# Solr原理详解

****一、概述****

Lucene是一个Java语言编写的利用倒排原理实现的文本检索类库，Solr是以Lucene为基础实现的文本检索应用服务，SolrCloud是Solr4.0版本开发出的具有开创意义的基于Solr和Zookeeper的分布式搜索方案，主要思想是使用Zookeeper作为集群的配置信息中心。也可以说，SolrCloud是Solr的一种部署方式，除SolrCloud之外，Solr还可以以单机方和多机Master-Slaver方式进行部署。分布式索引是指当索引越来越大，一个单一的系统无法满足磁盘需求的时候，或者一次简单的查询实在要耗费很多时间的时候，我们就可以使用solr的分布式索引了。在分布式索引中，原来的大索引，将会分成多个小索引，solr可以将这些小索引返回的结果合并，然后返回给客户端。

****二、SolrCloud的基本概念****

SolrCloud模式下有Cluster，Node，Collection，Shard，LeaderCore，ReplicationCore等重要概念。

1、Cluster集群：Cluster是一组Solr节点，逻辑上作为一个单元进行管理，整个集群必须使用同一套schema和SolrConfig。

2、Node节点：一个运行Solr的JVM实例。

3、Collection：在SolrCloud集群中逻辑意义上的完整的索引,常常被划分为一个或多个Shard，这些Shard使用相同的Config Set，如果Shard数超过一个，那么索引方案就是分布式索引。SolrCloud允许客户端用户通过Collection名称引用它，这样用户不需要关心分布式检索时需要使用的和Shard相关参数。

4、Core: 也就是Solr Core，一个Solr中包含一个或者多个Solr Core，每个Solr Core可以独立提供索引和查询功能，Solr Core的提出是为了增加管理灵活性和共用资源。SolrCloud中使用的配置是在Zookeeper中的，而传统的Solr Core的配置文件是在磁盘上的配置目录中。

5、Config Set: Solr Core提供服务必须的一组配置文件,每个Config Set有一个名字。最小需要包括solrconfig.xml和schema.xml，除此之外，依据这两个文件的配置内容，可能还需要包含其它文件,如中文索引需要的词库文件。Config Set存储在Zookeeper中，可以重新上传或者使用upconfig命令进行更新，可使用Solr的启动参数bootstrap\_confdir进行初始化或更新。

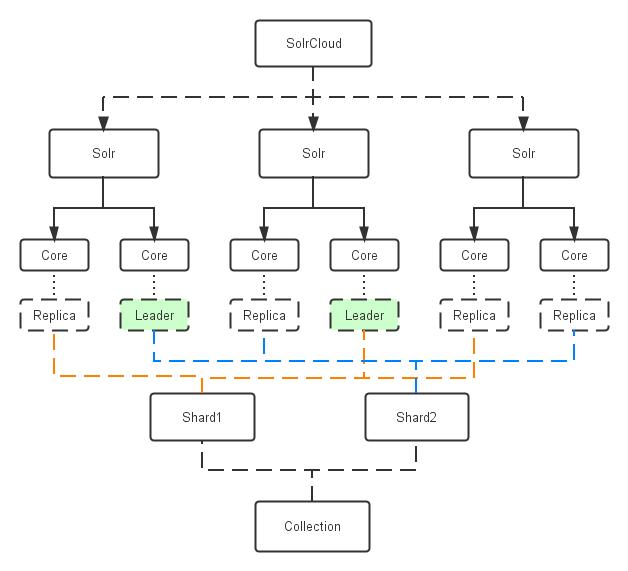
6、Shard分片: Collection的逻辑分片。每个Shard被分成一个或者多个replicas，通过选举确定哪个是Leader。

7、Replica: Shard的一个拷贝。每个Replica存在于Solr的一个Core中。换句话说一个SolrCore对应着一个Replica，如一个命名为“test”的collection以numShards=1创建，并且指定replicationFactor为2，这会产生2个replicas，也就是对应会有2个Core，分别存储在不同的机器或者Solr实例上，其中一个会被命名为test\_shard1\_replica1，另一个命名为test\_shard1\_replica2，它们中的一个会被选举为Leader。

8、 Leader: 赢得选举的Shard replicas，每个Shard有多个Replicas，这几个Replicas需要选举来确定一个Leader。选举可以发生在任何时间，但是通常他们仅在某个Solr实例发生故障时才会触发。当进行索引操作时，SolrCloud会将索引操作请求传到此Shard对应的leader，leader再分发它们到全部Shard的replicas。

