/startup.sh
```

访问任4个solr管理后台;

```
http://192.168.200.131:8180/solr/index.html#/~collections  
http://192.168.200.131:8280/solr/index.html#/~collections  
http://192.168.200.131:8380/solr/index.html#/~collections  
http://192.168.200.131:8480/solr/index.html#/~collections
```

使用任何一个后台管理系统，创建逻辑结构

name:

config set:

numShards:

replicationFactor

Show advanced ☒

name:指定逻辑结构collection名称;

config set:指定使用Zookeeper中配置文件;

numShards:分片格式

replicationFactory:每个分片副本格式;

当然我们也可以通过Solr RestAPI完成上面操作

<http://其中一个solr节点的IP:8983/solr/admin/collections?action=CREATE&name=testcore&numShards=2&replicationFactor=2&collection.configName=myconf>

查询逻辑结构



- **Leader**
- Active
- Recovering
- Down
- Recovery Failed
- Inactive
- Gone
- **Replica Type:**
N(NRT),T(TLOG),
P(PULL)

查看物理结构

- ▼ solrcloud
 - ▼ solrhome1
 - > configsets
 - > connection_shard1_replica_n1
 - ▼ solrhome2
 - > configsets
 - > connection_shard1_replica_n3
 - ▼ solrhome3
 - > configsets
 - > connection_shard2_replica_n5
 - ▼ solrhome4
 - > configsets
 - > connection_shard2_replica_n7

说明一下：原来SolrCore的名字已经发生改变。

一旦我们搭建好集群后，每个SolrCore中是没有conf目录即没有配置文件。

整个集群的配置文件只有一份，Zookeeper集群中的配置文件。

2.2.1.7测试集群

向collection中添加数据

```
{id:"100",name:"zhangsan"}
```

由于整合集群逻辑上就一个collection,所以在任何一个solr节点都可以获取数据。

2.2.1.8 使用SolrJ操作集群

之前我们操作单节点的Solr，使用的是HttpSolrClient，操作集群我们使用的是CloudSolrClient。

1. 将CloudSolrClient交由spring管理。

```
@Bean
public CloudSolrClient cloudSolrClient() {
    //指定Zookeeper的地址。
    List<String> zkHosts = new ArrayList<>();
    zkHosts.add("192.168.200.131:2181");
    zkHosts.add("192.168.200.131:2182");
    zkHosts.add("192.168.200.131:2183");
    //参数2: 指定
    CloudSolrClient.Builder builder = new
    CloudSolrClient.Builder(zkHosts,Optional.empty());
    CloudSolrClient solrClient = builder.build();
    //设置连接超时时间
```

```

solrClient.setZkClientTimeout(30000);
solrClient.setZkConnectTimeout(30000);
//设置collection
solrClient.setDefaultCollection("collection");
return solrClient;
}

```

2.使用CloudSolrClient中提供的API操作Solr集群,和HttpSolrClient相同

索引

```

@Autowired
private CloudSolrClient cloudSolrClient;

@Test
public void testSolrCloudAddDocument() throws Exception {
    SolrInputDocument doc = new SolrInputDocument();
    doc.setField("id", 1);
    doc.setField("name", "java");
    cloudSolrClient.add(doc);
    cloudSolrClient.commit();
}

```

搜索

```

@Test
public void testSolrQuery() throws Exception {
    SolrQuery params = new SolrQuery();
    params.setQuery("*:~");
    QueryResponse response = cloudSolrClient.query(params);
    SolrDocumentList results = response.getResults();
    for (SolrDocument result : results) {
        System.out.println(result);
    }
}

```

2.2.2 SolrCloud的其他操作

2.2.2.1 SolrCloud使用中文分词器 (IK)

1.在每一个solr服务中, 引入IK分词器的依赖包

▼ tomcat4	ik-analyzer-solr5-5.x.jar	1165911	Executable Jar File	2020/2/15 0:11
> \$sys:solr.log.dirj	jackson-annotations-2.9.8.jar	66894	Executable Jar File	2020/2/23 10:06
> bin	jackson-core-2.9.8.jar	325619	Executable Jar File	2020/2/23 10:06
> conf	jackson-core-asl-1.9.13.jar	232248	Executable Jar File	2020/2/23 10:06
> ...				

2.在classes中引入停用词和扩展词库及配置文件

IKAnalyzer.cfg.xml	404	XML Document	2017/5/29 23:47
log4j2-console.xml	1432	XML Document	2020/2/23 10:06
log4j2.xml	2678	XML Document	2020/2/23 10:06
stopword.dic	8245	DIC File	2017/5/29 23:47

3.重启所有的tomcat

4.修改单机版solrcore中schema, 加入FileType

```
<fieldType name ="text_ik" class ="solr.TextField">
  <analyzer class="org.wltea.analyzer.lucene.IKAnalyzer"/>
</fieldType>
```

5.将schema重新提交到Zookeeper

Solr集群中的配置文件统一由Zookeeper进行管理，所以如果需要修改配置文件，我们需要将修改的文件上传的Zookeeper。

```
./zkcli.sh -zkhost
192.168.200.128:2181,192.168.200.128:2182,192.168.200.128:2183 -cmd putfile
/configs/myconf/managed-schema /usr/local/solr_home/collection1/conf/managed-
schema
```

6. 测试

Field Value (Index)

我是中国人

Analyse Fieldname / FieldType: text_ik Schema Browser

IKT	text	中国人	中国	国人
	raw_bytes	[e4 b8 ad e5 9b bd e4 ba ba]	[e4 b8 ad e5 9b bd]	[e5 9b bd e4 ba ba]
	start	2	2	3
	end	5	4	5
	positionLength	1	1	1
	type	CN_WORD	CN_WORD	CN_WORD
	termFrequency	1	1	1
	position	1	2	3

7.利用text_ik 创建相关的业务域

创建业务域，我们依然需要修改shema文件，这个时候，我们建议使用后台管理系统。

Add Field

Add Dynamic Field

Add Copy Field

name:

field type:

default:

☒ stored

☒ indexed

☐ uninvertible

☐ docValues

☐ multiValued

☐ required

Show omit options ☒

Show term vector options ☒

Show sort options ☒

☒ Add Field

☐ Cancel

8.查看schema文件是否被修改。

通过files查看集群中的配置文件，内容

```
<field name="item_title" type="text_ik" uninvertible="false" indexed="true" stored="true"/>
```

到这关于如何修改集群中的配置文件，已经如何管理Filed我们就讲解完毕。

2.2.2.2 查询指定分片数据

在SolrCloud中，如果我们想查询指定分片的数据也是允许的。

需求：查询shard1分片的数据

```
http://192.168.200.128:8180/solr/myCollection/select?q=*&shards=shard1
```

```
{
  - responseHeader: {
    zkConnected: true,
    status: 0,
    QTime: 472,
    - params: {
      q: "*:*",
      shards: "shard1"
    }
  },
  - response: {
    numFound: 1,
    start: 0,
    maxScore: 1,
    - docs: [
      - {
        id: "1",
        name: "zhangsang",
        _version_: 1664863223627645000
      }
    ]
  }
}
```

需求：查询shard1和shard2分片的数据

```
http://192.168.200.128:8180/solr/myCollection/select?q=*&shards=shard1,shard2
```

上面的操作都是从根据shard的id随机从shard的副本中获取数据。shard1的数据可能来自8380,8280
shard2的数据可能来自8480,8180



也可以指定具体的副本；

需求：获取shard1分片8280副本中的数据

```
http://192.168.200.128:8180/solr/myCollection/select?
q=*&shards=192.168.200.128:8280/solr/myCollection
```

需求：获取8280和8380副本中数据

```
http://192.168.200.128:8180/solr/myCollection/select?
q=*&shards=192.168.200.128:8280/solr/myCollection,192.168.200.128:8380/solr/myCollection
```

混合使用,通过Shard的ID随机获取+通过副本获取。

```
http://192.168.200.128:8180/solr/myCollection/select?
q=*&shards=shard1,192.168.200.128:8380/solr/myCollection
```

2.2.2.3 SolrCloud并发和线程池相关的一些配置

在SolrCloud中，我们也可以配置Shard并发和线程相关的配置，来优化集群性能。

主要的配置在solr.xml文件。

```
<solr>
  <!--集群相关的配置,solr服务的地址,端口,上下文solr.是否自动生成solrCore名称,zk超时时间-->
  <solrcloud>
    <str name="host">192.168.200.131</str>
    <int name="hostPort">8180</int>
    <str name="hostContext">${hostContext:solr}</str>
    <bool name="genericCoreNodeNames">${genericCoreNodeNames:true}</bool>
    <int name="zkClientTimeout">${zkClientTimeout:30000}</int>
    <int name="distribUpdateSoTimeout">${distribUpdateSoTimeout:600000}</int>
    <int name="distribUpdateConnTimeout">${distribUpdateConnTimeout:60000}</int>
    <str
name="zkCredentialsProvider">${zkCredentialsProvider:org.apache.solr.common.cloud.DefaultZkCredentialsProvider}</str>
    <str
name="zkACLProvider">${zkACLProvider:org.apache.solr.common.cloud.DefaultZkACLProvider}</str>
  </solrcloud>

