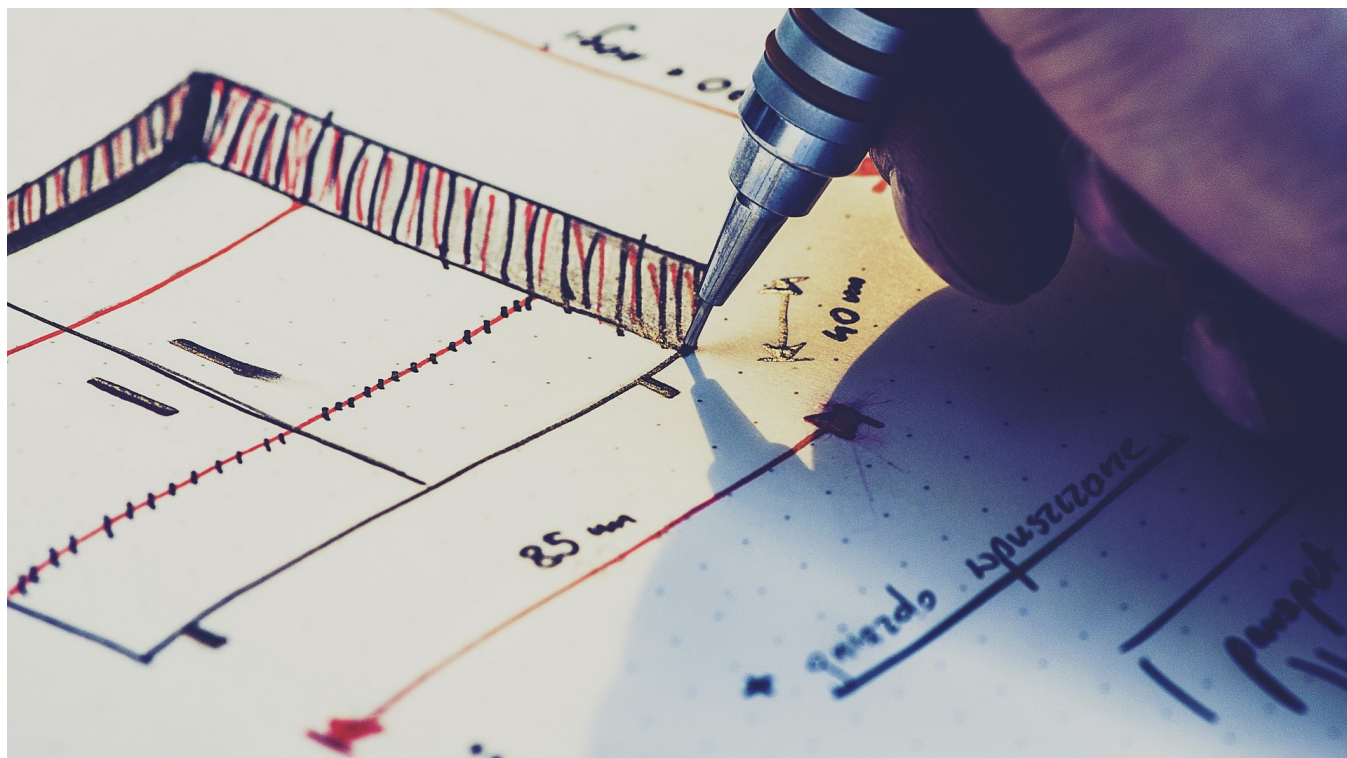


29 | Kafka动态配置了解下？

2019-08-08 胡夕



你好，我是胡夕。今天我要和你讨论的主题是：**Kafka**的动态**Broker**参数配置。

什么是动态**Broker**参数配置？

在开始今天的分享之前，我们先来复习一下设置**Kafka**参数，特别是**Broker**端参数的方法。

在**Kafka**安装目录的config路径下，有个server.properties文件。通常情况下，我们会指定这个文件的路径来启动**Broker**。如果要设置**Broker**端的任何参数，我们必须在这个文件中显式地增加一行对应的配置，之后启动**Broker**进程，令参数生效。我们常见的做法是，一次性设置好所有参数之后，再启动**Broker**。当后面需要变更任何参数时，我们必须重启**Broker**。但生产环境中的服务器，怎么能随意重启呢？所以，目前修改**Broker**端参数是非常痛苦的过程。

