第3章 热门品类Top10分析

学习目标

- 掌握热门品类 Top10 分析实现思路。
- 掌握如何创建 Spark 连接并读取数据集。
- 掌握利用 Spark 获取业务数据。
- 掌握利用 Spark 统计品类的行为类型。
- 掌握利用 Spark 过滤品类的行为类型。
- 掌握利用 Spark 合并相同品类的行为类型。
- 掌握利用 Spark 根据品类的行为类型进行排序。
- 掌握将数据持久化到 HBase 数据库。
- 熟悉通过 Spark On YARN 运行程序。

品类指商品所属分类，用户在访问电商网站时，通常会产生很多行为，如查看商品的信息、将感兴趣的商品加入购物车和购买商品等，这些行为都将作为数据被网站存储。本章通过对电商网站存储的用户行为数据进行分析，从而统计出排名前十的热门品类。

3.1 数据集分析

某电商网站 2019 年 11 月产生的用户行为数据存储在文件 user_session.txt，该文件中的每一行数据都表示一个用户行为，所有行为都与商品和用户有关。由于原始数据集较大(13.7GB)，对硬件配置要求较高，考虑到读者学习的便捷性，从原始数据集中抽取了 500 万条数据(约 1GB)进行分析。虽然数据比较多，但是数据内容格式基本类似，这里选取其中一条数据进行分析，具体如下。

```
{"user_session":"0000007c-adbf-4ed7-af17-d1fef9763d67","event_type":"view","category_
id":"2053013553090134275","user_id":"560165420","product_id":"8900305","address_name":
"Maryland","event_time":"2019-11-18 09:16:19"}
```

上述数据包含很多字段，每个字段都代表特定的含义，具体介绍如下。

- user_session：用于标识用户行为的唯一值。
- event_type：表示用户行为的类型，包括 view(查看)、cart(加入购物车)和 purchase(购买)行为。

- category_id：表示商品品类 ID。
- user_id：表示用户 ID。
- product_id：表示商品 ID。
- address_name：表示产生事件的区域。
- event_time：表示产生事件的具体时间。

注：本书的配套资源会为读者提供数据集文件 user_session.txt。

3.2 实现思路分析

用户在访问电商网站时，通常会针对商品产生很多行为事件，如查看、加入购物车和购买。首先需要分别统计各个品类商品的查看次数、加入购物车次数以及购买次数。接下来，将同一品类中商品的查看、加入购物车以及购买次数进行合并。然后，自定义排序规则、按照各个品类中商品的查看、加入购物车和购买次数进行降序排序，获取排名前十的品类，就是热门品类 Top10。排序时，优先按照各个品类商品的查看次数降序排列，如果查看次数相同，则按照各个品类商品的加入购物车次数进行降序排列。如果查看次数和加入购车次数都相同，那么按照各品类商品的购买次数进行降序排列。最后，将同一品类中商品的查看、加入购物车和购买次数映射到自定义排序规则中进行排序处理。有关热门品类 Top10 的分析过程如图 3-1 所示。

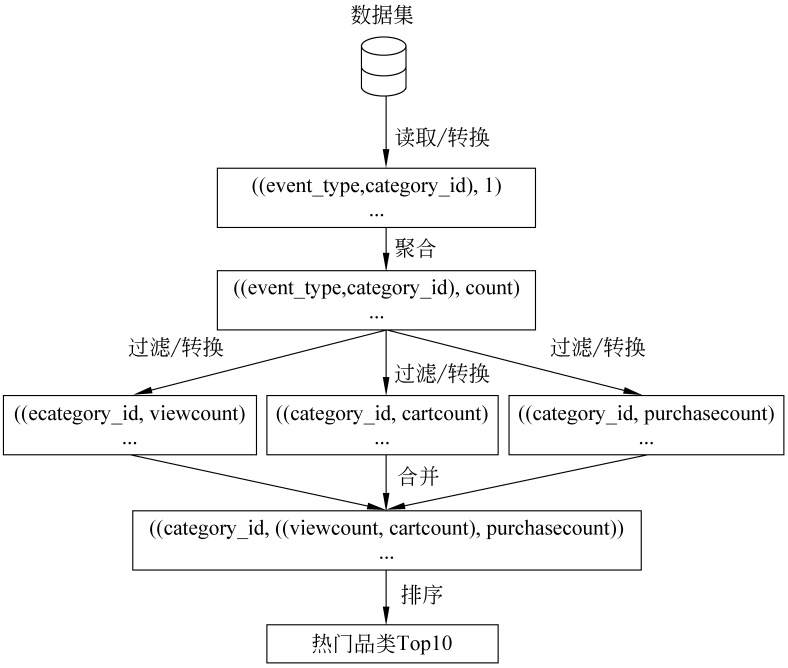

图 3-1 热门品类 Top10 的分析过程

针对图 3-1 中热门品类 Top10 的分析过程讲解如下。

- 读取/转换：读取数据集中的行为类型(event_type)和品类 ID(category_id)数据，为

了便于后续聚合处理时，将相同 Key 的 Value 值进行累加，计算每个品类中不同行为出现的总次数，这里需要对输出结果的数据格式进行转换处理，将行为类型和品类 ID 作为 Key，值 1 作为 Value。

- 聚合：统计各个品类的查看、加入购物车和购买次数。
- 过滤/转换：将聚合结果进行过滤处理，并分为 3 部分数据，第一部分数据包含各个品类查看次数，第二部分数据包含各个品类加入购物车次数，第三部分包含各个品类购买次数。对过滤后的 3 部分数据进行转换处理，去除数据中的行为类型字段。此步目的是后续合并操作时，明确同一品类中不同行为类型所处位置。
- 合并：将 Key 值相同的 Value 进行合并处理，目的是将相同品类的查看次数、加入购物车次数和购买次数合并到一行。
- 排序：对每个品类中查看次数(viewcount)、加入购物车次数(cartcount)和购买次数(purchasecount)进行排序处理，在排序过程会涉及 3 类值的排序，因此这里需要使用 Spark 的二次排序，在排序操作时使用自定义排序的方式进行处理。