  <!-- 分片并发和线程相关的信息-->
  <shardHandlerFactory name="shardHandlerFactory"
    class="HttpShardHandlerFactory">
    <int name="socketTimeout">${socketTimeout:600000}</int>
    <int name="connTimeout">${connTimeout:60000}</int>
    <str name="shardswhitelist">${solr.shardswhitelist:}</str>
  </shardHandlerFactory>

</solr>
```

相关参数

参数名	描述	默认值
socketTimeout	指客户端和服务端建立连接后,客户端从服务器读取数据的timeout	distribUpdateSoTimeout
connTimeout	指客户端和服务端建立连接的timeout	distribUpdateConnTimeout
maxConnectionsPerHost	最大并发数量	20
maxConnections	最大连接数量	10000
corePoolSize	线程池初始化线程数量	0
maximumPoolSize	线程池中线程的最大数量	Integer.MAX_VALUE
maxThreadIdleTime	设置线程在被回收之前空闲的最大时间	5秒
sizeOfQueue	指定此参数，那么线程池会使用队列来代替缓冲区，对于追求高吞吐量的系统而，可能希望配置为-1。对于追求低延迟的系统可能希望配置合理的队列大小来处理请求。	-1
fairnessPolicy	用于选择JVM特定的队列的公平策略：如果启用公平策略，分布式查询会以先进先出的方式来处理请求，但是是以有损吞吐量为代价；如果禁用公平策略，会提高集群查询吞吐量，但是以响应延迟为代价。	false

参数名	描述	默认值
useRetries	是否启 HTTP 连接自动重试机制，避免由 HTTP 连接池 的限制或者竞争导致的 IOException ，默认未开启	false

2.3.2 基于docker的集群搭建

2.3.2.1 环境准备

1.搭建docker

要想在docker上搭建solr集群，首先安装docker的环境。这个就不再演示，如果没有学过docker的同学可以参考下面的视频地址进行学习。

<https://www.boxuegu.com/freecourse/detail-1553.html>

如果学习过但是忘了如何搭建，参考一下地址。

<https://www.runoob.com/docker/centos-docker-install.html>

2. 拉取zookeeper镜像和solr的镜像，采用的版本是3.4.14和7.7.2

```
docker pull zookeeper:3.4.14
docker pull solr:7.7.2
```

查看拉取镜像

```
docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
docker.io/solr	7.7.2	eb2129dc999e	3 weeks ago	807.2 MB
docker.io/zookeeper	3.4.14	8cf98592b865	3 weeks ago	256.2 MB
jdk8	latest	d60a80636d1d	6 months ago	584 MB
192.168.171.138:5050/mynginx	latest	424fbeb030f5	6 months ago	121.7 MB
mynginx	latest	424fbeb030f5	6 months ago	121.7 MB
docker.io/mysql	latest	2151acc12881	7 months ago	444.6 MB
docker.io/registry	latest	f32a97de94e1	11 months ago	25.76 MB
docker.io/tomcat	7-jre7	e1ac7618b15d	2 years ago	454.3 MB
docker.io/redis	latest	1fb7b6c8c0d0	2 years ago	106.6 MB
docker.io/nginx	latest	1e5ab59102ce	2 years ago	108.3 MB
docker.io/centos	7	196e0ce0c9fb	2 years ago	196.6 MB

2.3.2.2 搭建zookeeper集群

搭建zookeeper集群，我们需要利用我们刚才拉取的zookeeper的镜像创建3个容器。并且让他们产生集群关系。

单独创建一个桥接网卡

```
docker network create itcast-zookeeper
docker network ls
```

1.容器1创建

```
docker run
  -id
  --restart=always
  -v /opt/docker/zookeeper/zoo1/data:/data
  -v /opt/docker/zookeeper/zoo1/data/log:/data/log
```

```

-e ZOO_MY_ID=1
-e ZOO_SERVERS="server.1=zookeeper1:2888:3888 server.2=zookeeper2:2888:3888
server.3=zookeeper3:2888:3888"
-p 2181:2181
--name=zookeeper1
--net=itcast-zookeeper
--privileged
zookeeper:3.4.14

docker run -id --restart=always -v /opt/docker/zookeeper/zoo1/data:/data -v
/opt/docker/zookeeper/zoo1/data/log:/data/log -e ZOO_MY_ID=1 -e
ZOO_SERVERS="server.1=zookeeper1:2888:3888 server.2=zookeeper2:2888:3888
server.3=zookeeper3:2888:3888" -p 2181:2181 --name=zookeeper1 --privileged --
net=itcast-zookeeper zookeeper:3.4.14

```

说明:

- restart:docker重启, 容器重启。
- d:守护式方式运行方式, 除此之外还有-it
- v:目录挂载, 用宿主机/opt/docker/zookeeper/zoo1/data, 映射容器的/data目录
- e:指定环境参数。分别指定了自己的id,集群中其他节点的地址, 包含通信端口和投票端口
- name:容器的名称
- p:端口映射, 用宿主机2181映射容器 2181,将来我们就需要通过本机2181访问容器。
- privileged:开启特权, 运行修改容器参数
- net=itcast-zookeeper 指定桥接网卡
- zookeeper:3.4.14: 创建容器的镜像

查看创建好的容器

```
docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
c8f84ecb9b2c	zookeeper:3.4.14	"/docker-entrypoint.s"	About a minute ago	Up About a minute	0.0.0.0:2181->2181/tcp, 0.0.0.0:2888->2888/tcp, 0.0.0.0:3888->3888/tcp	zookeeper1

查看宿主机的目录,该目录映射的就是zookeeper1容器中的data目录。

```
cd /opt/docker/zookeeper/zoo1/data
```

查看Zookeeper节点的id

```
cat myid
```

2.容器2创建

```

docker run -d --restart=always
-v /opt/docker/zookeeper/zoo2/data:/data
-v /opt/docker/zookeeper/zoo2/data/log:/data/log
-e ZOO_MY_ID=2
-e ZOO_SERVERS="server.1=zookeeper1:2888:3888 server.2=zookeeper2:2888:3888
server.3=zookeeper3:2888:3888"
-p 2182:2181
--name=zookeeper2
--net=itcast-zookeeper

```

```
--privileged
zookeeper:3.4.14
```

```
docker run -d --restart=always -v /opt/docker/zookeeper/zoo2/data:/data -v
/opt/docker/zookeeper/zoo2/data/log:/data/log -e ZOO_MY_ID=2 -e
ZOO_SERVERS="server.1=zookeeper1:2888:3888 server.2=zookeeper2:2888:3888
server.3=zookeeper3:2888:3888" -p 2182:2181 --name=zookeeper2 --net=itcast-
zookeeper --privileged zookeeper:3.4.14
```

说明:

需要修改目录挂载。

修改Zookeeper的id

端口映射: 用宿主机2182 映射容器 2181

容器名称: zookeeper2

3.容器3创建

```
docker run -d --restart=always
-v /opt/docker/zookeeper/zoo3/data:/data
-v /opt/docker/zookeeper/zoo3/data/log:/data/log
-e ZOO_MY_ID=3
-e ZOO_SERVERS="server.1=zookeeper1:2888:3888 server.2=zookeeper2:2888:3888
server.3=zookeeper3:2888:3888"
-p 2183:2181
--name=zookeeper3
--net=itcast-zookeeper
--privileged
zookeeper:3.4.14
```

```
docker run -d --restart=always -v /opt/docker/zookeeper/zoo3/data:/data -v
/opt/docker/zookeeper/zoo3/data/log:/data/log -e ZOO_MY_ID=3 -e
ZOO_SERVERS="server.1=zookeeper1:2888:3888 server.2=zookeeper2:2888:3888
server.3=zookeeper3:2888:3888" -p 2183:2181 --name=zookeeper3 --net=itcast-
zookeeper --privileged zookeeper:3.4.14
```

说明:

需要修改目录挂载。

修改Zookeeper的id

端口映射: 用宿主机2183 映射容器 2181

容器名称: zookeeper3

查看容器创建情况

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
8128e0278d58	zookeeper:3.4.14	"/docker-entrypoint.s"	8 seconds ago	Up 7 seconds	2888/tcp, 3888/tcp, 0.0.0.0:2183->2181/tcp	zookeeper3
65fe69aa7d14	zookeeper:3.4.14	"/docker-entrypoint.s"	About a minute ago	Up About a minute	2888/tcp, 3888/tcp, 0.0.0.0:2182->2181/tcp	zookeeper2
06c4bdeb24a7	zookeeper:3.4.14	"/docker-entrypoint.s"	3 minutes ago	Up 3 minutes	2888/tcp, 0.0.0.0:2181->2181/tcp, 3888/tcp	zookeeper1

2.3.2.3 测试Zookeeper集群的搭建情况

使用yum安装nc指令

```
yum install -y nc
```

方式1:

通过容器的ip查看Zookeeper容器状态

查看三个Zookeeper容器的ip


```
docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
zookeeper1
docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
zookeeper2
docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
zookeeper3
```

查看Zookeeper集群的状态

