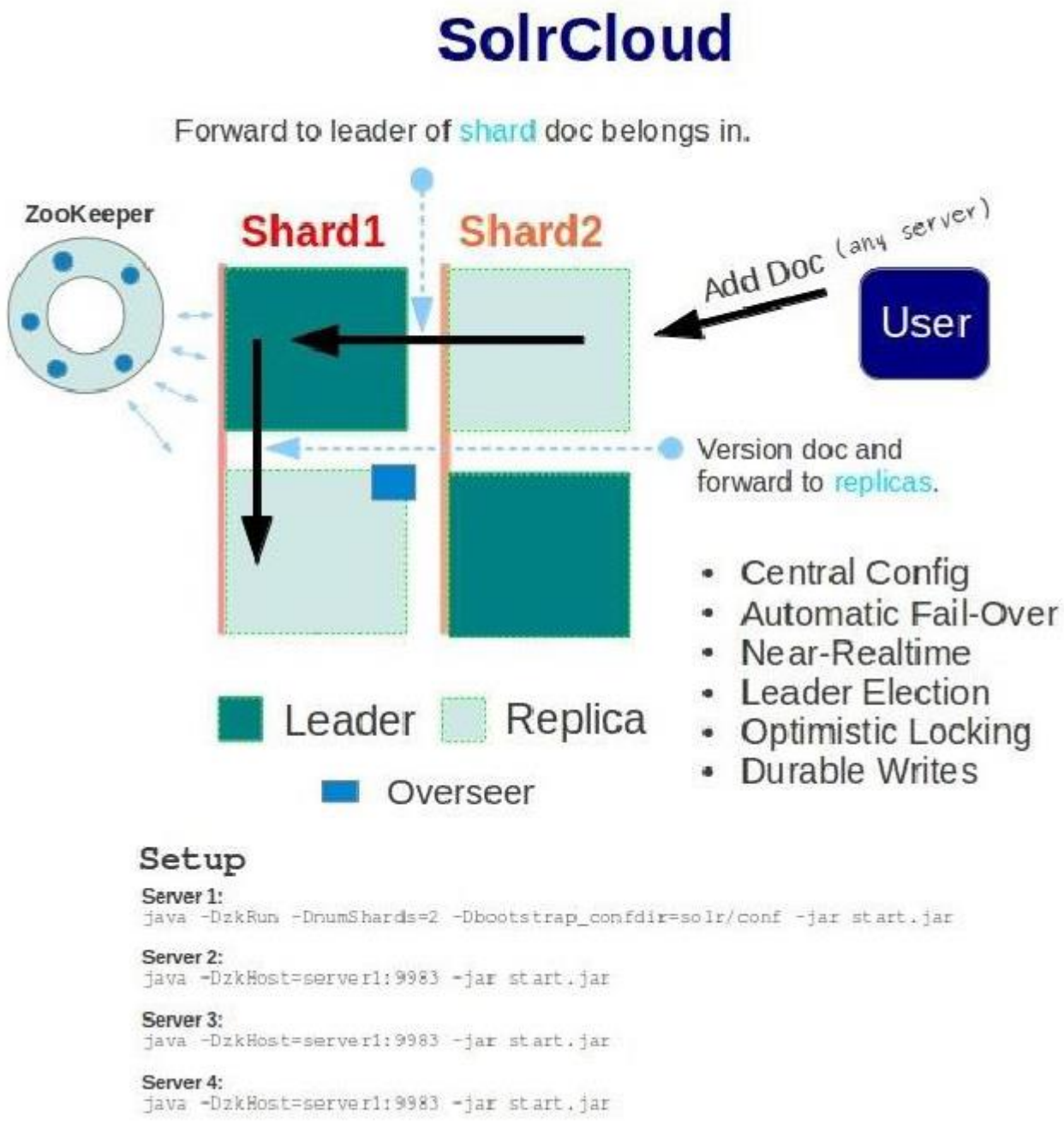
SolrCloud 使用教程、原理介绍

发布于 2013 年 8 月 24 日，属于 搜索 分类，7,446 浏览数

SolrCloud 是基于 Solr 和 Zookeeper 的分布式搜索方案，是正在开发中的 Solr4.0 的核心组件之一，它的主要思想是使用 Zookeeper 作为集群的配置信息中心。

它有几个特色功能：①集中式的配置信息 ②自动容错 ③近实时搜索 ④查询时自动负载均衡。



Setup

Server 1:

```
java -DzkRun -DnumShards=2 -Dbootstrap_confdir=solr/conf -jar start.jar
```