3.3　实现热门品类 Top10

实现热门品类 Top10 分析的程序由 Java 编程语言实现。目前，Java 的主流开发工具有两种：Eclipse 工具和 IntelliJ IDEA 工具。我们可以在这两种开发工具中编写 Java 代码来实现热门品类 Top10 分析。由于 IntelliJ IDEA 工具内置了很多优秀的插件，在智能代码助手、代码自动提示、重构、CVS 整合、代码分析等方面有着不错的表现，因此本项目将使用 IntelliJ IDEA 作为 Java 开发工具。

3.3.1　创建项目

本项目使用的 IntelliJ IDEA 版本为 2018.3，读者可通过 IntelliJ IDEA 官网下载并安装程序，关于 IntelliJ IDEA 的下载安装这里不做赘述(注意：安装 IntelliJ IDEA 之前需要安装 JDK 并在系统环境变量中配置 JDK，本项目使用的 JDK 版本为 1.8)。

Maven 便于维护和管理项目依赖，因此本项目将通过构建 Maven 现目实现相关需求。接下来，详细讲解如何在 IntelliJ IDEA 中构建 Maven 项目 SparkProject，具体步骤如下。

1. 创建 Maven 项目

打开 IntelliJ IDEA 开发工具，进入 IntelliJ IDEA 欢迎界面，具体如图 3-2 所示。

在图 3-2 中单击 Configure 右侧的下拉箭头，依次选择 Project Defaults→Project Structure 命令，配置项目使用的 JDK，如图 3-3 所示。

在图 3-3 中配置完 JDK 后，单击 OK 按钮返回 IntelliJ IDEA 欢迎界面。

单击图 3-2 中的 Create New Project 按钮创建新项目，在弹出的 New Project 窗口左侧选择 Maven，即创建 Maven 项目，如图 3-4 所示。