```
echo stat|nc ip 2181
echo stat|nc ip 2181
echo stat|nc ip 2181
```

```
[root@pinyougou-docker conf]# docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' zookeeper1
172.19.0.2
[root@pinyougou-docker conf]# docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' zookeeper2
172.19.0.3
[root@pinyougou-docker conf]# docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' zookeeper3
172.19.0.4
[root@pinyougou-docker conf]# echo stat|nc 172.19.0.2 2181
Zookeeper version: 3.4.14-4c25d480e66aadd371de8bd2fd8da255ac140bcf, built on 03/06/2019 16:18 GMT
Clients:
 /172.19.0.1:57564[0](queued=0,recved=1,sent=0)
 /192.168.200.1:2783[1](queued=0,recved=100,sent=100)
Latency min/avg/max: 0/0/2
Received: 205
Sent: 100
Connections: 2
Outstanding: 0
Zxid: 0x100000001
Mode: follower
Node count: 4
[root@pinyougou-docker conf]# echo stat|nc 172.19.0.3 2181
Zookeeper version: 3.4.14-4c25d480e66aadd371de8bd2fd8da255ac140bcf, built on 03/06/2019 16:18 GMT
Clients:
 /172.19.0.1:57690[0](queued=0,recved=1,sent=0)
Latency min/avg/max: 0/0/0
Received: 1
Sent: 0
Connections: 1
Outstanding: 0
Zxid: 0x200000000
Mode: leader
Node count: 4
Proposal sizes last/min/max: -1/-1/-1
[root@pinyougou-docker conf]# echo stat|nc 172.19.0.4 2181
Zookeeper version: 3.4.14-4c25d480e66aadd371de8bd2fd8da255ac140bcf, built on 03/06/2019 16:18 GMT
Clients:
 /172.19.0.1:44366[0](queued=0,recved=1,sent=0)
Latency min/avg/max: 0/0/0
Received: 1
Sent: 0
Connections: 1
Outstanding: 0
Zxid: 0x100000001
Mode: follower
Node count: 4
[root@pinyougou-docker conf]#
```

方式2：通过宿主机的ip查询Zookeeper容器状态

```
[root@pinyoyougou-docker ~]# echo stat|nc 192.168.200.128 2181
Zookeeper version: 3.4.14-4c25d480e66aadd371de8bd2fd8da255ac140bcf, built on 03/06/2019 16:18 GMT
Clients:
/192.168.200.128:33332[0] (queued=0,recved=1,sent=0)

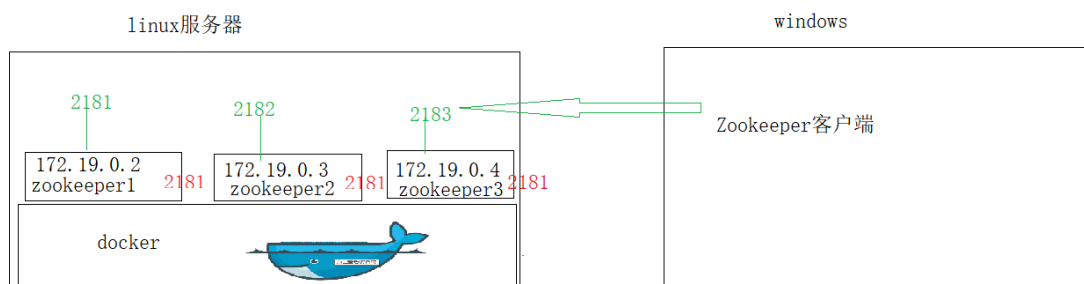
Latency min/avg/max: 0/0/0
Received: 7
Sent: 6
Connections: 1
Outstanding: 0
Zxid: 0x0
Mode: follower
Node count: 4
[root@pinyoyougou-docker ~]# echo stat|nc 192.168.200.128 2182
Zookeeper version: 3.4.14-4c25d480e66aadd371de8bd2fd8da255ac140bcf, built on 03/06/2019 16:18 GMT
Clients:
/192.168.200.128:44046[0] (queued=0,recved=1,sent=0)

Latency min/avg/max: 0/0/0
Received: 2
Sent: 1
Connections: 1
Outstanding: 0
Zxid: 0x100000000
Mode: leader
Node count: 4
Proposal sizes last/min/max: -1/-1/-1
[root@pinyoyougou-docker ~]# echo stat|nc 192.168.200.128 2183
Zookeeper version: 3.4.14-4c25d480e66aadd371de8bd2fd8da255ac140bcf, built on 03/06/2019 16:18 GMT
Clients:
/192.168.200.128:34084[0] (queued=0,recved=1,sent=0)

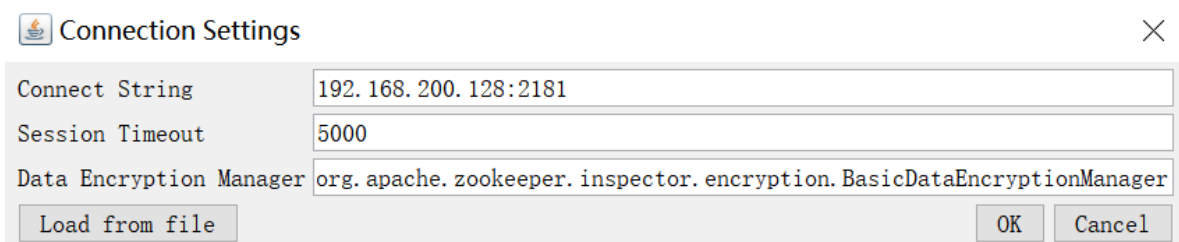
Latency min/avg/max: 0/0/0
Received: 2
Sent: 1
Connections: 1
Outstanding: 0
Zxid: 0x100000000
Mode: follower
Node count: 4
```

2.3.2.4 zookeeper集群的架构

Zookeeper集群架构



Zookeeper客户端连接Zookeeper容器



2.3.2.3 搭建solr集群

搭建Solr集群，我们需要利用我们刚才拉取的solr的镜像创建4个容器。并且需要将集群交由Zookeeper管理，Solr容器内部使用jetty作为solr的服务器。

1.容器1创建

```
docker run --name solr1 --net=itcast-zookeeper -d -p 8983:8983 solr:7.7.2 bash
-c '/opt/solr/bin/solr start -f -z
zookeeper1:2181,zookeeper2:2181,zookeeper3:2181'
--name:指定容器名称
--net:指定网卡，之前创建的桥接网卡
-d:后台运行
-p:端口映射，宿主机8983映射容器中8983
-c:
/opt/solr/bin/solr:通过容器中脚本，指定Zookeeper的集群地址
```

2.容器2创建

```
docker run --name solr2 --net=itcast-zookeeper -d -p 8984:8983 solr:7.7.2 bash
-c '/opt/solr/bin/solr start -f -z
zookeeper1:2181,zookeeper2:2181,zookeeper3:2181'
--name:指定容器名称
--net:指定网卡，之前创建的桥接网卡
-d:后台运行
-p:端口映射，宿主机8983映射容器中8984
-c:/opt/solr/bin/solr:通过容器中脚本，指定Zookeeper的集群地址
```

3.容器3创建

```
docker run --name solr3 --net=itcast-zookeeper -d -p 8985:8983 solr:7.7.2 bash
-c '/opt/solr/bin/solr start -f -z
zookeeper1:2181,zookeeper2:2181,zookeeper3:2181'
--name:指定容器名称
--net:指定网卡，之前创建的桥接网卡
-d:后台运行
-p:端口映射，宿主机8983映射容器中8984
-c:/opt/solr/bin/solr:通过容器中脚本，指定Zookeeper的集群地址
```

4.容器4创建

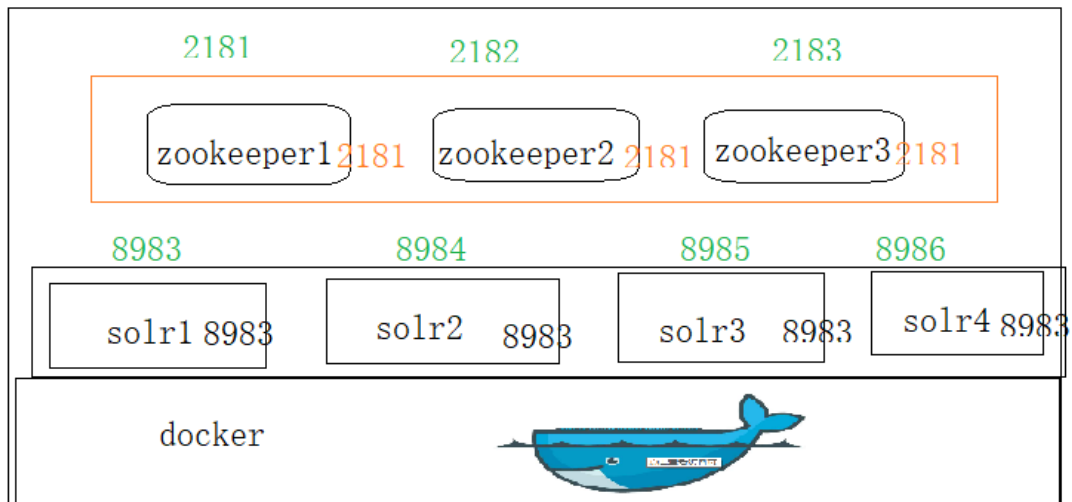
```
docker run --name solr4 --net=itcast-zookeeper -d -p 8986:8983 solr:7.7.2 bash
-c '/opt/solr/bin/solr start -f -z
zookeeper1:2181,zookeeper2:2181,zookeeper3:2181'
--name:指定容器名称
--net:指定网卡，之前创建的桥接网卡
-d:后台运行
-p:端口映射，宿主机8983映射容器中8985
-c:/opt/solr/bin/solr:通过容器中脚本，指定Zookeeper的集群地址
```

5.查看结果

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
71eb0a16766c	solr:7.7.2	"docker-entrypoint.sh"	5 seconds ago	Up 3 seconds	0.0.0.0:8986->8983/tcp	solr4
c0358dc79791	solr:7.7.2	"docker-entrypoint.sh"	27 seconds ago	Up 26 seconds	0.0.0.0:8985->8983/tcp	solr3
61ab963a0037	solr:7.7.2	"docker-entrypoint.sh"	54 seconds ago	Up 53 seconds	0.0.0.0:8984->8983/tcp	solr2
eb84f8e71f3a	solr:7.7.2	"docker-entrypoint.sh"	About a minute ago	Up About a minute	0.0.0.0:8983->8983/tcp	solr1

2.3.2.5 整体架构

linux



2.3.2.6 创建集群的逻辑结构

1.通过端口映射访问solr的后台系统（注意关闭防火墙）