Server 2:

```
java -DzkHost=server1:9983 -jar start.jar
```

Server 3:

```
java -DzkHost=server1:9983 -jar start.jar
```

Server 4:

```
java -DzkHost=server1:9983 -jar start.jar
```

下面看看 wiki 的文档：

1、SolrCloud

SolrCloud 是指 Solr 中一套新的潜在的分发能力。这种能力能够通过参数让你建立起一个高可用、
容错的 Solr 服务集群。当你需要大规模，容错，分布式索引和检索能力时使用 SolrCloud(solr 云)。

看看下面“启动”部分内容，快速的学会怎样启动一个集群。后面有 3 个快速简单的例子，它们展现怎样启动一个逐步越来越复杂的集群。检出例子之后，需要翻阅后面的部分了解更加细节的信息。

2、关于 SolrCores 和 Collections 的一点儿东西

对于单独运行的 Solr 实例，它有个东西叫 SolrCore(Solr.xml 中配置的),它是本质上独立的索引块。如果你打算多个索引块，你就创建多个 SolrCores。当同时部署 SolrCloud 的时，独立的索引块可以跨越多个 Solr 实例。这意味着一个单独的索引块能由不同服务器设备上多个 SolrCore 的索引块组成。我们把组成一个逻辑索引块的所有 SolrCores 叫做一个独立索引块儿(collection)。一个独立索引块是本质上一个独立的跨越多个 SolrCore 索引块的索引块，同时索引块尽可能随着多余的设备进行缩放。如果你想把你的两个 SolrCore Solr 建立成 SolrCloud,你将有 2 个独立索引块，每个有多个独立里的 SolrCores 组成。

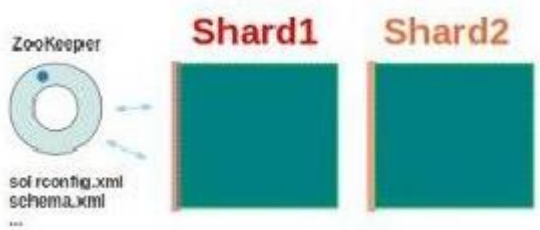
3、启动

下载 Solr4-Beta 或更高版本。

如果你还没了解，通过简单的 Solr 指南让自己熟悉 Solr。**注意**：在通过指南了解云特点前，重设所有的配置和移除指南的文档.复制带有预先存在的 Solr 索引的例子目录将导致文档计数关闭 Solr 内嵌使用了 Zookeeper 作为集群配置和协调运作的仓储。协调考虑作为一个包含所有 Solr 服务信息的分布式文件系统。

如果你想用一个其他的而不是 8983 作为 Solr 端口，去看下面’ Parameter Reference’部分下的关于 solr.xml 注解

例 A:简单的 2 个 shard 集群



这个例子简单的创建了一个代表一个独立索引块的两个不同的 shards 的两个 solr 服务组成

的集群。从我们将需要两个 solr 服务器，简单的复制例子目录副本作为第二个服务器时，要确保没有任何索引存在。

```
rm -r example/solr/collection1/data/*
cp -r example example2
```

下面的命令启动了一个 solr 服务同时引导了一个新的 solr 集群

```
cd example
java -Dbootstrap_confdir=./solr/collection1/conf
-Dcollection.configName=myconf -DzkRun -DnumShards=2 -jar start.jar
```

•-DzkRun 该参数将促使一个内嵌的 zookeeper 服务作为 Solr 服务的部分运行起来。
•-Dbootstrap_confdir=./solr/collection1/conf 当我们还没有 zookeeper 配置时，这个参数导致本地路径./solr/conf 作为“myconf”配置被加载.“myconf”来自下面的“collection.configName”参数定义。
•-Dcollection.configName=myconf 设置用于新集合的配置名。省略这个参数将导致配置名字为默认的“configuration1”
•-DnumShards=2 我们打算将索引分割到逻辑分区的个数。

浏览 http://localhost:8983/solr/#/~cloud 地址，看看集群的状态。你从 zookeeper 浏览程序能看到 Solr 配置文件被加载成”myconf”，同时一个新的叫做”collection1”的文档集合被创建.collection1 下是碎片列表，这些碎片组成了那完整的集合。

现在我们想要启动我们的第二个服务器;因为我们没明确设置碎片 id，它将自动被设置成 shard2，启动第二个服务器，指向集群

```
cd example2
java -Djetty.port=7574 -DzkHost=localhost:9983 -jar start.jar
```

•-Djetty.port=7574 只是一种用于让 Jetty servlet 容器使用不同端口的方法
•-DzkHost=localhost:9983 指向全体包含集群状态的 Zookeeper 服务.这个例子中，我们运行的是一个内嵌在第一个 solr 服务器中的独立的 Zookeeper 服务.默认情况下，一个内嵌的 Zookeeper 服务运行在 9983 端口上。