在图 3-4 中，单击 Next 按钮，配置 Maven 项目的组织名(GroupId)和项目工程名(ArtifactId)，如图 3-5 所示。

图 3-2 IntelliJ IDEA 欢迎界面

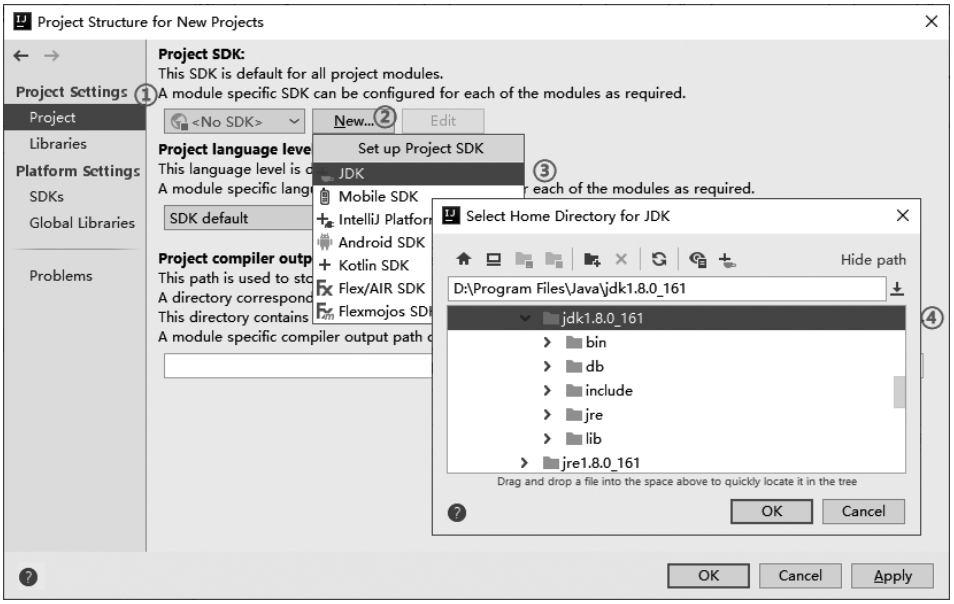

图 3-3 配置项目使用的 JDK

在图 3-5 中，单击 Next 按钮，配置项目名称（Project name）和项目本地的存放目录（Project location），如图 3-6 所示。

在图 3-6 中，单击 Finish 按钮，完成项目 SparkProject 的创建。项目 SparkProject 的初始结构如图 3-7 所示。

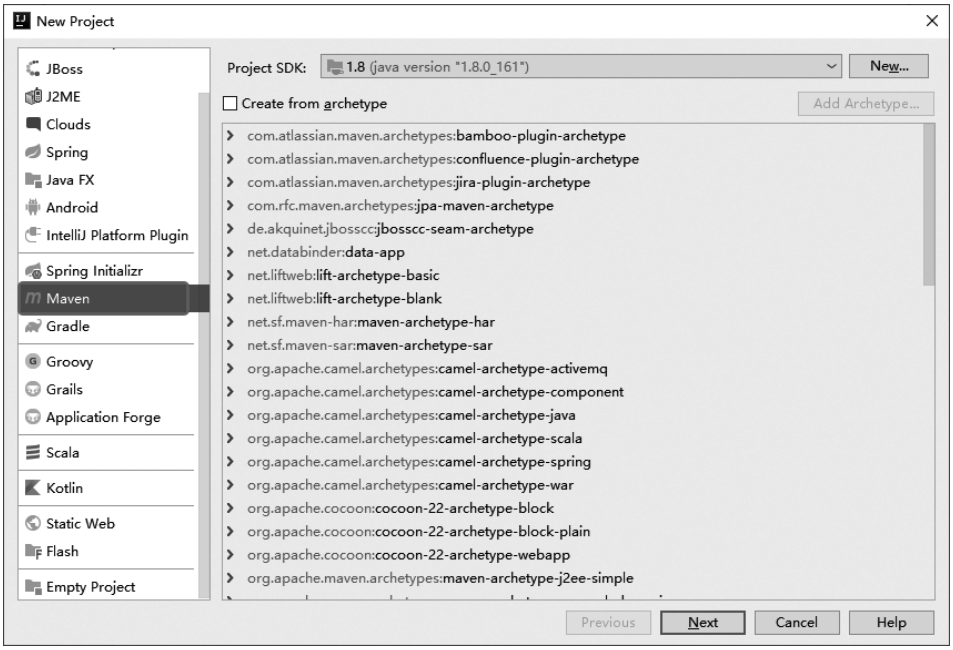

图 3-4　创建 Maven 项目

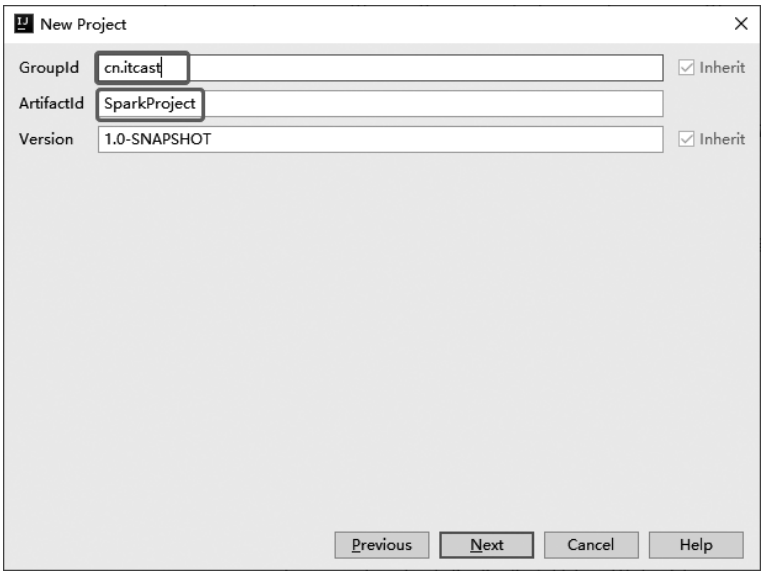

图 3-5　配置组织名和工程名

2. 导入依赖

本项目所需要的依赖包括 JSON、HBase 和 Spark，在文件 pom.xml 中添加这些依赖方式，具体代码如文件 3-1 所示。

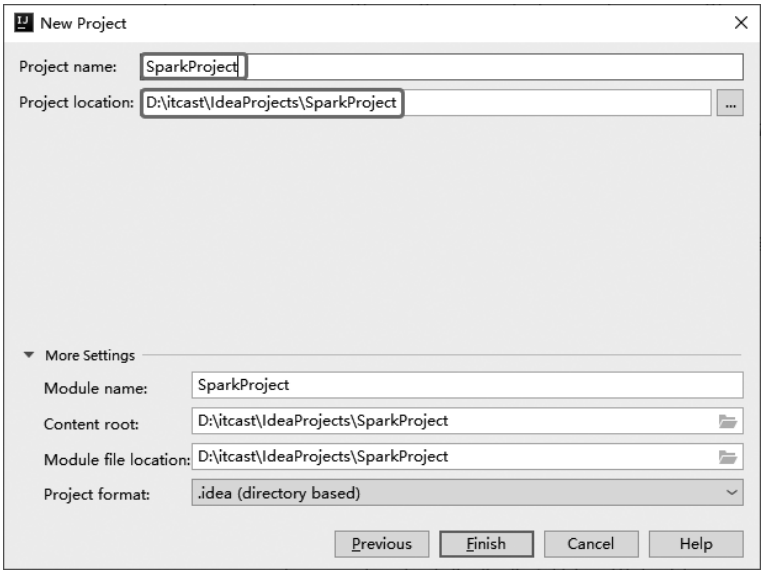

图 3-6　配置项目名称和本地存放目录

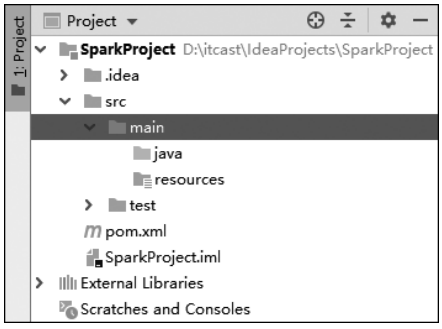

图 3-7　项目 SparkProject 的初始结构

文件 3-1　pom.xml

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>io.netty</groupId>
            <artifactId>netty-all</artifactId>
            <version>4.1.18.Final</version>
        </dependency>
    </dependencies>
</dependencyManagement>
<dependencies>
      <!--JSON 依赖-->
      <dependency>
           <groupId>com.alibaba</groupId>
```

```
14              <artifactId>fastjson</artifactId>
15              <version>1.2.62</version>
16        </dependency>
17        <!--HBase 依赖-->
18        <dependency>
19              <groupId>org.apache.hbase</groupId>
20              <artifactId>hbase-client</artifactId>
21              <version>1.2.1</version>
22        </dependency>
23        <dependency>
24              <groupId>org.apache.hbase</groupId>
25              <artifactId>hbase-common</artifactId>
26              <version>1.2.1</version>
27        </dependency>
28        <!--Spark 依赖-->
29        <dependency>
30              <groupId>org.apache.spark</groupId>
31              <artifactId>spark-core_2.11</artifactId>
32              <version>2.3.2</version>
33              <exclusions>
34                    <exclusion>
35                          <groupId>io.netty</groupId>
36                          <artifactId>netty</artifactId>
37                    </exclusion>
38              </exclusions>
39        </dependency>
40  </dependencies>
41  <build>
42        <plugins>
43              <plugin>
44                    <groupId>org.apache.maven.plugins</groupId>
45                    <artifactId>maven-compiler-plugin</artifactId>
46                    <configuration>
47                          <source>1.8</source>
48                          <target>1.8</target>
49                    </configuration>
50              </plugin>
51              <plugin>
52                 <artifactId>maven-assembly-plugin</artifactId>
53                 <configuration>
54                    <appendAssemblyId>false</appendAssemblyId>
55                    <descriptorRefs>
56                       <descriptorRef>jar-with-dependencies</descriptorRef>
57                    </descriptorRefs>
58                    <archive>
59                        <manifest>
60                            <!--此处指定 main 方法入口的 class -->
61                     <mainClass>cn.itcast.top10.CategoryTop10</mainClass>
62                        </manifest>
63                    </archive>
```