9、Zookeeper: Zookeeper提供分布式锁功能，这对SolrCloud是必须的，主要负责处理Leader的选举。Solr可以以内嵌的Zookeeper运行，也可以使用独立的Zookeeper，并且Solr官方建议最好有3个以上的主机。

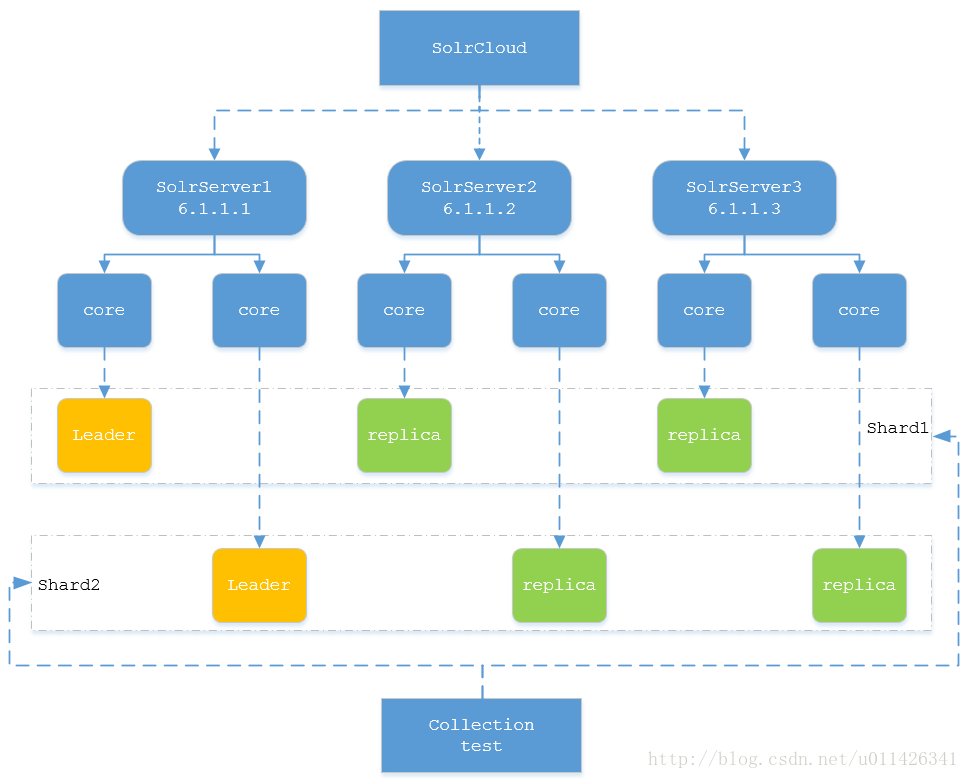
****三、SolrCloud中完整索引（Collection）的逻辑图****



在SolrCloud模式下Collection是访问Cluster的入口，这个入口有什么用呢？比如说集群里面有好多台机器，那么访问这个集群通过哪个地址呢，必须有一个接口地址，Collection就是这个接口地址。可见Collection是一个逻辑存在的东西，因此是可以跨Node的，在任意节点上都可以访问Collection。Shard其实也是逻辑存在的，因此Shard也是可以跨Node的； 1个Shard下面可以包含0个或者多个Replication，但1个Shard下面能且只能包含一个Leader  
如果Shard下面的Leader挂掉了，会从Replication里面再选举一个Leader。

此处需要注意的是在Solr4.0中，可以在Solr AdminGUI里面增加和删除Core，如果Shard里最后一个Core被删除了，Shard是不会自动删除的，这会导致集群出错， 而且如果Shard里面所有的Core宕机了，会导致不能继续插入新的记录，从而导致查询会受到影响，其实如果一个Shard下的所有Core宕机了，SolrCloud应该允许插入到其它存活的Shard里面，这在后期版本中的Solr中应该会被支持。

****四、SolrCloud索引操作的基本架构图****

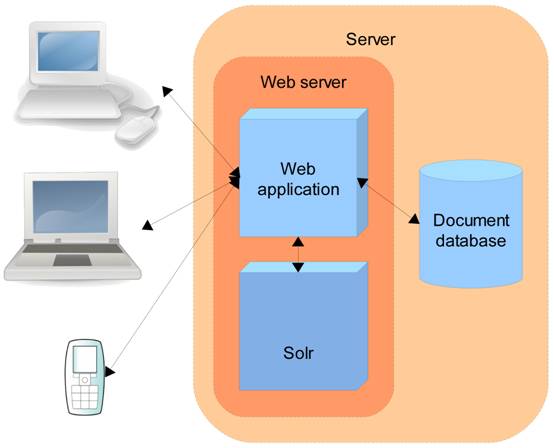


图中所示为拥有3个Solr节点的集群，索引分布在两个Shard里面，每个Shard包含两个Solr节点，一个是Leader节点，一个是Replica节点，此外集群中有一个负责维护集群状态信息的Overseer节点，它是一个总控制器。

