

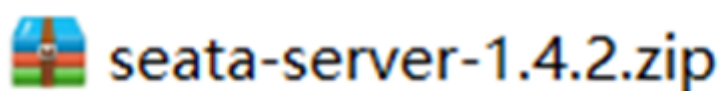
seata的部署和集成

一、部署Seata的tc-server

1.下载

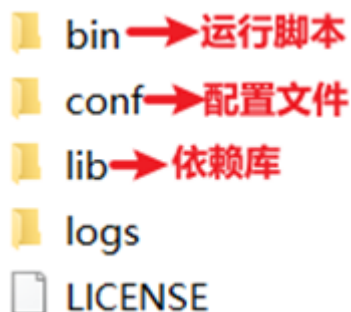
首先我们要下载seata-server包，地址在<http://seata.io/zh-cn/blog/download.html>

当然，课前资料也准备好了：



2.解压

在非中文目录解压缩这个zip包，其目录结构如下：



3.修改配置

修改conf目录下的registry.conf文件：

名称

- logback
- META-INF
- file.conf
- file.conf.example
- logback.xml
- README.md
- README-zh.md
- registry.conf

内容如下：

```
registry {
  # tc服务的注册中心类，这里选择nacos，也可以是eureka、zookeeper等
  type = "nacos"

  nacos {
    # seata tc 服务注册到 nacos的服务名称，可以自定义
    application = "seata-tc-server"
    serverAddr = "127.0.0.1:8848"
    group = "DEFAULT_GROUP"
    namespace = ""
    cluster = "SH"
    username = "nacos"
    password = "nacos"
  }
}

config {
  # 读取tc服务端的配置文件的方式，这里是从nacos配置中心读取，这样如果tc是集群，可以共享配置
  type = "nacos"
  # 配置nacos地址等信息
  nacos {
    serverAddr = "127.0.0.1:8848"
    namespace = ""
    group = "SEATA_GROUP"
    username = "nacos"
    password = "nacos"
    dataId = "seataServer.properties"
  }
}
```

4.在nacos添加配置

特别注意，为了让tc服务的集群可以共享配置，我们选择了nacos作为统一配置中心。因此服务端配置文件seataServer.properties文件需要在nacos中配好。

格式如下：

新建配置

* Data ID:

seataServer.properties

* Group:

DEFAULT_GROUP

更多高级选项

描述:

配置格式:

☐ TEXT

☐ JSON

☐ XML

☐ YAML

☐ HTML

☒ Properties

* 配置内容: ? :

1

配置内容如下：

```
# 数据存储方式，db代表数据库
store.mode=db
store.db.datasource=druid
store.db.dbType=mysql
store.db.driverClassName=com.mysql.jdbc.Driver
store.db.url=jdbc:mysql://127.0.0.1:3306/seata?
useUnicode=true&rewriteBatchedStatements=true
store.db.user=root
store.db.password=123
store.db.minConn=5
store.db.maxConn=30
store.db.globalTable=global_table
store.db.branchTable=branch_table
store.db.queryLimit=100
store.db.lockTable=lock_table
store.db.maxWait=5000
# 事务、日志等配置
server.recovery.committingRetryPeriod=1000
server.recovery.asynCommittingRetryPeriod=1000
server.recovery.rollbackingRetryPeriod=1000
server.recovery.timeoutRetryPeriod=1000
server.maxCommitRetryTimeout=-1
server.maxRollbackRetryTimeout=-1
server.rollbackRetryTimeoutUnlockEnable=false
server.undo.logSaveDays=7
server.undo.logDeletePeriod=86400000

# 客户端与服务端传输方式
transport.serialization=seata
transport.compressor=none
# 关闭metrics功能，提高性能
```

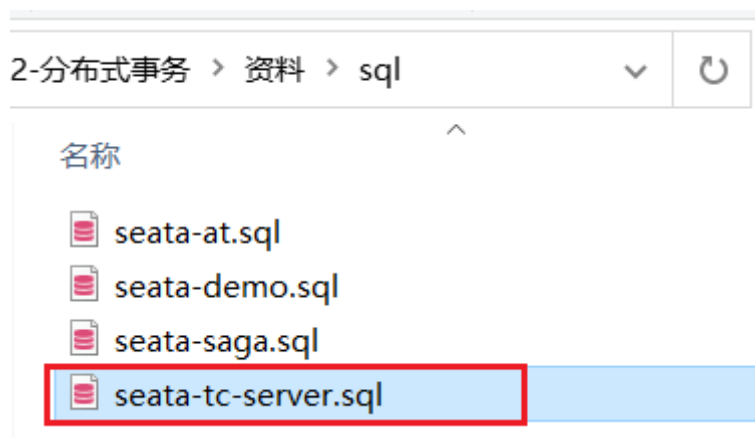
```
metrics.enabled=false
metrics.registryType=compact
metrics.exporterList=prometheus
metrics.exporterPrometheusPort=9898
```