```
64                </configuration>
65                <executions>
66                    <execution>
67                        <id>make-assembly</id>
68                        <phase>package</phase>
69                        <goals>
70                            <goal>assembly</goal>
71                        </goals>
72                    </execution>
73                </executions>
74            </plugin>
75        </plugins>
76    </build>
```

文件 3-1 中：第 1～9 行代码主要是对项目中 Netty 依赖进行多版本管理，避免本地运行出现多个版本的 Netty，导致程序出现 NoSuchMethodError 异常；第 12～16 行代码引入 JSON 依赖，用于解析 JSON 数据；第 18～27 行代码引入 HBase 依赖，用于操作 HBase 数据库；第 29～39 行代码引入 Spark 依赖，用于开发 Spark 数据分析程序；第 43～50 行代码指定 Maven 编译的 JDK 版本，如果不指定，Maven 3 默认用 JDK 1.5，Maven 2 默认用 JDK 1.3；第 51～74 行代码配置程序打包方式并指定程序主类。

3. 创建项目目录

在项目 SparkProject 中右击 java 目录，在弹出的快捷菜单中依次选择 New→Package，从而新建 Package 包，具体如图 3-8 所示。

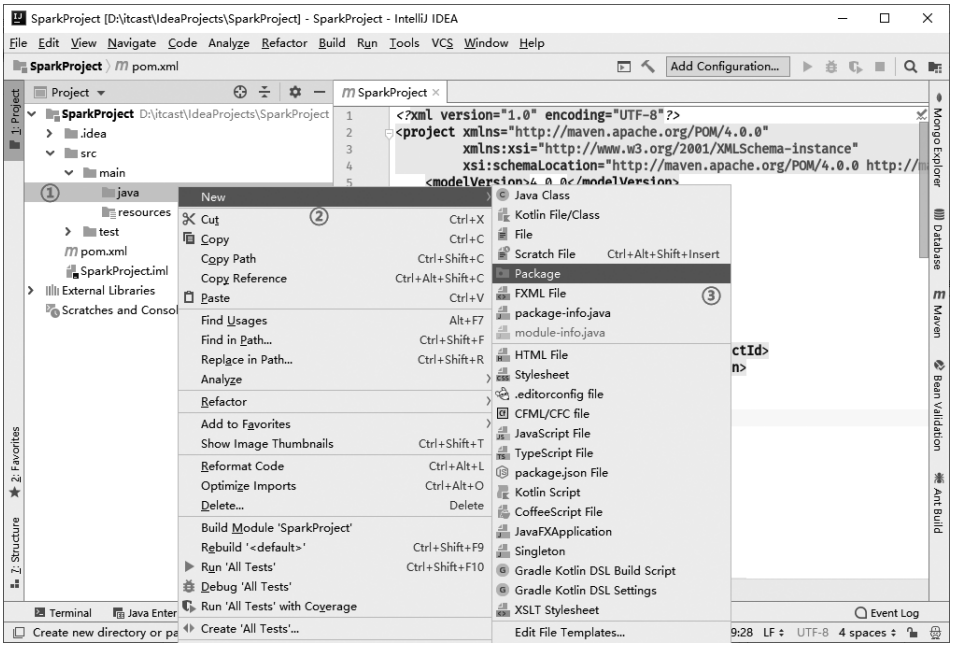

图 3-8　新建 Package 包的步骤

通过如图 3-8 所示的操作后，会弹出 New Package 对话框，在文本输入框 Enter new package name 中输入 cn.itcast.top10 设置 Package 名称，用于存放实现热门品类 Top10 分析的 Java 文件，如图 3-9 所示。

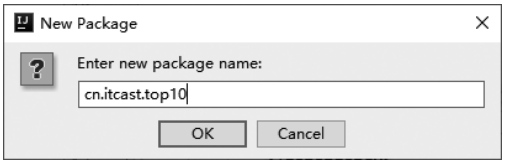

图 3-9　设置 Package 名称

在图 3-9 中单击 OK 按钮完成 Package 包的创建。

4. 创建程序主类

右击包 cn.itcast.top10，在弹出的快捷菜单中依次选择 New→Java Class 新建 Java 类，具体如图 3-10 所示。

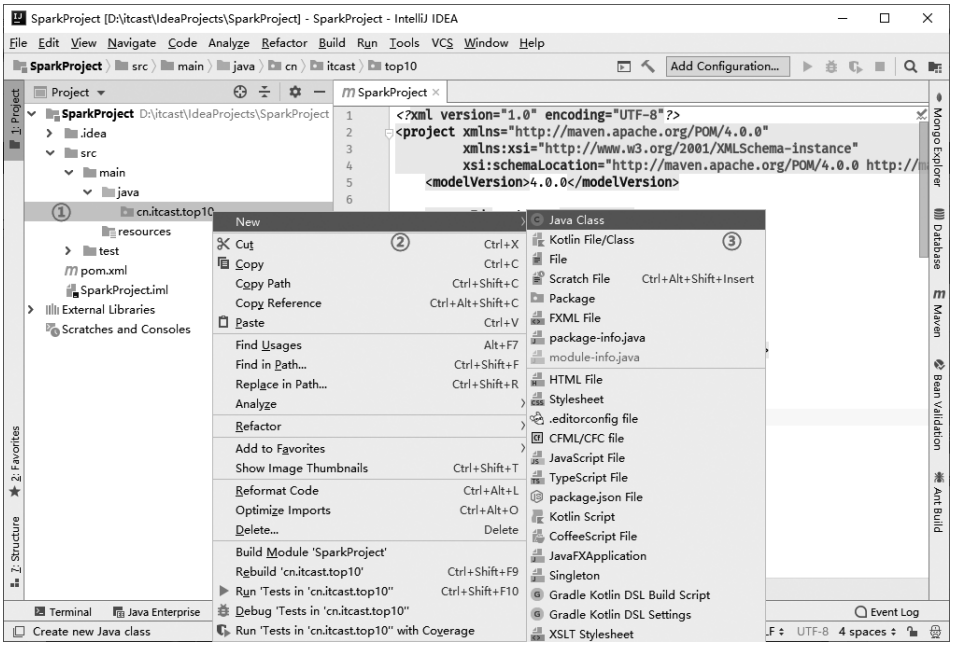

图 3-10　新建 Java 类

通过如图 3-10 所示的操作后，会弹出 Create New Class 对话框，在文本框 Name 中输入 CategoryTop10 设置类名称，在类中实现热门品类 Top10 分析，具体如图 3-11 所示。

3.3.2　创建 Spark 连接并读取数据集

在类 CategoryTop10 中定义 main()方法，该方法是 Java 程序执行的入口，在 main()方法中实现 Spark 程序，具体代码如文件 3-2 所示。

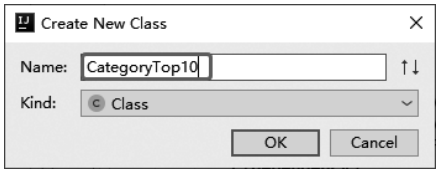

图 3-11 设置 Java 类名称

文件 3-2 CategoryTop10.java

