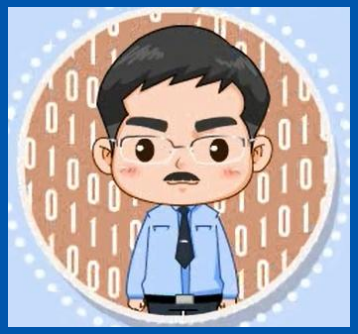


《Flink编程基础（Scala版）》

教材官网： http://dblab.xmu.edu.cn/post/flink/

温馨提示：编辑幻灯片母版，可以修改每页PPT的厦大校徽和底部文字

第8章 FlinkCEP

（PPT版本号：2021年9月版本）

扫一扫访问教材官网

林子雨 博士/副教授

厦门大学计算机科学与技术系

E-mail: ziyulin@xmu.edu.cn

主页：http://dblab.xmu.edu.cn/linziyu

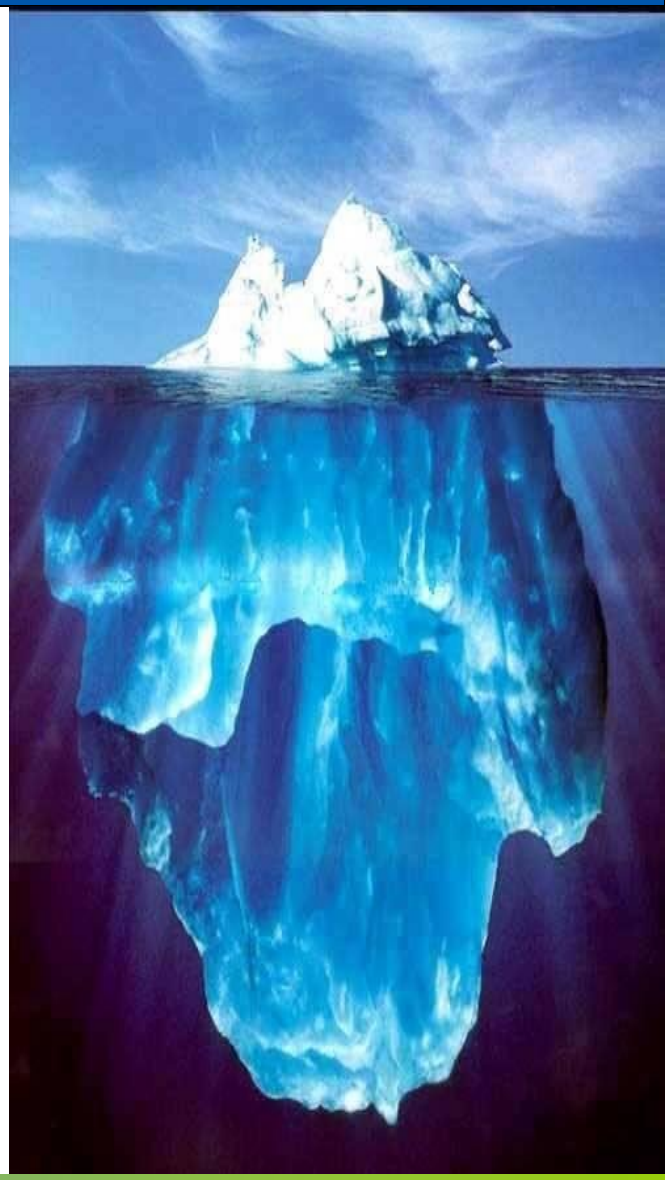
配套教材

林子雨，陶继平 编著
《Flink编程基础（Scala版）》
清华大学出版社出版发行
ISBN: 978-7-302-58367-7 2021年9月第1版
教材官网：http://dblab.xmu.edu.cn/post/flink/
E-mail:ziyulin@xmu.edu.cn

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Flink技术铺平道路
深入浅出，有效降低Flink技术学习门槛
资源全面，构建全方位一站式在线服务体系

提纲

8.1 概述
8.2 Pattern API
8.3 模式的检测
8.4 应用实例



高校大数据课程
公 共 服 务 平 台

百度搜索厦门大学数据库实验室网站访问平台

8.1 概述

复杂事件处理（Complex Event Processing，简称CEP）将来自多个来源的数据组合在一起，以推断出更复杂情况的事件或模式。复杂事件处理的目标是识别有意义的事件，并尽快对其做出响应。可见，复杂事件处理天生带有流数据处理的一些特征，具体如下：

