Spark Streaming：实时计算 Kafka 数据

一、实验简介

1. 实验描述

• 通过创建 Kafka topic，使用 Kafka Producer 产生消息，然后通过编写 spark

Streaming 程序处理这些消息。

- 主要步骤：
  - 创建 Spark Streaming 项目工程
  - 编写 streaming 程序
  - 启动 Zookeeper，Kafka 集群
  - 创建 topic
  - 启动 Kafka 生产者
  - 准备作业环境
  - 提交作业

2. 实验环境

• 虚拟机数量：3(分别为：server0、server1、server2)

• 系统版本：Centos 7.8

• Zookeeper 版本： Apache Zookeeper 3.5.10

• Kafka 版本：kafka_2.11-2.4.1

• Spark 版本：Apache Spark 2.4.8

3. 相关技能

• 使用 IDEA 开发 spark Streaming 程序

• Spark Streaming 编程

• Kafka topic 的创建

• 使用 Kafka 生产者产生消息

• Linux 命令使用

4. 知识点

• 常见 Linux 命令的使用• IDEA 的使用

• Spark Streaming 编程方法

• spark jar 包的提交

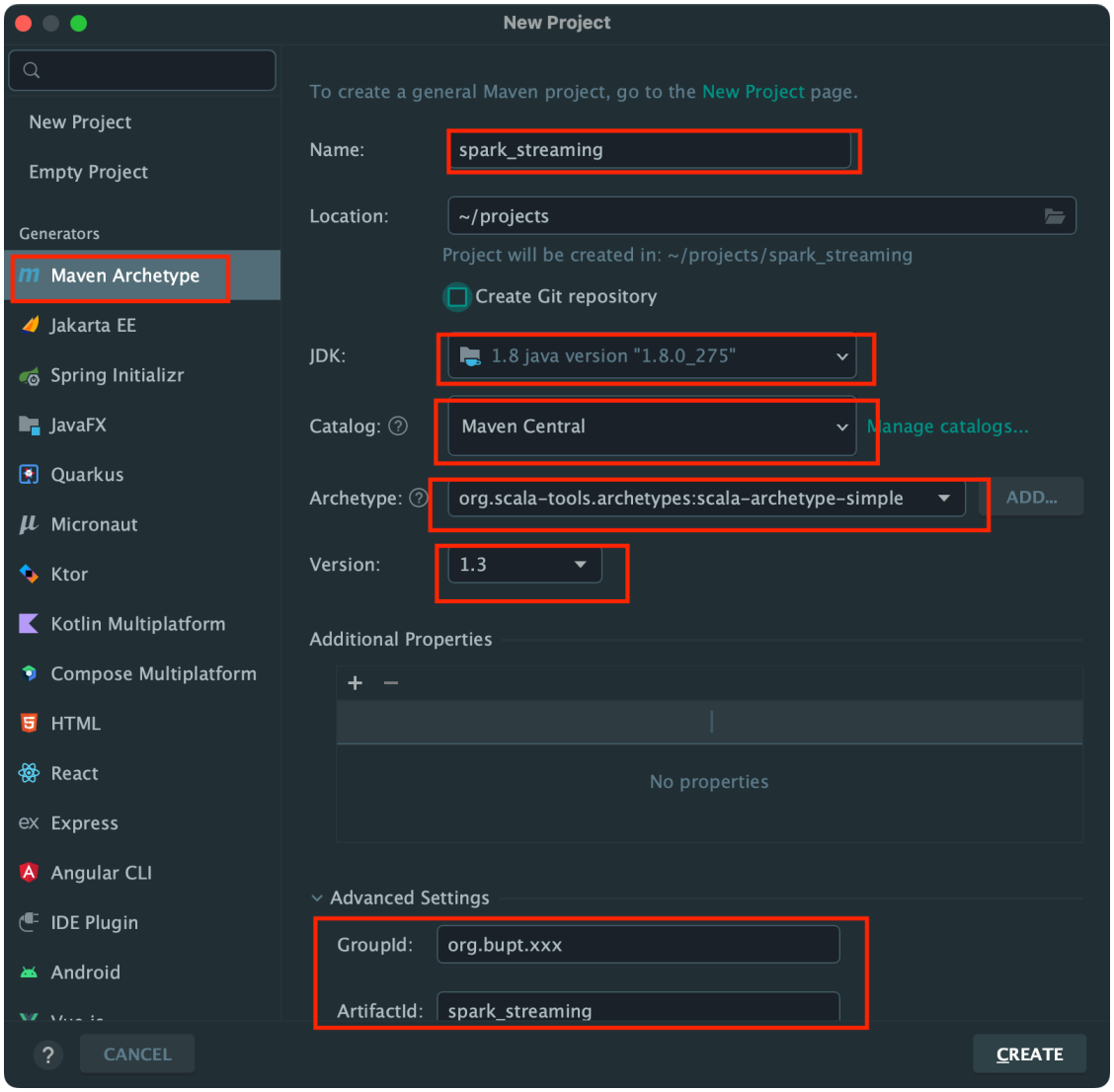
二、Spark Streaming代码编写

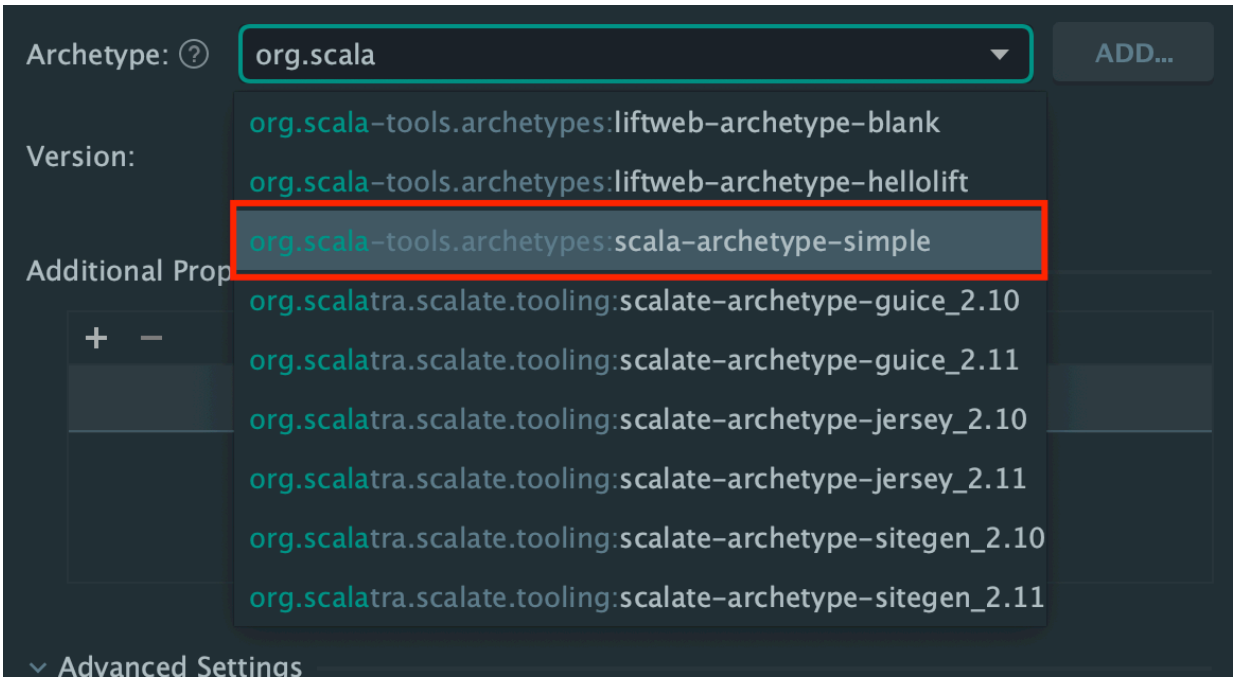
1. 创建Scala Maven项目

选择 `New Project` 创建项目, 出现如下界面, 根据下图进行配置

其中 `archetype` 的配置见第二张图, 在框内输入关键词进行搜索. 由于 `maven central` 在国内访问比较慢, 可以配置加速器/VPN来提速

