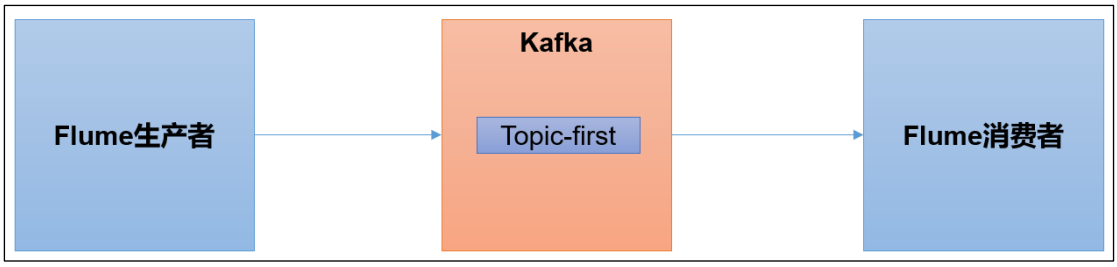
尚硅谷大数据技术之 Kafka（外部系统集成）

（作者：尚硅谷研究院）

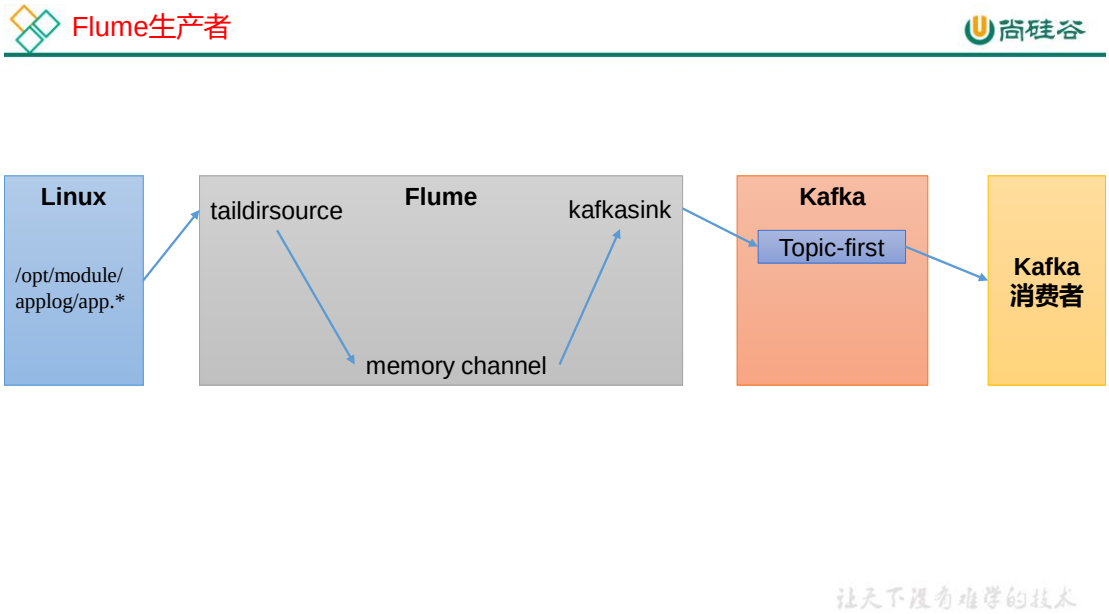
版本：V3.3

第 1 章 集成 Flume

Flume 是一个在大数据开发中非常常用的组件。可以用于 Kafka 的生产者，也可以用于 Flume 的消费者。



1.1 Flume 生产者



（1）启动 kafka 集群

```
[atguigu@hadoop102 ~]$ zk.sh start
[atguigu@hadoop102 ~]$ kf.sh start
```

（2）启动 kafka 消费者

```
[atguigu@hadoop103 kafka]$ bin/kafka-console-consumer.sh --
```

```
bootstrap-server hadoop102:9092 --topic first
```

（3）Flume 安装步骤

在 hadoop102 主机上安装 Flume。

详见：尚硅谷大数据技术之 Flume

尚硅谷大数据技术之Flume（V3.0）.c

（4）配置 Flume

在 hadoop102 节点的 Flume 的 job 目录下创建 file_to_kafka.conf

```
[atguigu@hadoop102 flume]$ mkdir jobs
[atguigu@hadoop102 flume]$ vim jobs/file_to_kafka.conf
```

配置文件内容如下

```
# 1 组件定义
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# 2 配置 source
a1.sources.r1.type = TAILDIR
a1.sources.r1.filegroups = f1
a1.sources.r1.filegroups.f1 = /opt/module/applog/app.*
a1.sources.r1.positionFile =
/opt/module/flume/taildir_position.json

# 3 配置 channel
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

# 4 配置 sink
a1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink
a1.sinks.k1.kafka.bootstrap.servers =
hadoop102:9092,hadoop103:9092,hadoop104:9092
a1.sinks.k1.kafka.topic = first
a1.sinks.k1.kafka.flumeBatchSize = 20
a1.sinks.k1.kafka.producer.acks = 1
a1.sinks.k1.kafka.producer.linger.ms = 1

# 5 拼接组件
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

（5）启动 Flume

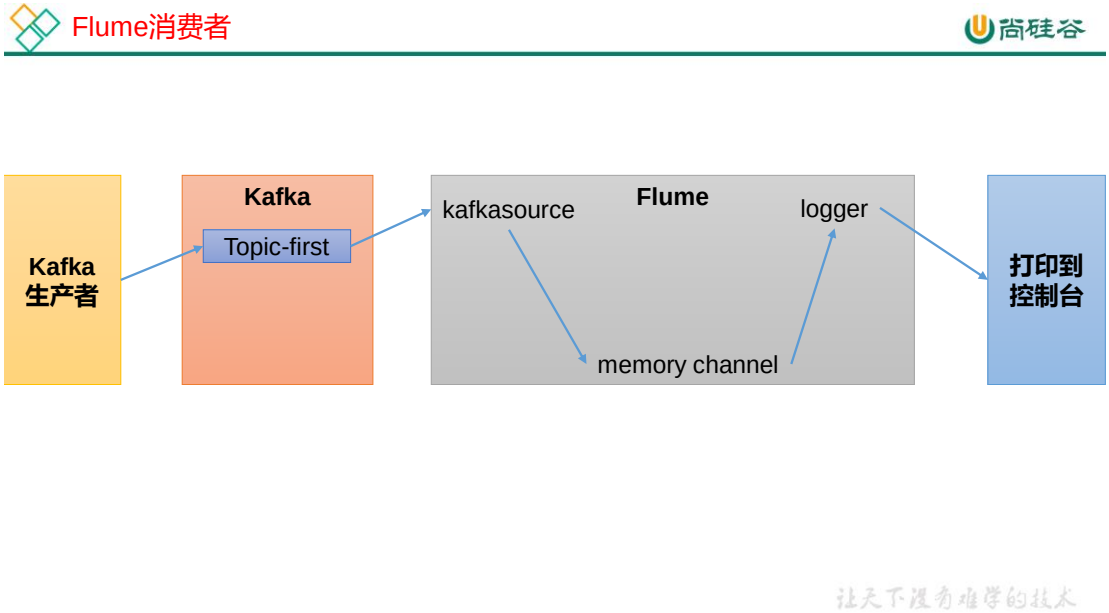
```
[atguigu@hadoop102 flume]$ bin/flume-ng agent -c conf/ -n a1 -f
jobs/file_to_kafka.conf &
```

（6）向/opt/module/applog/app.log 里追加数据，查看 kafka 消费者消费情况