如果你刷新浏览器，你现在在 collection1 下应该看到两个 shard1 和 shard2。

接下来，索引一些文档。如果你想要快速完成一些，像 java 你能够使用 CloudSolrServer solrj 的实现和简单的用 Zookeeper 地址初始化它就可以了.或者也可以随便选择某个 solr 实例添加文档，这些文档自动的被导向属于他们的地方。

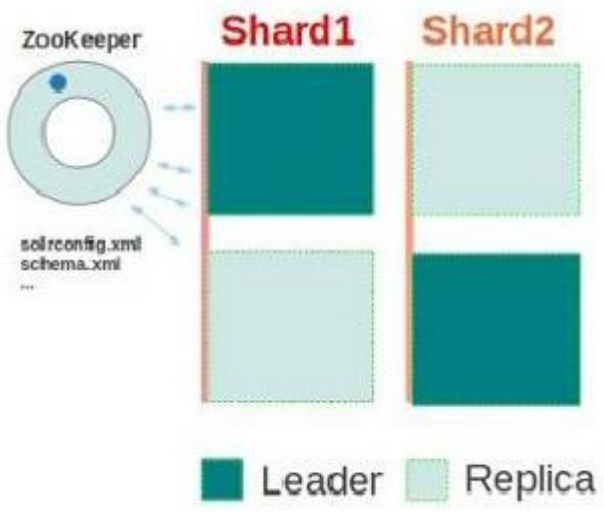
```
cd exampledocs
java -Durl=http://localhost:8983/solr/collection1/update -jar post.jar
ipod_video.xml
java -Durl=http://localhost:8983/solr/collection1/update -jar post.jar
monitor.xml
java -Durl=http://localhost:8983/solr/collection1/update -jar post.jar
mem.xml
```

现在，发起一个获得覆盖整个集合的分布式搜索服务搜索到的任何一个服务的结果的请求：

```
http://localhost:8983/solr/collection1/select?q=*:*
```

如果在任一个点你希望重新加载或尝试不同的配置,你能通过关闭服务器后简单的删除 solr/zoo_data 目录方式删除所有 zookeeper 的云状态。

例 B：简单的支持副本的两个碎片集群



有先前的例子，这个例子通过再创建 shard1 和 shard2 副本很容易创建起来。额外的 shard 副本用于高可用性和容错性，或用于提高集群的检索能能力。

首先，通过前一个例子， 我们已经有两个碎片和一些文档索引在里面了。然后简单的复制那两个服务:

```
cp -r example exampleB
cp -r example2 example2B
```

然后在不同的端口下启动这两个新的服务，每个都在各自的视窗下执行

```
cd exampleB
java -Djetty.port=8900 -DzkHost=localhost:9983 -jar start.jar
cd example2B
java -Djetty.port=7500 -DzkHost=localhost:9983 -jar start.jar
```

刷新 zookeeper 浏览页面，核实 4 个 solr 节点已经启动，并且每个 shard 有两个副本，因为我们已经告诉 Solr 我们想要 2 个逻辑碎片，启动的实例 3 和 4 自动的被负值为两个 shards 的副
本。

现在，向任何一个服务器发送请求来查询这个集群

```
http://localhost:7500/solr/collection1/select?q=*:*
```

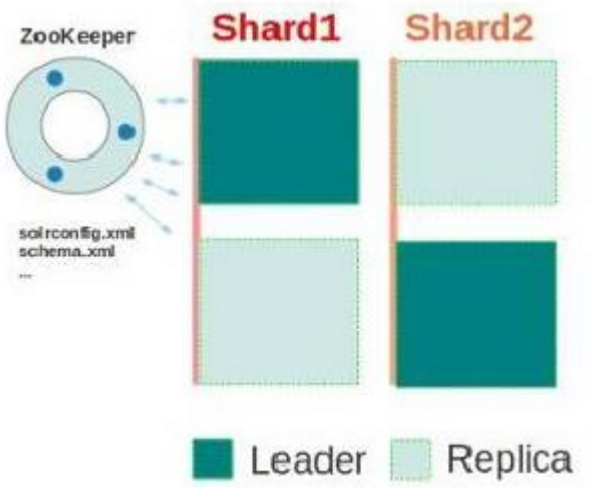
多次发送这个请求，来研究 solr 服务的日志。你应该能够发现 Solr 实现跨副本的负载均衡请求，用不同的服务满足每一个请求。 浏览器发送请求的服务器里有一个顶层请求的日志描述，也会有被合并起来生成完成响应的子请求的日志描述。