基于这个痛点，社区于1.1.0版本中正式引入了动态**Broker**参数（Dynamic Broker Configs）。所谓动态，就是指修改参数值后，无需重启**Broker**就能立即生效，而之前在server.properties中配置参数则称为静态参数（Static Configs）。显然，动态调整参数值而无需重启服务，是非常实用的功能。如果你想体验动态**Broker**参数的话，那就赶快升级到1.1版本吧。

当然了，当前最新的2.3版本中的**Broker**端参数有200多个，社区并没有将每个参数都升级成动态参数，它仅仅是把一部分参数变成了可动态调整。那么，我们应该如何分辨哪些参数是动态参数呢？

如果你打开1.1版本之后（含1.1）的**Kafka**官网，你会发现[Broker Configs](#)表中增加了Dynamic

Update Mode列。该列有3类值，分别是read-only、per-broker和cluster-wide。我来解释一下它们的含义。

- **read-only**。被标记为read-only的参数和原来的参数行为一样，只有重启Broker，才能令修改生效。
- **per-broker**。被标记为per-broker的参数属于动态参数，修改它之后，只会在对应的Broker上生效。
- **cluster-wide**。被标记为cluster-wide的参数也属于动态参数，修改它之后，会在整个集群范围内生效，也就是说，对所有Broker都生效。你也可以为具体的Broker修改cluster-wide参数。

我来举个例子说明一下per-broker和cluster-wide的区别。Broker端参数listeners想必你应该不陌生吧。它是一个per-broker参数，这表示你只能为单个Broker动态调整listeners，而不能直接调整一批Broker的listeners。log.retention.ms参数是cluster-wide级别的，Kafka允许为集群内所有Broker统一设置一个日志留存时间值。当然了，你也可以为单个Broker修改此值。

使用场景

你可能会问，动态Broker参数的使用场景都有哪些呢？实际上，因为不必重启Broker，动态Broker参数的使用场景非常广泛，通常包括但不限于以下几种：

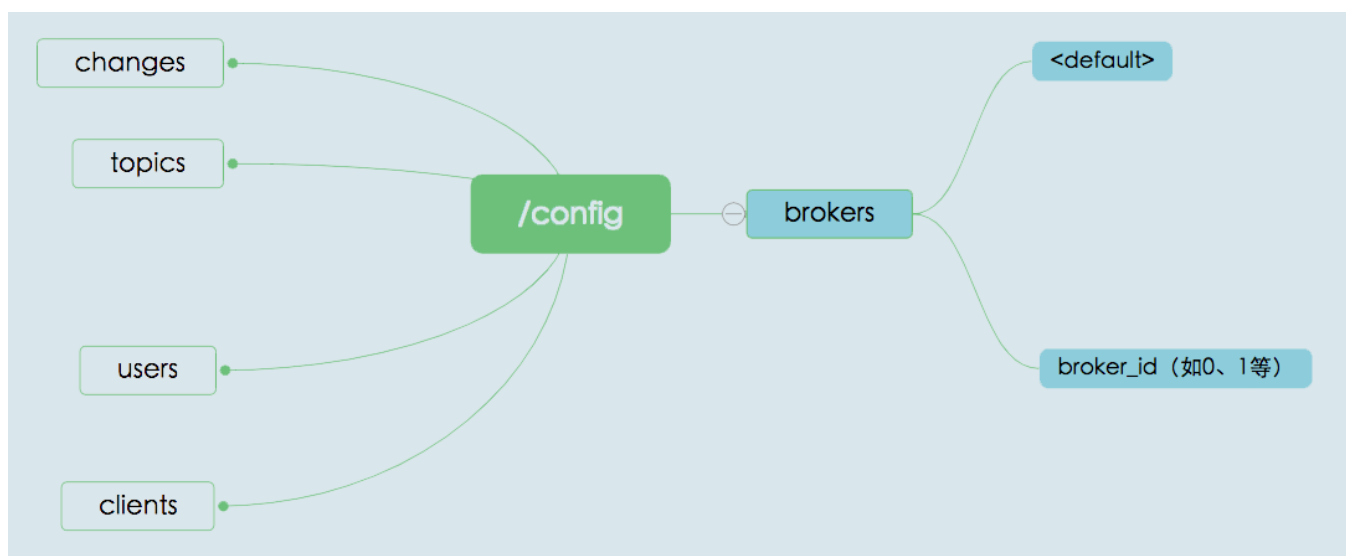
- 动态调整Broker端各种线程池大小，实时应对突发流量。
- 动态调整Broker端连接信息或安全配置信息。
- 动态更新SSL Keystore有效期。
- 动态调整Broker端Compact操作性能。
- 实时变更JMX指标收集器(JMX Metrics Reporter)。