```
[atguigu@hadoop102 module]$ mkdir applog
[atguigu@hadoop102 applog]$ echo hello >> /opt/module/applog/app.log
```

（7）观察 kafka 消费者，能够看到消费的 hello 数据

1.2 Flume 消费者



（1）配置 Flume

在 hadoop102 节点的 Flume 的/opt/module/flume/jobs 目录下创建 kafka_to_file.conf

```
[atguigu@hadoop102 jobs]$ vim kafka_to_file.conf
```

配置文件内容如下

```
# 1 组件定义
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# 2 配置 source
a1.sources.r1.type = org.apache.flume.source.kafka.KafkaSource
a1.sources.r1.batchSize = 50
a1.sources.r1.batchDurationMillis = 200
a1.sources.r1.kafka.bootstrap.servers = hadoop102:9092
a1.sources.r1.kafka.topics = first
a1.sources.r1.kafka.consumer.group.id = custom.g.id

# 3 配置 channel
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

# 4 配置 sink
a1.sinks.k1.type = logger
```

```
# 5 拼接组件
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

（2）启动 Flume

```
[atguigu@hadoop102 flume]$ bin/flume-ng agent -c conf/ -n a1 -f jobs/kafka_to_file.conf -Dflume.root.logger=INFO,console
```

（3）启动 kafka 生产者

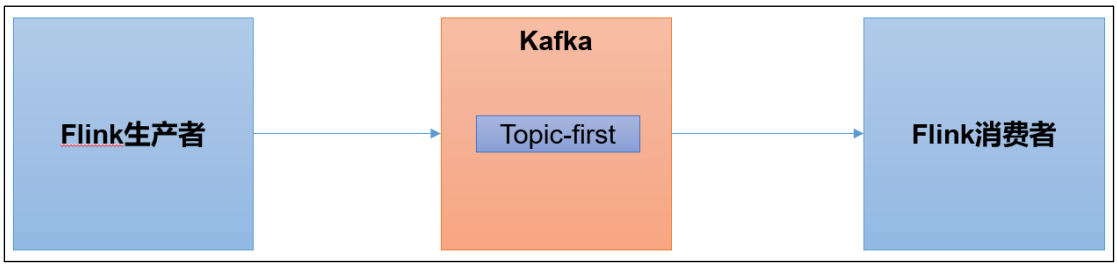
```
[atguigu@hadoop103 kafka]$ bin/kafka-console-producer.sh --bootstrap-server hadoop102:9092 --topic first
```

并输入数据，例如：hello world

（4）观察控制台输出的日志

第 2 章 集成 Flink

Flink 是一个在大数据开发中非常常用的组件。可以用于 Kafka 的生产者，也可以用于 Flink 的消费者。



1）Flink 环境准备

（1）创建一个 maven 项目 flink-kafka

（2）添加配置文件

```
<dependencies>
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-java</artifactId>
        <version>1.13.0</version>
    </dependency>

    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-streaming-java_2.12</artifactId>
        <version>1.13.0</version>
    </dependency>

    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-clients_2.12</artifactId>
        <version>1.13.0</version>
    </dependency>
```

```
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-connector-kafka_2.12</artifactId>
        <version>1.13.0</version>
    </dependency>
</dependencies>
```

（3）将 log4j.properties 文件添加到 resources 里面，就能更改打印日志的级别为 error

```
log4j.rootLogger=error, stdout,R
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd
HH:mm:ss,SSS}  %5p --- [%50t]  %-80c(line:%5L)  :  %m%n

log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=../log/agent.log
log4j.appender.R.MaxFileSize=1024KB
log4j.appender.R.MaxBackupIndex=1

log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%d{yyyy-MM-dd
HH:mm:ss,SSS}  %5p --- [%50t]  %-80c(line:%6L)  :  %m%n
```

（4）在 java 文件夹下创建包名为 com.atguigu.flink

2.1 Flink 生产者

（1）在 com.atguigu.flink 包下创建 java 类：FlinkKafkaProducer1

```
package com.atguigu.flink;

import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;

import java.util.ArrayList;
import java.util.Properties;

public class FlinkKafkaProducer1 {

    public static void main(String[] args) throws Exception {
        // 0 初始化 flink 环境
        StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
        env.setParallelism(3);

        // 1 读取集合中数据
        ArrayList<String> wordsList = new ArrayList<>();
        wordsList.add("hello");
        wordsList.add("world");
```

```
        DataStream<String> stream = env.fromCollection(wordsList);

        // 2 kafka 生产者配置信息
        Properties properties = new Properties();
        properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "hadoop102:9092");

        // 3  创建 kafka 生产者
        FlinkKafkaProducer<String> kafkaProducer = new FlinkKafkaProducer<>(
                "first",
                new SimpleStringSchema(),
                properties
        );

        // 4  生产者和 flink 流关联
        stream.addSink(kafkaProducer);

        // 5  执行
        env.execute();
    }
}
```