`xxx` 需要替换为 `名字缩写-学号` 的形式





2. 导入依赖

编辑`pom.xml`覆盖原内容, 注意修改自己的groupId

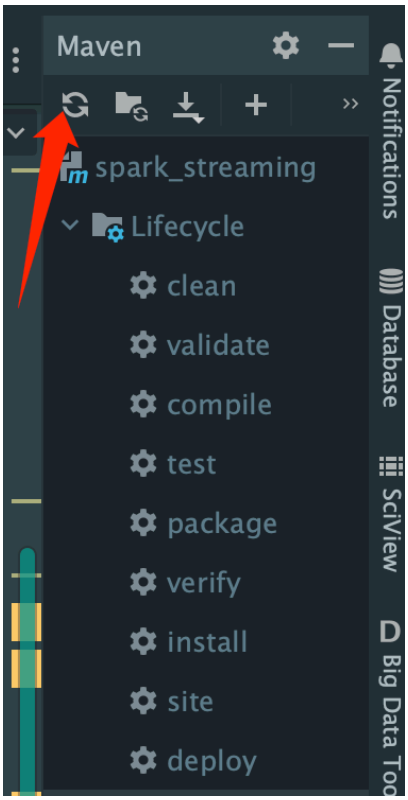
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.bupt.xxx</groupId>
    <artifactId>spark_streaming</artifactId>
    <version>1.0-SNAPSHOT</version>
    <name>${project.artifactId}</name>


    <properties>
        <maven.compiler.source>1.5</maven.compiler.source>
        <maven.compiler.target>1.5</maven.compiler.target>
        <encoding>UTF-8</encoding>
        <scala.version>2.11.12</scala.version>
        <spark.version>2.4.8</spark.version>
        <hadoop.version>2.7.7</hadoop.version>
    </properties>

    <!-- 配置阿里云仓库 -->
    <repositories>
        <repository>
            <id>aliyun-repos</id>
            <url>https://maven.aliyun.com/repository/public</url>
            <releases>
                <enabled>true</enabled>
```

```
        </releases>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>

<pluginRepositories>
    <pluginRepository>
        <id>scala-tools.org</id>
        <name>Scala-Tools Maven2 Repository</name>
        <url>http://scala-tools.org/repo-releases</url>
    </pluginRepository>
</pluginRepositories>

<dependencies>
    <dependency>
        <groupId>org.scala-lang</groupId>
        <artifactId>scala-library</artifactId>
        <version>${scala.version}</version>
    </dependency>

    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-core_2.11</artifactId>
        <version>${spark.version}</version>
        <exclusions>
            <exclusion>
                <groupId>org.scala-lang</groupId>
                <artifactId>scala-library</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-streaming_2.11</artifactId>
        <version>${spark.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-streaming-kafka-0-10_2.11</artifactId>
        <version>2.4.8</version>
        <exclusions>
            <exclusion>
                <groupId>org.apache.kafka</groupId>
                <artifactId>kafka-clients</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
```

```
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>2.4.1</version>
        </dependency>
    </dependencies>
    <build>
        <sourceDirectory>src/main/scala</sourceDirectory>
        <testSourceDirectory>src/test</testSourceDirectory>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>2.2.1</version>
                <configuration>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>

                </configuration>
                <executions>
                    <execution>
                        <id>make-assembly</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.scala-tools</groupId>
                <artifactId>maven-scala-plugin</artifactId>
                <executions>
                    <execution>
                        <goals>
                            <goal>compile</goal>
                            <goal>testCompile</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <scalaVersion>${scala.version}</scalaVersion>
                    <args>
                        <arg>-target:jvm-1.5</arg>
                    </args>
                </configuration>
            </plugin>
```