```
http://192.168.200.128:8983/solr
http://192.168.200.128:8984/solr
http://192.168.200.128:8985/solr
http://192.168.200.128:8986/solr
```

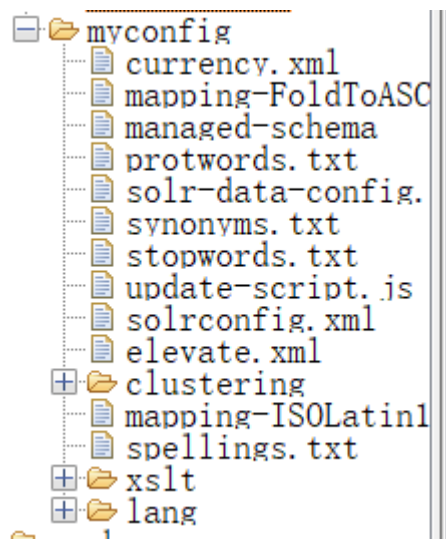
2.上传solr配置文件到Zookeeper集群。

进入到某个solr容器中，使用/opt/solr/server/scripts/cloud-scripts/zkcli.sh。上传solr的配置文件。

位置/opt/solr/example/example-DIH/solr/solr/conf到Zookeeper集群
zookeeper1:2181,zookeeper2:2181,zookeeper3:2181。

```
docker exec -it solr1 /opt/solr/server/scripts/cloud-scripts/zkcli.sh -zkhost
zookeeper1:2181,zookeeper2:2181,zookeeper3:2181 -cmd upconfig -confdir
/opt/solr/server/solr/configsets/sample_techproducts_configs/conf -confname
myconfig
```

3.使用zooInterceptor查看配置文件上传情况



4.使用后台管理系统创建connection

name:

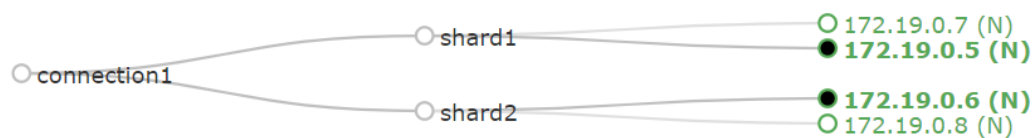
config set:

numShards:

replicationFactor:

Show advanced ☒

5. 查看集群的逻辑结构



到这基于docker的solr集群我们就讲解完毕。

6.测试集群

```
{id:1,name:"zx"}
```

2.3.2.7 solr配置文件修改

1.在linux服务器上需要有一份solr的配置文件,修改宿主机中的配置文件。

之前单机版solr的/usr/local/solr_home下,就有solr的配置文件。

/usr/local/solr_home下面的配置文件, schema已经配置了基于IK分词器的FieldType

2.将宿主机中的配置文件, 替换到某个solr容器中。

```
docker cp /usr/local/solr_home/collection1/conf/managed-schema  
solr1:/opt/solr/server/solr/configsets/sample_techproducts_configs/conf/managed-  
schema
```

3.将容器中修改后managed-schema配置文件, 重新上传到Zookeeper集群。

进入到solr1容器中, 使用zkcli.sh命令, 将最新的配置文件上传到Zookeeper集群

容器文件位置: /opt/solr/server/solr/configsets/sample_techproducts_configs/conf/managed-schema

集群文件位置: /configs/myconfig/managed-schema

```
docker exec -it solr1 /opt/solr/server/scripts/cloud-scripts/zkcli.sh -zkhost  
zookeeper1:2181,zookeeper2:2181,zookeeper3:2181 -cmd putfile  
/configs/myconfig/managed-schema  
/opt/solr/server/solr/configsets/sample_techproducts_configs/conf/managed-schema
```

4.在Files中查看, 修改后的配置文件

5.将IK分词器的jar包, 及配置文件复制到4个solr容器中。

5.1 将IK分词器上传到linux服务器

5.2 使用docker cp命令进行jar包的拷贝

```
docker cp /root/ik-analyzer-solr5-5.x.jar solr1:/opt/solr/server/solr-  
webapp/webapp/WEB-INF/lib/ik-analyzer-solr5-5.x.jar
```

```
docker cp /root/ik-analyzer-solr5-5.x.jar solr2:/opt/solr/server/solr-  
webapp/webapp/WEB-INF/lib/ik-analyzer-solr5-5.x.jar
```

```
docker cp /root/ik-analyzer-solr5-5.x.jar solr3:/opt/solr/server/solr-  
webapp/webapp/WEB-INF/lib/ik-analyzer-solr5-5.x.jar
```

```
docker cp /root/ik-analyzer-solr5-5.x.jar solr4:/opt/solr/server/solr-  
webapp/webapp/WEB-INF/lib/ik-analyzer-solr5-5.x.jar
```

5.3 使用docker cp命令进行配置文件的拷贝

将IK分析器的相关配置文件复制到/root/classes目录中

```
docker cp /root/classes solr1:/opt/solr/server/solr-webapp/webapp/WEB-  
INF/classes
```

```
docker cp /root/classes solr2:/opt/solr/server/solr-webapp/webapp/WEB-INF/classes
```

```
docker cp /root/classes solr3:/opt/solr/server/solr-webapp/webapp/WEB-INF/classes
```


```
docker cp /root/classes solr4:/opt/solr/server/solr-webapp/webapp/WEB-INF/classes
```


6、重启所有solr容器


```
docker restart solr1  
docker restart solr2  
docker restart solr3  
docker restart solr4
```

7、使用text_ik创建业务域

不再使用修改配置文件的方式，直接利用后台管理系统进行。

 Add Field

 Add Dynamic Field

 Add Copy Field

name:

field type:

default:

☒ stored

☒ indexed

☐ uninvertible

☐ docValues


☐ multiValued


☐ required

Show omit options ☒

Show term vector options ☒

Show sort options ☒

 Add Field

 Cancel

```
{id:2,book_name:"java编程思想"}
```

2.3.2.7 使用SolrJ操作solr集群

1.修改Zookeeper地址和connection名称

```
@Bean
public CloudSolrClient cloudSolrClient() {
    //指定Zookeeper的地址。
    List<String> zkHosts = new ArrayList<>();
    zkHosts.add("192.168.200.128:2181");
    zkHosts.add("192.168.200.128:2182");
    zkHosts.add("192.168.200.128:2183");
    //参数2: 指定
```



```

        CloudSolrClient.Builder builder = new
CloudSolrClient.Builder(zkHosts,Optional.empty());
        CloudSolrClient solrClient = builder.build();
        //设置连接超时时间
        solrClient.setZkClientTimeout(30000);
        solrClient.setZkConnectTimeout(30000);
        //设置collection
        solrClient.setDefaultCollection("myCollection");
        return solrClient;
    }

```

2. 将之前的索引和搜索的代码执行一下