- 复杂性。需要处理多个来源的数据；
- 低延迟。在一些应用中需要做出秒级或毫秒级的响应；
- 高吞吐。需要处理海量的事件流并从中挖掘出有意义的事件。

常见的CEP用例包括欺诈检测、复杂系统中的监视和警报、检测网络入侵或可疑用户行为等。

8.1 概述

•虽然通过DataStream API也可以达到处理复杂事件的目的，但是往往需要十分复杂的逻辑才能表达出不同事件到达的顺序、次数等特征，具有很大的局限性。因此，Flink提供了专门用于复杂事件处理的库FlinkCEP。

•FlinkCEP组件栈构建在DataStream API之上，提供了用于定义输入事件流模式的Pattern API，并将Pattern API定义好的模式应用在输入流上，构建出一个事件流PatternStream，最后使用PatternStream的select方法从输入事件流中抽取与之前定义好的模式相匹配的事件序列。

8.1 概述

在使用FlinkCEP库之前，需要将FlinkCEP的依赖导入到pom.xml文件中。以下是FlinkCEP对应的Maven依赖：

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-cep-scala_2.12</artifactId>
    <version>1.11.2</version>
</dependency>
```

8.2 Pattern API

8.2.1 个体模式
8.2.2 复合模式
8.2.3 模式组
8.2.4 匹配后跳过策略

8.2.1 个体模式

一个个体模式可以是单例模式，也可以是循环模式。它们的区别在于单例模式每次只接收单个事件，而循环模式可以接收一个或多个事件。个体模式在默认情况下都是单例模式，不过可以通过定义量词将其转换为循环模式。每个模式可以对其定义若干个条件来控制该模式是否要开始接收事件。以下代码通过begin方法定义了一个个体模式start：

```
val start = Pattern.begin("start")
.where(_.getName.startsWith("start"))
```

8.2.1 个体模式

1. 量词

Pattern API提供的量词能灵活地控制单例模式匹配的次数，将单例模式转换成循环模式。通过定义单例模式的量词，既可以指定匹配的次数，也可以指定匹配次数的范围。下面以单例模式start为例，介绍不同量词的效果。

（1）times方法：指定匹配的次数。

```
// 匹配4次
start.times(4)
// 匹配2、3或4次
start.times(2, 4)
```

8.2.1 个体模式

（2）optional方法：将该模式标记为可选的，即该模式要么不匹配，要么就按照指定的次数匹配。

```
// 要么匹配4次，要么不匹配
start.times(4).optional()
// 要么匹配2、3或4次， 要么不匹配
start.times(2, 4).optional()
```

（3）greedy方法：将该模式标记为贪婪的，即该模式在完成指定次数匹配的情况下尽可能多地匹配。

```
// 匹配2到4次，并且希望尽可能多地匹配
start.times(2, 4).greedy()
```

8.2.1 个体模式

（4）oneOrMore方法：希望匹配一次或多次。

```
// 匹配1次或多次
start.oneOrMore()
// 要么匹配1次或多次，要么不匹配
start.oneOrMore().optional()
```

（5）timesOrMore方法：指定匹配次数为若干次或以上。

```
// 匹配2次或更多次
start.timesOrMore(2)
// 匹配0次或2次或更多次
start.timesOrMore(2).optional()
```

8.2.1 个体模式

2. 条件

对于每一个个体模式，都可以设置一些条件来控制该模式是否要开始接收事件或停止接收事件。只有当事件的属性满足预设的条件才可以被模式接收。Pattern API为可以为个体模式定义5种不同类型的条件，分别是迭代条件、简单条件、复合条件、停止条件和邻近条件。

8.2.1 个体模式

（1）迭代条件

迭代条件能够根据之前已接收事件的属性或这些事件的子集属性的统计信息，来判断是否要继续接收后续的事件。下面的代码演示了迭代条件的使用。首先通过subtype方法将事件转换为SubEvent，在where中使用ctx.getEventsForPattern方法获取模式middle接收的所有事件，并计算出这些事件的总价格。如果下一个事件的名称以foo开头，且其价格与之前价格的总和不超过5.0，则该模式middle将接收这一事件。要注意的是，这里用到的ctx.getEventsForPattern方法会获取之前接收的所有事件，因此它的开销可能很大。Flink官方建议我们尽量减少该方法的使用，以免对系统的性能造成影响。

```
middle.oneOrMore()
    .subtype(classOf[SubEvent])
    .where(
        (value, ctx) => {
            lazy val sum = ctx.getEventsForPattern("middle").map(_.getPrice).sum
            value.getName.startsWith("foo") && sum + value.getPrice < 5.0
        }
    )
```

8.2.1 个体模式

（2）简单条件

该类型的条件继承自IterativeCondition类，它根据事件本身的属性来判断是否要接收该事件。以下代码规定了只有名称以foo开头的事件才会被模式start接收。