```
public class CategoryTop10{
    public static void main(String[] arg){
    //实现热门品类 Top10 分析
    }
}
```

在文件 3-2 的 main()方法中，创建 JavaSparkContext 和 SparkConf 对象，JavaSparkContext 对象用于实现 Spark 程序，SparkConf 对象用于配置 Spark 程序相关参数，具体代码如下。

```
SparkConf conf =new SparkConf();
//设置 Application 名称为 top10_category
conf.setAppName("top10_category");
JavaSparkContext sc =new JavaSparkContext(conf);
```

在文件 3-2 的 main()方法中，调用 JavaSparkContext 对象的 textFile()方法读取外部文件，将文件中的数据加载到 textFileRDD，具体代码如下。

```
JavaRDD<String> textFileRDD =sc.textFile(arg[0]);
```

上述代码中，通过变量 arg[0]指定文件路径，目的是执行提交 Spark 程序到 YARN 集群运行的命令中，通过参数指定文件路径。

3.3.3 获取业务数据

在文件 3-2 的 main()方法中，使用 mapToPair()算子转换 textFileRDD 的每一行数据，用于获取每一行数据中的行为类型和品类 ID 数据，将转换结果加载到 transformRDD，具体代码如下。

```
JavaPairRDD<Tuple2<String,String>,Integer> transformRDD =textFileRDD
        .mapToPair(
                new PairFunction<
                        String,
                        Tuple2<String, String>, Integer>() {
    @Override
    public Tuple2<Tuple2<String, String>, Integer> call(String s)
            throws Exception {
```

```
            //将数据转换为 JSON 对象
            JSONObject json =JSONObject.parseObject(s);
            String category_id =json.getString("category_id");
            String event_type =json.getString("event_type");
            return new Tuple2<>(
                    new Tuple2<>(category_id,event_type),
                    new Integer(1));
        }
    });
```

上述代码中，首先将 textFileRDD 中的每一行数据转换为 JSON 对象；然后获取 JSON 对象中的 category_id(品类 ID)和 event_type(行为类型)；最后将 category_id、event_type 和值 1 添加到 Tuple2 对象中。

3.3.4　统计品类的行为类型

在文件 3-2 的 main()方法中，使用 reduceByKey()算子对 transformRDD 进行聚合操作，用于统计每个品类中商品被查看、加入购物车和购买的次数，将统计结果加载到 aggregationRDD，具体代码如下。

```
JavaPairRDD<Tuple2<String, String>, Integer> aggregationRDD =
        transformRDD.reduceByKey(
                new Function2<Integer, Integer, Integer>() {
    @Override
    public Integer call(Integer integer1, Integer integer2)
            throws Exception {
        return integer1 +integer2;
    }
});
```

3.3.5　过滤品类的行为类型

在文件 3-2 的 main()方法中，首先使用 filter()算子过滤 aggregationRDD 每一行数据中行为类型为加入购物车和购买的数据，只保留行为类型为查看的数据，然后使用 mapToPair()算子对过滤结果进行转换，获取每一行数据中品类被查看次数和品类 ID 数据，最终将转换结果加载到 getViewCategoryRDD，具体代码如下。

```
JavaPairRDD<String,Integer> getViewCategoryRDD =aggregationRDD
    .filter(new Function<Tuple2<Tuple2<String, String>, Integer>
                        , Boolean>() {
        @Override
        public Boolean call(Tuple2<Tuple2<String, String>
                , Integer> tuple2) throws Exception {
            //获取行为类型
            String action =tuple2._1._2;
            return action.equals("view");
        }
```

```
11        }).mapToPair(
12                new PairFunction<Tuple2<Tuple2<String, String>
13                        , Integer>, String, Integer>() {
14                    @Override
15                    public Tuple2<String, Integer>
16                    call(Tuple2<Tuple2<String, String>, Integer> tuple2)
17                            throws Exception {
18                        return new Tuple2<>(tuple2._1._1,tuple2._2);
19                    }
20                });
```

上述代码中，第 9 行通过 equals()方法判断获取的行为类型是否为 view(查看)并将判断结果作为返回值，若返回值为 True，则进行后续转换操作。

在文件 3-2 的 main()方法中，首先使用 filter()算子过滤 aggregationRDD 每一行数据中行为类型为查看和购买的数据，只保留行为类型为加入购物车的数据，然后使用 mapToPair()算子对过滤结果进行转换，获取每一行数据中品类被加入购物车的次数和品类 ID 数据，最终将转换结果加载到 getCartCategoryRDD，具体代码如下。

```
1    JavaPairRDD<String,Integer> getCartCategoryRDD =aggregationRDD
2            .filter(new Function<Tuple2<Tuple2<String, String>, Integer>
3                    , Boolean>() {
4                @Override
5                public Boolean call(Tuple2<Tuple2<String, String>
6                        , Integer> tuple2) throws Exception {
7                    String action =tuple2._1._2;
8                    return action.equals("cart");
9                }
10           }).mapToPair(new PairFunction<Tuple2<Tuple2<String, String>
11                   , Integer>, String, Integer>() {
12               @Override
13               public Tuple2<String, Integer>
14               call(Tuple2<Tuple2<String, String>, Integer> tuple2)
15                       throws Exception {
16                   return new Tuple2<>(tuple2._1._1,tuple2._2);
17               }
18           });
```

上述代码中，第 8 行通过 equals()方法判断获取的行为类型是否为 cart(加入购物车)并将判断结果作为返回值，若返回值为 True，则进行后续转换操作。

在文件 3-2 的 main()方法中，首先使用 filter()算子过滤 aggregationRDD 每一行数据中行为类型为查看和加入购物车的数据，只保留行为类型为购买的数据，然后使用 mapToPair()算子对过滤结果进行转换，获取每一行数据中品类被购买次数和品类 ID 数据，最终将转换结果加载到 getPurchaseCategoryRDD，具体代码如下。