集群的所有状态信息都放在Zookeeper集群中统一维护。从图中还可以看到，任何一个节点都可以接收索引创建或者更新的请求，然后再将这个请求转发到索引文档所应该属于的那个Shard的Leader节点，Leader节点更新结束完成后，最后将版本号和文档转发给同属于一个Shard的replicas节点。

****五、SolrCloud的工作模式****

首先来看下索引和Solr实体对照图



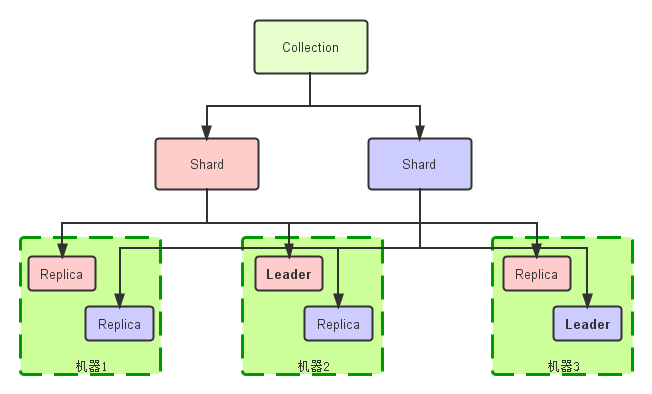
SolrCloud中包含有多个Solr Instance,而每个Solr Instance中包含有多个Solr Core，Solr Core对应着一个可访问的Solr索引资源，每个Solr Core对应着一个Replica或者Leader,这样，当Solr Client通过Collection访问Solr集群的时候，便可通过Shard分片找到对应的Replica即SolrCore，从而就可以访问索引文档了。

IMG_256IMG_256

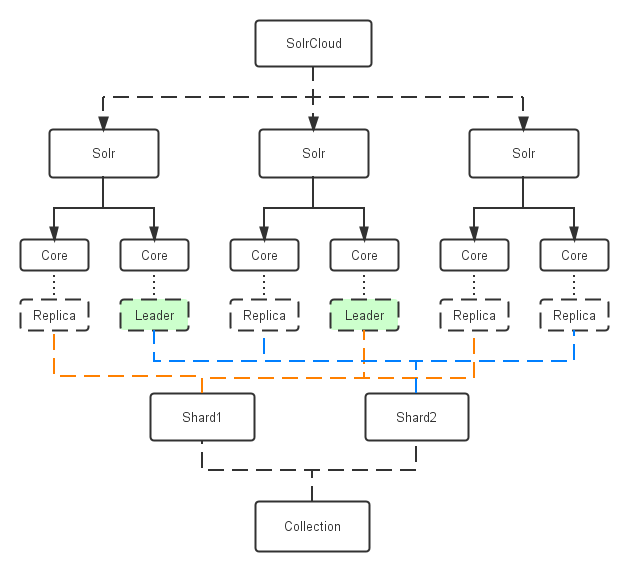
在SolrCloud模式下，同一个集群里所有Core的配置是统一的，Core有leader和replication两种角色，每个Core一定属于一个Shard，Core在Shard中扮演Leader还是replication由Solr内部Zookeeper自动协调。

访问SolrCloud的过程：Solr Client向Zookeeper咨询Collection的地址，Zookeeper返回存活的节点地址供访问，插入数据的时候由SolrCloud内部协调数据分发（内部使用一致性哈希）。

1. ****SolrCloud创建索引和更新索引****



* [索引和Solr实体对照图](http://www.processon.com/view/link/526f1a880cf2ddc32cf56edc)



****<一>、不得不知道的索引存储细节****

当Solr客户端发送add/update请求给CloudSolrServer,CloudSolrServer会连接至Zookeeper获取当前SolrCloud的集群状态，并会在/clusterstate.json 和/live\_nodes中注册watcher，便于监视Zookeeper和SolrCloud，这样做的好处有以下两点：