为了证明故障切换的可用性，在任何一个服务器（除了运行 Zookeeper 的服务器）运行窗口下按下 CTRL-C。一旦那个服务实例停止运行，就向剩余运行的服务器的任何一个发送另一个查询请求，你应该仍然看到完整的结果。

SolrCloud 能够持续提供完整查询服务直到最后一个维护每一个碎片的服务器当机.你能通过明确地关闭各个实例和查询结果来证明这种高可用性。如果你关掉维护某一个碎片的所有服务器，请求其他服务器时将会返回 503 错误结果。要想从依然运行的碎片服务中返回可用的正确文档，需要添加 shards.tolerant=true 查询参数。

SolrCloud 用多主服务+监控者服务实现。这意味着某写节点或副本扮演一种特殊角色。你不需要担忧你关掉的是主服务或是集群监控，如果你关掉它们中的一个，故障切换功能将自动选出新的主服务或新的监控服务，新的服务将无缝接管各自的工作。任何 Solr 实例都能被升级为这些角色中的某个角色。

例 C：支持副本和完整 zookeeper 服务的两个碎片集群



例 B 的问题是可以 有足够多的服务器保证任何一个损坏的服务器的信息幸存下来，但确只有一个包含集群状态的 zookeeper 服务器。如果那台 zookeeper 服务器损坏，分布式查询将仍然工作在上次 zookeeper 保存的集群状态下，可能没有机会改变集群的状态了。

在 a zookeeper ensemble(zookeeper 集群)中运行多个 zookeeper 服务器保证 zookeeper 服务的高可用性。每个 zookeeper 服务器需要知道集群中其他服务器，且这些服务器中的一个主服务器需要提供服务。例如： 3 台组成的 zookeeper 集群允许任一个故障，而后通过剩余 2 个协商一个主服务继续提供服务。5 台的需要允许一次 2 个服务器故障。

对于用于生产服务时，推荐运行一个额外的 zookeeper 集群而不是运行内嵌在 Solr 里面的 zookeeper 服务。在这里你能阅读更多关于建立 zookeeper 集群的知识。对于这个例子，为了简单点我们将用内嵌的 zookeeper 服务。

首先停止 4 个服务器并清理 zookeeper 数据

```
1 rm -r example*/solr/zoo_data
```

我们将分别在 8983，7574，8900，7500 端口运行服务。默认的方式是在主机端口+1000 运行内嵌的 zookeeper 服务，所以如果我们用前 3 个服务器运行内嵌 zookeeper，地址分别是 localhost:9983，localhost:8574，localhost:9900。

为了方便，我们上传第一个服务器 solr 配置到集群。你将注意到第一个服务器阻塞了，直到你启动了第二个服务器。这是因为 zookeeper 需要达到足够的服务器数目才允许操作。

```
1 cd example
2 java -Dbootstrap_confdir=./solr/collection1/conf -
```

```
Dcollection.configName=myconf -DzkRun -
DzkHost=localhost:9983,localhost:8574,localhost:9900 -DnumShards=2 -jar
start.jar
cd example2
java -Djetty.port=7574 -DzkRun -
DzkHost=localhost:9983,localhost:8574,localhost:9900 -jar start.jar
cd exampleB
java -Djetty.port=8900 -DzkRun -
DzkHost=localhost:9983,localhost:8574,localhost:9900 -jar start.jar
cd example2B
java -Djetty.port=7500 -
DzkHost=localhost:9983,localhost:8574,localhost:9900 -jar start.jar
```

现在我们已经运行起由 3 个内嵌 zookeeper 服务组成的集群，尽管某个服务丢失依然保持工作。通过 CTRL+C 杀掉例 B 的服务器然后刷新浏览器看 zookeeper 状态可以断定 zookeeper 服务仍然工作。**注意**：当运行在多个 host 上时，在每个需要设置对应的 hostname，-DzkRun=hostname:port，-DzkHosts 使用准确的 host 名字和端口，默认的 localhost 将不能工作。

4、ZooKeeper

为了容错和高可用多个 zookeeper 服务器一起运行，这种模式叫做 an ensemble。对于生产环境，推荐额外运行 zookeeper ensemble 来代替 solr 内嵌的 zookeeper 服务。关于下载和运行 ensemble 的更多信息访问 Apache ZooKeeper 。 更特殊的是，试着启动和管理 zookeeper。它的运行实际上非常简单。你能容银让 Solr 运行 Zookeeper，但是记住一个 zookeeper 集群不容易进行动态修改。除非将来支持动态修改，否则最好在回滚重启时进行修改。与 Solr 分开处理通常是最可取的。