（2）启动 Kafka 消费者

```
[atguigu@hadoop104 kafka]$ bin/kafka-console-consumer.sh --bootstrap-server hadoop102:9092 --topic first
```

（3）执行 FlinkKafkaProducer1 程序，观察 kafka 消费者控制台情况

2.2 Flink 消费者

（1）在 com.atguigu.flink 包下创建 java 类：FlinkKafkaConsumer1

```
package com.atguigu.flink;

import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.StringDeserializer;

import java.util.Properties;

public class FlinkKafkaConsumer1 {
    public static void main(String[] args) throws Exception {

        // 0  初始化 flink 环境
        StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
        env.setParallelism(3);
```

```
            // 1 kafka 消费者配置信息
            Properties properties = new Properties();
            properties.setProperty(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
"hadoop102:9092");

            // 2  创建 kafka 消费者
            FlinkKafkaConsumer<String> kafkaConsumer = new FlinkKafkaConsumer<>(
                        "first",
                        new SimpleStringSchema(),
                        properties
            );

            // 3  消费者和 flink 流关联
            env.addSource(kafkaConsumer).print();

            // 4  执行
            env.execute();
      }
}
```

（2）启动 FlinkKafkaConsumer1 消费者

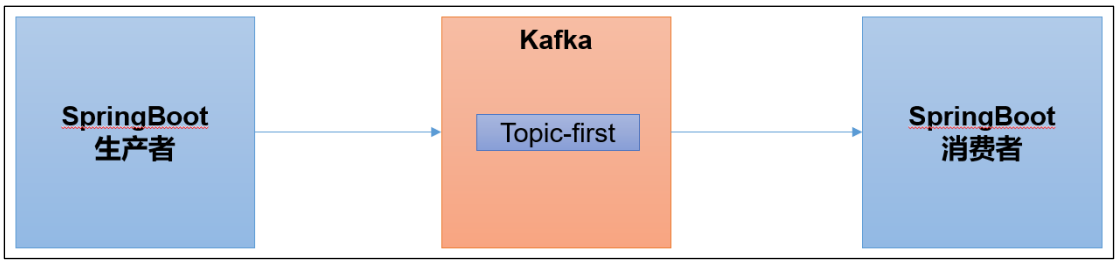
（3）启动 kafka 生产者

```
[atguigu@hadoop103 kafka]$ bin/kafka-console-producer.sh --bootstrap-server hadoop102:9092 --topic first
```