1、CloudSolrServer获取到SolrCloud的状态后，它可直接将document发往SolrCloud的leader，从而降低网络转发消耗。

2、注册watcher有利于建索引时候的负载均衡，比如如果有个节点leader下线了，那么CloudSolrServer会立马得知，那它就会停止往已下线的leader发送document。

此外，CloudSolrServer 在发送document时候需要知道发往哪个shard？对于建好的SolrCloud集群，每一个shard都会有一个Hash区间，当Document进行update的时候，SolrCloud就会计算这个Document的Hash值，然后根据该值和shard的hash区间来判断这个document应该发往哪个shard，Solr使用documentroute组件来进行document的分发。目前Solr有两个DocRouter类的子类CompositeIdRouter(Solr默认采用的)类和ImplicitDocRouter类，当然我们也可以通过继承DocRouter来定制化我们的document route组件。

举例来说当Solr Shard建立时候，Solr会给每一个shard分配32bit的hash值的区间，比如SolrCloud有两个shard分别为A,B，那么A的hash值区间就为80000000-ffffffff，B的hash值区间为0-7fffffff。默认的CompositeIdRouter hash策略会根据document ID计算出唯一的Hash值，并判断该值在哪个shard的hash区间内。

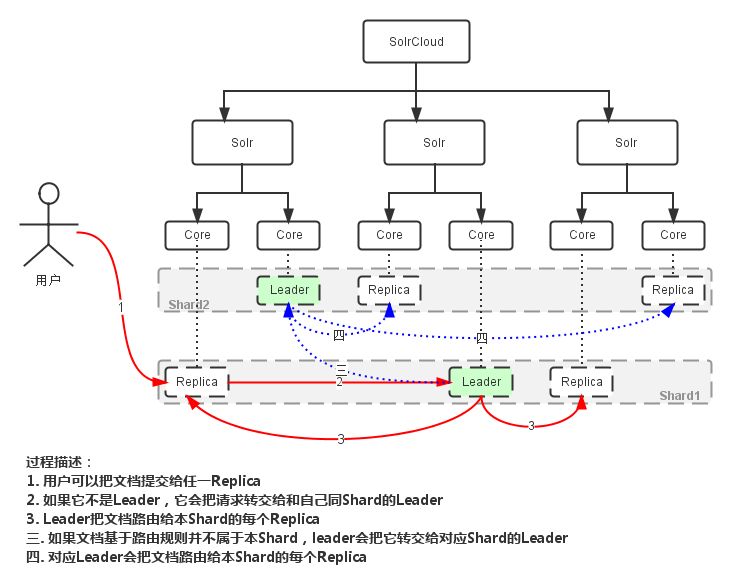
SolrCloud对于Hash值的获取提出了以下两个要求：

1、hash计算速度必须快，因为hash计算是分布式建索引的第一步。

2、 hash值必须能均匀的分布于每一个shard，如果有一个shard的document数量大于另一个shard，那么在查询的时候前一个shard所花的时间就会大于后一个，SolrCloud的查询是先分后汇总的过程，也就是说最后每一个shard查询完毕才算完毕，所以SolrCloud的查询速度是由最慢的shard的查询速度决定的。