```
start.where(event => event.getName.startsWith("foo"))
```

（3）复合条件

复合条件即多个条件的组合，多个顺序排序的where方法代表逻辑与，使用or方法连接条件代表逻辑或。以下代码规定只有名称以foo开头或价格小于2.0的事件才会被模式middle接收。

```
middle.where(event => event.getName.startsWith("foo"))
.or(event => event.getPrice < 2.0)
```

8.2.1 个体模式

（4）停止条件

对于循环模式，可以指定停止条件让其在满足条件时停止接收事件。FlinkCEP提供了until方法来规定停止条件，需要注意的是，until方法必须与oneOrMore方法一起使用。如下代码用until方法指定停止条件为遇到事件名称为end的事件时，停止接收事件。

```
middle.oneOrMore().until(event => event.getName() == “end”)
```

8.2.1 个体模式

（5）邻近条件

一个循环模式一次可以接收一个或多个事件，邻近条件规定了每个事件之间的相邻关系。在FlinkCEP中可以对循环模式规定3种不同的邻近条件，分别是严格邻近、宽松邻近和非确定宽松邻近。严格近邻规定，所有匹配的事件应该是严格地一个接一个地出现，中间不能有任何不匹配的事件；而宽松邻近允许匹配的事件之间出现不匹配的事件；非确定宽松邻近则进一步放开了条件，允许在匹配过程中忽略已经匹配的条件。默认情况下，循环模式满足宽松邻近条件。

FlinkCEP提供了consecutive方法用于定义循环模式的严格邻近条件，使用allowCombinations来定义非确定宽松邻近。以下的代码以严格邻近模式为例，为循环模式middle指定严格邻近条件，只有严格邻近的且名称以foo开头的事件才会被middle接收。

```
middle.where(event => event.getName.startsWith("foo"))
.consecutive()
```

8.2.2 复合模式

复合模式就是将不同的个体模式组合起来，因而复合模式又被称为模式序列。将个体模式组合起来的就是上面提到过的邻近条件。不过，在复合模式中，邻近条件的种类要比在循环模式中提到的要丰富。除了上面提到的三种邻近条件，还有不让某一模式出现在当前模式之后的邻近条件，以及为模式定义一个时间约束。

首先，每个复合模式都必须以一个初始模式为开头，如下代码所示：

```
val start = Pattern.begin("start")
.where(_.getName.startsWith("start"))
```

接着就可以用邻近条件将不同的个体模式连接起来。

1. 严格邻近

期望所有的模式在匹配时都严格地按照定义的顺序出现，中间不出现任何不满足模式的事件，在FlinkCEP中用next方法实现。下面的代码规定start之后的模式必须是模式middle。

```
val strict = start.next("middle")
.where(event => event.getName.startsWith("foo"))
.times(3)
```

相应地，也可以规定不让一个模式紧跟着另一个模式发生，在FlinkCEP中用notNext方法实现。下面的代码规定模式start之后的模式不能是not。

```
val strictNot = start.notNext("not")
.where(event => event.getName.startsWith("foo"))
```

2. 宽松邻近

允许两个匹配的模式之间出现不匹配的事件，在FlinkCEP中用followedBy方法实现。下面的代码规定start之后的模式是模式middle，但是中间可以有不匹配的事件。

```
val relaxed = start.followedBy("middle")
.where(event => event.getName.startsWith("foo"))
.times(3)
```

相应地，也可以规定不让某个模式在两个匹配的模式之间发生匹配，在FlinkCEP中用notFollowedBy方法实现。需要注意的是，模式序列不能以notFollowedBy结束。下面的代码规定模式not不能在模式start和模式start之间发生匹配。

```
val relaxedNot = start.notFollowedBy("not")
.where(event => event.getName.startsWith("foo"))
.followedBy("middle").where(_.getPrice < 2.0)
```

3. 非确定宽松邻近

非确定宽松邻近进一步放松了连续性，允许其他匹配忽略某些匹配的事件，在FlinkCEP中通过followedByAny方法实现。下面的代码规定模式start和模式middle为非确定宽松邻近：

```
val nonDetermin = start.followedByAny("middle")
.where(event => event.getName.startsWith("foo"))
```

4. 时间约束

可以给每个模式一个时间约束，在规定时间内到达的事件发生的匹配才是有效的，在FlinkCEP中通过within方法实现。以下的代码规定模式middle必须在10秒之内完成匹配，否则就不再接收事件：

```
val strict = start.next("middle")
.where(event => event.getName.startsWith("foo"))
.within(Time.seconds(10)
```

通过以上的几个连接条件，我们就可以完整地定义一个复合模式pattern。首先，模式pattern在事件流中寻找一个名称以start为开头的事件。接着，模式pattern将寻找严格连续出现且名称以foo开头的3个事件。最后，模式pattern需要在10秒内到达的事件中找到2个以上价格为2.0以下的事件。

