

Kafka

一、项目概述

二、环境准备

(一) 安装 Java

(二) 安装 Kafka

第二种方法(运行失败)

安装错误：<https://github.com/Homebrew/brew>

1. 安装Homebrew： 如果你还没有安装Homebrew， 可以通过以下命令安装：

2. Sh /bin/bash -c "\$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install....

3. 安装Java： Kafka依赖于Java， 因此需要确保Java已经安装。你可以通过以下命令安装Java：

4. Sh brew install openjdk

5. 安装Kafka： 使用Homebrew安装Kafka：

6. Sh brew install kafka

7. 启动Zookeeper： Kafka依赖于Zookeeper来管理集群状态。启动Zookeeper：

8. Sh zkServer start

9. 启动Kafka服务器： 在另一个终端窗口中， 启动Kafka服务器：

10. Sh kafka-server-start /usr/local/etc/kafka/server.properties

11. 验证安装： 你可以通过创建一个主题并发送和接收消息来验证Kafka是否安装成功。

12. 创建一个主题：

13. Sh kafka-topics --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions...

14. 列出所有主题：

15. Sh kafka-topics --list --bootstrap-server localhost:9092

16. 发送消息：

17. Sh kafka-console-producer --broker-list localhost:9092 --topic test

18. 接收消息：

19. Sh kafka-console-consumer --bootstrap-server localhost:9092 --topic test --from-beginning

20. 通过以上步骤， 你应该能够在Mac上成功安装并运行Kafka。

(三) 安装 Maven

三、项目创建与依赖导入

(一) 创建 Maven 项目

(二) 导入依赖

四、案例复现

(一) 基础数据传输案例

注意改为

创建 “in” 主题

创建 “out” 主题

总结

1. 项目功能的主动性

2. 与 Kafka 交互的实现

3. 数据处理流程的驱动

(二) 计算数字总和案例

注意修改新版本的

创建sumIn主题

创建sumOut主题

(三) WordCount 案例

注意

创建 “wordin” 主题

创建 “wordout” 主题

(四) 窗口操作案例

1. 跳跃时间窗口 (Hopping time window)

2. 滚动时间窗口 (Tumbling time window)

3. 会话窗口 (Session window)

4. 滑动时间窗口 (Sliding time window)

一、项目概述

本项目旨在复现基于 Kafka Streaming 的多个数据处理案例，包括基础数据传输、计算数字总和、WordCount 以及不同类型的窗口操作（跳跃时间窗口、滚动时间窗口、会话窗口、滑动时间窗口），以帮助初学者理解和掌握 Kafka Streaming 的基本功能和使用方法。

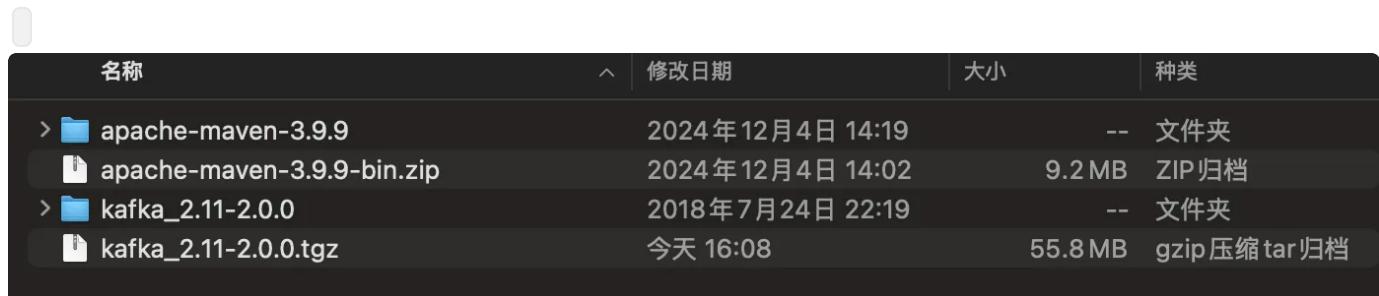
二、环境准备

(一) 安装 Java

1. 从 Oracle 官网 (<https://www.oracle.com/java/technologies/javase-downloads.html>) 下载适合您操作系统的 Java Development Kit (JDK) 版本，建议选择 Java 8 或更高版本。
2. 按照安装向导完成 Java 的安装。安装完成后，打开命令行终端，输入 `java -version` 命令，如果能够正确显示 Java 版本信息，则表示安装成功。

(二) 安装 Kafka

1. 从 Kafka 官方网站 (<https://kafka.apache.org/downloads>) 下载最新稳定版本的 Kafka。
2. 解压下载的压缩包到您选择的目录，例如在 Linux 系统下，可以使用命令 `tar -zxvf kafka_<version>.tgz` (将 `<version>` 替换为实际下载的版本号)。



3. 进入解压后的 Kafka 目录，编辑配置文件 `config/server.properties`。主要配置项如下：
 - `broker.id`：为每个 Kafka broker 设置唯一的 ID，例如 `broker.id=0`。
 - `listeners`：指定 Kafka 监听的地址和端口，如 `listeners=PLAINTEXT://localhost:9092` (表示使用本地地址和 9092 端口进行通信，如果在服务器环境中，需根据实际情况修改为服务器的 IP 地址)。
 - `log.dirs`：设置 Kafka 存储日志文件的目录，例如 `log.dirs=/tmp/kafka-logs` (可根据实际需求修改目录路径)。
4. 启动 Zookeeper (Kafka 依赖 Zookeeper 进行协调管理)。在 Kafka 目录下，执行命令 `bin/zookeeper-server-start.sh config/zookeeper.properties` (Linux 系统下)。如果是 Windows 系统，则执行 `bin\windows\zookeeper-server-start.bat config\zookeeper.properties`。