```

@Test
    public void testSolrCloudAddDocument() throws Exception {
        SolrInputDocument doc = new SolrInputDocument();
        doc.setField("id", "3");
        doc.setField("book_name", "葵花宝典");

        cloudSolrClient.add(doc);
        cloudSolrClient.commit();
    }

@Test
    public void testSolrQuery() throws Exception {
        SolrQuery params = new SolrQuery();
        params.setQuery("*:~");
        QueryResponse response = cloudSolrClient.query(params);
        SolrDocumentList results = response.getResults();
        for (SolrDocument result : results) {
            System.out.println(result);
        }
    }
}

```

3. 执行之前的代码会报错

Caused by: org.apache.http.conn.ConnectTimeoutException: Connect to 172.19.0.6:8983 [/172.19.0.6] failed: connect timed out

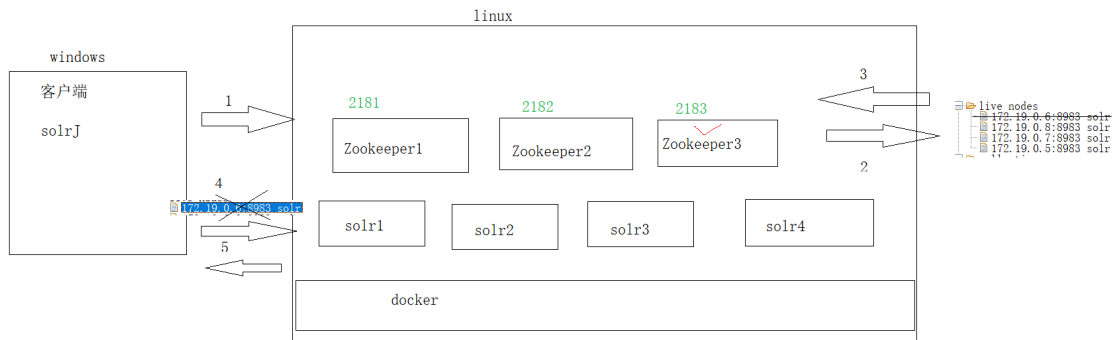
错误的原因：连接不上172.19.0.6.8983。

这个容器是谁？

Host	Node	CPU	Heap	Disk usage	Requests	Collections	Replicas
172.19.0.5 Linux 984.8Mb Java 11 Load: 0.25 show details...	8983_solr Uptime: 1h 53m show details...	0%	23%	2.3Kb	RPM: 0 p95: 90ms	haha	haha_s2r6 (1 docs)
172.19.0.6 Linux 984.8Mb Java 11 Load: 0.25 show details...	8983_solr Uptime: 1h 53m show details...	0%	13%	2.3Kb	RPM: 0.01 p95: 192ms	haha	haha_s2r4 (1 docs)
172.19.0.7 Linux 984.8Mb Java 11 Load: 0.25 show details...	8983_solr Uptime: 1h 53m show details...	0%	17%	69.0b	RPM: 0 p95: 967ms	haha	haha_s1r2 (0 docs)
172.19.0.8 Linux 984.8Mb Java 11 Load: 0.25 show details...	8983_solr Uptime: 1h 53m show details...	0%	18%	69.0b	RPM: 0 p95: 407ms	haha	haha_s1r1 (0 docs)

为什么连接不上呢？

4. 错误的原因



1. 客户端将添加的请求发送给Zookeeper集群，由Zookeeper的某个节点处理请求。eg: Zookeeper3
2. Zookeeper根据存储的solr活动节点的列表，获取指定的solr节点地址，并且将solr节点地址返回给客户端。
3. 客户端根据solr节点地址，访问solr节点完成添加操作。
4. 客户端获取执行结果

当前问题，Zookeeper返回solr节点的地址，客户端是无法连接的。

5. 正确的测试方案

将客户端应用部署到docker容器中，容器之间是可以相互通信的。

如何将我们的应用部署到docker容器中。

5.1 修改代码，将添加文档的操作写在controller方法中，便于我们进行测试；

```
@RestController
@RequestMapping("/test")
public class TestController {
    @Autowired
    private CloudSolrClient cloudSolrClient;
    @RequestMapping("/addDocument")
    public Map<String, Object> addDocument(String id, String bookName)
    throws IOException, SolrServerException {
        SolrInputDocument doc = new SolrInputDocument();
        doc.setField("id", id);
        doc.setField("book_name", bookName);
        cloudSolrClient.add(doc);
        cloudSolrClient.commit();
        Map<String, Object> result = new HashMap<>();
        result.put("flag", true);
        result.put("msg", "添加文档成功");
        return result;
    }
}
```

5.2 将spring boot项目打成可以运行jar包。

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

5.3 使用package命令打成jar

```
clean package -DskipTests=true
```

5.4 将jar包上传到linux服务器制作镜像

5.4.1 上传

5.4.2 编写DockerFile文件

```
#基础镜像
FROM java:8
#作者
MAINTAINER itcast
#将jar包添加到镜像app.jar
ADD solr_solrj_other-1.0-SNAPSHOT.jar app.jar
#运行app的命令 java -jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

5.4.3 创建镜像

```
docker build -t myapp:1.0 ./ 使用当前目录的DockerFile文件创建镜像myapp，版本1.0
```

查看镜像

```
docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
myapp	1.0	b60a86b0bfb5	About a minute ago	643.1 MB

5.4.4 利用myapp:1.0镜像创建容器

```
docker run -d --net=itcast-zookeeper -p 8888:8080 myapp:1.0 #使用本地8888映射容器中8080
```

5.4.5 查看日志

```
docker logs -f 容器id
```

5.5.6 访问controller

```
http://192.168.200.128:8888/test/addDocument?id=999&bookName=葵花宝典
```

注意：如果发现连接不上Zookeeper，注意关闭防火墙；

```
systemctl stop firewalld
```

3.Solr查询高级

3.1 函数查询

在Solr中提供了大量的函数，允许我们在查询的时候使用函数进行动态运算。

3.2.1 函数调用语法

函数调用语法和java基本相同,格式为:函数名(实参)，实参可以是常量（100,1.45,"hello"），也可以是域名，下面是函数调用例子。

```
functionName()  
functionName(...)  
functionName1(functionName2(...),...)
```

数据准备：为了便于我们后面的讲解，采用的案例数据,员工信息。

业务域：

员工姓名,工作，领导的员工号，入职日期，月薪，奖金

```
<field name="emp_ename" type="text_ik" indexed="true" stored="true"/>  
<field name="emp_ejob" type="text_ik" indexed="true" stored="true"/>  
<field name="emp_mgr" type="string" indexed="true" stored="true"/>  
<field name="emp_hiredate" type="pdate" indexed="true" stored="true"/>  
<field name="emp_sal" type="pfloat" indexed="true" stored="true"/>  
<field name="emp_comm" type="pfloat" indexed="true" stored="true"/>  
<field name="emp_cv" type="text_ik" indexed="true" stored="false"/>  
<field name="emp_deptno" type="string" indexed="true" stored="true"/>
```

data-import.xml

```
<entity name="emp" query="select * from emp">  
  <!-- 每一个field映射着数据库中列与文档中的域，column是数据库列，name是solr的  
域(必须是在managed-schema文件中配置过的域才行) -->  
  <field column="empno" name="id"/>  
  <field column="ename" name="emp_ename"/>  
  <field column="ejob" name="emp_ejob"/>  
  <field column="mgr" name="emp_mgr"/>  
  <field column="hiredate" name="emp_hiredate"/>  
  <field column="sal" name="emp_sal"/>  
  <field column="comm" name="emp_comm"/>  
  <field column="cv" name="emp_cv"/>  
  <field column="deptno" name="emp_deptno"/>  
</entity>
```

3.2.2 函数的使用

3.2.2.1 将函数返回值作为关键字

在solr中通常的搜索方式都是根据关键字进行搜索。底层首先要对关键字分词，在倒排索引表中根据词查找对应的文档返回。

```
item_title:三星手机
```

需求：查询年薪在30000~50000的员工。

主查询条件:查询所有:

过滤条件: 利用product()函数计算员工年薪, product函数是Solr自带函数, 可以计算2个数之积。{!frange}指定范围。l表示最小值, u表示最大值。{!frange}语法是Function Query Parser解析器中提供的一种语法。

```
q=*:*&fq={!frange l=30000 u=50000}product(emp_sal,12)
```

需求: 查询年薪+奖金大于50000的员工

通过sum函数计算年薪+奖金。

```
q=*:*&fq={!frange l=50000}sum(product(emp_sal,12),emp_comm)
上面查询不出来数据, 原因是有些员工emp_comm为空值。
emp_comm进行空值处理。判断emp_comm是否存在, 不存在指定0, 存在, 本身的值
exists判断域是否存在
if指定存在和不存在值
{!frange l=40000}sum(product(emp_sal,12),if(exists(emp_comm),emp_comm,0))
```

3.2.2.1.2 将函数返回值作为伪域

伪域: 文档中本身不包含的域, 利用函数生成。

需求: 查询所有的文档, 每个文档中包含年薪域。

```
q=*:*&
fl=*,product(emp_sal,12)
指定别名: yearsql:product(emp_sal,12) 便于将来封装
```

3.2.2.1.3 根据函数进行排序

需求: 查询所有的文档, 并且按照 (年薪+奖金) 之和降序排序

```
q=*:*&
sort=sum(product(emp_sal,12),if(exists(emp_comm),emp_comm,0)) desc
```

3.2.2.1.4 使用函数影响相关度排序 (了解)

之前在讲解查询的时候, 我们说的相关度排序, Lucene底层会根据查询的词和文档关系进行相关度打分。

打分高的文档排在前面, 打分低的排在后面。打分的2个指标tf,df;

使用函数可以影响相关度排序。

需求: 查询emp_cv中包含董事长的员工, 年薪高的员工相关度打分高一点。

q=emp_cv:董事长

```
AND _val_:"product(emp_sal,12)" #用来影响相关度排序,不会影响查询结果
```

可以简单这么理解

lucene本身对查询文档有一个打分叫score，每个文档分数可能不同。

通过函数干预后，打分score += 函数计算结果

val:规定

另外一种写法：