```
val pattern = Pattern.begin("start")
.where(_.getName.startsWith("start"))
.followedBy("middle")
.where(_.getName.startsWith("foo"))
    .times(3).consecutive()
.followedBy("end").where(_.getPrice < 2.0)
.timesOrMore(2)
.within(Time.seconds(10))
```

8.2.3 模式组

上面介绍的复合模式是以begin方法开头，再通过一些邻近条件将个体模式连接起来形成的，同样地，复合模式也可以通过上面的条件连接起来，形成一个模式组。在模式组上也可以指定量词和循环模式中介绍的邻近条件。从逻辑上看，复合模式可以看作是模式组的条件。每个复合模式完成各自内部的匹配后，最后在模式组的层面对匹配的结果进行汇总。以下代码展示了模式组gPattern的定义。

```
val gPattern = Pattern.begin(Pattern.begin("start")
    .where(_.getName.startsWith("start"))
    .followedBy("start_middle")
    .where(_.getName.startsWith("foo"))
    )
    .next(Pattern.begin("next_start")
        .where(_.getName.startsWith("buy"))
        .followedBy("next_middle")
        .where(_.getPrice < 2.0)
    ).times(3).consecutive()
```

8.2.4 匹配后跳过策略

对于给定的一个模式，其在事件流中可能会发生多次匹配，与此同时，一个事件可能被分配到多次匹配中。为了控制一个事件被匹配的次数，可以指定匹配后跳过策略。FlinkCEP提供了AfterMatchSkipStrategy，其中有5种跳过策略，分别是：

（1）NO_SKIP：任意一次匹配都不会被跳过。可以通过下面代码指定匹配后跳过策略为NO_SKIP。

```
val skipStrategy = AfterMatchSkipStrategy.noSkip()
```

（2）SKIP_TO_NEXT：丢弃以同一个事件开始的所有部分匹配。可以通过下面代码指定匹配后跳过策略为SKIP_TO_NEXT。

```
val skipStrategy = AfterMatchSkipStrategy.skipToNext()
```

（3）SKIP_PAST_LAST_EVENT：丢弃匹配开始后但结束之前的所有部分匹配。可以通过下面代码指定匹配后跳过策略为SKIP_PAST_LAST_EVENT。

```
val skipStrategy = AfterMatchSkipStrategy.skipPastLastEvent()
```

（4）SKIP_TO_FIRST：丢弃在匹配开始后但在指定事件第一次发生前开始的所有部分匹配，这里需要指定一个有效的patternName。可以通过下面代码指定匹配后跳过策略为SKIP_TO_FIRST。

```
val skipStrategy = AfterMatchSkipStrategy.skipToFirst(patternName)
```

（5）SKIP_TO_LAST：丢弃在匹配开始后但在指定事件最后一次发生前开始的所有部分匹配，这里需要指定一个有效的patternName。可以通过下面代码指定匹配后跳过策略为SKIP_TO_LAST。

```
val skipStrategy = AfterMatchSkipStrategy.skipToLast(patternName)
```

在创建模式时，先定义一个匹配后跳过策略，然后就可以将其应用到begin方法中。如果没有指定，Flink会默认将匹配后跳过策略指定为NO_SKIP。下面代码指定了匹配后跳过策略为SKIP_TO_NEXT，并将其应用到模式start上。

```
val skipStrategy = AfterMatchSkipStrategy.skipToNext()
val start = Pattern.begin("start", skipStrategy)
  .where(_.getName.startsWith("start"))
```

8.3 模式的检测

在定义好要查找的复合模式或模式组后，就可以将其应用到输入流中，以检测潜在的匹配。想要从事件流中提取出与模式相匹配的事件必须将定义好的模式与输入流相结合，创建一个PatternStream类型的模式流，后续的匹配事件提取都是基于这一类型。首先定义一个输入流input，这里的输入流可以是DataStream也可以是KeyedStream。要注意的是，使用DataStream时并行度只能是1。接着定义一个模式，或者也可以有选择性地定义一个comparator，用于根据事件到达时间或时间戳对事件进行排序。最后，FlinkCEP提供了CEP.pattern方法来生成模式流，代码如下。