5. 启动 Kafka broker。在另一个命令行窗口中，执行 `bin/kafka-server-start.sh config/server.properties` (Linux 系统) 或 `bin\windows\kafka-server-start.bat config\server.properties` (Windows 系统)。

```

socket.receive.buffer.bytes = 102400
socket.request.max.bytes = 104857600
socket.send.buffer.bytes = 102400
ssl.allow.dn.changes = false
ssl.allow.samething = false
ssl.cipher.suites = []
ssl.client.auth = none
ssl.enabled.protocols = [TLSv1.2, TLSv1.3]
ssl.endpoint.identification.algorithm = https
ssl.engine.factory.class = null
ssl.key.password = null
ssl.keymanager.algorithm = SunX509
ssl.keystore.certificate.chain = null
ssl.keystore.key = null
ssl.keystore.location = null
ssl.keystore.password = null
ssl.keystore.type = JKS
ssl.principal.mapping.rules = DEFAULT
ssl.protocol.version = 1.3
ssl.provider = null
ssl.secure.random.implementation = null
ssl.trustmanager.algorithm = PKIX
ssl.truststore.certificates = null
ssl.truststore.location = null
ssl.truststore.password = null
ssl.truststore.type = JKS
telemetry.max.bytes = 1048576
transaction.abort.timed.out.transaction.cleanup.interval.ms = 10000
transaction.max.timeout.ms = 900000
transaction.partition.verify.enable = true
transaction.remove.expired.transaction.cleanup.interval.ms = 3600000
transaction.state.log.load.buffer.size = 5242880
transaction.state.log.segment.bytes = 104857600
transaction.state.log.segment.factor = 1
transaction.state.log.replication.factor = 1
transaction.state.log.segment.bytes = 104857600
transactional.id.expiration.ms = 604800000
unclean.leader.election.enable = false
unclean.leader.election.interval.ms = 300000
unstable.api.versions.enable = false
unstable.feature.versions.enable = false
zookeeper.clientCnxnSocket = null
zookeeper.connect = localhost:2181
zookeeper.connection.timeout.ms = 10000
zookeeper.max.in.flight.requests = 10
zookeeper.metadata.migration.enable = false
zookeeper.metadata.migration.min.batch.size = 200
zookeeper.session.timeout.ms = 18000
zookeeper.ssl.aci = false
zookeeper.ssl.cipher.suites = null
zookeeper.ssl.client.enable = false
zookeeper.ssl.crl.enable = false
zookeeper.ssl.enabled.protocols = null
zookeeper.ssl.endpoint.identification.algorithm = HTTPS
zookeeper.ssl.keystore.location = null
zookeeper.ssl.keystore.password = null
zookeeper.ssl.keystore.type = null
zookeeper.ssl.ocsp.enable = false
zookeeper.ssl.protocol = TLSv1.2
zookeeper.ssl.truststore.location = null
zookeeper.ssl.truststore.password = null
zookeeper.ssl.truststore.type = null
(kafka_server.KafkaConfig)
[2024-12-08 16:36:45,542] INFO [ThrottledChannelReaper-Fetch]: Starting (kafka.server.ClientQuotaManager$ThrottledChannelReaper)
[2024-12-08 16:36:45,542] INFO [ThrottledChannelReaper-Producer]: Starting (kafka.server.ClientQuotaManager$ThrottledChannelReaper)

```

第二种方法(运行失败)

在Mac上安装Kafka可以通过以下步骤完成：

安装错误： <https://github.com/Homebrew/brew>

1. 安装Homebrew： 如果你还没有安装Homebrew，可以通过以下命令安装：

2. Sh

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

3. 安装Java： Kafka依赖于Java，因此需要确保Java已经安装。你可以通过以下命令安装Java：

4. Sh

```
brew install openjdk
```

5. 安装Kafka： 使用Homebrew安装Kafka：

6. Sh

```
brew install kafka
```

7. 启动Zookeeper: Kafka依赖于Zookeeper来管理集群状态。启动Zookeeper:

8. Sh

```
zkServer start
```

9. 启动Kafka服务器: 在另一个终端窗口中, 启动Kafka服务器:

10. Sh

```
kafka-server-start /usr/local/etc/kafka/server.properties
```

11. 验证安装: 你可以通过创建一个主题并发送和接收消息来验证Kafka是否安装成功。

12. 创建一个主题:

13. Sh

```
kafka-topics --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic test
```

14. 列出所有主题:

15. Sh

```
kafka-topics --list --bootstrap-server localhost:9092
```

16. 发送消息:

17. Sh

```
kafka-console-producer --broker-list localhost:9092 --topic test
```

18. 接收消息:

19. Sh

```
kafka-console-consumer --bootstrap-server localhost:9092 --topic test --from-beginning
```

20. 通过以上步骤, 你应该能够在Mac上成功安装并运行Kafka。

(三) 安装 Maven

1. 从 Maven 官方网站 (<https://maven.apache.org/download.cgi>) 下载最新版本的 Maven 二进制压缩包。
2. 解压压缩包到您指定的目录，例如在 Linux 系统下使用 `tar -zxvf apache-maven-<version>.tar.gz` (将 `<version>` 替换为实际下载的版本号)。
3. 配置 Maven 环境变量。编辑 `~/.bashrc` 文件 (如果使用的是其他 Shell 环境，对应编辑相应的配置文件)，添加以下内容 (假设 Maven 解压到 `/opt/maven` 目录下，根据实际情况修改路径)：

```
Bash |  
1 export MAVEN_HOME=/opt/maven  
2 export PATH=$PATH:$MAVEN_HOME/bin
```

保存后，在命令行执行 `source ~/.bashrc` 使配置生效。在命令行输入 `mvn -version`，如果能正确显示 Maven 版本信息，则表示安装成功。

三、项目创建与依赖导入

(一) 创建 Maven 项目

1. 打开命令行终端，进入您想要创建项目的目录。
2. 执行以下命令创建一个 Maven 项目：

```
Bash |  
1 mvn archetype:generate -DgroupId=com.example -DartifactId=kafka-streaming-demo -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

其中，`groupId` 是项目的组织唯一标识符，`artifactId` 是项目的唯一标识符，这里创建了一个名为 `kafka-streaming-demo` 的项目。

(二) 导入依赖

1. 进入项目目录 `kafka-streaming-demo`，找到 `pom.xml` 文件，使用文本编辑器打开。
2. 在 `<dependencies>` 标签内添加以下依赖项：

```

1 <dependency>
2   <groupId>org.apache.kafka</groupId>
3   <artifactId>kafka-clients</artifactId>
4   <version>2.0.0</version>
5 </dependency>
6 <dependency>
7   <groupId>org.apache.kafka</groupId>
8   <artifactId>kafka_2.11</artifactId>
9   <version>2.0.0</version>
10 </dependency>
11 <dependency>
12   <groupId>org.apache.kafka</groupId>
13   <artifactId>kafka-streams</artifactId>
14   <version>2.0.0</version>
15 </dependency>

```

这些依赖项用于在项目中使用 Kafka 相关功能，版本号可根据实际情况调整，但需确保相互兼容。

3. 保存 `pom.xml` 文件后，在命令行执行 `mvn clean install` 命令，Maven 会自动下载并安装相关依赖到本地仓库。

四、案例复现

(一) 基础数据传输案例

1. 在项目的 `src/main/java/com/example` 目录下（根据项目的 groupId 和 artifactId 可能有所不同），创建一个名为 `MyStreamDemo.java` 的 Java 类。
2. 复制以下代码到 `MyStreamDemo.java` 文件中：