基于以上两点，SolrCloud采用了MurmurHash 算法以提高hash计算速度和hash值的均匀分布。

****<二>、Solr创建索引可以分为5个步骤（如下图所示）：****

********

1、用户可以把新建文档提交给任意一个Replica（Solr Core）。

2、如果它不是leader，它会把请求转给和自己同Shard的Leader。

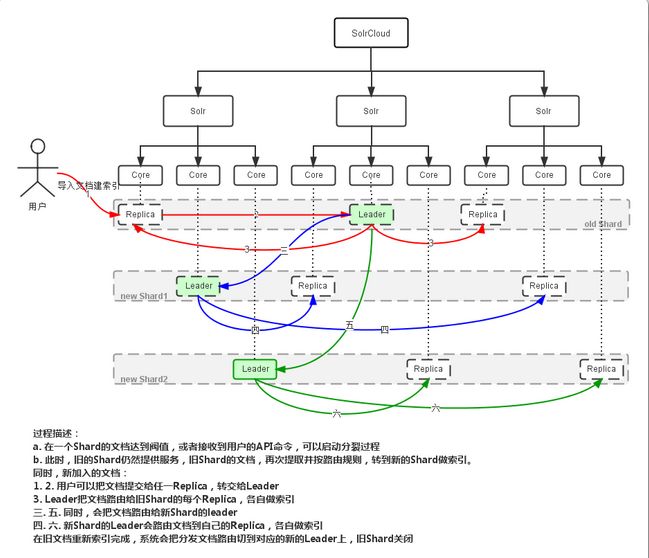
3、Leader把文档路由给本Shard的每个Replica。

III、如果文档基于路由规则(如取hash值)并不属于当前的Shard，leader会把它转交给对应Shard的Leader。

VI、对应Leader会把文档路由给本Shard的每个Replica。

需要注意的是，添加索引时，单个document的路由非常简单，但是SolrCloud支持批量添加索引，也就是说正常情况下可对N个document同时进行路由。这时SolrCloud会根据document路由的去向分开存放document，即对document进行分类，然后进行并发发送至相应的shard，这就需要较高的并发能力。

****<三>、更新索引的关键点：****

********

1、 Leader接受到update请求后，先将update信息存放到本地的update log，同时Leader还会给document分配新的version，对于已存在的document，如果新的版本高就会抛弃旧版本，最后发送至replica。

2、一旦document经过验证以及加入version后，就会并行的被转发至所有上线的replica。SolrCloud并不会关注那些已经下线的replica，因为当他们上线时候会有recovery进程对他们进行恢复。如果转发的replica处于recovering状态，那么这个replica就会把update放入updatetransaction 日志。

3、当leader接受到所有的replica的反馈成功后，它才会反馈客户端成功。只要shard中有一个replica是active的，Solr就会继续接受update请求。这一策略其实是牺牲了一致性换取了写入的有效性。这其中有一个重要参数：leaderVoteWait参数，它表示当只有一个replica时候，这个replica会进入recovering状态并持续一段时间等待leader的重新上线。如果在这段时间内leader没有上线，那么他就会转成leader，其中可能会有一些document丢失。当然可以使用majority quorum来避免这个情况,这跟Zookeeper的leader选举策略一样，比如当多数的replica下线了，那么客户端的write就会失败。

4、索引的commit有两种，一种是softcommit，即在内存中生成segment，document是可见的(可查询到)但是没写入磁盘，断电后数据会丢失。另一种是hardcommit，直接将数据写入磁盘且数据可见。

****<四>、对Solr更新索引和创建索引的几点总结：****

1、leader转发的规则

1）请求来自leader转发：那么就只需要写到本地ulog,不需要转发给leader，也不需要转发给其它replicas。如果replica处于非active状态，就会将update请求接受并写入ulog，但不会写入索引。如果发现重复的更新就会丢弃旧版本的更新。

2）请求不是来自leader，但自己就是leader，那么就需要将请求写到本地，顺便分发给其他的replicas。

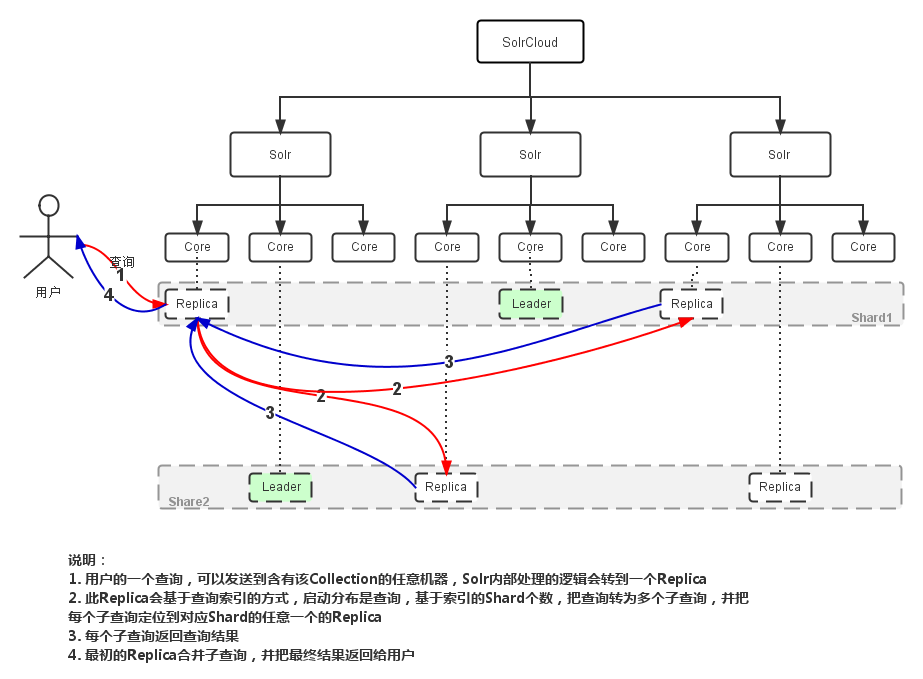
3）请求不是来自leader，但自己又不是leader,也就是该更新请求是最原始的更新请求，那么需要将请求写到本地ulog，顺便转发给leader,再由leader分发。每commit一次，就会重新生成一个ulog更新日志，当服务器挂掉，内存数据丢失的时候，数据就可以从ulog中恢复。