```
JavaPairRDD<String,Integer> getPurchaseCategoryRDD =aggregationRDD
    .filter(new Function<Tuple2<Tuple2<String, String>, Integer>
            , Boolean>() {
        @Override
        public Boolean call(Tuple2<Tuple2<String, String>
                , Integer> tuple2) throws Exception {
            String action =tuple2._1._2;
            return action.equals("purchase");
        }
    }).mapToPair(new PairFunction<Tuple2<Tuple2<String, String>
            , Integer>, String, Integer>() {
        @Override
        public Tuple2<String, Integer>
        call(Tuple2<Tuple2<String, String>, Integer> tuple2)
                throws Exception {
            return new Tuple2<>(tuple2._1._1,tuple2._2);
        }
    });
```

上述代码中，第 8 行通过 equals()方法判断获取的行为类型是否为 purchase(购买)并将判断结果作为返回值，若返回值为 True，则进行后续转换操作。

3.3.6　合并相同品类的行为类型

在文件 3-2 的 main()方法中，使用 leftOuterJoin()(左外连接)算子合并 getViewCategoryRDD、getCartCategoryRDD 和 getPurchaseCategoryRDD，用于合并同一品类的查看次数、加入购物车次数和购买次数，将合并结果加载到 joinCategoryRDD，具体代码如下。

```
JavaPairRDD<String,Tuple2<Integer, Optional<Integer>>>
        tmpJoinCategoryRDD =getViewCategoryRDD
        .leftOuterJoin(getCartCategoryRDD);
JavaPairRDD<String,
        Tuple2<Tuple2<Integer, Optional<Integer>>,
                Optional<Integer>>> joinCategoryRDD =
        tmpJoinCategoryRDD.leftOuterJoin(getPurchaseCategoryRDD);
```

上述代码中，首先通过 leftOuterJoin()算子合并 getViewCategoryRDD 和 getCartCategoryRDD，将合并结果加载到 tmpJoinCategoryRDD，然后通过 leftOuterJoin()算子合并 tmpJoinCategoryRDD 和 getPurchaseCategoryRDD，将合并结果加载到 joinCategoryRDD。

Optional 类是一个包含有可选值的包装类，它既可以含有对象也可以为空，主要为了解决空指针异常的问题，因为某些品类中的商品可能被查看但并未被购买或加入购物车。

3.3.7　根据品类的行为类型进行排序

在包 cn.itcast.top10 中创建文件 CategorySortKey.java，用于实现自定义排序。在类 CategorySortKey 中继承比较器接口 Comparable 和序列化接口 Serializable，并实现

Comparable 接口的 compareTo()方法，具体代码如文件 3-3 所示。

文件 3-3 CategorySortKey.java

```
import java.io.Serializable;
public class CategorySortKey implements Comparable<CategorySortKey>
        ,Serializable{
    //查看次数
    private int viewCount;
    //加入购物车次数
    private int cartCount;
    //购买次数
    private int purchaseCount;
    //定义类的构造方法
    public CategorySortKey(
            int viewcount,
            int cartCount,
            int purchaseCount)
    {
        this.viewCount =viewcount;
        this.cartCount =cartCount;
        this.purchaseCount =purchaseCount;
    }
    //定义属性的 getter 和 setter 方法
    ⋮
    @Override
    public int compareTo(CategorySortKey other) {
        if(viewCount -other.getViewCount() !=0) {
            return (int) (viewCount -other.getViewCount());
        } else if(cartCount -other.getCartCount() !=0) {
            return (int) (cartCount -other.getCartCount());
        } else if(purchaseCount -other.getPurchaseCount() !=0) {
            return (int) (purchaseCount -other.getPurchaseCount());
        }
        return 0;
    }
}
```

在文件 3-3 中，第 22～32 行代码，重写接口 Comparable 的 compareTo()方法，在方法内部实现对象的比较，比较的规则为返回值等于 0 表示相等；返回值小于 0 表示小于；返回值大于 0 表示大于。比较的优先级按照 viewCount、cartCount 和 purchaseCount 的顺序。

在文件 3-2 的 main()方法中，使用 mapTopair()算子转换 joinCategoryRDD，将 joinCategoryRDD 中品类被查看次数、加入购物车次数和购买次数映射到自定义排序类 CategorySortKey，通过 transCategoryRDD 加载转换结果，具体代码如下。

```
JavaPairRDD<CategorySortKey,String> transCategoryRDD =joinCategoryRDD
    .mapToPair(new PairFunction<Tuple2<String,
     Tuple2<Tuple2<Integer, Optional<Integer>>,Optional<Integer>>>,
```

```
4      CategorySortKey,String>() {
5      @Override
6      public Tuple2<CategorySortKey,String> call(Tuple2<String,
7              Tuple2<Tuple2<Integer, Optional<Integer>>,
8              Optional<Integer>>> tuple2) throws Exception {
9                   String category_id =tuple2._1;
10                  int viewcount =tuple2._2._1._1;
11                  int cartcount =0;
12                  int purchasecount =0;
13                  //判断品类被加入购物车次数是否为空
14                  if (tuple2._2._1._2.isPresent()){
15                      cartcount =tuple2._2._1._2.get().intValue();
16                  }
17                  //判断品类被购买次数是否为空
18                  if (tuple2._2._2.isPresent()){
19                      purchasecount =tuple2._2._2.get().intValue();
20                  }
21                  /*将 viewcount、cartcount 和 purchasecount 映射到
22                  类 CategorySortKey 的构造方法中*/
23                  CategorySortKey sortKey =
24          new CategorySortKey(viewcount, cartcount, purchasecount);
25                  return new Tuple2<>(sortKey,category_id);
26              }
27          });
```

上述代码中的 isPresent()方法用于判断 Optional 类型的数据是否为空，若值为空则通过 get()方法获取值，并通过 intValue()方法指定获取的值为 Int 类型。

在文件 3-2 的 main()方法中，通过 sortByKey()算子对 transCategoryRDD 进行排序操作，使 transCategoryRDD 中品类被查看次数、加入购物车次数和购买次数根据自定义排序类 CategorySortKey 指定的排序规则进行排序，将排序结果加载到 sortedCategoryRDD，具体代码如下。