```
1 import org.apache.kafka.common.serialization.Serdes;
2 import org.apache.kafka.streams.KafkaStreams;
3 import org.apache.kafka.streams.StreamsBuilder;
4 import org.apache.kafka.streams.StreamsConfig;
5 import org.apache.kafka.streams.Topology;
6
7 import java.util.Properties;
8 import java.util.concurrent.CountDownLatch;
9
10 public class MyStreamDemo {
11     public static void main(String[] args) {
12         // 创建Properties对象，配置Kafka Streaming配置项
13         Properties prop = new Properties();
14         prop.put(StreamsConfig.APPLICATION_ID_CONFIG, "demo");
15             // 配置任务名称
16         prop.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092")
17             ;
18             // 配置Kafka主机IP和端口，根据实际情况修改
19         prop.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());    // 配置Key值类型
20         prop.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass()); // 配置Value值类型
21
22         // 创建流构造器
23         StreamsBuilder builder = new StreamsBuilder();
24
25         // 用构造好的builder将in数据写入到out
26         builder.stream("in").to("out");
27
28         // 构建Topology结构
29         Topology topology = builder.build();
30         final KafkaStreams kafkaStreams = new KafkaStreams(topology, prop)
31     ;
32
33         // 固定的启动方式
34         CountDownLatch latch = new CountDownLatch(1);
35         Runtime.getRuntime().addShutdownHook(new Thread("stream") {
36             @Override
37             public void run() {
38                 kafkaStreams.close();
39                 latch.countDown();
40             }
41         });
42
43         kafkaStreams.start();
44         try {
```

```
41         latch.await();
42     } catch (InterruptedException e) {
43         e.printStackTrace();
44     }
45
46     System.exit(0);
47 }
48 }
```

1. 在命令行中，进入 Kafka 安装目录，使用以下命令创建“in” 和 “out” topic（如果已经存在则无需重复创建）：

代码有问题：

```
Bash |
```

```
1 bin/kafka-topics.sh --create --topic in --partitions 1 --replication-factor
  1 --zookeeper localhost:2181
2 bin/kafka-topics.sh --create --topic out --partitions 1 --replication-factor
```

注意改为

从你执行创建 Kafka 主题的命令及报错信息来看，问题在于你使用的 Kafka 版本（3.9.0）中，`kafka-topics.sh` 命令的参数发生了变化，不再使用 `--zookeeper` 参数来指定 Zookeeper 地址。在这个版本中，应该使用 `--bootstrap-server` 参数来指定 Kafka 集群的地址（在单机环境下就是 Kafka broker 的地址）。

以下是正确的命令格式：

创建“in”主题

```
Bash |
```

```
1 bin/kafka-topics.sh --create --topic in --partitions 1 --replication-factor
  1 --bootstrap-server localhost:9092
```

创建“out”主题

```
Bash |
```

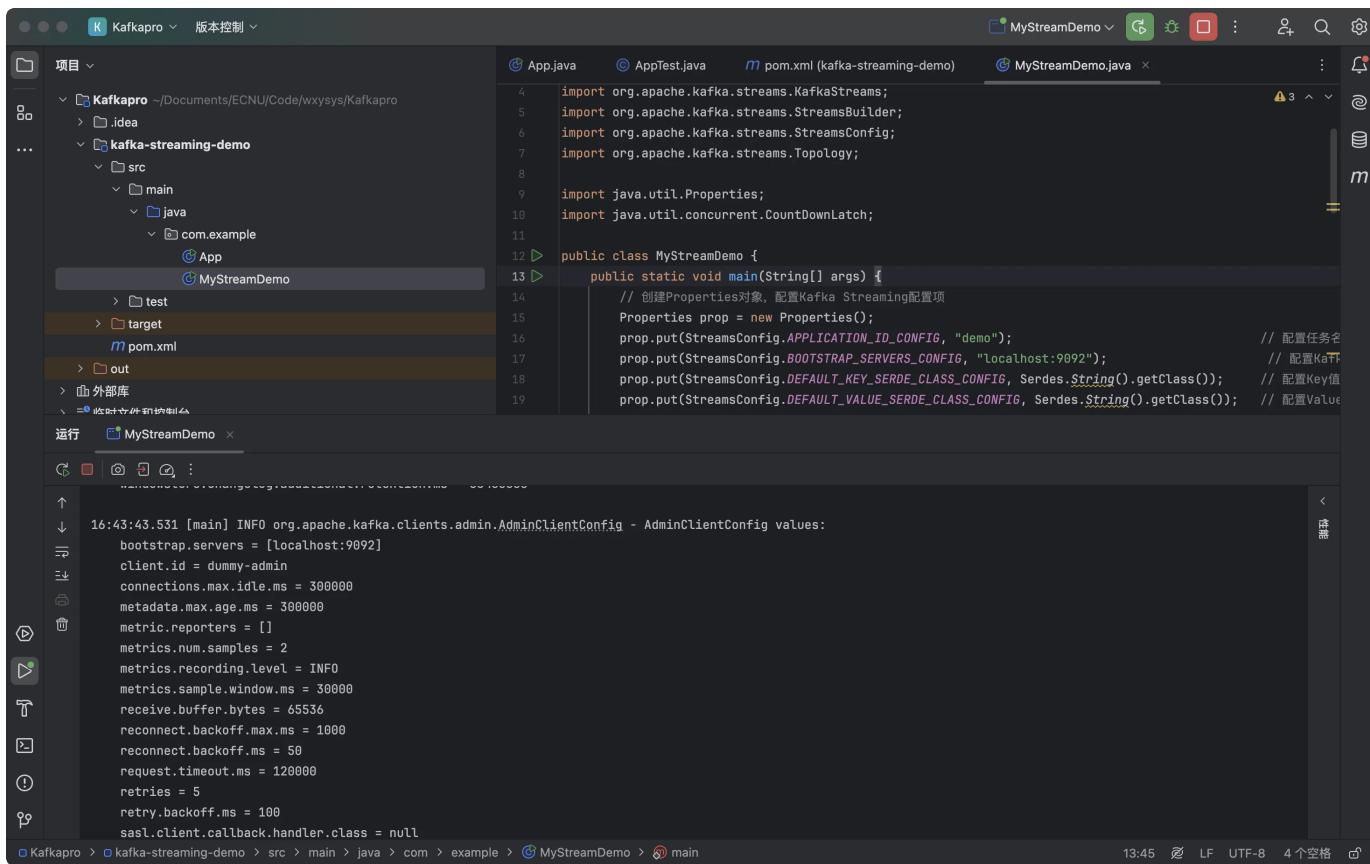
```
1 bin/kafka-topics.sh --create --topic out --partitions 1 --replication-factor
  1 --bootstrap-server localhost:9092
```

请注意，如果你在实际环境中 Kafka broker 监听的端口不是 9092，需要将上述命令中的端口号修改为实际的端口号。

在 Kafka 3.9.0 版本中，`--bootstrap-server` 参数用于与 Kafka 集群进行通信，而 Zookeeper 的连接信息是通过 Kafka 的配置文件（`config/server.properties`）中的 `zookeeper.connect` 参数来配置的，Kafka 客户端（包括 `kafka-topics.sh` 等工具）会根据配置文件中的信息自动连接到 Zookeeper。因此，在使用 Kafka 相关命令行工具时，需要按照新版本的参数规范来操作。