```
val patternStream = CEP.pattern(input, pattern)
```

8.3.1 匹配事件提取

创建好PatternStream后，输入事件流中与已定义模式相匹配的事件序列就已经存储在patternStream中。此时就可以用PatternStream提供的select和flatSelect方法从检测到的事件序列中提取结果了。

select方法需要输入一个选择函数（select function）为参数，每个被成功匹配的事件序列都会调用它。选择函数以一个Map[String, Iterable[IN]]来接收匹配到的事件序列，其中key是每个模式的名称，而value是所有接收到的事件的Iterable类型。每次调用选择函数只会返回一个结果。以下代码是一个选择函数的定义，该函数会从PatternStream中提取出与模式start和模式end相匹配的事件序列。

```
def selectFn(pattern: Map[String, Iterable[IN]]): OUT = {
  val startEvent = pattern.get("start").get.next
  val endEvent = pattern.get("end").get.next
  OUT(startEvent, endEvent)
}
```

8.3.1 匹配事件提取

如果需要通过flatSelect方法来提取事件序列，则需要定义一个flat选择函数。flat选择函数与选择函数类似，不过flat选择函数使用Collector作为返回结果的容器，因此每次调用可以返回任意数量的结果。以下代码是一个flat选择函数的定义，该函数会从PatternStream中提取出与模式start和模式end相匹配的事件序列。

```
def flatSelectFn(pattern: Map[String, Iterable[IN]]): collector:
COLLECTOR[OUT] = {
  val startEvent = pattern.get("start").get.next
  val endEvent = pattern.get("end").get.next
  for (i <- 0 to startEvent.getValue){
    collector.collect(OUT(startEvent, endEvent))
  }
}
```

8.3.2 超时事件提取

对于事件流中的事件，如果其到达的时间超过了模式中within方法规定的时间窗口，与当前模式部分匹配的事件序列就会被丢弃，这些事件被称为超时事件。FlinkCEP提供了select方法和flatSelect方法来提取超时事件，且这一方法可以在提取匹配事件的同时提取超时事件。下面一段代码使用flatSelect方法将提取匹配事件与超时事件，最终用getSideOutput方法将超时事件输出。

8.3.2 超时事件提取

```
// 创建一个事件流
val patternStream = CEP.pattern(input, pattern)
// 定义一个OutputTag并命名为late-data
val outputTag = OutputTag[String](" late-data")

val result = patternStream.flatSelect(outputTag){
    // 提取超时事件
    (pattern: Map[String, Iterable[Event]], timestamp: Long, out:
Collector[TimeoutEvent]) =>
        out.collect(TimeoutEvent())
} { // 提取正常事件
    (pattern: mutable.Map[String, Iterable[Event]], out:
Collector[ComplexEvent]) =>
        out.collect(ComplexEvent())
}
// 调用getSideOutput并指定outputTag将超时事件输出
val timeoutResult = result.getSideOutput(outputTag)
```

8.4 应用实例

通过以上的介绍，我们对FlinkCEP如何处理复杂事件有了一个基本的了解。一般来说，编写FlinkCEP独立应用程序包括如下步骤：

1. 创建一个DataStream用于接收事件流。通常在开发应用程序的过程中，会用样例类来表示接收到的事件，以方便后续对事件的处理。
2. 通过Pattern API定义事件模式。用户可以用Pattern API中提供的方法来自定义希望从事件流中提取的事件模式。
3. 用CEP.pattern方法将输入事件流与模式结合起来。此时Flink会根据定义好的模式对事件流中的事件进行模式匹配，并把所有与模式相匹配的事件序列用PatternStream返回。
4. 调用PatternStream的select方法或flatSelect方法从匹配的事件序列中提取需要输出的事件。这里往往需要重写PatternSelectFunction中的select方法或flatSelect方法以满足用户对输出事件的格式需求，通常也需要用样例类来表示输出事件。

8.4 应用实例

下面通过一个案例来完整地介绍编写FlinkCEP应用程序的步骤。下面是某购物网站对一些用户的点击行为（click）、加入购物车行为（order）以及购买行为（buy）的部分记录：

Adam,click,1558430815185
Adam,buy,1558430815865
Adam,order,1558430815985
Berry,buy,1558430815988
Adam,click,1558430816068
Berry,order,1558430816074
Carl,click,1558430816151
Carl,buy,1558430816641
Dennis,buy,1558430817128
Carl,click,1558430817165
Ella,click,1558430818652

8.4 应用实例

我们需要从中找出用户在点击商品后立即购买的操作，并将用户名以及点击与购买的时间输出到控制台中。完整代码如下：

```
import java.util

import org.apache.flink.cep.PatternSelectFunction
import org.apache.flink.cep.scala.CEP
import org.apache.flink.cep.scala.pattern.Pattern
import org.apache.flink.streaming.api.TimeCharacteristic
import org.apache.flink.streaming.api.scala._
// 定义输入事件的样例类
case class UserAction(userName: String, eventType: String, eventTime:
Long)
// 定义输出事件的样例类
case class ClickAndBuyAction(userName: String, clickTime: Long,
buyTime: Long)
```

8.4 应用实例