在这些使用场景中，动态调整线程池大小应该算是最实用的功能了。很多时候，当Kafka Broker入站流量（inbound data）激增时，会造成Broker端请求积压（Backlog）。有了动态参数，我们就能够动态增加网络线程数和I/O线程数，快速消耗一些积压。当突发流量过去后，我们也能将线程数调整回来，减少对资源的浪费。整个过程都不需要重启Broker。你甚至可以将这套调整线程数的动作，封装进定时任务中，以实现自动扩缩容。

如何保存？

由于动态配置的特殊性，它必然有和普通只读参数不同的保存机制。下面我来介绍一下Kafka是如何保存动态配置的。

首先，Kafka将动态Broker参数保存在ZooKeeper中，具体的znode路径如下图所示。



我来解释一下图中的内容。**changes**是用来实时监测动态参数变更的，不会保存参数值；**topics**是用来保存**Kafka**主题级别参数的。虽然它们不属于动态**Broker**端参数，但其实它们也是能够动态变更的。

users和**clients**则是用于动态调整客户端配额（**Quota**）的**znode**节点。所谓配额，是指**Kafka**运维人员限制连入集群的客户端的吞吐量或者是限定它们使用的**CPU**资源。

分析到这里，我们就会发现，**/config/brokers znode**才是真正保存动态**Broker**参数的地方。该**znode**下有两大类子节点。第一类子节点就只有一个，它有个固定的名字叫**< default >**，保存的是前面说过的**cluster-wide**范围的动态参数；另一类则以**broker.id**为名，保存的是特定**Broker**的**per-broker**范围参数。由于是**per-broker**范围，因此这类子节点可能存在多个。

我们一起来看一张图片，它展示的是我的一个**Kafka**集群环境上的动态**Broker**端参数。

```

[zk: ... (CONNECTED) 0] ls /config/brokers
[0, <default>, 1]
[zk: ... (CONNECTED) 1] get /config/brokers/<default>
{"version":1,"config":{"num.io.threads":"12","num.network.threads":"5"}}
cZxid = 0x1f
ctime = Thu Jun 20 08:24:47 CST 2019
mZxid = 0x4c
mtime = Thu Jun 20 08:45:05 CST 2019
pZxid = 0x1f
cversion = 0
dataVersion = 6
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 72
numChildren = 0
[zk: ... (CONNECTED) 2] get /config/brokers/0
{"version":1,"config":{"num.io.threads":"16","num.network.threads":"2"}}
cZxid = 0x29
ctime = Thu Jun 20 08:26:50 CST 2019
mZxid = 0x55
mtime = Thu Jun 20 08:45:32 CST 2019
pZxid = 0x29
cversion = 0
dataVersion = 4
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 72
numChildren = 0
[zk: ... (CONNECTED) 3] get /config/brokers/1
{"version":1,"config":{"num.io.threads":"8"}}
cZxid = 0x59
ctime = Thu Jun 20 08:45:44 CST 2019
mZxid = 0x59
mtime = Thu Jun 20 08:45:44 CST 2019
pZxid = 0x59
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 45

```

在这张图中，我首先查看了/config/brokers下的子节点，我们可以看到，这里面有<default>节点和名为0、1的子节点。<default>节点中保存了我设置的cluster-wide范围参数；0和1节点中分别保存了我为Broker 0和Broker 1设置的per-broker参数。

接下来，我分别展示了cluster-wide范围和per-broker范围的参数设置。拿num.io.threads参数为例，其cluster-wide值被动态调整为12，而在Broker 0上被设置成16，在Broker 1上被设置成8。我为Broker 0和Broker 1单独设置的值，会覆盖掉cluster-wide值，但在其他Broker上，该参数默认值还是按12计算。