```
AND _query_:"{!func}product(emp_sal,12)"
```

3.2.2.1.5 Solr内置函数和自定义函数

常用内置函数

函数	说明	举例
abs(x)	返回绝对值	abs(-5)
def(field,value)	获取指定域中值，如果文档没有指定域，使用value作为默认值；	def(emp_comm,0)
div(x,y)	除法，x除以y	div(1,5)
dist	计算两点之间的距离	dis(2,x,y,0,0)
docfreq(field,val)	返回某值在某域出现的次数	docfreq(title,'solr')
field(field)	返回该field的索引数量	field('title')
hsin	曲面圆弧上两点之间的距离	hsin(2,true,x,y,0,0)
idf	Inverse document frequency 倒排文档频率	idf(field,'solr')
if	if(test,value1,value2)	if(termfreq(title,'solr'),popularity,42)
linear(x,m,c)	就是 $m*x+c$,等同于 $sum(product(m,x),c)$	linear(1,2,4)= $1 \times 2 + 4 = 6$
log(x)	以10为底，x的对数	log(sum(x,100))
map(x,min,max,target)	如果x在min和max之间, $x=target$,否则 $x=x$	map(x,0,0,1)
max(x,y,...)	返回最大值	max(2,3,0)
maxdoc	返回索引的个数，查看有多少文档，包括被标记为删除状态的文档	maxdoc()
min(x,y,...)	返回最小值	min(2,4,0)
ms	返回两个参数间毫秒级的差别	ms(datefield1,2000-01-01T00:00:00Z)
norm(field)	返回该字段索引值的范数	norm(title)
numdocs	返回索引的个数，查看有多少文档，不包括被标记为删除状态的文档	numdocs()
ord	根据顺序索引发货结果	ord(title)
pow(x,y)	返回x的y次方	pow(x,log(y))
product(x,y)	返回多个值得乘积	product(x,2)
query	返回给定的子查询的得分，或者文档不匹配的默认值	query(subquery,default)
recip(x,m,a,b)	相当于 $a/(m*x+b)$,a,m,b是常量，x是变量	recip(myfield,m,a,b)
rord	按ord的结果反序返回	
scale	返回一个在最大值和最小值之间的值	scale(x,1,3)
sqedist	平方欧氏距离计算	sqedist(x_td,y_td,0,0)

函数	说明	举例
sqrt	返回指定值得平方根	sqrt(x)sqrt(100)
strdist	计算两个字符串之间的距离	strdist("SOLR",id,edit)
sub	返回x-y	sub(field1,field2)
sum(x,y)	返回指定值的和	sum(x,y,...)
sumtotaltermfreq	返回所有totaltermfreq的和	
termfreq	词出现的次数	termfreq(title,'solr')
tf	词频	tf(text,'solr')
top	功能类似于ord	
totaltermfreq	返回这个词在该字段出现的次数	ttf(title,'memory')
and	返回true值当且仅当它的所有操作为true	and(not(exists(popularity)),exists(price))
or	返回true值当有一个操作为true	or(value1,value2)
xor	返回false值如果所有操作都为真	xor(field1,field2)
not	排除操作	not(exists(title))
exists	如果字段存在返回真	exists(title)
gt,gte,lt,lte,eq	比较函数	2 gt 1

自定义函数

自定义函数流程：

准备：

```
<dependency>
  <groupId>org.apache.solr</groupId>
  <artifactId>solr-core</artifactId>
  <version>7.7.2</version>
</dependency>
```

- 1.定义一个类继承 ValueSource 类，用于封装函数执行结果。
- 2.定义一个类继承ValueSourceParser，用来解析函数调用语法，并且可以解析参数类型，以及返回指定结果。
3. 在solrconfig.xml中配置文件中配置ValueSourceParser实现类的全类名

需求：定义一个solr字符串拼接的函数，将来函数的调用语法concat(域1,域2)，完成字符串拼接。

第一步：

```
/**
 * 用来解析函数调用语法，并且将解析的参数返回给第一个类处理。
 * concat(field1,field2)
 */
```



```

public class ConcatValueSourceParser extends ValueSourceParser {
    //对函数调用语法进行解析, 获取参数, 将参数封装到ValueSource, 返回函数执行结果
    public ValueSource parse(FunctionQParser functionQParser) throws SyntaxError
    {
        //1. 获取函数调用参数, 由于参数类型可能是字符串, int类型, 域名, 所以封装成ValueSource
        //1.1 获取第一个参数封装对象
        ValueSource arg1 = functionQParser.parseValueSource();
        //1.2 获取第二个参数封装对象
        ValueSource arg2 = functionQParser.parseValueSource();

        //返回函数执行结果
        return new ConcatValueSource(arg1, arg2);
    }
}

```

第二步:

```

public class ConcatValueSource extends ValueSource {

    //使用构造方法接收ConcatValueSourceParser解析获取的参数
    private ValueSource arg1;
    private ValueSource arg2;

    public ConcatValueSource(ValueSource arg1, ValueSource arg2) {
        this.arg1 = arg1;
        this.arg2 = arg2;
    }

    //返回arg1和arg2计算结果
    public FunctionValues getValues(Map map, LeafReaderContext
leafReaderContext) throws IOException {
        FunctionValues values1 = arg1.getValues(map, leafReaderContext); //第一个
参数
        FunctionValues values2 = arg2.getValues(map, leafReaderContext); //第二个
参数

        //返回一个字符串的结果
        return new StrDocValues(this) {
            @Override
            public String strVal(int doc) throws IOException {
                //根据参数从文档中获取值
                return values1.strVal(doc) + values2.strVal(doc);
            }
        };
    }
}

```

第三步: 在solrconfig.xml配置

```

<valueSourceParser name="concat"
class="cn.itcast.funcation.ConcatValueSourceParser" />

```

第四步: 打包, 并且在solr/WEB-INF/lib 目录中配置

第五步: 测试

```

fl=*,concat(id,emp_ename)

```

3.2 地理位置查询

在Solr中也支持地理位置查询，前提是我们需要将地理位置的信息存储的Solr中，其中地理位置中必须包含一个比较重要的信息，经度和纬度，经度范围[-180,180]，纬度范围[-90,90]。对于经度来说东经为正，西经为负，对于纬度来说，北纬正，南纬为负。

3.2.1 数据准备

首先我们先来完成数据准备，需要将地理位置信息存储到Solr中；

1.在MySQL导入地理位置信息

name是地址名称

position:经纬度坐标，中间用,分割（纬度,经度）

2.schema.xml中定义业务域

name域的定义

```
<field name="address_name" type="text_ik" indexed="true" stored="true"/>
```

注意：

坐标域定义

域类型location，这域类型就是用来定义地理位置域。

使用location域类型定义的域，底层会被拆分成2个域进行存储，分别是经度和纬度。

这个域将来会被拆分成2个域进行存储。

所以需要定义动态域对底层操作进行支持。

```
<field name="address_position" type="location" indexed="true"
stored="true"/>
```

以下在schema中已经定义好了。

```
<fieldType name="location" class="solr.LatLonType"
subFieldSuffix="_coordinate"/>
<dynamicField name="*_coordinate" type="tdouble" indexed="true"
stored="false"/>
```

3.使用DataImport导入数据

```
<entity name="address" query="select * from address">
  <!-- 每一个field映射着数据库中列与文档中的域，column是数据库列，name是solr的
域(必须是在managed-schema文件中配置过的域才行) -->
  <field column="id" name="id"/>
  <field column="name" name="address_name"/>
  <field column="position" name="address_position"/>
</entity>
```

4.测试

```
{
  "address_name": "庄河",
  "id": "296",
  "address_position": "22.97, 39.7",
  "_version_": 1659668029089251335,
  "address_position_0_coordinate": 22.97,
  "address_position_1_coordinate": 39.7},
{
  "address_name": "乌鲁木齐",
  "id": "1849",
  "address_position": "87.68, 43.77",
  "_version_": 1659668029149020163,
  "address_position_0_coordinate": 87.68,
  "address_position_1_coordinate": 43.77},
{
  "address_name": "克拉玛依",
  "id": "1850",
  "address_position": "84.77, 45.59",
  "_version_": 1659668029149020164,
  "address_position_0_coordinate": 84.77,
  "address_position_1_coordinate": 45.59},
{
```

id: 主键

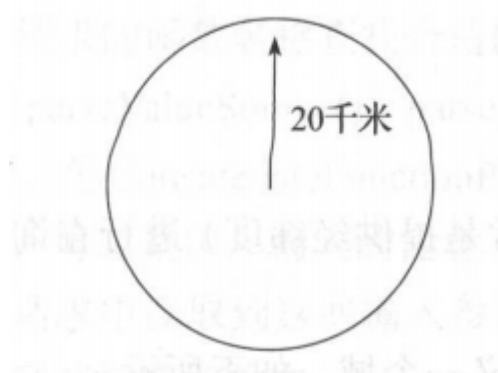
address_name: 位置名称

address_position: 位置坐标

还多出2个域: address_position_0_coordinate, address_position_1_coordinate 基于动态域生成的维度和经度坐标。

3.2.2 简单地理位置查询

Solr中 提供了一种特殊的查询解析器(geofilt)，它能解析以指定坐标（包含经纬度）为中心。以指定距离（千米）为半径画圆，坐标落在圆面积中的文档，



需求：查询离西安半径 30 千米范围内的城市。

1. 查询西安的坐标

address_name: 西安