```
object UserActionDetect {
  def main(args: Array[String]): Unit = {
    val env = StreamExecutionEnvironment.getExecutionEnvironment
    env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)

    val dataList = List(
      UserAction("Adam", "click", 1558430815185L),
      UserAction("Adam", "buy", 1558430815865L),
      UserAction("Adam", "order", 1558430815985L),
      UserAction("Berry", "buy", 1558430815988L),
      UserAction("Adam", "click", 1558430816068L),
      UserAction("Berry", "order", 1558430816074L),
      UserAction("Carl", "click", 1558430816151L),
      UserAction("Carl", "buy", 1558430816641L),
      UserAction("Dennis", "buy", 1558430817128L),
      UserAction("Carl", "click", 1558430817165L),
      UserAction("Ella", "click", 1558430818652L),
    )
```

8.4 应用实例

```
// 1. 创建输入事件流
  val userLogStream = env.fromCollection(dataList)
    .assignAscendingTimestamps(_.eventTime)
    .keyBy(_.userName)
  // 2. 用户自定义模式
  val userActionPattern = Pattern.begin[UserAction]("begin")
    .where(_.eventType == "click")
    .next("next")
    .where(_.eventType == "buy")
  // 3. 调用CEP.pattern方法寻找与模式匹配的事件
  val patternStream = CEP.pattern(userLogStream, userActionPattern)
  // 4. 输出结果
  val result = patternStream.select(new ClickAndBuyMatch())

  result.print()

  env.execute()
 }
}
```

8.4 应用实例

```
// 重写select方法
class ClickAndBuyMatch() extends PatternSelectFunction[UserAction,
ClickAndBuyAction] {
  override def select(map: util.Map[String, util.List[UserAction]]):
ClickAndBuyAction = {
    val click: UserAction = map.get("begin").iterator().next()
    val buy: UserAction = map.get("next").iterator().next()
    ClickAndBuyAction(click.userName, click.eventTime, buy.eventTime)
  }
}
```

8.4 应用实例

在运行这段代码前，需要先启动Flink。然后，创建项目文件夹，命令如下：

```
$cd ~
$mkdir FlinkCEP
$cd FlinkCEP
$mkdir -p src/main/scala
$vim src/main/scala/UserActionDetect.scala
```

8.4 应用实例

将上面的代码复制到UserActionDetect.scala中，接着创建一个pom.xml文件，并输入如下内容：

```
<project>
    <groupId>cn.edu.xmu.dblab</groupId>
    <artifactId>simple-project</artifactId>
    <modelVersion>4.0.0</modelVersion>
    <name>Simple Project</name>
    <packaging>jar</packaging>
    <version>1.0</version>
    <dependencies>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-cep-scala_2.12</artifactId>
            <version>1.11.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-scala_2.12</artifactId>
            <version>1.11.2</version>
        </dependency>
```

8.4 应用实例

```
<!-- https://mvnrepository.com/artifact/org.apache.flink/flink-streaming-scala -->
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-streaming-scala_2.12</artifactId>
            <version>1.11.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-clients_2.12</artifactId>
            <version>1.11.2</version>
        </dependency>
</dependencies>
```

8.4 应用实例

```
<build>
    <plugins>
        <plugin>
            <groupId>net.alchim31.maven</groupId>
            <artifactId>scala-maven-plugin</artifactId>
            <version>3.4.6</version>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
</project>
```

8.4 应用实例

接着在终端中执行下面的命令将项目打包成jar包：

```
$cd ~/FlinkCEP
$/usr/local/maven/bin/mvn package
```

最后将jar包提交给Flink运行：

```
$/usr/local/flink/bin/flink run -c \
> UserActionDetect ./target/simple-project-1.0.jar
```

运行成功会将会得到如下的结果：