```
JavaPairRDD<CategorySortKey,String> sortedCategoryRDD =
                transCategoryRDD.sortByKey(false);
```

上述代码中，sortByKey()算子的参数为 false，表示使用自定义排序类的比较方式进行排序。

在文件 3-2 的 main()方法中，使用 take()算子获取 sortedCategoryRDD 前 10 个元素，即热门品类 Top10 分析结果，将分析结果加载到 top10CategoryList，具体代码如下。

```
List<Tuple2<CategorySortKey, String>> top10CategoryList =
              sortedCategoryRDD.take(10);
```

上述代码中，take()算子的参数为 10，表示获取 sortedCategoryRDD 前 10 个元素。

3.3.8　数据持久化

本项目使用 HBase 数据库作为数据持久化工具，HBase 分布式数据库通过 HDFS 和

ZooKeeper 实现数据的高可用和冗余，从而确保数据库和数据的安全性。接下来，分步骤讲解如何将热门品类 Top10 分析结果持久化到 HBase 数据库中。

1. 封装 HBase 工具类

为了避免后续环节重复编写数据库连接和数据库操作的相关代码，这里将 HBase 数据库连接工具类和 HBase 数据库操作工具类进行封装，具体实现步骤如下。

(1) 在项目 SparkProject 的 java 目录新建 Package 包 cn.itcast.hbase，用于存放实现数据持久化的 Java 文件。在包 cn.itcast.hbase 下创建文件 HbaseConnect.java，用于实现封装 HBase 数据库连接工具类，具体代码如文件 3-4 所示。

文件 3-4　HbaseConnect.java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.MasterNotRunningException;
import org.apache.hadoop.hbase.ZooKeeperConnectionException;
import org.apache.hadoop.hbase.client.Connection;
import org.apache.hadoop.hbase.client.ConnectionFactory;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import java.io.IOException;
public class HbaseConnect {
    public static Configuration conf;
    public static Connection conn;
    public static HBaseAdmin hbaseAdmin;
    static {
        //创建 HBase 配置信息
        conf =HBaseConfiguration.create();
        //配置 ZooKeeper 集群地址
        conf.set("hbase.zookeeper.quorum", "spark01,spark02,spark03");
        //配置 ZooKeeper 端口号
        conf.set("hbase.zookeeper.property.clientPort", "2181");
        try {
            //通过 HBase 配置获取 HBase 数据库连接对象
            conn =ConnectionFactory.createConnection(conf);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public static HBaseAdmin getHBaseAdmin() throws IOException{
        try {
            //获取 HBase 数据库操作对象
            hbaseAdmin =(HBaseAdmin)(conn.getAdmin());
        } catch (MasterNotRunningException e) {
            e.printStackTrace();
        } catch (ZooKeeperConnectionException e) {
            e.printStackTrace();
        }
        return hbaseAdmin;
```

```
37        }
38        public static Connection getConnection(){
39            return conn;
40        }
41        public static synchronized void closeConnection(){
42            if(conn!=null){
43                try {
44                    conn.close();
45                } catch (IOException e) {
46                    e.printStackTrace();
47                }
48            }
49        }
50    }
```

在文件 3-3 中，第 38～40 行代码创建返回值类型为 Connection 的方法 getConnection()，用于获取 HBase 数据库连接；第 41～49 行代码创建方法 closeConnection()，用于关闭 HBase 数据库连接。

需要注意的是，若运行项目 SparkProject 的环境中未配置 IP 映射，则需要在配置 Zookeeper 集群地址时使用 IP 地址而不是主机名。

（2）在项目 SparkProject 的包 cn.itcast.hbase 下创建文件 HbaseUtils.java，用于实现封装 HBase 数据库操作工具类，具体代码如文件 3-5 所示。

文件 3-5　HbaseUtils.java

```
1    import org.apache.hadoop.hbase.HColumnDescriptor;
2    import org.apache.hadoop.hbase.HTableDescriptor;
3    import org.apache.hadoop.hbase.TableName;
4    import org.apache.hadoop.hbase.client.HBaseAdmin;
5    import org.apache.hadoop.hbase.client.Put;
6    import org.apache.hadoop.hbase.client.Table;
7    import org.apache.hadoop.hbase.util.Bytes;
8    import java.io.IOException;
9    public class HbaseUtils {
10       public static void createTable(String tableName,
11                                      String... columFamilys)
12               throws IOException {
13           //获取 HBase 数据表操作对象
14           HBaseAdmin admin =HbaseConnect.getHBaseAdmin();
15           //判断表是否存在
16           if (admin.tableExists(tableName)){
17               //关闭表
18               admin.disableTable(tableName);
19               //删除表
20               admin.deleteTable(tableName);
21           }
22           //HTableDescriptor 类包含了表的名字以及表的列族信息
```

```
23          HTableDescriptor hd
24                  =new HTableDescriptor(TableName.valueOf(tableName));
25          for (String cf : columFamilys) {
26              hd.addFamily(new HColumnDescriptor(cf));
27          }
28          //通过 createTable()方法创建 HBase 数据表
29          admin.createTable(hd);
30          admin.close();
31      }
32      public static void putsToHBase(String tableName,
33                                     String rowkey,
34                                     String cf,
35                                     String[] column,
36                                     String[] value)
37              throws Exception {
38          //获取指定 HBase 数据表的操作对象
39          Table table =HbaseConnect
40                  .getConnection()
41                  .getTable(TableName.valueOf(tableName));
42          //通过 Put 对象存储插入数据表的内容
43          Put puts =new Put(rowkey.getBytes());
44          for (int i =0;i<column.length;i++){
45              puts.addColumn(
46                      Bytes.toBytes(cf),
47                      Bytes.toBytes(column[i]),
48                      Bytes.toBytes(value[i]));
49          }
50          //向指定数据表中插入数据
51          table.put(puts);
52          table.close();
53      }
54  }
```