当 Solr 运行内嵌 zookeeper 服务时，默认使用 solr 端口+1000 作为客户端口，另外，solr 端口+1 作为 zookeeper 服务端口，solr 端口+2 作为主服务选举端口。所以第一个例子中，Solr 运行在 8983 端口，内嵌 zookeeper 使用 9983 作为客户端端口，9984 和 9985 作为服务端口。

鉴于确保 Zookeeper 搭建起来是非常快速的，要记住几点：Solr 不强烈要求使用 Zookeeper，很多情况优化可能不是必须的。另外，当添加更多的 Zookeeper 节点对读数据性能会有帮助，但也轻微的降低了写的性能。再有，当集群处于稳定运行时，solr 与 Zookeeper 没有太多的交互。如果你需要优化 zookeeper，这里有几个有益的要点：

1.ZooKeeper 在专用的机器上运行效果最好。ZooKeeper 提供的是一种及时服务，专用设备有助于确保及时服务的响应。当然，专用机器并不是必须的。
2.ZooKeeper 在事务日志和快照在不同的硬盘驱动器上工作效果最好
3.如果配置支持 solr 的 Zookeeper，各自使用独立的硬盘驱动器有助于性能。

5、通过集合 api 管理集合

你可以通过集合 API 管理集合。在这个 API 壳下，一般使用 CoreAdmin API 异步管理每台服务器上的 SolrCores，对于如果你自己处理使独立的 CoreAdmin API 通过替换 action 参数就能调用每个服务器来说，这无疑是一个必不可少好东西。

①创建接口

```
1 http://localhost:8983/solr/admin/collections?action=CREATE&amp;name=mycollection&amp;numSh
```

相关参数：

name:将被创建的集合的名字
numShards:集合创建时需要创建逻辑碎片的个数
replicationFactor:每个文档副本数。replicationFactor(复制因子)为 3 意思是每个逻辑碎片将有 3 份副本。注：Solr4.0 中，replicationFactor 是 additional * 副本数 ，而不是副本总数
maxShardsPerNode:一个创建操作将展开创建 numShards*replicationFactor 碎片副本遍布在你的 Solr 节点上，公平分布，同一个碎片的两个副本不会在同一个 Solr 节点上。如果创建操作完成时 Solr 损坏，该操作不会创建出新集合的任何部分。该参数用来防止在同一个 Solr 节点创建太多副本，默认参数 1.如果它的值与整体集合中 numShards*replicationFactor 副本数分布到正常活跃的 Solr 节点的数不符，将不能创建任何东西
createNodeSet:如果不提供该参数，创建操作将创建碎片副本展开分布到所有活跃的 Solr 节点上。提供该参数改变用于创建碎片副本的节点集合。参数值的格式是："<node-name1>,<node-name2>,...,<node-nameN>"，
例如：
createNodeSet=localhost:8983_solr,localhost:8984_solr,localhost:8985_solr
collection.configName:用于新集合的配置文件的名称。如果不提供该参数将使用集合名称作为配置文件的名称。

Solr4.2
相关参数：
name:将被创建的集合别名的名字
collections:逗号分隔的一个或多个集合别名的列表

②删除接口

```
1 http://localhost:8983/solr/admin/collections?action=DELETE&name=mycollection
```

相关参数：

```
1 <strong>name</strong>:将被删除的集合的名字
```

③重新加载接口

```
1 http://localhost:8983/solr/admin/collections?action=RELOAD&name=mycollection
```

相关参数：
name:将被重载的集合的名字

④分割碎片接口

```
1 http://localhost:8983/solr/admin/collections?action=SPLITSHARD&collection=&lt;collection_n
```

Solr4.3
相关参数：
collection:集合的名字
shard:将被分割的碎片 ID

这个命令不能用于使用自定义哈希的集群，因为这样的集群没有一个明确的哈希范围。 它只用于具有 plain 或 compositeid 路由的集群。

该命令将分割给定的碎片索引对应的那个碎片成两个新碎片。通过将碎片范围划分成两个相等的分区和根据新碎片范围分割出它在父碎片(被分的碎片)中的文档。新碎片将被命名为 appending_0 和_1。例如：shard=shard1 被分割，新的碎片将被命名为 shard1_0 和 shard1_1。一旦新碎片被创建，它们就被激活同时父碎片(被分的碎片)被暂停因此将没有新的请求到父碎片(被分的碎片)。