```
5> ClickAndBuyAction(Adam,1558430815185,1558430815865)
6> ClickAndBuyAction(Carl,1558430816151,1558430816641)
```

8.5 本章小结

FlinkCEP是Flink生态系统中用于实现复杂事件处理的组件。与之前流式处理的过程不同，FlinkCEP需要用户自定义一个模式，根据这一模式就能自动寻找事件流中与模式相匹配的事件序列。Pattern API提供了丰富的模式定义形式，具有强大的表达能力。FlinkCEP提供了PatternSelectFunction对象用于抽取与模式相匹配的事件序列，同时也能提取出匹配结果中的超时事件序列。本章按照以上的顺序介绍了FlinkCEP中的各个部分，最后通过一个实例演示了编写FlinkCEP程序的基本步骤。

附录A：主讲教师林子雨简介

主讲教师：林子雨

单位：厦门大学计算机科学系
E-mail: ziyulin@xmu.edu.cn
个人网页：http://dblab.xmu.edu.cn/post/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn

扫一扫访问个人主页

林子雨，男，1978年出生，博士（毕业于北京大学），全国高校知名大数据教师，现为厦门大学计算机科学系副教授，曾任厦门大学信息科学与技术学院院长助理、晋江市发展和改革局副局长。中国计算机学会数据库专业委员会委员，中国计算机学会信息系统专业委员会委员。国内高校首个“数字教师”提出者和建设者，厦门大学数据库实验室负责人，厦门大学云计算与大数据研究中心主要建设者和骨干成员，2013年度、2017年度和2020年度厦门大学教学类奖教金获得者，荣获2019年福建省精品在线开放课程、2018年厦门大学高等教育成果特等奖、2018年福建省高等教育教学成果二等奖、2018年国家精品在线开放课程。主要研究方向为数据库、数据仓库、数据挖掘、大数据、云计算和物联网，并以第一作者身份在《软件学报》《计算机学报》和《计算机研究与发展》等国家重点期刊以及国际学术会议上发表多篇学术论文。作为项目负责人主持的科研项目包括1项国家自然科学青年基金项目(No.61303004)、1项福建省自然科学青年基金项目(No.2013J05099)和1项中央高校基本科研业务费项目(No.2011121049)，主持的教改课题包括1项2016年福建省教改课题和1项2016年教育部产学协作育人项目，同时，作为课题负责人完成了国家发改委城市信息化重大课题、国家物联网重大应用示范工程区域试点泉州市工作方案、2015泉州市互联网经济调研等课题。中国高校首个“数字教师”提出者和建设者，2009年至今，“数字教师”大平台累计向网络免费发布超过1000万字高价值的研究和教学资料，累计网络访问量超过1000万次。打造了中国高校大数据教学知名品牌，编著出版了中国高校第一本系统介绍大数据知识的专业教材《大数据技术原理与应用》，并成为京东、当当网等网店畅销书籍；建设了国内高校首个大数据课程公共服务平台，为教师教学和学生学习大数据课程提供全方位、一站式服务，年访问量超过200万次，累计访问量超过1000万次。

附录B：大数据学习路线图

大数据学习路线图访问地址：http://dblab.xmu.edu.cn/post/10164/

附录C：林子雨大数据系列教材

了解全部教材信息：http://dblab.xmu.edu.cn/post/bigdatabook/

附录D：《大数据导论（通识课版）》教材

开设全校公共选修课的优质教材

本课程旨在实现以下几个培养目标：

- 引导学生步入大数据时代，积极投身大数据的变革浪潮之中
- 了解大数据概念，培养大数据思维，养成数据安全意识
- 认识大数据伦理，努力使自己的行为符合大数据伦理规范要求
- 熟悉大数据应用，探寻大数据与自己专业的应用结合点
- 激发学生基于大数据的创新创业热情

高等教育出版社 ISBN:978-7-04-053577-8 定价：32元 版次：2020年2月第1版
教材官网：http://dblab.xmu.edu.cn/post/bigdataintroduction/

附录E：《大数据导论》教材

- 林子雨 编著《大数据导论》
- 人民邮电出版社，2020年9月第1版
- ISBN:978-7-115-54446-9 定价：49.80元

教材官网：http://dblab.xmu.edu.cn/post/bigdata-introduction/

开设大数据专业导论课的优质教材

扫一扫访问教材官网

附录F：《大数据技术原理与应用（第3版）》教材

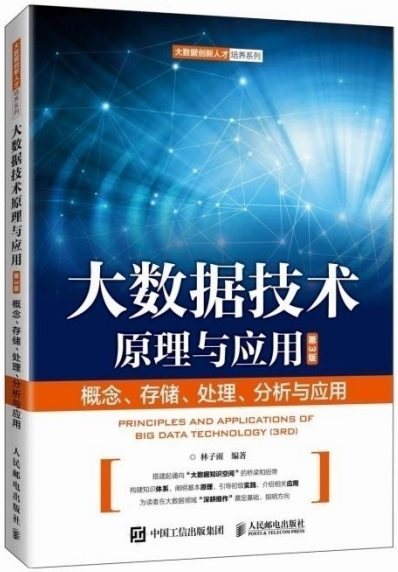