（4）观察 IDEA 控制台数据打印

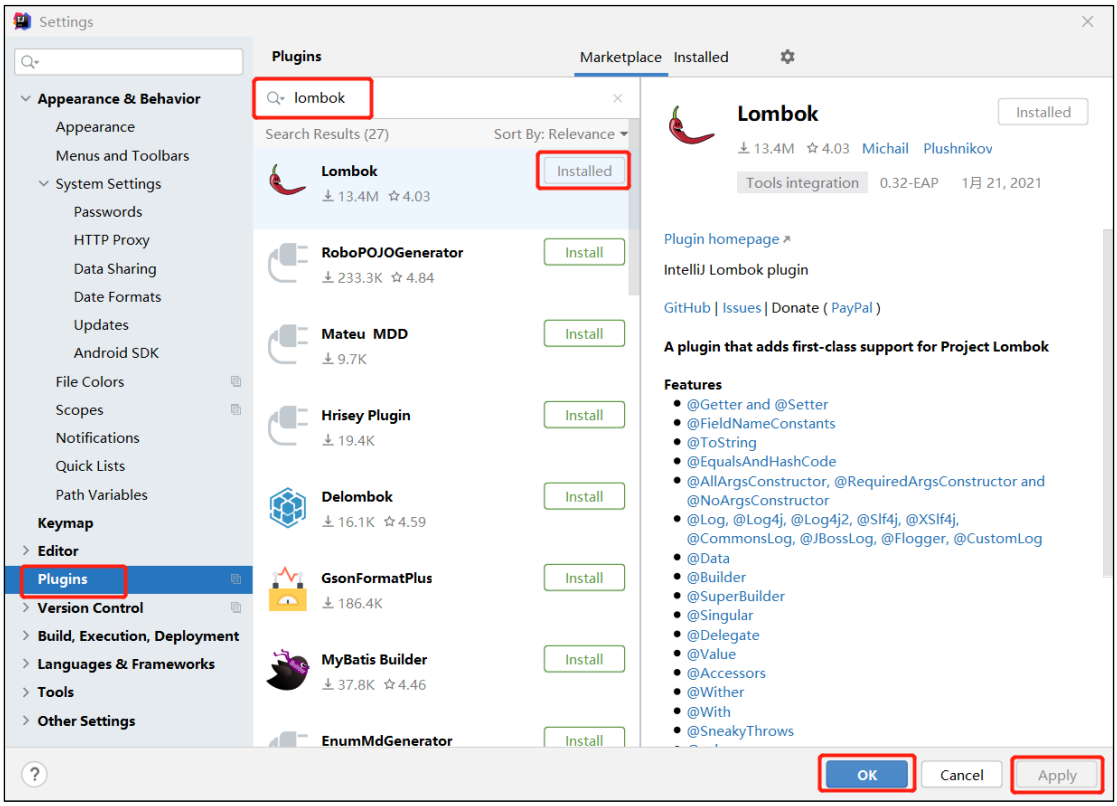
第 3 章 集成 SpringBoot

SpringBoot 是一个在 JavaEE 开发中非常常用的组件。可以用于 Kafka 的生产者，也可以用于 SpringBoot 的消费者。



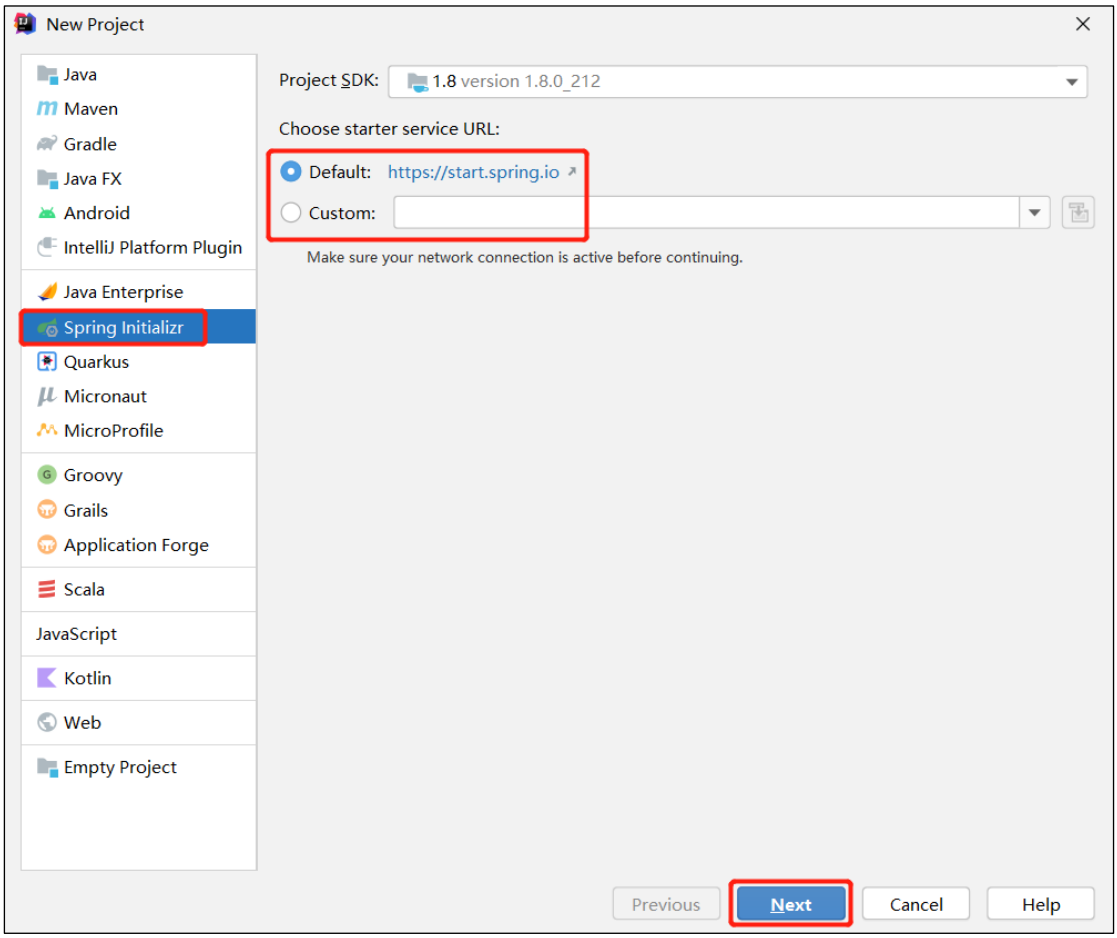
1）在 IDEA 中安装 lombok 插件

在 Plugins 下搜索 lombok 然后在线安装即可，安装后注意重启