```
(base) mikaelnow@MikaelMacAir kafka_2.12-3.9.0 % bin/kafka-topics.sh --create --topic in --partitions 1 --replication-factor 1 --bootstrap-server localhost:9092
Created topic in.
(base) mikaelnow@MikaelMacAir kafka_2.12-3.9.0 % bin/kafka-topics.sh --create --topic out --partitions 1 --replication-factor 1 --bootstrap-server localhost:9092
Created topic out.
(base) mikaelnow@MikaelMacAir kafka_2.12-3.9.0 %
```

1. 运行 `MyStreamDemo` 类（在 IDE 中直接运行或在命令行中使用 `mvn exec:java -Dexec.mainClass="com.example.MyStreamDemo"` 命令运行，注意根据实际项目包名修改类路径）。



2. 程序启动后，打开新的命令行窗口，进入 Kafka 安装目录，使用以下命令向“in” topic 发送数据：

```
Bash |
```

```
1 bin/kafka-console-producer.sh --broker-list localhost:9092 --topic in
```

在生产者窗口输入一些数据，然后在另一个命令行窗口中，使用以下命令消费“out” topic 的数据，检查是否接收到从“in” topic 传输过来的数据：

1 bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic ou t --from-beginning

```

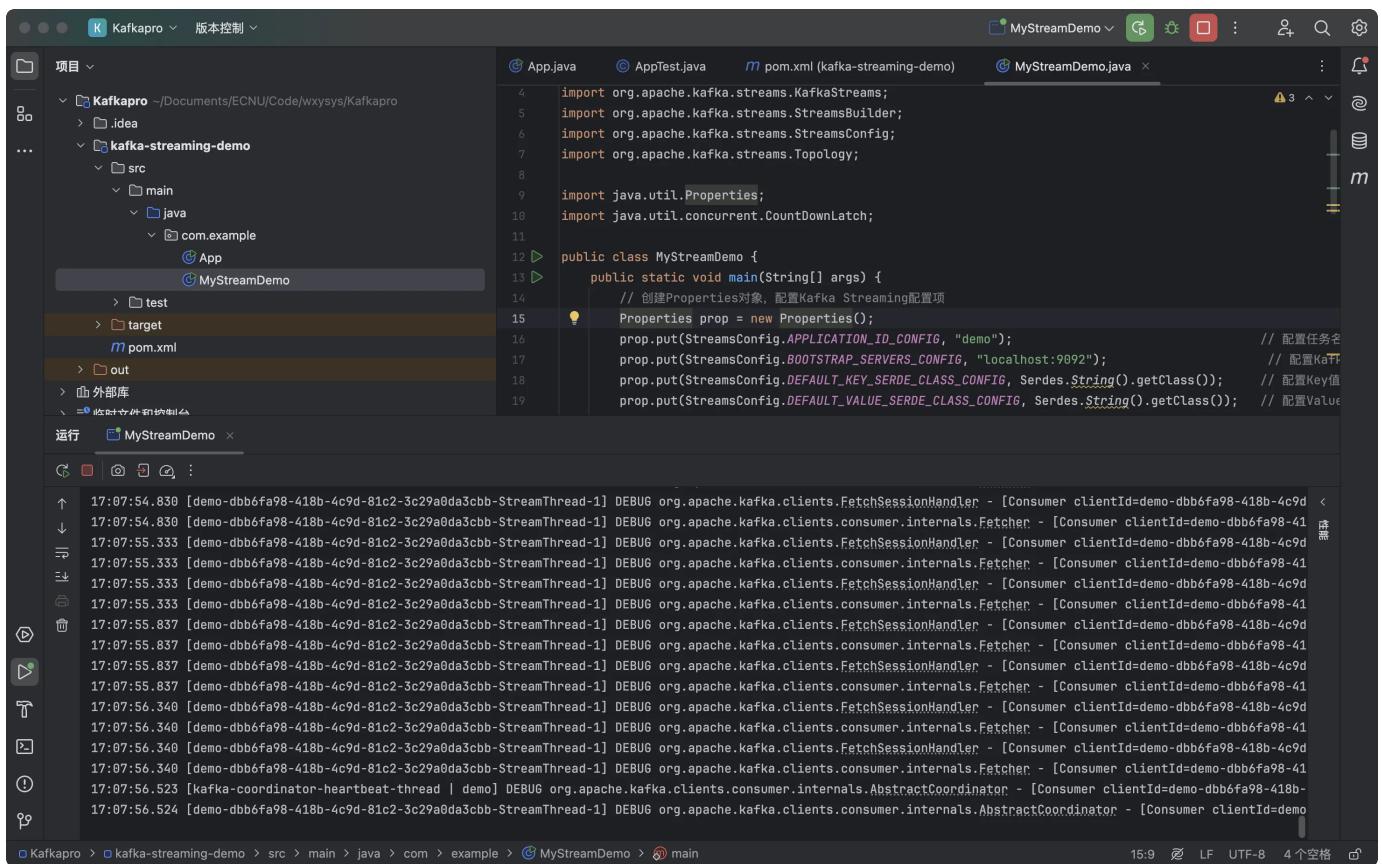
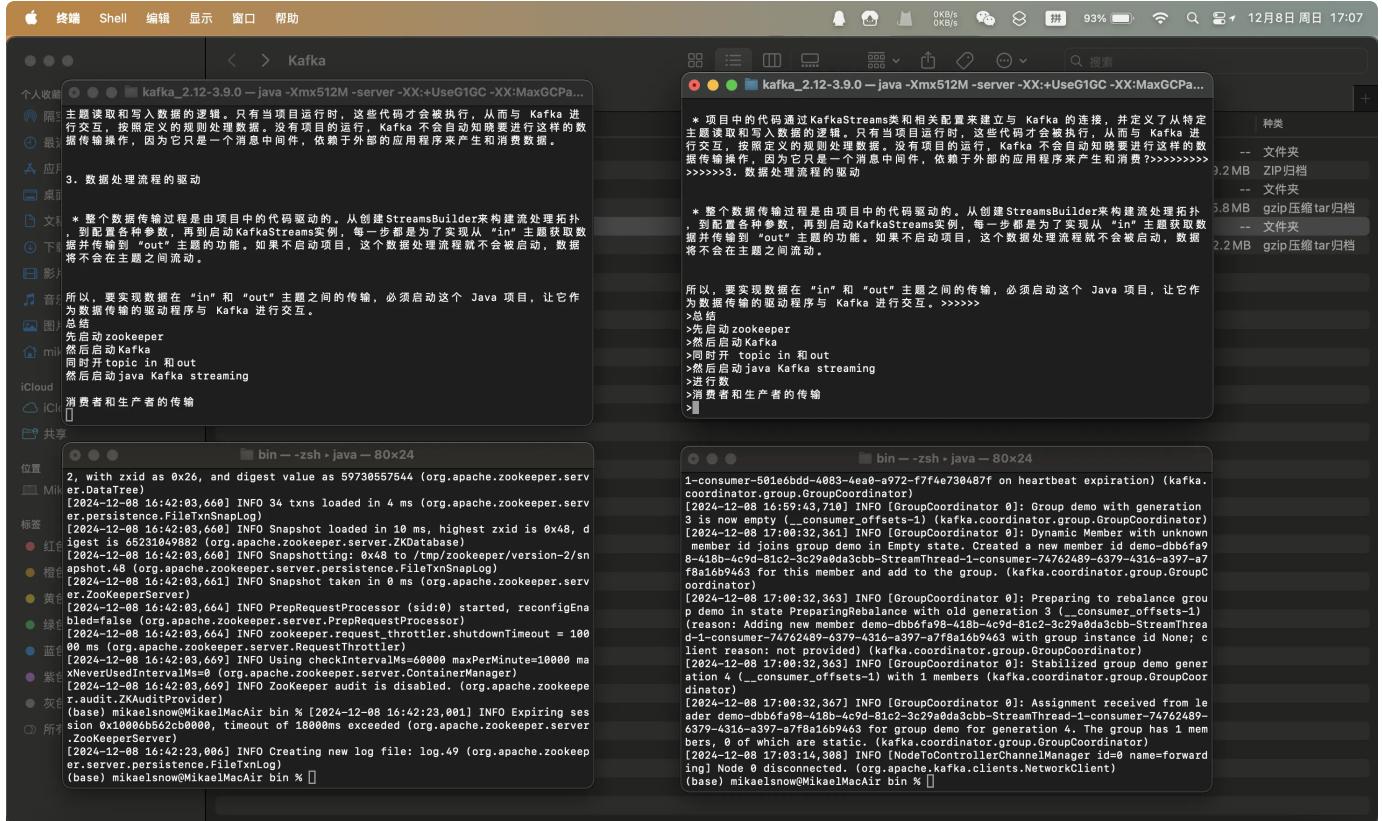
Last login: Sun Dec 8 16:46:00 on ttys003
(base) mikaelnow@MikaelMacAir ~ % cd /Users/mikaelnow/Documents/ECNU/Code/Kafka
(base) mikaelnow@MikaelMacAir kafka_2.12-3.9.0 % bin/kafka-console-consumer.sh
--bootstrap-server localhost:9092 --topic out --from-beginning
Hello
Completed Task!
Mikael admission-
Kafka zookeeper setting
successfully!
[ 9.9
  9.9-bin.zip
gz
tgz
]