```
        </plugins>
    </build>

</project>
```

编辑后, IDEA会提示重新加载依赖, 点击即可, 如果没有提示,则点击maven工具栏中的图标



3. 编写流处理代码

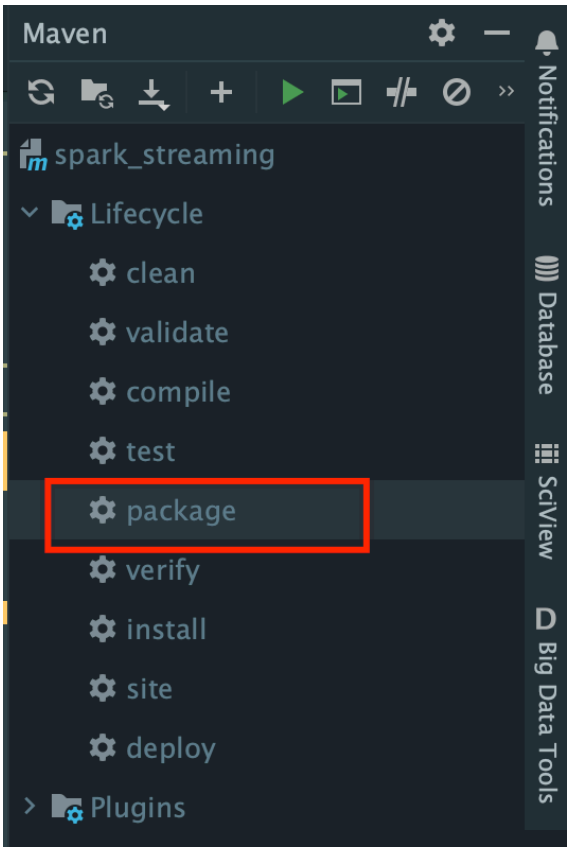
- 删除 test 目录下的所有代码
- 编写`org.bupt.xxx.App`对象代码, **其中注意Kafka的配置部分需要修改**

```
package org.bupt.xxx

import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.SparkConf
import org.apache.spark.streaming.dstream.{DStream, InputDStream}
import org.apache.spark.streaming.kafka010._
import org.apache.kafka.clients.consumer._

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

    // 中创建本地的 sparkconf 对象，包含 2 个执行线程，APP 名字命名为“kafka_streaming”
    val conf = new SparkConf().setAppName("kafka_streaming").setMaster("local[2]")

    // 创建本地的 StreamingContext 对象，第一参数为上一步创建 sparkconf 对象， 第二个参数为处理的
时间片间隔时间，设置为 3 秒
```

```
    val ssc = new StreamingContext(conf, Seconds(3))

    // Kafka 消费者配置
    val kafkaParams = Map[String, Object](
      "bootstrap.servers" -> "server0:9092", // 此处需要修改为自己的主机名称
      "key.deserializer" -> classOf[StringDeserializer],
      "value.deserializer" -> classOf[StringDeserializer],
      "group.id" -> "xxx", // 此处需要修改为自己的 姓名缩写-学号
      "auto.offset.reset" -> "latest",
      "enable.auto.commit" -> (false: java.lang.Boolean)
    )

    // Kafka 主题设置
    val topics: Set[String] = List("test_kafka").toSet

    // 创建 DStream，返回接收到的输入数据
    val stream: InputDStream[ConsumerRecord[String, String]] =
KafkaUtils.createDirectStream[String, String](ssc, LocationStrategies.PreferConsistent,
ConsumerStrategies.Subscribe[String, String](topics, kafkaParams))

    //  每一个 stream 都是一个 ConsumerRecord，获取每一个 stream 的 value
    var lines: DStream[String] = stream.map((record: ConsumerRecord[String, String]) =>
{
      record.value()
    })
    // 使用 flatMap 和 Split 对收到的字符串进行分割
    val words: DStream[String] = lines.flatMap(line => line.split(" "))

    // map 操作将独立的单词映射到(word，1)元组，reduceByKey 操作对 pairs 执行 reduce 操作获得
（单词，词频）元组
    val wordcounts = words.map(word => (word, 1)).reduceByKey((c1, c2) => c1 + c2)