```
{
  "address_name": "西安",
  "id": "3816",
  "address_position": "34.27, 108.95",
  "_version_": 1659668610796224515,
  "address_position_0_coordinate": 34.27,
  "address_position_1_coordinate": 108.95} ]
}}
```

2. 查询距离西安30千米范围内的城市

```
q=*&
fq={!geofilt sfield=address_position pt=34.27,108.95 d=30}
```

sfield:指定坐标域

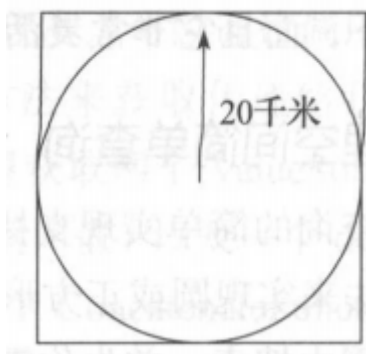
pt:中心

d:半径

```
{
  "address_name": "西安",
  "id": "3816",
  "address_position": "34.27, 108.95",
  "_version_": 1659668610796224515,
  "address_position_0_coordinate": 34.27,
  "address_position_1_coordinate": 108.95},
{
  "address_name": "长安",
  "id": "3817",
  "address_position": "34.18, 108.97",
  "_version_": 1659668610796224516,
  "address_position_0_coordinate": 34.18,
  "address_position_1_coordinate": 108.97},
{
  "address_name": "咸阳",
  "id": "3857",
  "address_position": "34.36, 108.72",
  "_version_": 1659668610801467393,
  "address_position_0_coordinate": 34.36,
  "address_position_1_coordinate": 108.72},
{
  "address_name": "临潼",
  "id": "3882",
  "address_position": "34.38, 109.22",
  "_version_": 1659668610805661701,
  "address_position_0_coordinate": 34.38,
  "address_position_1_coordinate": 109.22}]
```

除了可以查询落在圆面积中的地址外，对于精确度不高的查询，也可以查询落在外切正方形中的地理位置。

这种方式比圆的方式效率要高。



```
q=*:*&
ft={!bbox sfield=address_position pt=34.27,108.95 d=30}
```

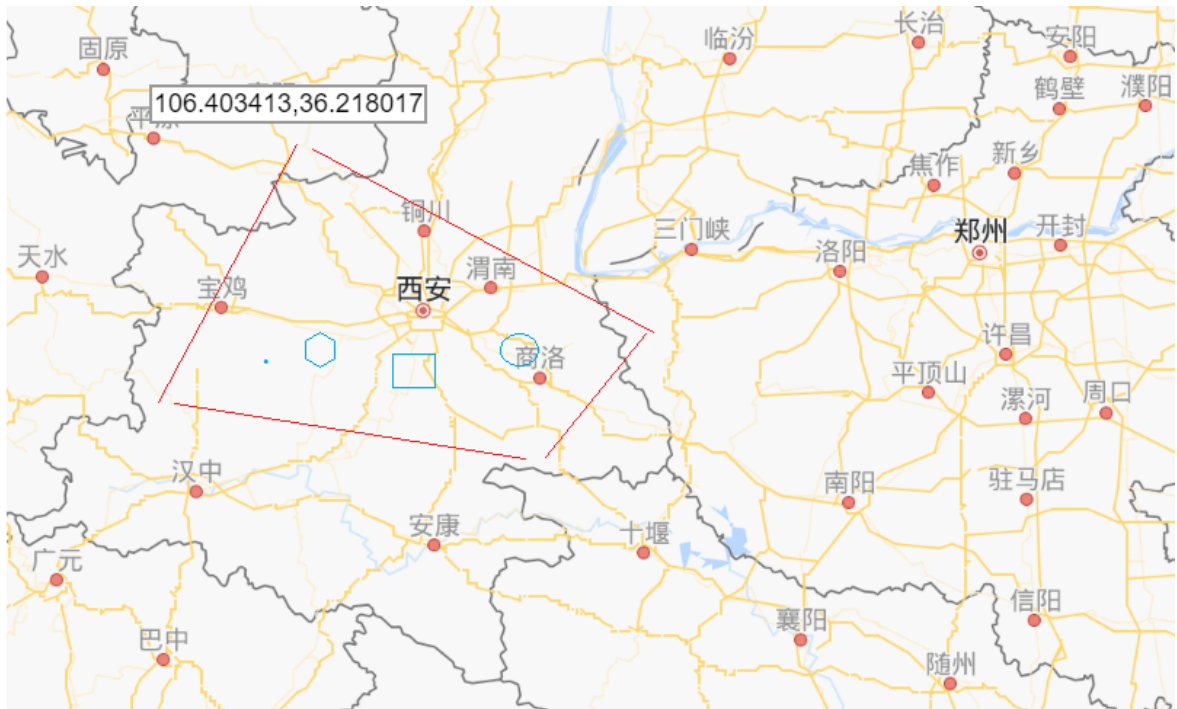
3. 在solr中还提供了一些计算地理位置的函数，比较常用的函数geodist,该函数可以计算2个地理位置中间的距离。调用方式：geodist(sfield,latitude,longitude);

需求：计算其他城市到北京的距离，作为伪域展示

```
q=*:*&
fl=*,distance:geodist(address_position,39.92,116.46)
sort=geodist(address_position,39.92,116.46) asc
```

3.2.3 地理位置高级查询

之前我们在讲解简单地理位置查询的时候，都是基于正方形或者基于圆进行。有时候我们在进行地理位置查询的时候需要基于其他形状。在Solr中提供了一个类SpatialRecursivePrefixTreeFieldType (RPT) 可以完成基于任意多边形查询。



3.2.3.1对地理数据进行重写索引

- 1.修改address_position域类型

之前的域定义，域类型location

```
<field name="address_position" type="location" indexed="true" stored="true"/>
```

修改后的域定义,域类型location_rpt

```
<field name="address_position" type="location_rpt" indexed="true" stored="true"
multivalued="true"/>
```

location_rpt类型是schema文件已经提供好的一个类型，class就是RPT

```
<fieldType name="location_rpt" class="solr.SpatialRecursivePrefixTreeFieldType"
    spatialContextFactory="JTS"
    geo="true" distErrPct="0.025" maxDistErr="0.001"
    distanceUnits="kilometers" />
```

2. location_rpt类型，允许存储的数据格式。

点：

纬度 经度
纬度,经度

任意形状

POLYGON((坐标1,坐标2...))

将来我们可以查询一个点，或者任意形状是否在另外一个几何图形中。

3. 复制jst的jar包到solr/WEB-INF/classes目录

```
<dependency>
  <groupId>org.locationtech.jts</groupId>
  <artifactId>jts-core</artifactId>
  <version>1.16.1</version>
</dependency>
```

jst: 用于构建图形的jar包

4. 插入测试数据,address_position域中存储的是一个几何图形

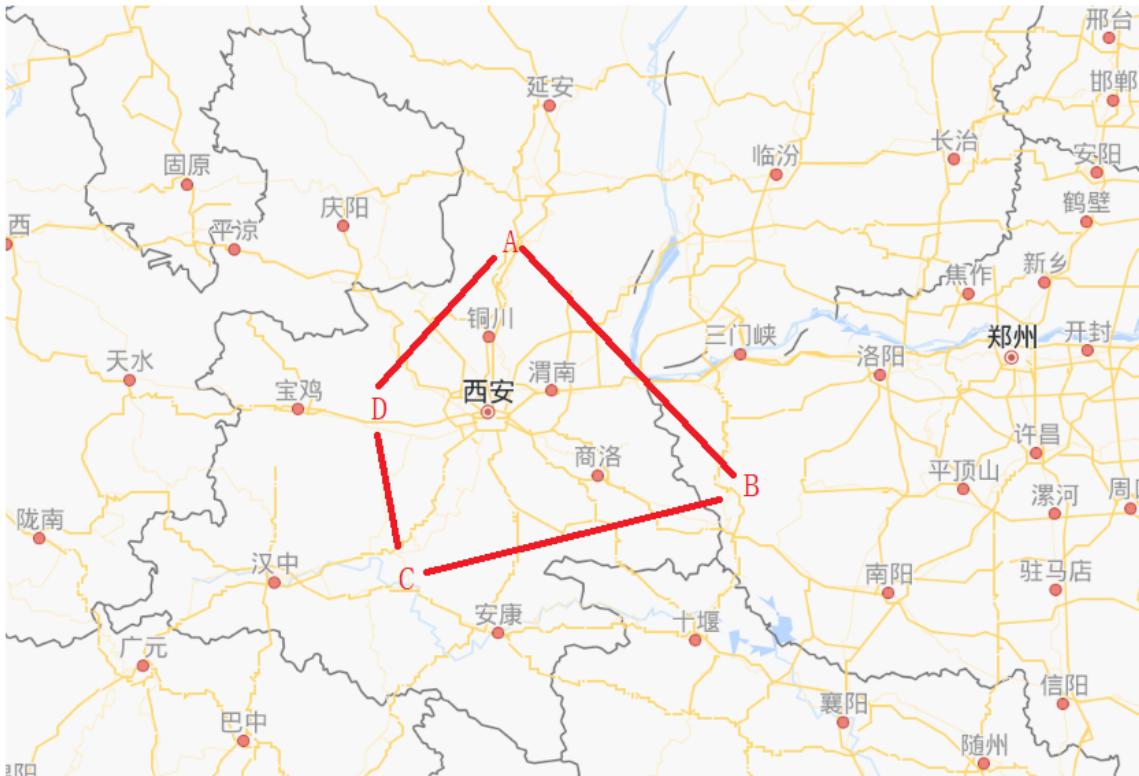
```
insert into address values(8989,'三角形任意形状','POLYGON((108.94224
35.167441,108.776664 34.788863,109.107815 34.80404,108.94224 35.167441))');
```



5. 使用DataImport重新进行索引

3.2.3.2 基于任意图形进行查询

查询基于A,B,C,D这4个点围城的图形有哪些城市。



可以使用百度地图的坐标拾取得知A,B,C,D四个点的坐标。

<http://api.map.baidu.com/lbsapi/getpoint/index.html>

A 109.034226,35.574317

B 111.315491,33.719499

C 108.243142,33.117791

D 107.98558,34.37802

利用ABCD四个点构建几个图形，查询地理坐标在该几何图形中的文档。