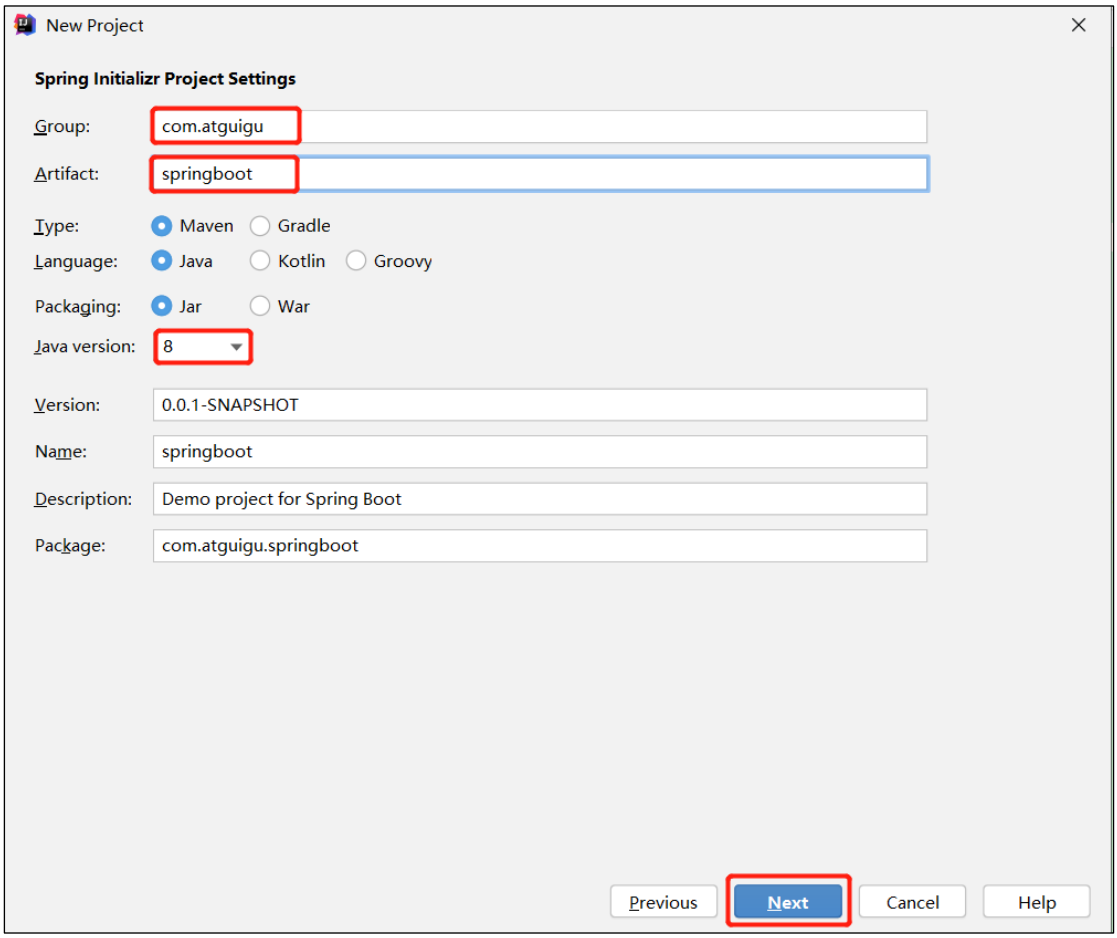
2）SpringBoot 环境准备

（1）创建一个 Spring Initializr

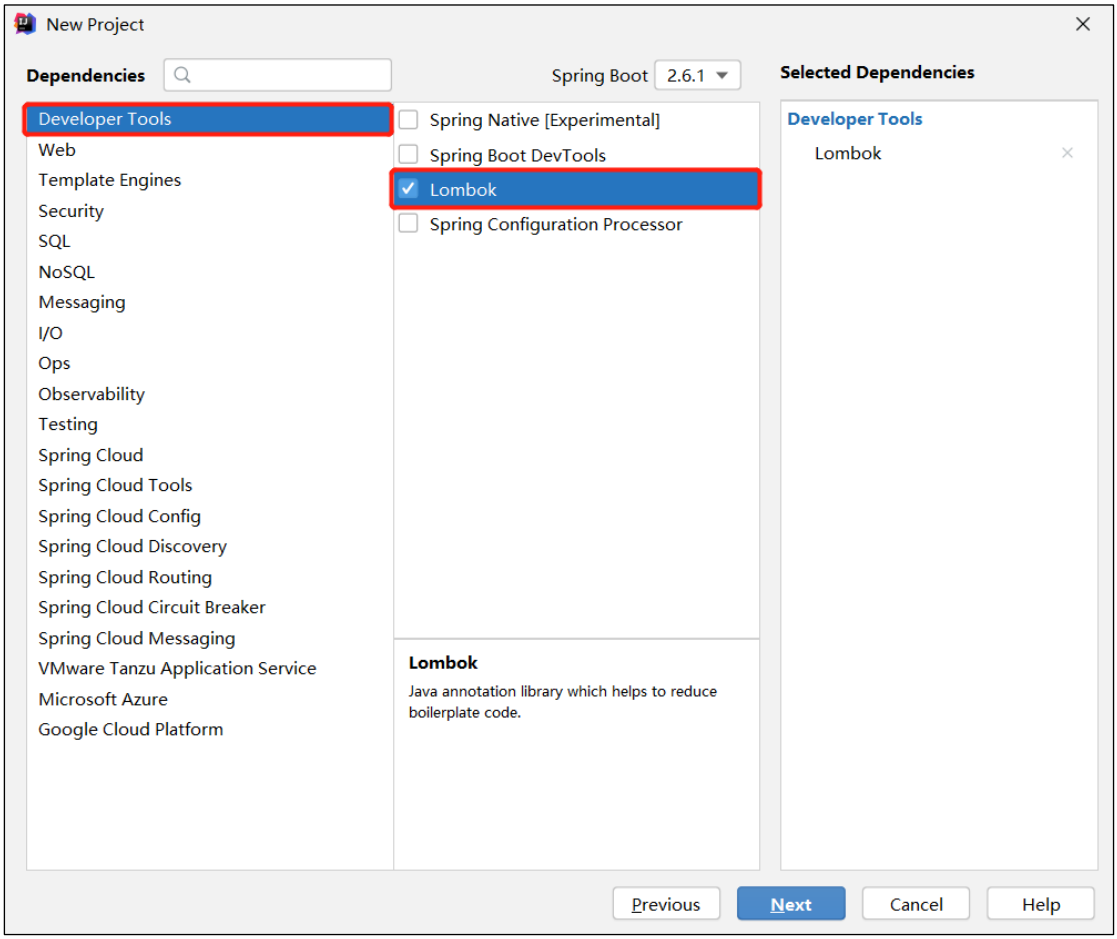


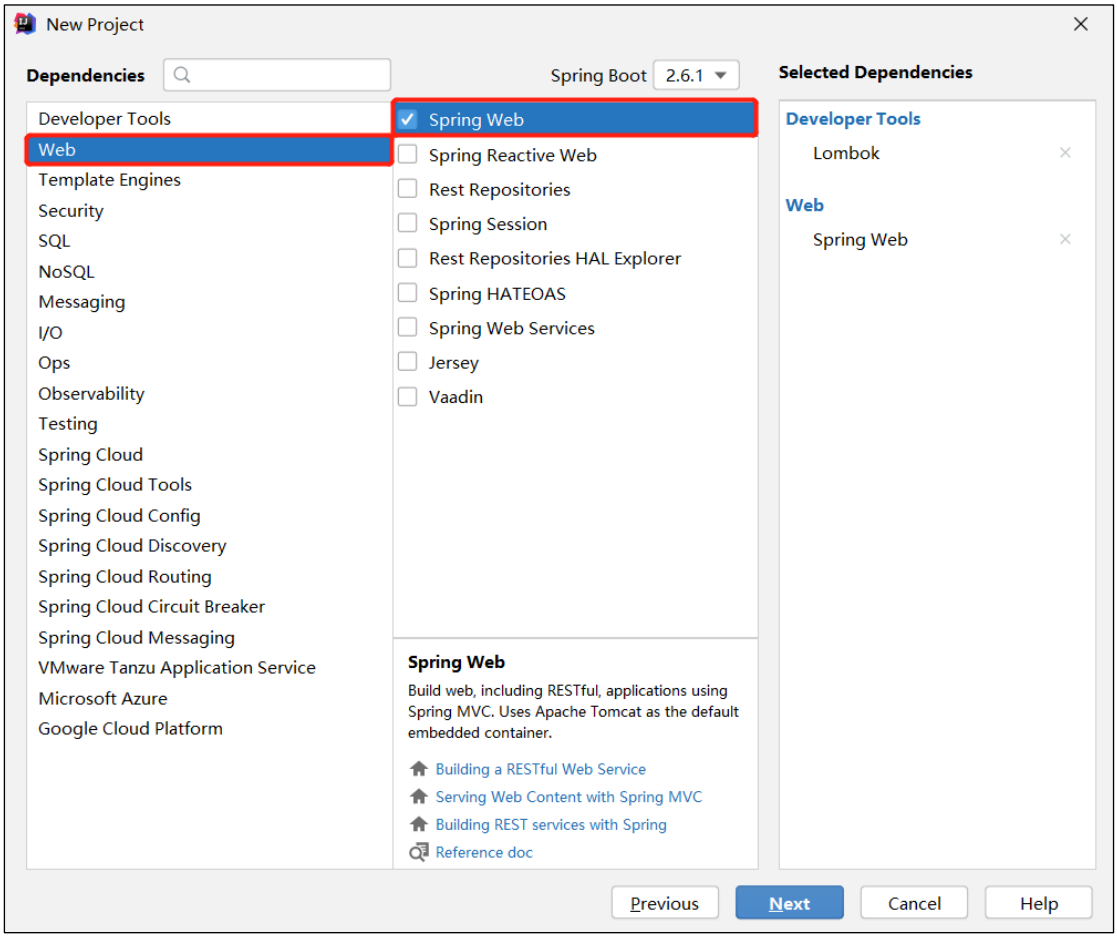
注意：有时候 SpringBoot 官方脚手架不稳定，我们切换国内地址 https://start.aliyun.com