如果我们再把静态参数加进来一起讨论的话，cluster-wide、per-broker和static参数的优先级是这样的：per-broker参数 > cluster-wide参数 > static参数 > Kafka默认值。

另外，如果你仔细查看上图中的**ephemeralOwner**字段，你会发现它们的值都是**0x0**。这表示这些**znode**都是持久化节点，它们将一直存在。即使**ZooKeeper**集群重启，这些数据也不会丢失，这样就能保证这些动态参数的值会一直生效。

如何配置？

讲完了保存原理，我们来说说如何配置动态**Broker**参数。目前，设置动态参数的工具行命令只有一个，那就是**Kafka**自带的**kafka-configs**脚本。接下来，我来以**unclean.leader.election.enable**参数为例，演示一下如何动态调整。

下面这条命令展示了如何在集群层面设置全局值，即设置**cluster-wide**范围值。

```
$ bin/kafka-configs.sh --bootstrap-server kafka-host:port --entity-type brokers --entity-default --alter --add-config unclean.leader.election.enable=true
Completed updating default config for brokers in the cluster,
```

总体来说命令很简单，但有一点需要注意。如果要设置**cluster-wide**范围的动态参数，需要显式指定**entity-default**。现在，我们使用下面的命令来查看一下刚才的配置是否成功。

```
$ bin/kafka-configs.sh --bootstrap-server kafka-host:port --entity-type brokers --entity-default --describe
Default config for brokers in the cluster are:
unclean.leader.election.enable=true sensitive=false synonyms={DYNAMIC_DEFAULT_BROKER_CONFIG:unclean.leader.election.enable}
```

从输出来看，我们成功地在全局层面上设置该参数值为**true**。注意**sensitive=false**的字眼，它表明我们要调整的参数不是敏感数据。如果我们调整的是类似于密码这样的参数时，该字段就会为**true**，表示这属于敏感数据。

好了，调整完**cluster-wide**范围的参数，我来演示下如何设置**per-broker**范围参数。我们还是以**unclean.leader.election.enable**参数为例，我现在为ID为**1**的**Broker**设置一个不同的值。命令如下：

```
$ bin/kafka-configs.sh --bootstrap-server kafka-host:port --entity-type brokers --entity-name 1 --alter --add-config unclean.leader.election.enable=false
Completed updating config for broker: 1.
```

同样，我们使用下列命令，来查看一下刚刚的设置是否生效了。

```
$ bin/kafka-configs.sh --bootstrap-server kafka-host:port --entity-type brokers --entity-name 1 --describe  
Configs for broker 1 are:  
unclean.leader.election.enable=false sensitive=false synonyms={DYNAMIC_BROKER_CONFIG:unclean.leader.election.enable=false}
```

这条命令的输出信息很多。我们关注两点即可。

1. 在**Broker 1**层面上，该参数被设置成了**false**，这表明命令运行成功了。
2. 从倒数第二行可以看出，在全局层面上，该参数值依然是**true**。这表明，我们之前设置的**cluster-wide**范围参数值依然有效。

如果我们要删除**cluster-wide**范围参数或**per-broker**范围参数，也非常简单，分别执行下面的命令就可以了。

```
# 删除cluster-wide范围参数  
$ bin/kafka-configs.sh --bootstrap-server kafka-host:port --entity-type brokers --entity-default --alter --delete-config u  
Completed updating default config for brokers in the cluster,
```

```
# 删除per-broker范围参数  
$ bin/kafka-configs.sh --bootstrap-server kafka-host:port --entity-type brokers --entity-name 1 --alter --delete-config  
Completed updating config for broker: 1.
```

删除动态参数要指定**delete-config**。当我们删除完动态参数配置后，再次运行查看命令，结果如下：

```
# 查看cluster-wide范围参数  
$ bin/kafka-configs.sh --bootstrap-server kafka-host:port --entity-type brokers --entity-default --describe  
Default config for brokers in the cluster are:
```

```
# 查看Broker 1上的动态参数配置  
$ bin/kafka-configs.sh --bootstrap-server kafka-host:port --entity-type brokers --entity-name 1 --describe  
Configs for broker 1 are:
```