(base) mikaelnow@MikaelMacAir kafka_2.12-3.9.0 % bin/kafka-topics.sh --create --topic in --partitions 1 --replication-factor 1 --bootstrap-server localhost:9092
Created topic in.
(base) mikaelnow@MikaelMacAir kafka_2.12-3.9.0 % bin/kafka-topics.sh --create --topic out --partitions 1 --replication-factor 1 --bootstrap-server localhost:9092
Created topic out.
(base) mikaelnow@MikaelMacAir kafka_2.12-3.9.0 % bin/kafka-console-producer.sh --broker-list localhost:9092 --topic in
>Hello
>Completed Task!
>Mikael admission-
>Kafka zookeeper setting
>successfully!
[ 9.9
  9.9-bin.zip
gz
tgz
]

[ 2, with zxid as 0x26, and digest value as 59730557544 (org.apache.zookeeper.server.DataTree)
[2024-12-08 16:42:03,660] INFO 34 txns loaded in 4 ms (org.apache.zookeeper.server.persistence.FileTxnSnapshotLog)
[2024-12-08 16:42:03,660] INFO Snapshot loaded in 10 ms, highest zxid is 0x48, digest is 65231049892 (org.apache.zookeeper.server.ZKDatabase)
[2024-12-08 16:42:03,666] INFO Snapshotting: 0x48 to /tmp/zookeeper/version-2/snAPSHOT_48 (org.apache.zookeeper.server.persistence.FileTxnSnapshotLog)
[2024-12-08 16:42:03,666] INFO Snapshot taken in 0 ms (org.apache.zookeeper.server.ZooKeeper)
[2024-12-08 16:42:03,661] INFO Snapshot taken in 0 ms (org.apache.zookeeper.server.PreRequestProcessor)
[2024-12-08 16:42:03,664] INFO PreRequestProcessor (sid:0) started, reconfigEnabled=false (org.apache.zookeeper.server.PreRequestProcessor)
[2024-12-08 16:42:03,664] INFO zookeeper.request.throttler.shutdownTimeout = 10000 ms (org.apache.zookeeper.server.RequestThrottler)
[2024-12-08 16:42:03,669] INFO Using checkIntervalMs=60000 maxPerMinute=10000 maxNeverUsedIntervalMs=0 (org.apache.zookeeper.server.ContainerManager)
[2024-12-08 16:42:03,669] INFO Zookeeper audit is disabled. (org.apache.zookeeper.audit.ZKAuditProvider)
[base] mikaelnow@MikaelMacAir bin % [2024-12-08 16:42:23,001] INFO Expiring session 0x10006b562cb0000, timeout of 18000ms exceeded (org.apache.zookeeper.server.ZooKeeperServer)
[2024-12-08 16:42:23,006] INFO Creating new log file: log.49 (org.apache.zookeeper.server.persistence.FileTxnLog)