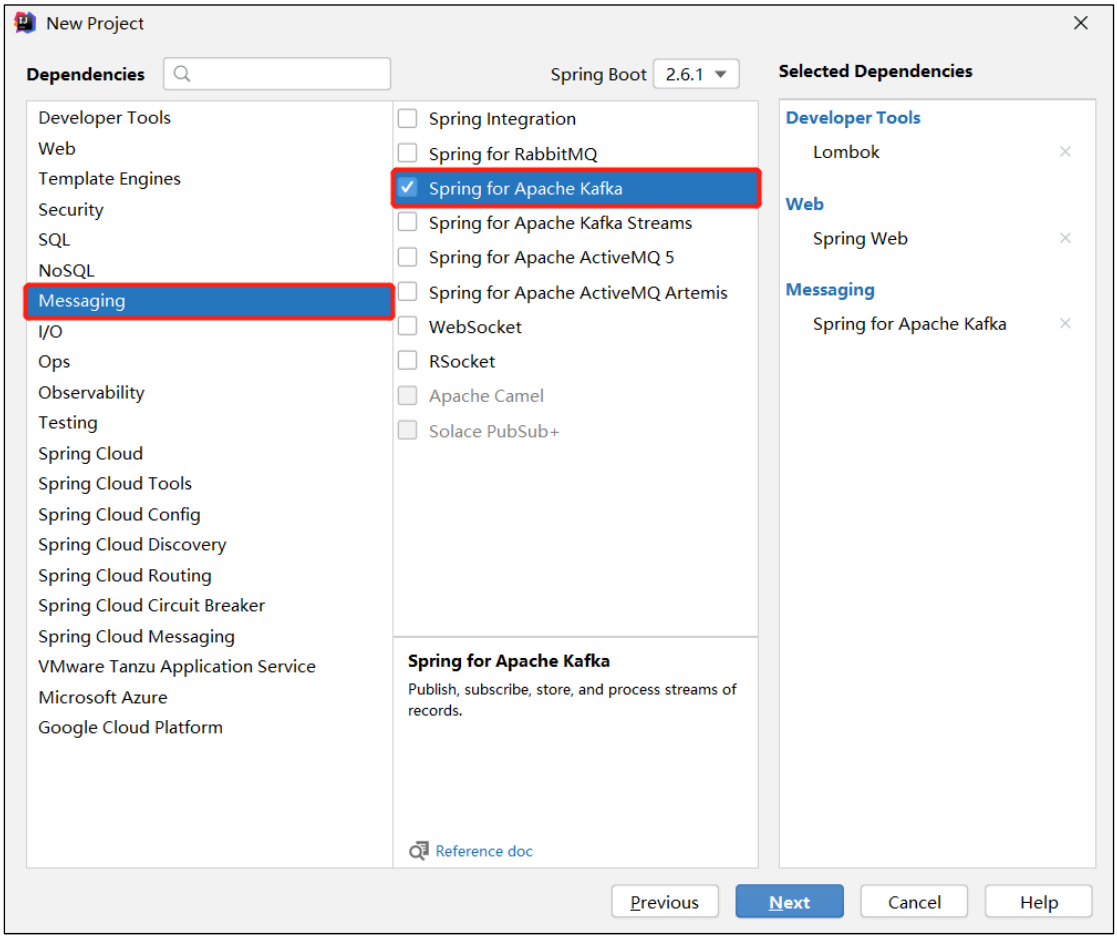
（2）项目名称 springboot

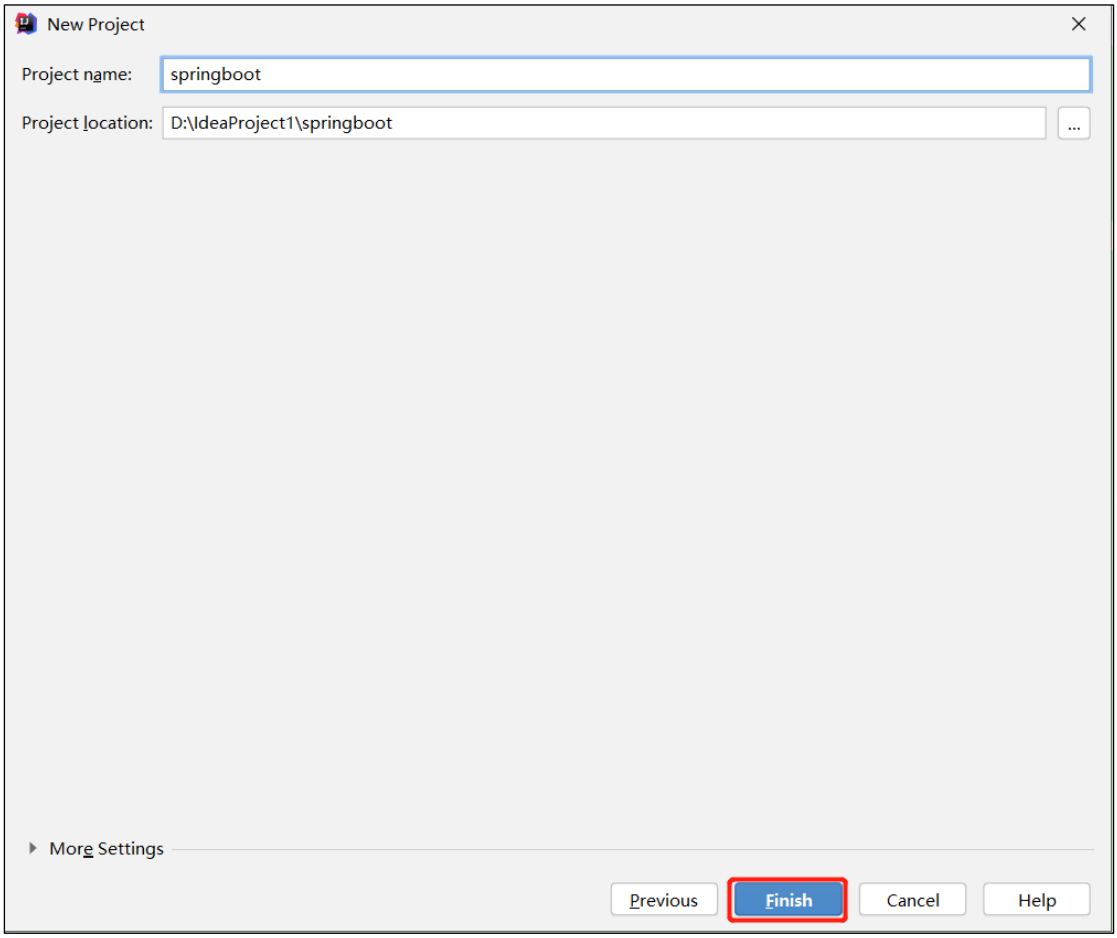

New Project
Spring Initializr Project Settings
Group:
com.atguigu
Artifact:
springboot
Type:
Maven
Gradle
Language:
Java
Kotlin
Groovy
Packaging:
Jar
War
Java version:
8
Version:
0.0.1-SNAPSHOT
Name:
springboot
Description:
Demo project for Spring Boot
Package:
com.atguigu.springboot
Previous
Next
Cancel
Help

（3）添加项目依赖


New Project
Dependencies
Spring Boot 2.6.1
Selected Dependencies
Developer Tools
Web
Template Engines
Security
SQL
NoSQL
Messaging
I/O
Ops
Observability
Testing
Spring Cloud
Spring Cloud Tools
Spring Cloud Config
Spring Cloud Discovery
Spring Cloud Routing
Spring Cloud Circuit Breaker
Spring Cloud Messaging
VMware Tanzu Application Service
Microsoft Azure
Google Cloud Platform
Spring Native [Experimental]
Spring Boot DevTools
Lombok
Spring Configuration Processor
Lombok
Java annotation library which helps to reduce boilerplate code.
Developer Tools
Lombok
Previous
Next
Cancel
Help


New Project
Dependencies
Spring Boot 2.6.1
Selected Dependencies
Developer Tools
Web
Template Engines
Security
SQL
NoSQL
Messaging
I/O
Ops
Observability
Testing
Spring Cloud
Spring Cloud Tools
Spring Cloud Config
Spring Cloud Discovery
Spring Cloud Routing
Spring Cloud Circuit Breaker
Spring Cloud Messaging
VMware Tanzu Application Service
Microsoft Azure
Google Cloud Platform
Spring Web
Spring Reactive Web
Rest Repositories
Spring Session
Rest Repositories HAL Explorer
Spring HATEOAS
Spring Web Services
Jersey
Vaadin
Spring Web
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
Building a RESTful Web Service
Serving Web Content with Spring MVC
Building REST services with Spring
Reference doc
Developer Tools
Lombok
Web
Spring Web
Previous
Next
Cancel
Help


New Project
Dependencies
Spring Boot 2.6.1
Selected Dependencies
Developer Tools
Web
Template Engines
Security
SQL
NoSQL
Messaging
I/O
Ops
Observability
Testing
Spring Cloud
Spring Cloud Tools
Spring Cloud Config
Spring Cloud Discovery
Spring Cloud Routing
Spring Cloud Circuit Breaker
Spring Cloud Messaging
VMware Tanzu Application Service
Microsoft Azure
Google Cloud Platform
Spring Integration
Spring for RabbitMQ
Spring for Apache Kafka
Spring for Apache Kafka Streams