该特征达到了无缝分割和无故障时间的要求。原来的碎片数据不会被删除。使用新 API 命令重载碎片用户自己决定。

该特性发布始于 Solr4.3，由于 4.3 发布版本发现了一些 bugs,所以要使用该特性推荐等待 4.3.1。

6、集合别名
别名允许你创建独立的指向一个或多个真是集合的虚拟集合，你能够在运行时修改别名。

①创建别名

```
http://localhost:8983/solr/admin/collections?action=CREATEALIAS&name=alias&collections=col
```

创建或修改别名。用于发送修改的别名应该只映射一个独立的集合。读的别名能映射一个或多个集合。

②移除存在的别名

```
http://localhost:8983/solr/admin/collections?action=DELETEALIAS&name=alias
```

移除存在的别名

7、经 CoreAdmin 创建 cores

经过 CoreAdmin 可以创建新的 Solrcores 和使新的 Solrcores 与一个集合关联，创建时附加云相关参数

•collection 这个 core 所属的集合的名字。默认 core 的名字
•shard 这个 core 代表的碎片的 id(可选-通常自动设置一个碎片 id)
•numShards 集合碎片个数。该参数只在集合第一个 core 创建时有用
•collection.<param>=<value> 集合创建时设置属性值，用 collection.configName=<configname> 指明新集合的配置文件

```
curl
'http://localhost:8983/solr/admin/cores?action=CREATE&name=mycore&collection=collection1&s
```

8、分布式请求

查询一个集合的所有碎片(注意:那个集合已经隐含在 URL 中了)

```
http://localhost:8983/solr/collection1/select?
```

查询一个兼容集合的所有碎片，明确详述如下:

```
http://localhost:8983/solr/collection1/select?collection=collection1_recent
```

查询多个兼容集合的所有碎片

```
http://localhost:8983/solr/collection1/select?collection=collection1_NY,collection1_NJ,col
```

查询集合的某些碎片。例子中，用户已经按日期对索引进行分割了，每月创建一个新的碎片

```
http://localhost:8983/solr/collection1/select?shards=shard_200812,shard_200912,shard_20100
```

明确指明以想要查询碎片的地址

```
http://localhost:8983/solr/collection1/select?shards=localhost:8983/solr,localhost:7574/so
```

明确指明以想要查询碎片的地址，同时为负载均衡和故障切换给出可选的地址

```
http://localhost:8983/solr/collection1/select?shards=localhost:8983/solr|localhost:8900/so
```

9、必须的配置

所有必须的配置已经安装在 Solr 的例子中。下面是如果你正在迁移旧的配置文件你需要添加什么，或不应该删除什么

10、schema.xml

字段_version_是必须定义的

```
<field name="_version_" type="long" indexed="true" stored="true"
multiValued="false"/>
```

11、solrconfig.xml

必要有一个更新日志的定义，这个应该被定义在 updateHandler 标签中

```
<!-- Enables a transaction log, currently used for real-time get."dir" -
the target directory for transaction logs, defaults to thesolr data
directory.  -->
<updateLog>
    <str name="dir">${solr.data.dir:}</str>
</updateLog>
```

必须有一个复制处理调用的定义

```
  <requestHandler name="/replication" class="solr.ReplicationHandler"
1 startup="lazy" />
```

必须有一个获取实际时间处理的调用的定义

```
1 <requestHandler name="/get" class="solr.RealTimeGetHandler">
2 <lst name="defaults">
3     <str name="omitHeader">true</str>
4 </lst>
5 </requestHandler>
```

必须有管理句柄的定义

```
1 <requestHandler name="/admin/" class="solr.admin.AdminHandlers"/>
```

DistributedUpdateProcessor 是默认更新链的部分也会被自动注入自定义更新链。

```
1 <updateRequestProcessorChain name="sample">
2   <processor class="solr.LogUpdateProcessorFactory" />
3   <processor class="solr.DistributedUpdateProcessorFactory"/>
4   <processor class="my.package.UpdateFactory"/>
5   <processor class="solr.RunUpdateProcessorFactory" />
6 </updateRequestProcessorChain>
```

如果不想让它自动注入到你的链中(说明你想使用 SolrCloud 功能，但想自己分发修改)那么在你的链
中指明下面的更新处理工厂: NoOpDistributingUpdateProcessorFactory

12、solr.xml

必须保留 admin path 作为默认 core

```
1 <cores adminPath="/admin/cores" />
```

13、可变集群

当你在一个集合中启动 SolrCore 时通过 numShards 参数可以控制集群的规模。 这个参数一般自动赋给关联每个实例的碎片。启动 numShards 个实例后所有的 SolrCores 都被均匀的加入每个碎片作为副本。