[ 12-08 16:52:51,562] INFO [GroupCoordinator @0]: Assignment received from leader demo-3efc5c87-592e-4eac-b1b8-384016b94cc8-StreamThread-1-consumer-501e6bdd-4083-4ea0-a972-f7f4e739487f for group demo for generation 2. The group has 1 members, 0 of which are static. (kafka.coordinator.group.GroupCoordinator)
(base) mikaelnow@MikaelMacAir bin % [2024-12-08 16:54:06,834] INFO [GroupCoordinator @0]: Dynamic member with unknown member id joins group console-consumer-50955 in Empty state. Created a new member id console-consumer-6d936a0e-bc28-4b4c-b33e-6a967bca9e98 and request the member to rejoin with this id. (kafka.coordinator.group.GroupCoordinator)
[2024-12-08 16:54:06,836] INFO [GroupCoordinator @0]: Preparing to rebalance group console-consumer-50955 in state PreparingRebalance with old generation 0 (_consumers_offsets-5) (reason: Adding new member console-consumer-6d936a0e-bc28-4b4c-b33e-6a967bca9e98 with group instance id None; client reason: need to re-join with the given member-id: console-consumer-6d936a0e-bc28-4b4c-b33e-6a967bca9e98) (kafka.coordinator.group.GroupCoordinator)
[2024-12-08 16:54:06,837] INFO [GroupCoordinator @0]: Stabilized group console-consumer-50955 generation 1 (_consumers_offsets-5) with 1 members (kafka.coordinator.group.GroupCoordinator)
[2024-12-08 16:54:06,841] INFO [GroupCoordinator @0]: Assignment received from leader console-consumer-6d936a0e-bc28-4b4c-b33e-6a967bca9e98 for group console-consumer-50955 for generation 1. The group has 1 members, 0 of which are static. (kafka.coordinator.group.GroupCoordinator)

```



总结

如果不启动你提供的这个 Java 项目，是不能进行数据从“in”主题到“out”主题的传输的，原因如下：

1. 项目功能的主动性

- 该 Java 项目是专门设计用于从 Kafka 的“in”主题读取数据，并将其传输到“out”主题的。它通过连接到 Kafka 集群（配置为 `localhost:9092`），利用 Kafka Streams API 构建了一个流处理拓扑来实现这个功能。如果项目不启动，就没有任何机制来主动触发数据的读取和传输操作。

2. 与 Kafka 交互的实现

- 项目中的代码通过 `KafkaStreams` 类和相关配置来建立与 Kafka 的连接，并定义了从特定主题读取和写入数据的逻辑。只有当项目运行时，这些代码才会被执行，从而与 Kafka 进行交互，按照定义的规则处理数据。没有项目的运行，Kafka 不会自动知晓要进行这样的数据传输操作，因为它只是一个消息中间件，依赖于外部的应用程序来产生和消费数据。

3. 数据处理流程的驱动

- 整个数据传输过程是由项目中的代码驱动的。从创建 `StreamsBuilder` 来构建流处理拓扑，到配置各种参数，再到启动 `KafkaStreams` 实例，每一步都是为了实现从“in”主题获取数据并传输到“out”主题的功能。如果不启动项目，这个数据处理流程就不会被启动，数据将不会在主题之间流动。

所以，要实现数据在“in”和“out”主题之间的传输，必须启动这个 Java 项目，让它作为数据传输的驱动程序与 Kafka 进行交互。

(二) 计算数字总和案例

- 在 `src/main/java/com/example` 目录下创建一个名为 `SumStreamDemo.java` 的 Java 类。
- 复制以下代码到 `SumStreamDemo.java` 文件中：

```

1 import org.apache.kafka.clients.consumer.ConsumerConfig;
2 import org.apache.kafka.common.serialization.Serdes;
3 import org.apache.kafka.streams.*;
4
5 import java.util.Properties;
6 import java.util.concurrent.CountDownLatch;
7
8 public class SumStreamDemo {
9     public static void main(String[] args) {
10         Properties prop = new Properties();
11         prop.put(StreamsConfig.APPLICATION_ID_CONFIG, "sum");
12         prop.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092")
13             ;
14         prop.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG, 3000);
15         prop.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false");
16         prop.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
17         prop.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
18         prop.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());
19
20         StreamsBuilder builder = new StreamsBuilder();
21         builder.stream("sumIn") // 从sumin主题获取数据
22             .map((key, value) -> new KeyValue<>("sum", value.toString()))
23             // 因为上面指定的是String类型的value值, 所以将数值转为String类型
24             .groupByKey()
25             .reduce((value1, value2) -> { // 数值相加功能, 顺带输出中间计算的结果, 逻辑和spark的reduce功能完全相同
26                 int sum = Integer.valueOf(value1) + Integer.valueOf(value2);
27                 System.out.println(Integer.valueOf(value1) + "+" + Integer.valueOf(value2) + " = " + sum);
28                 return Integer.toString(sum);
29             })
30             .toStream().to("sumOut"); // 重新转为stream后输出到sumout主题
31
32         Topology topology = builder.build();
33         final KafkaStreams kafkaStreams = new KafkaStreams(topology, prop)
34             ;
35
36         CountDownLatch latch = new CountDownLatch(1);
37         Runtime.getRuntime().addShutdownHook(new Thread("stream") {
38             @Override
39             public void run() {

```

```
37         kafkaStreams.close();
38         latch.countDown();
39     }
40 }
41
42     kafkaStreams.start();
43     try {
44         latch.await();
45     } catch (InterruptedException e) {
46         e.printStackTrace();
47     }
48
49     System.exit(0);
50 }
51 }
```

1. 在命令行中创建“sumIn”和“sumOut”topic（如果不存在）：

```
1 bin/kafka-topics.sh --create --topic sumIn --partitions 1 --replication-factor 1 --zookeeper localhost:2181
2 bin/kafka-topics.sh --create --topic sumOut --partitions 1 --replication-factor 1 --zookeeper localhost:2181
```

注意修改新版本的

创建 **sumIn** 主题

```
1 bin/kafka-topics.sh --create --topic sumIn --partitions 1 --replication-factor 1 --bootstrap-server localhost:9092
```

创建 **sumOut** 主题

```
1 bin/kafka-topics.sh --create --topic sumOut --partitions 1 --replication-factor 1 --bootstrap-server localhost:9092
```

请注意，如果你的 Kafka broker 监听的端口不是 9092，需要将上述命令中的端口号修改为实际的端口号。同时，确保 Kafka 已经正确安装、配置并启动，并且你具有足够的权限来执行这些命令。如果在执行命令过程中遇到权限问题，可以参考前面关于修改目录权限的内容，或者根据实际情况调整用户权限以确保能够成功创建主题。

1. 运行 `SumStreamDemo` 类。
2. 分别在不同命令行窗口启动“sumIn”和“sumOut”主题的生产者和消费者：
 - 向“sumIn”输入数字数据（如 `1`、`2`、`3` 等）：

```
▼ Bash |  
1 bin/kafka-console-producer.sh --broker-list localhost:9092 --topic sumIn
```

- 观察“sumOut”输出的结果，检查是否正确计算总和，同时可以测试修改 `StreamsConfig.APPLICATION_ID_CONFIG` 重新计算总和的功能（修改后需重新创建 topic 并运行程序）。

(三) WordCount 案例

1. 在 `src/main/java/com/example` 目录下创建一个名为 `WordCount.java` 的 Java 类。
2. 复制以下代码到 `WordCount.java` 文件中：