《大数据技术原理与应用——概念、存储、处理、分析与应用（第3版）》，由厦门大学计算机科学系林子雨博士编著，是国内高校第一本系统介绍大数据知识的专业教材。人民邮电出版社　ISBN:978-7-115-54405-6　定价：59.80元

全书共有17章，系统地论述了大数据的基本概念、大数据处理架构Hadoop、分布式文件系统HDFS、分布式数据 库HBase、NoSQL数据库、云数据库、分布式并行编程模型MapReduce、Spark、流计算、Flink、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在Hadoop、HDFS、HBase、MapReduce、Spark和Flink等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。

欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：
http://dblab.xmu.edu.cn/post/bigdata3

扫一扫访问教材官网



附录G：《大数据基础编程、实验和案例教程（第2版）》

本书是与《大数据技术原理与应用（第3版）》教材配套的唯一指定实验指导书

大数据教材

1+1黄金组合

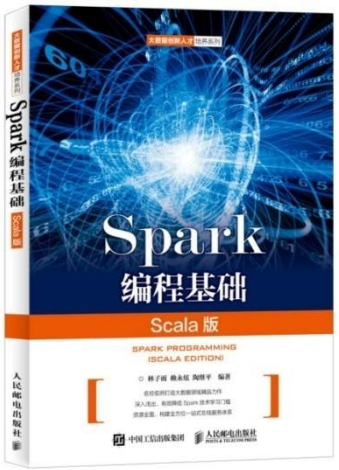
厦门大学林子雨编著

配套实验指导书

- 步步引导，循序渐进，详尽的安装指南为顺利搭建大数据实验环境铺平道路
- 深入浅出，去粗取精，丰富的代码实例帮助快速掌握大数据基础编程方法
- 精心设计，巧妙融合，八套大数据实验题目促进理论与编程知识的消化和吸收
- 结合理论，联系实际，大数据课程综合实验案例精彩呈现大数据分析全流程

林子雨编著《大数据基础编程、实验和案例教程（第2版）》
清华大学出版社 ISBN:978-7-302-55977-1 定价：69元 2020年10月第2版

附录H：《Spark编程基础（Scala版）》



《Spark编程基础（Scala版）》

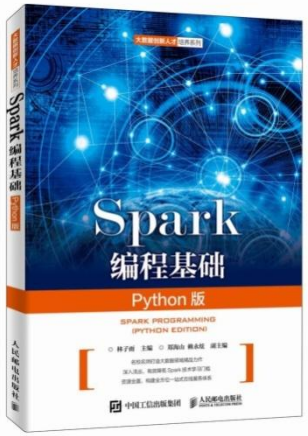
厦门大学 林子雨，赖永炫，陶继平 编著

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-48816-9
教材官网：http://dblab.xmu.edu.cn/post/spark/

本书以Scala作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Scala语言基础、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、授课视频、上机实验指南等。

附录I：《Spark编程基础（Python版）》



《Spark编程基础（Python版）》

厦门大学 林子雨，郑海山，赖永炫 编著

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-52439-3
教材官网： http://dblab.xmu.edu.cn/post/spark-python/

本书以Python作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Structured Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、上机实验指南等。

附录J：高校大数据课程公共服务平台

高校大数据课程

公 共 服 务 平 台

http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/

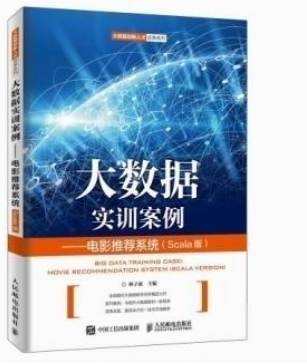
扫一扫访问平台主页

扫一扫观看3分钟FLASH动画宣传片

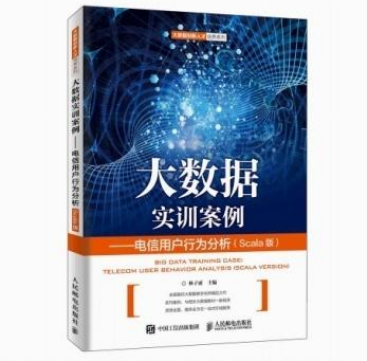
附录K：高校大数据实训课程系列案例教材

为了更好满足高校开设大数据实训课程的教材需求，厦门大学数据库实验室林子雨老师团队联合企业共同开发了《高校大数据实训课程系列案例》，目前已经完成开发的系列案例包括：

《电影推荐系统》（已经于2019年5月出版）
《电信用户行为分析》（已经于2019年5月出版）
《实时日志流处理分析》
《微博用户情感分析》
《互联网广告预测分析》
《网站日志处理分析》



系列案例教材将于2019年陆续出版发行，教材相关信息，敬请关注网页后续更新！
http://dblab.xmu.edu.cn/post/shixunkecheng/



扫一扫访问大数据实训课程系列案例教材主页

Department of Computer Science, Xiamen University, 2021