此时，刚才配置的所有动态参数都已经被成功移除了。

刚刚我只是举了一个参数的例子，如果你要知道动态Broker参数都有哪些，一种方式是在Kafka官网中查看Broker端参数列表，另一种方式是直接运行无参数的kafka-configs脚本，该脚本的说明文档会告诉你当前动态Broker参数都有哪些。我们可以先来看看下面这两张图。

```
For entity-type 'brokers':
log.message.timestamp.type
ssl.client.auth
log.retention.ms
sasl.login.refresh.window.jitter
sasl.kerberos.ticket.renew.window.factor
log.preallocate
log.index.size.max.bytes
sasl.login.refresh.window.factor
ssl.truststore.type
ssl.keymanager.algorithm
log.cleaner.io.buffer.load.factor
sasl.login.refresh.min.period.seconds
ssl.key.password
background.threads
log.retention.bytes
ssl.trustmanager.algorithm
log.segment.bytes
max.connections.per.ip.overrides
log.cleaner.delete.retention.ms
log.segment.delete.delay.ms
min.insync.replicas
ssl.keystore.location
ssl.cipher.suites
log.roll.jitter.ms
log.cleaner.backoff.ms
sasl.jaas.config
principal.builder.class
```

```
log.flush.interval.ms  
log.cleaner.dedupe.buffer.size  
log.flush.interval.messages  
advertised.listeners  
num.io.threads  
listener.security.protocol.map  
log.message.downconversion.enable  
sasl.enabled.mechanisms  
sasl.login.refresh.buffer.seconds  
ssl.truststore.password  
listeners
```

```
metric.reporters  
ssl.protocol  
sasl.kerberos.ticket.renew.jitter  
ssl.keystore.password  
sasl.mechanism.inter.broker.protocol  
log.cleanup.policy  
sasl.kerberos.principal.to.local.rules  
sasl.kerberos.min.time.before.relogin  
num.recovery.threads.per.data.dir  
log.cleaner.io.max.bytes.per.second  
log.roll.ms  
ssl.endpoint.identification.algorithm  
unclean.leader.election.enable  
message.max.bytes  
log.cleaner.threads  
log.cleaner.io.buffer.size  
max.connections.per.ip  
sasl.kerberos.service.name  
ssl.provider  
follower.replication.throttled.rate  
log.index.interval.bytes
```



```
log.cleaner.min.compaction.lag.ms
log.message.timestamp.difference.max.ms
ssl.enabled.protocols
log.cleaner.min.cleanable.ratio
replica.alter.log.dirs.io.max.bytes.per.second
ssl.keystore.type
ssl.secure.random.implementation
ssl.truststore.location
sasl.kerberos.kinit.cmd
leader.replication.throttled.rate
num.network.threads
compression.type
num.replica.fetchers
```

看到有这么多动态Broker参数，你可能会问：这些我都需要调整吗？你能告诉我最常用的几个吗？根据我的实际使用经验，我来跟你分享一些有较大几率被动态调整值的参数。

1.log.retention.ms。

修改日志留存时间应该算是一个比较高频的操作，毕竟，我们不可能完美地预估所有业务的消息留存时长。虽然该参数有对应的主题级别参数可以设置，但拥有在全局层面上动态变更的能力，依然是一个很好的功能亮点。

2.num.io.threads和num.network.threads。

这是我们在前面提到的两组线程池。就我个人而言，我觉得这是动态Broker参数最实用的场景了。毕竟，在实际生产环境中，Broker端请求处理能力经常要按需扩容。如果没有动态Broker参数，我们是无法做到这一点的。

3.与SSL相关的参数。

主要是4个参数（ssl.keystore.type、ssl.keystore.location、ssl.keystore.password和ssl.key.password）。允许动态实时调整它们之后，我们就能创建那些过期时间很短的SSL证书。每当我们调整时，Kafka底层会重新配置Socket连接通道并更新Keystore。新的连接会使用新的Keystore，阶段性地调整这组参数，有利于增加安全性。

4.num.replica.fetchers。

这也是我认为的最实用的动态**Broker**参数之一。**Follower**副本拉取速度慢，在线上**Kafka**环境中一直是一个老大难的问题。针对这个问题，常见的做法是增加该参数值，确保有充足的线程可以执行**Follower**副本向**Leader**副本的拉取。现在有了动态参数，你不需要再重启**Broker**，就能立即在**Follower**端生效，因此我说这是很实用的应用场景。