向集合添加 SolrCores 保证其启动是很简单的，你随时都可以进行这样的操作。新的 SolrCore 在激活前会把它的数据同步到碎片中的副本上。

如果你要在几台机器上启动集群，不仅添加副本需要花时间扩展成集群也需要时间，那么，你可以选择通过每部机器负责几个碎片的方式启动(使用多 SolrCores)，然后通过对碎片启动副本的形式迁移碎片到新的设备上，最终移除原来设备上的碎片。

Solr4.3 具备新的 API 接口， SPLITSHARD，分割已存在的碎片成两份新的碎片，相关介绍前面讲过。

14、近实时搜索

如果你希望启用近实时搜索，你可能需要修改放在 ZooKeeper 中的 solrconfig.xml 文件，添加启动软件自动提交配置，或者通过向集群发送软提交。详见看近实时搜索。

15、参数参考介绍

A.集群参数

numShards	默认值 1	文档被散列存储到所有碎片的数量。每个碎片有一个主碎片(领导者),每个主碎片有 N 个副本

B.Solr 云实例参数

参数在 solr.xml 中设置,实例的运行与系统环境变量也有关。重要事项：hostPort 是每个 Solr 实例运行的端口，zookeeper 通过该端口通知集群的状态(剩余等)，默认值是 8983。solr.xml 例子用的是 jetty.port 环境变量,所以如果你不想用 8983，那么你启动 Solr 前设置 jetty.port 变量，或修改 solr.xml 文件。

host	默认值 找到的第一个本地地址	如果自动找到的地址是错的，你可以通过这个参数修改
hostPort	默认值	这个参数是 Solr 使用的，默认使用的是 jettry 的配置文件

	jettry 的 系统配置	
host	默认值 solr	Solr 部署成 web 应用时的上下文路径。（注：在 Solr4.0 中，强制要求 hostContext 不能包含”/”或”_”字符。Solr4.1 开始，没有这个限制了，并且建议以”/”开头。运行 jetty 例子时，hostContext 既是 jetty 容器的 servlet 上下文地址也是 SolrCloud 上下文地址，例：-DhostContext=/solr）

C.云实例中 Zookeeper 参数

zkRun	默认值 localhost:solrPort+1000	配置该参数让 Solr 运行起内嵌的 Zookeeper 服务。参数设置了 Zookeeper 所在的服务器地址。这个值会出现在 zkHost 参数中，我们能够知道都有哪些 zk 服务器。简单模式可以使用默认值；注意：这个值必须是 zkHost 中的一个；另外，多服务器版的不能使用默认的 localhost。
zkHost	默认值 没有	ZooKeeper host 地址——你的 ZooKeeper 集群一般节点地址使用逗号分割
zkClientTimeout	默认值 15000	session 没有过期前提下，不和 ZooKeeper

	交互的最长间隔

zkClientTimeout 参数在 solr.xml 设置，也可以通过系统参数设置。

D.云核参数

shard	shard 的 ID,默认基于 numShards 自动分配	允许你指定 SolrCores 里的 shard 的 Id

碎片 id 可以在每个 core 元素的属性中配置

16、把配置文件给 ZooKeeper
A.配置启动引导参数

第一次启动 Solr，有两种使用系统参数加载初始化 Zookeeper 配置文件的方法。记住，只用于第一次启动或覆盖配置文件时。每次启动都要带着系统参数，系统参数会覆盖配置文件中名字相同的值。

1.看 solr.xml 文件，找到每个 SolrCore 加载的 conf 目录。 'config set' (配置集)的名字就是 SolrCore 的集合的名字，集合使用与它名字相同的配置集合。

	无默	如果启动时携带参数 -Dbootstrap_conf=true，每个 SolrCore 将把
bootstrap_conf	认值	自己的配置文件加载关联到它对应的集合上。

2.通过给的配置文件名字从指定目录下加载配置集。尽管没有指定集合到配置集的关联，如果只有一个配置集，集合将自动链接到这个配置集。

		如果你通过 -bootstrap_confdir=<目录≤ 启动， 指定的配置目录文件将会被提交到 ZooKeeper, 设置配置名可以用下面的参数：
bootstrap_confdir	无默认值	collection.configName
collection.configName	默认值	通过 bootstrap_confdir 设置配置集的名称

configuration1

B.命令行工具

CLI 工具也可以配置 Zookeeper。 它与上面两种方法一样。 它提供一些命令，能够配置集到集合关联，创建 Zookeeper 路径或移除，还能从 zookeeper 下载配置文件到本地。

```
usage: ZkCLI
-c,--collection <arg> for linkconfig: name of the collection
-cmd <arg> cmd to run: bootstrap, upconfig, downconfig,linkconfig,
makepath, clear
-d,--confdir <arg> for upconfig: a directory of configuration files
-h,--help bring up this help page
-n,--confname <arg> for upconfig, linkconfig: name of the config set
-r,--runzk <arg> run zk internally by passing the solr run port -only for
clusters on one machine (tests, dev)
-s,--solrhome <arg> for bootstrap, runzk: solrhome location
-z,--zkhost <arg> ZooKeeper host address
```