2、建索引的时候最好使用CloudSolrServer，因为CloudSolrServer直接向leader发送update请求，从而避免网络开销。

3、批量添加索引的时候，建议在客户端提前做好document的路由，在SolrCloud内进行文档路由，开销较大。

****七、SolrCloud索引的检索****

* IMG_256[索引和Solr实体对照图](http://www.processon.com/view/link/526f1a880cf2ddc32cf56edc)



在创建好索引的基础上，SolrCloud检索索引相对就比较简单了：

1、用户的一个查询，可以发送到含有该Collection的任意Solr的Server，Solr内部处理的逻辑会转到一个Replica。

2、此Replica会基于查询索引的方式，启动分布式查询，基于索引的Shard的个数，把查询转为多个子查询，并把每个子查询定位到对应Shard的任意一个Replica。

3、每个子查询返回查询结果。

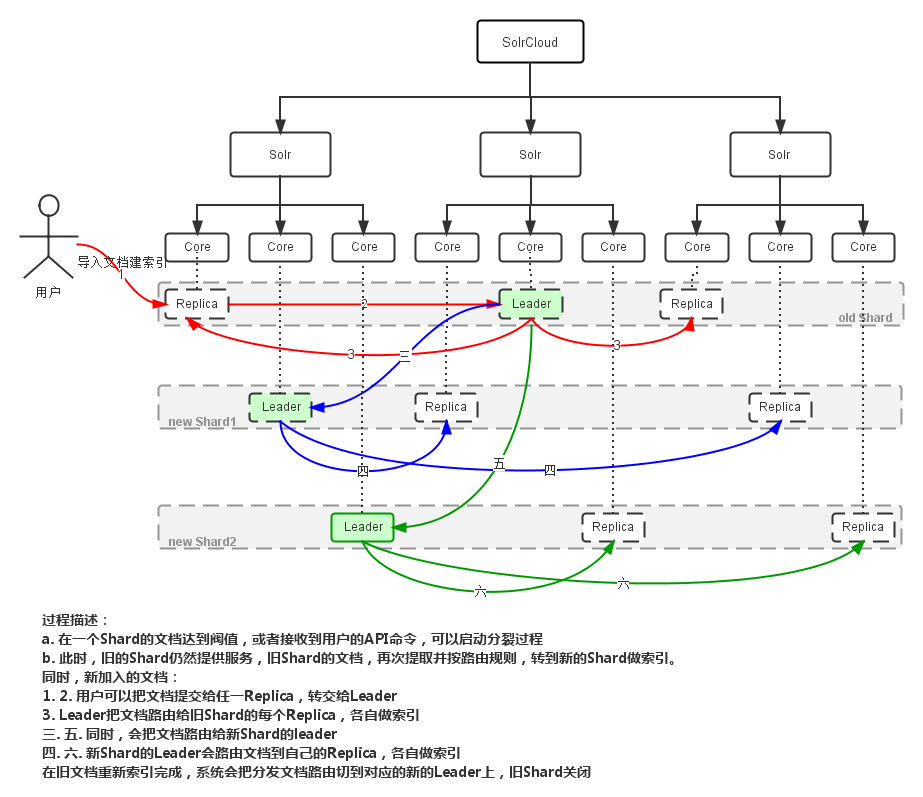
4、最初的Replica合并子查询，并把最终结果返回给用户。

****SolrCloud中提供NRT近实时搜索：****

SolrCloud支持近实时搜索，所谓的近实时搜索即在较短的时间内使得新添加的document可见可查，这主要基于softcommit机制(注意：Lucene是没有softcommit的，只有hardcommit)。上面提到Solr建索引时的数据是在提交时写入磁盘的，这是硬提交，硬提交确保了即便是停电也不会丢失数据；为了提供更实时的检索能力，Solr提供了一种软提交方式。软提交（soft commit）指的是仅把数据提交到内存，index可见，此时没有写入到磁盘索引文件中。在设计中一个通常的做法是：每1-10分钟自动触发硬提交，每秒钟自动触发软提交，当进行softcommit时候，Solr会打开新的Searcher从而使得新的document可见，同时Solr还会进行预热缓存及查询以使得缓存的数据也是可见的，这就必须保证预热缓存以及预热查询的执行时间必须短于commit的频率，否则就会由于打开太多的searcher而造成commit失败。