小结

好了，我们来小结一下。今天，我们重点讨论了**Kafka 1.1.0**版本引入的动态**Broker**参数。这类参数最大的好处在于，无需重启**Broker**，就可以令变更生效，因此能够极大地降低运维成本。除此之外，我还给出了动态参数的保存机制和设置方法。在专栏的后面，我还会给出动态参数设置的另一种方法，敬请期待。

Kafka动态参数配置

动态Broker参数

- 指修改参数值后，无需重启Broker就能立即生效的参数。

动态Broker参数常见的5种使用场景

- 动态调整Broker端各种线程池大小，实时应对突发流量。
- 动态调整Broker端连接信息或安全配置信息。
- 动态更新SSL Keystore有效期。
- 动态调整Broker端Compact操作性能。
- 实时变更JMX指标收集器。

有较大几率被调整的参数

- log.retention.ms。
- num.io.threads和num.network.threads。
- 与SSL相关的参数。
- num.replica.fetchers。



开放讨论

目前，社区只是将一部分Broker参数升级为动态参数，在实际使用过程中，你觉得还有哪些参数也应该变为可动态修改？

欢迎写下你的思考和答案，我们一起讨论。如果你觉得有所收获，也欢迎把文章分享给你的朋友。

Kafka 核心技术与实战

全面提升你的 Kafka 实战能力

胡夕

人人贷计算平台部总监

Apache Kafka Contributor



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

精选留言



外星人

1

你好，我们用的版本是1.1，请问下，动态参数有没有bug啊？

2019-08-08

作者回复

目前没有太严重的bug

2019-08-08



玉剑冰锋

1

这么看的话，是不是不管是动态参数还是静态参数，集群中的broker配置是可以不一样的对吧？我用0.11（测试环境），环境中三台kafka磁盘容量不一样，导致的问题就是另外两台磁盘已经到了警戒值，剩下那一台磁盘始终用不上，如果可以的话我直接调整一台的保留时间就好了

2019-08-08

作者回复

每个Broker可以是不同的配置。

2019-08-08



外星人

0

您好，我们生产上num.io.threads已经提高到了24，但是对应idle还是在0.3左右，是不是kafka pi哪个接口处理慢了？这种情况一般如何定位啊？谢谢

2019-08-22

作者回复

大概率是PRODUCE和FETCH请求。不过你最好结合`kafka.network:type=RequestMetrics,name=RequestQueueTimeMs,request={Produce|FetchConsumer|FetchFollower}`确认下是否需要调整`num.network.threads`

2019-08-22



无菇朋友

👍 0

老师 请问下 怎么查看每个broker的请求积压情况？

2019-08-20

作者回复

可以看看broker端的请求队列的JMX指标

2019-08-21



godtrue

👍 0

请问老师，kafka动态调整配置的实现原理是啥？它怎么做到不需重启broker就能使配置生效的？

2019-08-18

作者回复

把配置保存在ZooKeeper中，然后动态监听这个变更，一旦监听到立即处理

2019-08-19



Sunney

👍 0

老师你好，我想咨询一下NATS和Kafka的区别和各自的适用场景是什么？

2019-08-09



cricket1981

👍 0

follower为什么会拉取副本慢？它不负责读写只专心同步副本数据，增加线程就有用吗？会增加leader负担吗？

2019-08-09

作者回复

因为是异步的网络请求。一个Broker上可能有很多follower副本，增加线程可以均匀分散这些follower副本来去任务

2019-08-09



cricket1981

👍 0

怎样知道什么时候该调整这两组线程池大小？

2019-08-09

作者回复

可以监控请求队列的大小

2019-08-09



边城

👍 0

您好，动态参数会一直生效吗？是不是需要在`server.properties`里同步维护，防止下次重启时参数失效。

2019-08-08

| 作者回复

不会失效，即使重启

2019-08-08



玉剑冰锋

👍 0

如果可以每台设置不同的日志保存时间，这样的话会导致分区留存时间不一致，消费者消费的问题怎么办？

2019-08-08

| 作者回复

统一的机制。反正消费不到数据的话就发生位移越界的情况

2019-08-08