```

1 import org.apache.kafka.clients.consumer.ConsumerConfig;
2 import org.apache.kafka.common.serialization.Serdes;
3 import org.apache.kafka.streams.*;
4
5 import java.util.Arrays;
6 import java.util.Properties;
7 import java.util.concurrent.CountDownLatch;
8
9 - public class WordCount {
10 -     public static void main(String[] args) {
11         Properties prop = new Properties();
12         prop.put(StreamsConfig.APPLICATION_ID_CONFIG, "wordCount");
13         prop.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092")
14         ;
15         prop.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG, 3000);
16         prop.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false");
17         prop.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "latest");
18         prop.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
19         prop.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());
20
21         StreamsBuilder builder = new StreamsBuilder();
22         builder.stream("wordin")      // 从wordin导入数据
23             .flatMapValues((value) -> Arrays.asList(value.toString().split("\\\\s+")))    // 按照空白字符切割为多个单词
24             .map((key, value) -> new KeyValue<>(value, "1"))      // 转换
25             // 为 (单词, 1) 的键值对形式
26             .groupByKey()      // 根据单词分组
27             .count()          // 计算各分组value的数量
28             .toStream()
29             .map(((key, value) -> new KeyValue<>(key, key + " : " + value.toString())))
30             .to("wordout");    // 输出到wordout
31
32         Topology topology = builder.build();
33         final KafkaStreams kafkaStreams = new KafkaStreams(topology, prop)
34         ;
35
36         CountDownLatch latch = new CountDownLatch(1);
37         Runtime.getRuntime().addShutdownHook(new Thread("stream") {
38             @Override
39             public void run() {
40                 kafkaStreams.close();
41                 latch.countDown();
42             }
43         });
44     }
45 }

```

```
39         }
40     });
41
42     kafkaStreams.start();
43     try {
44         latch.await();
45     } catch (InterruptedException e) {
46         e.printStackTrace();
47     }
48     System.exit(0);
49 }
50 }
```

1. 在命令行中创建“wordin”和“wordout”topic（如果不存在）：

```
Bash |
```

```
1 bin/kafka-topics.sh --create --topic wordin --partitions 1 --replication-factor 1 --zookeeper localhost:2181
2 bin/kafka-topics.sh --create --topic wordout --partitions 1 --replication-factor 1 --zookeeper localhost:2181
```

注意

创建“wordin”主题

```
Bash |
```

```
1 bin/kafka-topics.sh --create --topic wordin --partitions 1 --replication-factor 1 --bootstrap-server localhost:9092
```

创建“wordout”主题

```
Bash |
```

```
1 bin/kafka-topics.sh --create --topic wordout --partitions 1 --replication-factor 1 --bootstrap-server localhost:9092
```

请注意，如果你的 Kafka broker 实际监听的端口不是 9092，要把上述命令中的端口号替换为实际的端口号，这样才能确保正确地创建相应的 Kafka 主题。

1. 运行 WordCount 类。

2. 使用生产者向“wordin”输入文本数据（如 `hello world`、`hello kafka stream` 等）：

```
1 bin/kafka-console-producer.sh --broker-list localhost:9092 --topic wordin
```

1. 通过消费者观察“wordout”输出的单词计数结果是否正确：

```
1 bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic wordout --from-beginning
```

(四) 窗口操作案例

1. 跳跃时间窗口 (Hopping time window)

1. 在 `src/main/java/com/example` 目录下创建一个名为 `WindowDemo1.java` 的 Java 类。
2. 复制以下代码到 `WindowDemo1.java` 文件中：

```

1 import org.apache.kafka.streams.KafkaStreams;
2 import org.apache.kafka.streams.StreamsBuilder;
3 import org.apache.kafka.streams.StreamsConfig;
4 import org.apache.kafka.streams.Topology;
5 import org.apache.kafka.streams.kstream.TimeWindowedKStream;
6 import org.apache.kafka.streams.kstream.Windowed;
7 import org.apache.kafka.common.serialization.Serdes;
8 import java.time.Duration;
9 import java.util.Properties;
10 import java.util.concurrent.CountDownLatch;
11
12 public class WindowDemo1 {
13     public static void main(String[] args) {
14         // 配置Kafka Streaming
15         Properties prop = new Properties();
16         prop.put(StreamsConfig.APPLICATION_ID_CONFIG, "window_demo1");
17         prop.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092")
18         ;
19         prop.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG, 3000);
20         prop.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false");
21         prop.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
22         prop.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
23         prop.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());
24
25         StreamsBuilder builder = new StreamsBuilder();
26         TimeWindowedKStream<String, String> windowdemo = builder.stream("windowdemo")
27             .flatMapValues(value -> {
28                 String[] split = value.toString().split("\\s+");
29                 return Arrays.asList(split);
30             })
31             .map((key, value) -> new KeyValue<>(value, "1"))
32             .groupByKey()
33                 // 使用TimeWindows定义时间窗口, of方法内传入毫秒值表示的size, 得
34                 // 到结果调用advanceBy方法限定创建窗口的时间间隔
35                 .windowedBy(TimeWindows.of(Duration.ofSeconds(5)).advanceBy(
36                     Duration.ofSeconds(3)))
37                 .count();
38
39         windowdemo.toStream().map((Windowed<String> windowed, Long count)
40             -> {
41                 String windowStart = String.valueOf(windowed.window().start())
42             ;
43             }
44         );
45     }
46 }

```

```

38         String>windowEnd = String.valueOf(windowed.window().end());
39         return new KeyValue<>(windowed.key(), windowed.key() + " : " +
40             count + " [ " + windowStart + " - " + windowEnd + " ]");
41     }).to("windowdemo-out");
42
43     Topology topology = builder.build();
44     final KafkaStreams kafkaStreams = new KafkaStreams(topology, prop);
45 ;
46     CountDownLatch latch = new CountDownLatch(1);
47     Runtime.getRuntime().addShutdownHook(new Thread("stream") {
48         @Override
49         public void run() {
50             kafkaStreams.close();
51             latch.countDown();
52         }
53     });
54
55     kafkaStreams.start();
56     try {
57         latch.await();
58     } catch (InterruptedException e) {
59         e.printStackTrace();
60     }
61     System.exit(0);
62 }
63 }
```

- 在命令行中创建“windowdemo” 和“windowdemo-out” topic (如果不存在) :

```

Bash | ▾

1 bin/kafka-topics.sh --create --topic windowdemo --partitions 1 --replication-factor 1 --zookeeper localhost:2181
2 bin/kafka-topics.sh --create --topic windowdemo-out --partitions 1 --replication-factor 1 --zookeeper localhost:2181
```

- 运行 `WindowDemo1` 类。
- 使用生产者向“windowdemo” 输入数据 (例如一些单词或数字, 用空格隔开) :

```

Bash | ▾

1 bin/kafka-console-producer.sh --broker-list localhost:9092 --topic windowdemo
```

1. 通过消费者观察“windowdemo-out”输出的结果，检查是否按照跳跃时间窗口的规则进行了计数和窗口划分：

```
Bash |  
1 bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic windowdemo-out --from-beginning
```

2. 滚动时间窗口 (Tumbling time window)

1. 在 `src/main/java/com/example` 目录下创建一个名为 `WindowDemo2.java` 的 Java 类。
2. 复制以下代码到 `WindowDemo2.java` 文件中：

1. 在命令行中创建“windowdemo-out-rolling”topic（如果不存在）：