最后说说在项目中近实时搜索的感受吧，近实时搜索是相对的，对于有客户需求，1分钟就是近实时了，而有些需求3分钟就是近实时了。对于Solr来说，softcommit越频繁实时性更高，而softcommit越频繁则Solr的负荷越大（commit越频繁越会生成小且多的segment，于是Solr merge出现的更频繁）。目前我们项目中的softcommit频率是3分钟，之前设置过1分钟而使得Solr在Index所占资源过多，从而大大影响了查询。所以近实时蛮困扰着我们的，因为客户会不停的要求你更加实时，目前项目中我们采用加入缓存机制来弥补这个实时性。

1. ****SolrShard Splitting的具体过程****



一般情况下，增加Shard和Replica的数量能提升SolrCloud的查询性能和容灾能力，但是我们仍然得根据实际的document的数量，document的大小，以及建索引的并发，查询复杂度，以及索引的增长率来统筹考虑Shard和Replica的数量。Solr依赖Zookeeper实现集群的管理，在Zookeeper中有一个Znode 是/clusterstate.json ，它存储了当前时刻下整个集群的状态。同时在一个集群中有且只会存在一个overseer，如果当前的overseer fail了那么SolrCloud就会选出新的一个overseer，就跟shard leader选取类似。

Shard分割的具体过程（old shard split为newShard1和newShard2）可以描述为：

a、在一个Shard的文档到达阈值，或者接收到用户的API命令，Solr将启动Shard的分裂过程。

b、此时，原有的Shard仍然会提供服务，Solr将会提取原有Shard并按路由规则，转到新的Shard做索引。同时，新加入的文档：

1.2．用户可以把文档提交给任意一个Replica,并转交给Leader。

3.Leader把文档路由给原有Shard的每个Replica，各自做索引。

III.V. 同时，会把文档路由给新的Shard的Leader

IV.VI．新Shard的Leader会路由文档到自己的Replica，各自做索引，在原有文档重新索引完成，系统会把分发文档路由切到对应的新的Leader上，原有Shard关闭。Shard只是一个逻辑概念，所以Shard的Splitting只是将原有Shard的Replica均匀的分不到更多的Shard的更多的Solr节点上去。

****九、Zookeeper:****

****<一>、SolrCloud中使用ZooKeeper主要实现以下三点功能：****

1、集中配置存储以及管理。

2、集群状态改变时进行监控以及通知。

3、shard leader的选举。

****<二>、 Znode与短链接****

Zookeeper的组织结构类似于文件系统，每一层是一个Znode，每一个Znode存储了一些元数据例如创建时间，修改时间以及一些小量的数据。需要主要的是，Zookeeper并不支持存放大数据，它只支持小于1M大小的数据，因为性能原因，Zookeeper将数据存放在内存中。

Zookeeper另一个重要的概念是短链接，当Zookeeper客户端与Zookeeper建立一个短连接后会在Zookeeper新建一个Znode，客户端会一直与Zookeeper进行通信并保证这个Znode一直存在。如果当客户端与Zookeeper的短连接断开，这个Znode就会消失。在SolrCloud中，/live\_nodes下存储了了所有客户端的短连接，表示有哪些Solr组成SolrCloud，具体来说就是当Solr跟Zookeeper保持短连接时，这些Solr主机就组成了SolrCloud，如果其中一个Solr的短连接断掉了，那么Live\_nodes下就少了一个Znode，SolrCloud也就少了一个主机，于是Zookeeper就会告诉其他剩余的Solr有一个Solr挂掉了，那么在今后进行查询以及leader数据分发的时候就不用再经过刚才那个Solr了。Zookeeper是通过watch知道有Solr挂了的,而Zookeeper维护的集群状态数据是存放在solr/zoo\_data目录下的。