```
address_position:"Intersects(POLYGON((109.034226 35.574317,
111.315491 33.719499,
108.243142 33.117791,
107.98558 34.37802,
109.034226 35.574317)))"
```

说明：

POLYGON是JST中提供的构建任意多边形的语法。

每个坐标之间使用,进行分割。

经度和纬度之间使用空格隔开

如果构建的是一个封闭图形，最后一个坐标要和第一个坐标相同。

Intersects是一个操作符

作用：匹配2个图形相交部分的文档。

IsWithin

作用：匹配包含在指定图形面积之内的文档，不含边界

Contains:

作用：匹配包含在指定图形面积之内的文档，含边界

3.3 JSON Facet

在Solr中对于Facet查询提供了另外一种查询格式。称为JSON Facet。使用JSON Facet有很多好处，书写灵活，便于自定义响应结果，便于使用分组统计函数等。

3.3.1 JSON Facet 入门

根据商品分类，分组统计每个分类对应的商品数量，要求显示前3个。

传统的Facet查询

```
q=*&
facet=true&
facet.field=item_category&
facet.limit=3
```

JSON Facet

```
q=*&
json.facet={
  category:{
    type:terms,
    field:item_category,
    limit:3
  }
}
测试
http://localhost:8080/solr/collection1/select?
q=*&json.facet=%7Bcategory:%7Btype:terms,field:item_category,limit:3%7D%7D
```

其中category是自定义的一个键。

type:指的是facet的类型

field:分组统计的域

```
category: {
  - buckets: [
    - {
      val: "手机",
      count: 726
    },
    - {
      val: "平板电视",
      count: 207
    },
    - {
      val: "电子书",
      count: 9
    },
    - {
      val: "音乐",
      count: 5
    },
  ],
}
```

3.3.2 JSON Facet的语法

```
json.facet = {
  <facet_name>:{
    type:<fact_type>,
    <other_facet_parameter>
  }
}
```

facet_name:可以任意取值，是为了将来解析结果使用。

type:取值terms,query,range.

terms根据域进行Facet查询,query根据查询解决进行facet查询。range根据范围进行facet查询。

other_facet_parameter:其他facet的一些参数。

根据上面的语法，再来完成一个需求

需求：分组统计2015年每月添加的商品数量。

```
q=*:*,
json.facet={
  month_count:{
    type:range,
    field:item_createtime,
    start:'2015-01-01T00:00:00Z',
    end:'2016-01-01T00:00:00Z',
    gap:'+1MONTH'
  }
}
```

```
http://localhost:8080/solr/collection1/select?q=*:*&json.facet=%7B
  month_count:%7B
    type:range,
    field:item_createtime,
    start:'2015-01-01T00:00:00Z',
    end:'2016-01-01T00:00:00Z',
```

```
gap: '%2B1MONTH'
%7D
%7D
```

```
- month_count: {
  - buckets: [
    - {
      val: "2015-01-01T00:00:00Z",
      count: 0
    },
    - {
      val: "2015-02-01T00:00:00Z",
      count: 0
    },
    - {
      val: "2015-03-01T00:00:00Z",
      count: 933
    },
    - {
      val: "2015-04-01T00:00:00Z",
      count: 0
    },
    - {
      val: "2015-05-01T00:00:00Z",
      count: 0
    },
    - {
      val: "2015-06-01T00:00:00Z",
      count: 0
    },
  ],
}
```

需求：统计品牌是华为的商品数量，分类是平板电视的商品数量。

```
q=*:*,
json.facet={
  brand_count:{
    type:query,
    q:"item_brand:华为"
  },
  category_count:{
    type:query,
    q:"item_category:平板电视"
  }
}
```

```
http://localhost:8080/solr/collection1/select?q=*:*&json.facet=%7B
  brand_count:%7B
    type:query,
    q:"item_brand:华为"
  %7D,
  category_count:%7B
```

```
    type:query,
    q:"item_category:平板电视"
%7D
%7D
```

3.3.3 SubFacet

在JSON Facet中, 还支持另外一种Facet查询, 叫SubFacet.允许用户在之前分组统计的基础上, 再次进行分组统计,可以实现pivot Facet (多维度分组查询功能) 查询的功能。

需求: 对商品的数据, 按照品牌进行分组统计,获取前5个分组数据。

```
q=*&
json.facet={
  top_brand:{
    type:terms,
    field:item_brand,
    limit:5
  }
}
http://localhost:8080/solr/collection1/select?q=*&
json.facet=%7B
  top_brand:%7B
    type:terms,
    field:item_brand,
    limit:5
  %7D
%7D
```

```

- top_brand: {
  - buckets: [
    - {
      val: "三星",
      count: 154
    },
    - {
      val: "苹果",
      count: 94
    },
    - {
      val: "中国移动",
      count: 85
    },
    - {
      val: "联想",
      count: 76
    },
    - {
      val: "华为",
      count: 67
    }
  ]
}

```

在以上分组的基础上，统计每个品牌，不同商品分类的数据。eg:统计三星手机多少个，三星平板电视多少？

```

q=*:*&
json.facet={
  top_brand:{
    type:terms,
    field:item_brand,
    limit:5,
    facet:{
      top_category:{
        type:terms,
        field:item_category,
        limit:3
      }
    }
  }
}
http://localhost:8080/solr/collection1/select?q=*:*&
json.facet=%7B
  top_brand:%7B
    type:terms,
    field:item_brand,
    limit:5,
    facet:%7B
      top_category:%7B

```

```
type:terms,
field:item_category,
limit:3
%7D
%7D
%7D
%7D
```

```
top_brand: {
  - buckets: [
    - {
      val: "三星",
      count: 154,
      - top_category: {
        - buckets: [
          - {
            val: "手机",
            count: 134
          },
          - {
            val: "平板电视",
            count: 13
          },
          - {
            val: "净化器",
            count: 4
          }
        ]
      }
    }
  ],
  - {
    val: "苹果",
    count: 94,
    - top_category: {
      - buckets: [
        - {
          val: "手机",
          count: 93
        },
        - {
          val: "网络原创",
          count: 1
        }
      ]
    }
  ]
},
},
```

3.3.4 聚合函数查询

在JSON Facet中，还支持另外一种强大的查询功能，聚合函数查询。在Facet中也提供了很多聚合函数，也成为Facet函数。很多和SQL中的聚合函数类似。

函数如下：

聚合函数	例子	描述
sum	sum(sales)	聚合求和
avg	avg(popularity)	聚合求平均值
min	min(salary)	最小值
max	max(mul(price,popularity))	最大值
unique	unique(author)	统计域中唯一值的个数
sumsq	sumsq(rent)	平方和
percentile	percentile(salary,30,60,10)	可以根据百分比进行统计

需求：查询华为手机价格最大值,最小值，平均值.

```
q=item_title:手机
&fq=item_brand:华为
&json.facet={
  avg_price:"avg(item_price)",
  min_price:"min(item_price)",
  max_price:"max(item_price)"
}

http://localhost:8080/solr/collection1/select?q=item_title:手机
&fq=item_brand:华为
&json.facet=%7B
  avg_price:"avg(item_price)",
  min_price:"min(item_price)",
  max_price:"max(item_price)"
%7D
```

查询结果：

```
- facets: {
  count: 66,
  avg_price: 1680.560606060606,
  min_price: 468,
  max_price: 3399
}
```

需求：查询每个品牌下，最高的手机的价格.

```
q=item_title:手机
&json.facet={
  categories:{
    type:terms,
    field:item_brand,
    facet:{
      x : "max(item_price)"
    }
  }
}
http://localhost:8080/solr/collection1/select?q=item_title:手机
&json.facet=%7Bcategories:%7Btype:field,field:item_brand,facet:%7Bx:"max(item_price)"%7D%7D%7D
```

结果:

```
facets: {
  count: 716,
  - categories: {
    - buckets: [
      - {
        val: "三星",
        count: 127,
        x: 9999
      },
      - {
        val: "苹果",
        count: 93,
        x: 7388
      },
      - {
        val: "中国移动",
        count: 77,
        x: 6980
      },
      - {
        val: "联想",
        count: 68,
        x: 1999
      },
      - {
        val: "华为",
        count: 66,
        x: 3399
      },
    ]
  }
}
```