```
Bash |  
1 bin/kafka-topics.sh --create --topic windowdemo-out-rolling --partitions 1  
--replication-factor 1 --zookeeper localhost:2181
```

1. 运行 `WindowDemo2` 类。
2. 同样使用生产者向“windowdemo”输入数据（与跳跃时间窗口测试时相同类型的数据即可）。
3. 通过消费者观察“windowdemo-out-rolling”输出的结果，验证滚动时间窗口的功能，即窗口之间无缝隙不重叠，且每个数据只属于一个窗口。

3. 会话窗口 (Session window)

1. 在 `src/main/java/com/example` 目录下创建一个名为 `WindowDemo3.java` 的 Java 类。
2. 复制以下代码到 `WindowDemo3.java` 文件中：

```

1 import org.apache.kafka.streams.KafkaStreams;
2 import org.apache.kafka.streams.StreamsBuilder;
3 import org.apache.kafka.streams.StreamsConfig;
4 import org.apache.kafka.streams.Topology;
5 import org.apache.kafka.streams.kstream.SessionWindows;
6 import org.apache.kafka.streams.kstream.Windowed;
7 import org.apache.kafka.common.serialization.Serdes;
8 import java.time.Duration;
9 import java.util.Properties;
10 import java.util.concurrent.CountDownLatch;
11
12 public class WindowDemo3 {
13     public static void main(String[] args) {
14         // 配置Kafka Streaming
15         Properties prop = new Properties();
16         prop.put(StreamsConfig.APPLICATION_ID_CONFIG, "window_demo3");
17         prop.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092")
18         ;
19         prop.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG, 3000);
20         prop.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false");
21         prop.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
22         prop.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
23         prop.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());
24
25         StreamsBuilder builder = new StreamsBuilder();
26         builder.stream("windowdemo")
27             .flatMapValues(value -> {
28                 String[] split = value.toString().split("\\s+");
29                 return Arrays.asList(split);
30             })
31             .map((key, value) -> new KeyValue<>(value, "1"))
32             .groupByKey()
33             // 使用SessionWindows定义会话窗口，传入毫秒值表示的gap时间间隔
34             .windowedBy(SessionWindows.with(Duration.ofSeconds(5)))
35             .count();
36
37         builder.table("windowdemo")
38             .toStream()
39             .map((Windowed<String> windowed, Long count) -> {
40                 String windowStart = String.valueOf(windowed.window().start());
41                 String windowEnd = String.valueOf(windowed.window().end());
42             });

```

```

41             return new KeyValue<>(windowed.key(), windowed.key() +
42         " : " + count + " [ " + windowStart + " - " + windowEnd + " ]");
43     }).to("windowdemo-out-session");
44
45     Topology topology = builder.build();
46     final KafkaStreams kafkaStreams = new KafkaStreams(topology, prop)
47 ;
48
49     CountDownLatch latch = new CountDownLatch(1);
50     Runtime.getRuntime().addShutdownHook(new Thread("stream") {
51         @Override
52         public void run() {
53             kafkaStreams.close();
54             latch.countDown();
55         }
56     });
57
58     kafkaStreams.start();
59     try {
60         latch.await();
61     } catch (InterruptedException e) {
62         e.printStackTrace();
63     }
64     System.exit(0);
65 }
```

1. 在命令行中创建“windowdemo-out-session” topic (如果不存在) :

```
Bash |
```

```
1 bin/kafka-topics.sh --create --topic windowdemo-out-session --partitions 1
--replication-factor 1 --zookeeper localhost:2181
```

1. 运行 `WindowDemo3` 类。
2. 向“windowdemo”输入数据，注意输入数据的时间间隔，以便测试会话窗口的功能。例如，快速输入几个数据，然后停顿超过 5 秒（根据设置的 gap 时间间隔）再输入数据，观察“windowdemo-out-session”输出的结果，查看会话窗口是否根据数据输入时间正确划分和计算。

4. 滑动时间窗口 (Sliding time window)

1. 在 `src/main/java/com/example` 目录下创建一个名为 `WindowDemo4.java` 的 Java 类。
2. 复制以下代码到 `WindowDemo4.java` 文件中：

```

1 import org.apache.kafka.streams.KafkaStreams;
2 import org.apache.kafka.streams.StreamsBuilder;
3 import org.apache.kafka.streams.StreamsConfig;
4 import org.apache.kafka.streams.Topology;
5 import org.apache.kafka.streams.kstream.JoinWindows;
6 import org.apache.kafka.streams.kstream.KStream;
7 import org.apache.kafka.streams.kstream.Windowed;
8 import org.apache.kafka.common.serialization.Serdes;
9 import java.time.Duration;
10 import java.util.Properties;
11 import java.util.concurrent.CountDownLatch;
12
13 public class WindowDemo4 {
14     public static void main(String[] args) {
15         // 配置Kafka Streaming
16         Properties prop = new Properties();
17         prop.put(StreamsConfig.APPLICATION_ID_CONFIG, "window_demo4");
18         prop.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092")
19         ;
20         prop.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG, 3000);
21         prop.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false");
22         prop.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
23         prop.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
24         prop.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());
25
26         StreamsBuilder builder = new StreamsBuilder();
27         KStream<String, String> stream1 = builder.stream("stream1");
28         KStream<String, String> stream2 = builder.stream("stream2");
29
30         stream1.join(stream2, (value1, value2) -> value1 + " - " + value2,
31                     JoinWindows.withTimeDifferenceAndGrace(Duration.ofSeconds(5),
32                         Duration.ofSeconds(1)))
33             .to("joined-stream");
34
35         Topology topology = builder.build();
36         final KafkaStreams kafkaStreams = new KafkaStreams(topology, prop)
37         ;
38
39         CountDownLatch latch = new CountDownLatch(1);
40         Runtime.getRuntime().addShutdownHook(new Thread("stream") {
41             @Override
42             public void run() {
43                 kafkaStreams.close();
44             }
45         });
46     }
47 }

```

```
41             latch.countDown();
42         }
43     });
44
45     kafkaStreams.start();
46     try {
47         latch.await();
48     } catch (InterruptedException e) {
49         e.printStackTrace();
50     }
51
52     System.exit(0);
53 }
54 }
```

1. 在命令行中创建“stream1”、“stream2” 和“joined-stream” topic (如果不存在) :

```
Bash |
```

```
1 bin/kafka-topics.sh --create --topic stream1 --partitions 1 --replication-factor 1 --zookeeper localhost:2181
2 bin/kafka-topics.sh --create --topic stream2 --partitions 1 --replication-factor 1 --zookeeper localhost:2181
3 bin/kafka-topics.sh --create --topic joined-stream --partitions 1 --replication-factor 1 --zookeeper localhost:2181
```

1. 运行 `WindowDemo4` 类。
2. 分别在不同的命令行窗口启动“stream1”和“stream2”的生产者，向“stream1”和“stream2”输入数据，观察“joined-stream”输出的结果，验证滑动时间窗口在 join 操作中的功能，即根据设置的时间间隔和容忍时间确定哪些数据在同一窗口内进行 join 操作。