****<三>、SolrCloud配置Zookeeper集群的基本过程****

****事例1****、单节点的Zookeeper，包含2个简单的Shard集群：把一个collection的索引数据分布到两个shard上去，并假定两个shard分别存储在两台Solr服务器上。

****集群构建的基本流程：****

先从第一台solr服务器说起：

1、启动一个嵌入式的Zookeeper服务器，作为集群状态信息的管理者。

2、将自己这个节点注册到/node\_states/目录。

3、同时将自己注册到/live\_nodes/目录下。

4、创建/overseer\_elect/leader，为后续Overseer节点的选举做准备，新建一个Overseer。

5、更新/clusterstate.json目录下json格式的集群状态信息

6、本机从Zookeeper中更新集群状态信息，维持与Zookeeper上的集群信息一致。

7、上传本地配置文件到Zookeeper中，供集群中其他solr节点使用。

8、启动本地的Solr服务器，

9、Solr启动完成后，Overseer会得知shard中有第一个节点进来，更新shard状态信息，并将本机所在节点设置为shard1的leader节点，并向整个集群发布最新的集群状态信息。

10、本机从Zookeeper中再次更新集群状态信息，第一台solr服务器启动完毕。

然后来看第二台solr服务器的启动过程：

1、本机连接到集群所在的Zookeeper。

2、将自己这个节点注册到/node\_states/目录下。

3、同时将自己注册到/live\_nodes/目录下。

4、本机从Zookeeper中更新集群状态信息，维持与Zookeeper上的集群信息一致。

5、从集群中保存的配置文件加载Solr所需要的配置信息。

6、启动本地solr服务器。

7、solr启动完成后，将本节点注册为集群中的shard，并将本机设置为shard2的Leader节点。

8、本机从Zookeeper中再次更新集群状态信息，第二台solr服务器启动完毕。

****示例2****、单节点的Zookeeper，包含2个shard的集群，每个shard中有replica节点。

如图所示，集群包含2个shard，每个shard中有两个solr节点，一个是leader，一个是replica节点， 但Zookeeper只有一个。

因为Replica节点，使得这个集群现在具备容错性了，背后的实质是集群的overseer会监测各个shard的leader节点，如果leader节点挂了，则会启动自动的容错机制，会从同一个shard中的其他replica节点集中重新选举出一个leader节点，甚至如果overseer节点自己也挂了，同样会自动在其他节点上启用新的overseer节点，这样就确保了集群的高可用性。

****示例3、****包含2个shard的集群，带shard备份和zookeeper集群机制

示例2中存在的问题是：尽管solr服务器具有容错机制，但集群中只有一个Zookeeper服务器来维护集群的状态信息，单点的存在即是不稳定的根源。如果这个Zookeeper服务器挂了，那么分布式查询还是可以工作的，因为每个solr服务器都会在内存中维护最近一次由Zookeeper维护的集群状态信息，但新的节点无法加入集群，集群的状态变化也不可知了。

因此，为了解决这个问题，需要对Zookeeper服务器也设置一个集群，让其也具备高可用性和容错性。有两种方式可选，一种是提供一个外部独立的Zookeeper集群，另一种是每个solr服务器都启动一个内嵌的Zookeeper服务器，再将这些Zookeeper服务器组成一个集群。

****总结：**** 通过以上的介绍，可看出SolrCloud相比Solr而言，有了很多的新特性，保证了整个Solr应用的High Availability。

1、集中式的配置信息

使用ZK进行集中配置。启动时可以指定把Solr的相关配置文件上传Zookeeper，多机器共用。这些ZK中的配置不会再拿到本地缓存，Solr直接读取ZK中的配置信息。另外配置文件的变动，所有机器都可以感知到， Solr的一些任务也是通过ZK作为媒介发布的，目的是为了容错，这使得Solr接收到任务，但在执行任务时崩溃的机器，在重启后，或者集群选出候选者时，可以再次执行这个未完成的任务。

2、SolrCloud对索引分片，并对每个分片创建多个Replication。每个Replication都可以对外提供服务。一个Replication挂掉不会影响索引服务，更强大的是，SolrCloud还能自动的在其它机器上帮你把失败机器上的索引Replication重建并投入使用。

3、近实时搜索：立即推送式的replication（也支持慢推送），可以在秒内检索到新加入索引。

4、查询时自动负载均衡：SolrCloud索引的多个Replication可以分布在多台机器上，均衡查询压力，如果查询压力大，可以通过扩展机器，增加Replication来减缓。

5、自动分发的索引和索引分片：发送文档到任何节点，SolrCloud都会转发到正确节点。

6、事务日志：事务日志确保更新无丢失，即使文档没有索引到磁盘。

除此之外，SolrCloud中还提供了其它一些特色功能：

1、可将索引存储在HDFS上

2、通过MR批量创建索引

3、强大的RESTful API

优秀的管理界面：主要信息一目了然，可以清晰的以图形化方式看到SolrCloud的部署分布，当然还有不可或缺的Debug功能。