C.例子

```
# try uploading a conf dir
java -classpath example/solr-webapp/WEB-INF/lib/*
org.apache.solr.cloud.ZkCLI -cmd upconfig -zkhost 127.0.0.1:9983 -confdir
example/solr/collection1/conf -confname conf1
# try linking a collection to a conf set
java -classpath example/solr-webapp/WEB-INF/lib/*
org.apache.solr.cloud.ZkCLI -cmd linkconfig -zkhost 127.0.0.1:9983 -
collection collection1 -confname conf1
# try bootstrapping all the conf dirs in solr.xml
java -classpath example/solr-webapp/WEB-INF/lib/*
org.apache.solr.cloud.ZkCLI  -cmd bootstrap  -zkhost 127.0.0.1:9983  -
solrhome example/solr
```

D.脚本

如果你不想使用 jettry 容器，example/cloud-scripts 目录下有系统脚本文件，它帮你处理了 classpath 和 class name，命令变成为：

```
sh zkcli.sh  -cmd linkconfig  -zkhost 127.0.0.1:9983  -collection
collection1 -confname conf1
```

17、ZooKeeper chroot

如果其他应用已经在用 Zookeeper，你想保持原应用的管理方式，又或者有多个分开的 solrcloud 集群共用一个 zookeeper，你可以用 zookeeper 的‘chroot’选项，来自 Zookeeper Session。

```
An optional "chroot" suffix may also be appended to the
connection string. This will run the client commands while
interpreting all paths relative to this root (similar to the
unix chroot command). If used the example would look like:
"127.0.0.1:4545/app/a" or
"127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002/app/a" where the
client would be rooted at "/app/a" and all paths would be
relative to this root - ie getting/setting/etc... "/foo/bar"
would result in operations being run on "/app/a/foo/bar" (from
the server perspective).
```

要使用这个特点，zkHost 参数用上”chroot”后缀，而后简单启动 solr

```
1  java -DzkHost=localhost:9983/foo/bar -jar start.jar
```

或者

```
1  java -DzkHost=zoo1:9983,zoo2:9983,zoo3:9983/foo/bar -jar start.jar
```

提示：Solr4.0 启动时你需要预先创建 ZooKeeper 路径，Solr4.1 以后再使用 eitherbootstrap_conf 或 boostrap_confdir 的路径是自动初始化的了。

18、已知的局限性

很少 Solr 搜索框架不支持分布式搜索的。有些情况下，一个框架无法支持分布式，不过，那只是时间和努力的问题。不支持标准分布式搜索的搜索框架，它的 SolrCloud 同样不支持。通过 distrib=false 控制是否使用分布式。

组群特点只能是在同一个存储碎片上的组群。必须自定义碎片特征才能使用组群特点。

如果要更新支持 SolrCloud 的 Solr4.0 到 4.1，注意 name_node 参数的定义方式已经改变了。这导致一种情况发生，由于 name_node 用 ip 地址代替了服务器的名字，使 SolrCloud 无法感知到这个 name_node。你可以通过 solr.xml 配置 ip 到服务器名的关联解决问题。

19、术语

Collection:	A single search index.
Shard:	A logical section of a single collection (also called Slice). Sometimes people will talk about “Shard” in a physical sense (a manifestation of a logical shard)
Replica:	A physical manifestation of a logical Shard, implemented as a single Lucene index on a SolrCore
Leader:	One Replica of every Shard will be designated as a Leader to coordinate indexing for that Shard
SolrCore:	Encapsulates a single physical index. One or more make up logical shards (or slices) which make up a collection.
Node:	A single instance of Solr. A single Solr instance can have multiple SolrCores that can be part of any number of collections.
Cluster:	All of the nodes you are using to host SolrCores.

英文原文地址

您可能喜欢：

Solr 索引数据同步(ReplicationHandler)功能搭建问题

Solr 开发过程中遇到的一下资料、问题整理(资料)

Solr 实现 并集式、多值、复杂 过滤查询（filterQuery）

Solr 对比 Lucene 的优势

Solr4.0(SolrCloud) & ElasticSearch(ES) 比较(二)