在文件 3-5 中，第 10～31 行代码定义方法 createTable()，用于创建 HBase 数据表。方法 createTable()包含参数 tableName 和 columFamilys，其中参数 tableName 表示数据表名称，参数 columFamilys 表示列族；第 32～52 行代码定义方法 putsToHBase()，用于向指定 HBase 数据表中插入数据。方法 putsToHBase()包含参数 tableName、rowkey、cf、column 和 value，其中参数 tableName 表示数据表名称，参数 rowkey 表示行键，参数 cf 表示列族，参数 column 表示行，参数 value 表示值。

2. 持久化热门品类 Top10 分析结果

在文件 3-2 的类 CategoryTop10 中添加方法 top10ToHbase()，用于将热门品类 Top10 分析结果持久化到 HBase 数据库中，该方法包含参数 top10CategoryList，表示热门品类 Top10 分析结果数据，具体代码如下。

```
1   public static void top10ToHbase(List<Tuple2<CategorySortKey, String>>
2                                       top10CategoryList) throws Exception
3   {
4       //创建数据表 top10 和列族 top10_category
5       HbaseUtils.createTable("top10","top10_category");
6       //创建数组 column,用于存储数据表 top10 的列名
7       String[] column =
8               {"category_id","viewcount","cartcount","purchasecount"};
9       String viewcount ="";
10      String cartcount ="";
11      String purchasecount ="";
12      String category_id ="";
13      int count =0;
14      //遍历集合 top10CategoryList
15      for (Tuple2<CategorySortKey, String> top10: top10CategoryList) {
16          count++;
17          //获取查看次数
18          viewcount =String.valueOf(top10._1.getViewCount());
19          //获取加入购物车次数
20          cartcount =String.valueOf(top10._1.getCartCount());
21          //获取购买次数
22          purchasecount =String.valueOf(top10._1.getPurchaseCount());
23          //获取品类 ID
24          category_id =top10._2;
25          //创建数组 value,用于存储数据表 top10 的值
26          String[] value =
27                  {category_id,viewcount,cartcount,purchasecount};
28          HbaseUtils.putsToHBase("top10",
29                  "rowkey_top"+count,
30                  "top10_category",
31                  column,
32                  value);
33      }
34  }
```

上述代码中,第 28～32 行,调用 HBase 数据库操作工具类的 putToHBase()方法,用于持久化热门品类 Top10 数据。putToHBase()方法包含 5 个参数：其中第 1 个参数为字符串 top10,表示数据表名称;第 2 个参数为字符串对象 count 和字符串 rowkey_top,表示数据表的行键;第 3 个参数为字符串 top10_category,表示数据表的列族;第 4 个参数为数组 column,数组中的每一个元素表示数据表的列名;第 5 个参数为数组 value,数组中的每一个元素表示数据表的值。

在文件 3-2 的 main()方法中,调用方法 top10ToHbase()并传入参数 top10CategoryList,用于在 Spark 程序中实现 top10ToHbase()方法,将热门品类 Top10 分析结果持久化到 HBase 数据库中的数据表 top10,具体代码如下。

```
//通过 try…catch 抛出异常
try {
    top10ToHbase(top10CategoryList);
} catch (Exception e) {
    e.printStackTrace();
}
//关闭 HBase 数据库连接
HbaseConnect.closeConnection();
//关闭 JavaSparkContext 连接
sc.close();
```

3.4 运行程序

热门品类 Top10 分析程序编写完成后，需要在 IntelliJ IDEA 中将程序封装成 jar 包，并上传到集群环境中，通过 spark-submit 将程序提交到 YARN 中运行，具体步骤如下。

1. 封装 jar 包

在 IntelliJ IDEA 主界面单击右侧 Maven 选项卡打开 Maven 窗口，如图 3-12 和图 3-13 所示。

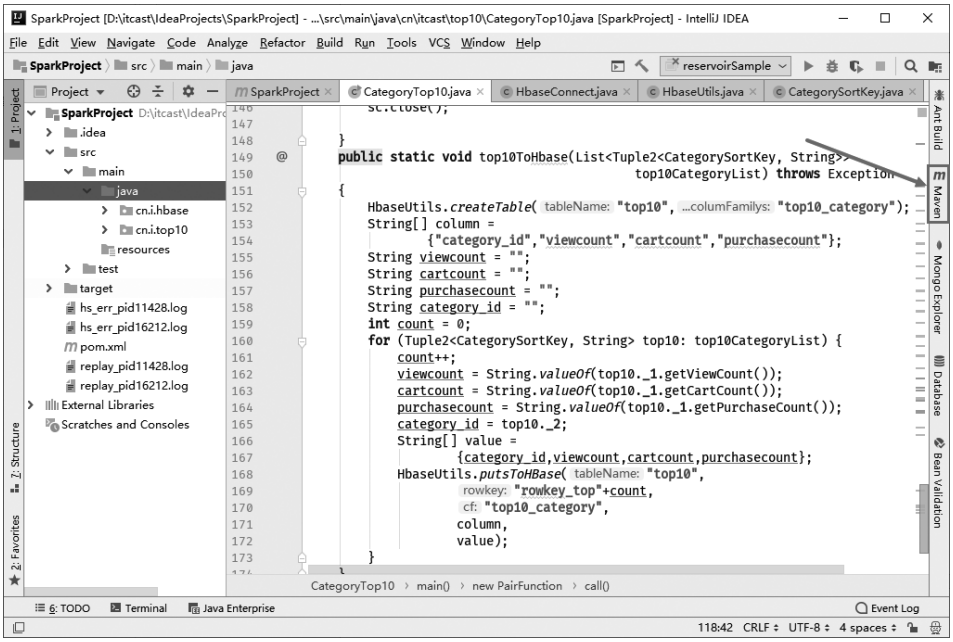

图 3-12　Maven 选项卡

在 Maven 窗口单击，展开 Lifecycle 目录，如图 3-14 所示。

双击 Lifecycle 目录中的 package 选项，IntelliJ IDEA 会自动将程序封装成 jar 包，封装完成后，若出现 BUILD SUCCESS 内容，则证明成功封装热门品类 Top10 分析程序为 jar