Spring for Apache ActiveMQ 5
Spring for Apache ActiveMQ Artemis
WebSocket
RSocket
Apache Camel
Solace PubSub+
Spring for Apache Kafka
Publish, subscribe, store, and process streams of records.
Reference doc
Developer Tools
Lombok
Web
Spring Web
Messaging
Spring for Apache Kafka
Previous
Next
Cancel
Help



（4）检查自动生成的配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.1</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.atguigu</groupId>
    <artifactId>springboot</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>springboot</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
```

```
        <dependency>
            <groupId>org.springframework.kafka</groupId>
            <artifactId>spring-kafka</artifactId>
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.kafka</groupId>
            <artifactId>spring-kafka-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

3.1 SpringBoot 生产者

（1）修改 SpringBoot 核心配置文件 application.propeties，添加生产者相关信息

```
# 应用名称
spring.application.name=atguigu_springboot_kafka

# 指定 kafka 的地址
spring.kafka.bootstrap-
servers=hadoop102:9092,hadoop103:9092,hadoop104:9092

#指定 key 和 value 的序列化器
spring.kafka.producer.key-
serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-
serializer=org.apache.kafka.common.serialization.StringSerializer
```

（2）创建 controller 从浏览器接收数据，并写入指定的 topic

```
package com.atguigu.springboot;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ProducerController {

    // Kafka 模板用来向 kafka 发送数据
    @Autowired
    KafkaTemplate<String, String> kafka;

    @RequestMapping("/atguigu")
    public String data(String msg) {
        kafka.send("first", msg);
        return "ok";
    }
}
```

（3）在浏览器中给/atguigu 接口发送数据

http://localhost:8080/atguigu?msg=hello

3.2 SpringBoot 消费者

（1）修改 SpringBoot 核心配置文件 application.propeties

```
# =========消费者配置开始=========
# 指定 kafka 的地址
spring.kafka.bootstrap-servers=hadoop102:9092,hadoop103:9092,hadoop104:9092

# 指定 key 和 value 的反序列化器
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer

#指定消费者组的 group_id
spring.kafka.consumer.group-id=atguigu
# =========消费者配置结束=========
```

（2）创建类消费 Kafka 中指定 topic 的数据

```
package com.atguigu.springboot;

import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.annotation.KafkaListener;

@Configuration
```