==其中的数据库地址、用户名、密码都需要修改成你自己的数据库信息。==

5.创建数据库表

特别注意：tc服务在管理分布式事务时，需要记录事务相关数据到数据库中，你需要提前创建好这些表。

新建一个名为seata的数据库，运行课前资料提供的sql文件：



这些表主要记录全局事务、分支事务、全局锁信息：

```
SET NAMES utf8mb4;
SET FOREIGN_KEY_CHECKS = 0;

-- -----
-- 分支事务表
-- -----

DROP TABLE IF EXISTS `branch_table`;
CREATE TABLE `branch_table` (
  `branch_id` bigint(20) NOT NULL,
  `xid` varchar(128) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `transaction_id` bigint(20) NULL DEFAULT NULL,
  `resource_group_id` varchar(32) CHARACTER SET utf8 COLLATE utf8_general_ci
NULL DEFAULT NULL,
  `resource_id` varchar(256) CHARACTER SET utf8 COLLATE utf8_general_ci NULL
DEFAULT NULL,
  `branch_type` varchar(8) CHARACTER SET utf8 COLLATE utf8_general_ci NULL
DEFAULT NULL,
  `status` tinyint(4) NULL DEFAULT NULL,
  `client_id` varchar(64) CHARACTER SET utf8 COLLATE utf8_general_ci NULL
DEFAULT NULL,
  `application_data` varchar(2000) CHARACTER SET utf8 COLLATE utf8_general_ci
NULL DEFAULT NULL,
  `gmt_create` datetime(6) NULL DEFAULT NULL,
  `gmt_modified` datetime(6) NULL DEFAULT NULL,
```

```

PRIMARY KEY (`branch_id`) USING BTREE,
INDEX `idx_xid`(`xid`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT =
Compact;

-----
-- 全局事务表
-----

DROP TABLE IF EXISTS `global_table`;
CREATE TABLE `global_table` (
  `xid` varchar(128) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `transaction_id` bigint(20) NULL DEFAULT NULL,
  `status` tinyint(4) NOT NULL,
  `application_id` varchar(32) CHARACTER SET utf8 COLLATE utf8_general_ci NULL
  DEFAULT NULL,
  `transaction_service_group` varchar(32) CHARACTER SET utf8 COLLATE
  utf8_general_ci NULL DEFAULT NULL,
  `transaction_name` varchar(128) CHARACTER SET utf8 COLLATE utf8_general_ci
  NULL DEFAULT NULL,
  `timeout` int(11) NULL DEFAULT NULL,
  `begin_time` bigint(20) NULL DEFAULT NULL,
  `application_data` varchar(2000) CHARACTER SET utf8 COLLATE utf8_general_ci
  NULL DEFAULT NULL,
  `gmt_create` datetime NULL DEFAULT NULL,
  `gmt_modified` datetime NULL DEFAULT NULL,
  PRIMARY KEY (`xid`) USING BTREE,
  INDEX `idx_gmt_modified_status`(`gmt_modified`, `status`) USING BTREE,
  INDEX `idx_transaction_id`(`transaction_id`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT =
Compact;

SET FOREIGN_KEY_CHECKS = 1;


```


6.启动TC服务

进入bin目录，运行其中的seata-server.bat即可：

```
D:) > lesson > seata > bin
```

名称

 seata-server.bat

 seata-server.sh

启动成功后，seata-server应该已经注册到nacos注册中心了。

打开浏览器，访问nacos地址：<http://localhost:8848>，然后进入服务列表页面，可以看到seata-tc-server的信息：

NACOS 1.4.1	public				
配置管理	服务列表 public				
服务管理	服务名称 <input type="text" value="请输入服务名称"/> 分组名称 <input type="text" value="请输入分组名称"/> 隐藏空服务: <input checked="" type="checkbox"/> 查询				
服务列表					
订阅者列表					
权限控制					
	服务名	分组名称	集群数目	实例数	健康实例数
	seata-tc-server	DEFAULT_GROUP	1	1	1

二、微服务集成seata

1.引入依赖

首先，我们需要在微服务中引入seata依赖：

```
<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-seata</artifactId>
  <exclusions>
    <!-- 版本较低，1.3.0，因此排除-->
    <exclusion>
      <artifactId>seata-spring-boot-starter</artifactId>
      <groupId>io.seata</groupId>
    </exclusion>
  </exclusions>
</dependency>
<!-- seata starter 采用1.4.2版本-->
<dependency>
  <groupId>io.seata</groupId>
  <artifactId>seata-spring-boot-starter</artifactId>
  <version>${seata.version}</version>
</dependency>
```

2.修改配置文件

需要修改application.yml文件，添加一些配置：

```
seata:
  registry: # TC服务注册中心的配置，微服务根据这些信息去注册中心获取tc服务地址
            # 参考tc服务自己的registry.conf中的配置
  type: nacos
  nacos: # tc
    server-addr: 127.0.0.1:8848
    namespace: ""
    group: DEFAULT_GROUP
    application: seata-tc-server # tc服务在nacos中的服务名称
```

```
cluster: SH
tx-service-group: seata-demo # 事务组，根据这个获取tc服务的cluster名称
service:
  vgroup-mapping: # 事务组与TC服务cluster的映射关系
    seata-demo: SH
```