    // 打印结果
    wordcounts.print()

    // Streaming 启动的流程，调用 start()函数启动，awaitTermination()函数 表示等待处理结束的信
号。
    ssc.start()
    ssc.awaitTermination()
  }
}
```

4. 打包Jar包

执行package, 会在target目录下生成两个jar包, 将其中名称为`spark_streaming-1.0-SNAPSHOT-jar-with-dependencies.jar`的文件重命名为`spark_streaming.jar`. 该文件将所有依赖都集成在jar包中, 使得我们可以在spark集群中使用spark不具有的依赖(如kafka-clients).



三、 实验环境启动

1. 启动HDFS/Yarn

在 server0 上执行 `start-dfs.sh`

在 server1 上执行 `start-yarn.sh`

2. 启动Zookeeper和Kafka

在三台节点上执行 `zkServer.sh start`

等待zookeeper启动完毕后, 在三台节点的kafka安装目录`opt/module/kafka-2.4.1`下执行

```
bin/kafka-server-start.sh -daemon config/server.properties
```

3. 创建Kafka Topic

在 `server0` 上的kafka安装目录`opt/module/kafka-2.4.1`下执行

```
bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 3 --topic test_kafka
```

--create 表示创建一个 topic

--bootstrap-servers localhost:9092 表示连接kafka服务的主机名和端口号, 由于本地有一个kafka实例,连接本地就行

--replication-factor 1 表示每个切片的副本数量是 1

--partitions 3 表示分区数量是 3

--topic test_kafka 表示创建一个名为“test_kafka”的 topic 主题

4. 上传Jar包

将打包出的`spark_streaming.jar`上传到`server0`的`/opt/software`目录下

四、 提交Spark任务

1. 提交Spark Streaming任务

在`server0`中执行命令: **注意替换xxx**为自己的名字缩写-学号. 该终端需要持续运行, 不能关闭

```
cd /opt/software
spark-submit --class org.bupt.xxx.App --master yarn --deploy-mode client
spark_streaming.jar
```

2. 向Kafka Topic中生产数据

- 打开一个新的`server0`终端

```
cd /opt/module/kafka-2.4.1
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test_kafka
```

kafka-console-producer.sh 表示启动一个生产者

--broker-list localhost:9092 表示 broker 服务列表中的 本地 服务和端口号

--topic test_kafka 表示向名为 test_kafka 的 topic 中生产数据

- 在出现的命令shell中随意输入单词, 如

```
>^C[hadoop@hadoop1-2022110946 kafka-2.4.1]$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test_kafka
>hi hello bupt speaker scala spark hello scala bupt bupt byr byr byr
```

3. 观察任务的输出

由于终端会一直输出大量信息, 可以ctrl-C 结束任务后返回查看

```
22/12/03 22:15:48 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 399.0, whose tasks have all completed, from pool
22/12/03 22:15:48 INFO scheduler.DAGScheduler: ResultStage 399 (print at App.scala:46) finished in 0.007 s
22/12/03 22:15:48 INFO scheduler.DAGScheduler: Job 199 finished: print at App.scala:46, took 0.008064 s
-------------------------------------------
Time: 1670076948000 ms
-------------------------------------------
(scala,2)
(hello,2)
(spark,1)
(bupt,3)
(byr,3)
(hi,1)
(speaker,1)
```

五、关闭集群

> 关闭过程要有序缓慢的进行, 关闭一个组件后, 通过jps检查已经关闭后, 再关闭下一个组件

1. 关闭Yarn和HDFS

在 server0 上执行 `stop-dfs.sh`

在 server1 上执行 `stop-yarn.sh`

2. 关闭Kafka

在三台节点上执行

```
cd /opt/module/kafka-2.4.1
./bin/kafka-server-stop.sh
```

3. 关闭Zookeeper

在三台节点上执行

```
zkServer.sh stop
```

4. 将集群三个节点关机

六、扩展实验

尝试理解 scala object 里的程序代码，可以做一些修改。

比如针对输入数据的解析方式、数据累加的方式等修改并展示。