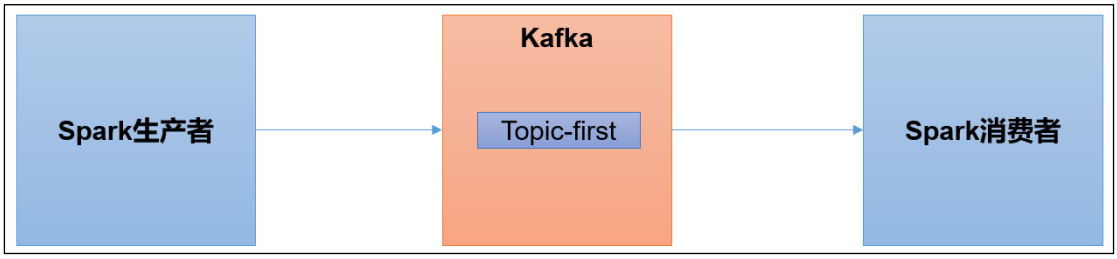
```
public class KafkaConsumer {

    // 指定要监听的 topic
    @KafkaListener(topics = "first")
    public void consumeTopic(String msg) { // 参数: 收到的 value
        System.out.println("收到的信息: " + msg);
    }
}
```

（3）向 first 主题发送数据

```
[atguigu@hadoop102 kafka]$ bin/kafka-console-producer.sh --bootstrap-server hadoop102:9092 --topic first
>
```

第 4 章 集成 Spark

Spark 是一个在大数据开发中非常常用的组件。可以用于 Kafka 的生产者，也可以用于 Spark 的消费者。



1）Scala 环境准备

尚硅谷大数据技术之Scala（3.8）.do

2）Spark 环境准备

（1）创建一个 maven 项目 spark-kafka

（2）在项目 spark-kafka 上点击右键，Add Framework Support=》勾选 scala

（3）在 main 下创建 scala 文件夹，并右键 Mark Directory as Sources Root=>在 scala 下创建包名为 com.atguigu.spark

（4）添加配置文件

```
<dependencies>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-streaming-kafka-0-10_2.12</artifactId>
        <version>3.0.0</version>
    </dependency>
</dependencies>
```

（5）将 log4j.properties 文件添加到 resources 里面，就能更改打印日志的级别为 error

```
log4j.rootLogger=error, stdout,R
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd
HH:mm:ss,SSS}  %5p --- [%50t]  %-80c(line:%5L)  :  %m%n

log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=../log/agent.log
log4j.appender.R.MaxFileSize=1024KB
log4j.appender.R.MaxBackupIndex=1

log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%d{yyyy-MM-dd
HH:mm:ss,SSS}  %5p --- [%50t]  %-80c(line:%6L)  :  %m%n
```

4.1 Spark 生产者

（1）在 com.atguigu.spark 包下创建 scala Object：SparkKafkaProducer

```
package com.atguigu.spark

import java.util.Properties
import          org.apache.kafka.clients.producer.{KafkaProducer,
ProducerRecord}

object SparkKafkaProducer {

  def main(args: Array[String]): Unit = {

    // 0 kafka 配置信息
    val properties = new Properties()
    properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
"hadoop102:9092,hadoop103:9092,hadoop104:9092")
    properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
classOf[StringSerializer])
    properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
classOf[StringSerializer])

    // 1 创建 kafka 生产者
    var producer = new KafkaProducer[String, String](properties)

    // 2 发送数据
    for (i <- 1 to 5){
      producer.send(new
ProducerRecord[String,String]("first","atguigu" + i))
    }

    // 3 关闭资源
    producer.close()
  }
}
```

（2）启动 Kafka 消费者

```
[atguigu@hadoop104  kafka]$  bin/kafka-console-consumer.sh  --
bootstrap-server hadoop102:9092 --topic first
```

（3）执行 SparkKafkaProducer 程序，观察 kafka 消费者控制台情况

4.2 Spark 消费者

（1）添加配置文件

```
<dependencies>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-streaming-kafka-0-10_2.12</artifactId>
        <version>3.0.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-core_2.12</artifactId>
        <version>3.0.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-streaming_2.12</artifactId>
        <version>3.0.0</version>
    </dependency>
</dependencies>
```

（2）在 com.atguigu.spark 包下创建 scala Object：SparkKafkaConsumer

```
package com.atguigu.spark

import org.apache.kafka.clients.consumer.{ConsumerConfig, ConsumerRecord}
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.spark.SparkConf
import org.apache.spark.streaming.dstream.{DStream, InputDStream}
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.kafka010.{ConsumerStrategies, KafkaUtils, LocationStrategies}

object SparkKafkaConsumer {

   def main(args: Array[String]): Unit = {

      //1.创建 SparkConf
      val sparkConf: SparkConf = new
SparkConf().setAppName("sparkstreaming").setMaster("local[*]")

      //2.创建 StreamingContext
      val ssc = new StreamingContext(sparkConf, Seconds(3))

      //3.定义 Kafka 参数：kafka 集群地址、消费者组名称、key 序列化、value 序列化
      val kafkaPara: Map[String, Object] = Map[String, Object](
         ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG ->
"hadoop102:9092,hadoop103:9092,hadoop104:9092",
         ConsumerConfig.GROUP_ID_CONFIG -> "atguiguGroup",
         ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG ->
classOf[StringDeserializer],
         ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG ->
classOf[StringDeserializer]
      )
```

```
    //4.读取 Kafka 数据创建 DStream
    val kafkaDStream: InputDStream[ConsumerRecord[String, String]] = KafkaUtils.createDirectStream[String, String](
        ssc,
        LocationStrategies.PreferConsistent, //优先位置
        ConsumerStrategies.Subscribe[String, String](Set("first"), kafkaPara)// 消费策略：（订阅多个主题，配置参数）
    )

    //5.将每条消息的 KV 取出
    val valueDStream: DStream[String] = kafkaDStream.map(record => record.value())

    //6.计算 WordCount
    valueDStream.print()

    //7.开启任务
    ssc.start()
    ssc.awaitTermination()
  }
}
```

（3）启动 SparkKafkaConsumer 消费者

（4）启动 kafka 生产者

```
[atguigu@hadoop103 kafka]$ bin/kafka-console-producer.sh --bootstrap-server hadoop102:9092 --topic first
```

（5）观察 IDEA 控制台数据打印