三、TC服务的高可用和异地容灾

1.模拟异地容灾的TC集群

计划启动两台seata的tc服务节点：

节点名称	ip地址	端口号	集群名称
seata	127.0.0.1	8091	SH
seata2	127.0.0.1	8092	HZ

之前我们已经启动了一台seata服务，端口是8091，集群名为SH。

现在，将seata目录复制一份，起名为seata2

修改seata2/conf/registry.conf内容如下：

```
registry {
  # tc服务的注册中心类，这里选择nacos，也可以是eureka、zookeeper等
  type = "nacos"

  nacos {
    # seata tc 服务注册到 nacos的服务名称，可以自定义
    application = "seata-tc-server"
    serverAddr = "127.0.0.1:8848"
    group = "DEFAULT_GROUP"
    namespace = ""
    cluster = "HZ"
    username = "nacos"
    password = "nacos"
  }
}

config {
  # 读取tc服务端的配置文件的方式，这里是从nacos配置中心读取，这样如果tc是集群，可以共享配置
  type = "nacos"
  # 配置nacos地址等信息
  nacos {
    serverAddr = "127.0.0.1:8848"
    namespace = ""
    group = "SEATA_GROUP"
    username = "nacos"
    password = "nacos"
    dataId = "seataServer.properties"
  }
}
```

进入seata2/bin目录，然后运行命令：

```
seata-server.bat -p 8092
```

打开nacos控制台，查看服务列表：

服务名	分组名称	集群数目	实例数	健康实例数
seata-tc-server	DEFAULT_GROUP	2	2	2

点进详情查看：

集群：HZ

元数据过滤

key

value

添加过滤

IP	端口	临时实例	权重	健康状态	元数据
192.168.150.1	8092	true	1	true	

集群：SH

元数据过滤

key

value

添加过滤

IP	端口	临时实例	权重	健康状态	元数据
192.168.150.1	8091	true	1	true	

2.将事务组映射配置到nacos

接下来，我们需要将tx-service-group与cluster的映射关系都配置到nacos配置中心。

新建一个配置：

新建配置

* Data ID: client.properties

* Group: SEATA_GROUP

[更多高级选项](#)

描述:

配置格式: ☐ TEXT ☐ JSON ☐ XML ☐ YAML ☐ HTML ☒ Properties

* 配置内容: ? :

1

配置的内容如下:

```
# 事务组映射关系
service.vgroupMapping.seata-demo=SH

service.enableDegrade=false
service.disableGlobalTransaction=false
# 与TC服务的通信配置
transport.type=TCP
transport.server=NIO
transport.heartbeat=true
transport.enableClientBatchSendRequest=false
transport.threadFactory.bossThreadPrefix=NettyBoss
transport.threadFactory.workerThreadPrefix=NettyServerNIOworker
transport.threadFactory.serverExecutorThreadPrefix=NettyServerBizHandler
transport.threadFactory.shareBossWorker=false
transport.threadFactory.clientSelectorThreadPrefix=NettyClientSelector
transport.threadFactory.clientSelectorThreadSize=1
transport.threadFactory.clientWorkerThreadPrefix=NettyClientWorkerThread
transport.threadFactory.bossThreadSize=1
transport.threadFactory.workerThreadSize=default
transport.shutdown.wait=3
# RM配置
client.rm.asyncCommitBufferLimit=10000
client.rm.lock.retryInterval=10
client.rm.lock.retryTimes=30
client.rm.lock.retryPolicyBranchRollbackOnConflict=true
client.rm.reportRetryCount=5
client.rm.tableMetaCheckEnable=false
client.rm.tableMetaCheckerInterval=60000
client.rm.sqlParserType=druid
client.rm.reportSuccessEnable=false
client.rm.sagaBranchRegisterEnable=false
# TM配置
client.tm.commitRetryCount=5
client.tm.rollbackRetryCount=5
client.tm.defaultGlobalTransactionTimeout=60000
client.tm.degradeCheck=false
client.tm.degradeCheckAllowTimes=10
client.tm.degradeCheckPeriod=2000
```

```
# undo日志配置
client.undo.dataValidation=true
client.undo.logSerialization=jackson
client.undo.onlyCareUpdateColumns=true
client.undo.logTable=undo_log
client.undo.compress.enable=true
client.undo.compress.type=zip
client.undo.compress.threshold=64k
client.log.exceptionRate=100
```

3.微服务读取nacos配置

接下来，需要修改每一个微服务的application.yml文件，让微服务读取nacos中的client.properties文件：

```
seata:
  config:
    type: nacos
    nacos:
      server-addr: 127.0.0.1:8848
      username: nacos
      password: nacos
      group: SEATA_GROUP
      data-id: client.properties
```

重启微服务，现在微服务到底是连接tc的SH集群，还是tc的HZ集群，都统一由nacos的client.properties来决定了。