第 5 章 HBase过滤器

学习目标

- 了解过滤器原理，能够说出基于过滤器查询数据的流程。
- 掌握过滤器的使用，能够灵活运用不同类型的过滤器查询数据。

过滤器(Filter)是一种在 HBase 中用于查询和筛选数据的工具。它可以根据行键、列族、列标识等信息快速过滤掉表中不符合特定条件的数据，从而提高查询性能和减少数据传输。HBase 提供了多种类型的过滤器，如值过滤器、行过滤器等，用户可以根据实际需求选择适合的过滤器类型来指定过滤条件。在 HBase 中，可以通过 HBase Java API 或 HBase Shell 使用过滤器。本章重点介绍使用 HBase Java API 来使用过滤器的方法。

5.1 过滤器原理

过滤器的实现原理是，当客户端基于过滤器读取 HBase 的数据时，HBase 只会将表中符合过滤器指定条件的数据返回给客户端，这样可以避免不符合过滤器指定条件的数据被传输到客户端，减轻网络传输和客户端的压力。

那么过滤器是如何避免不符合过滤器条件的数据被传输到客户端的呢？这主要是因为 HBase 采用谓词下推(predicate push down)的规则执行过滤器的操作，所谓谓词下推是指将过滤器尽可能下推到距离数据源最近的地方执行，以尽早完成数据的过滤。同样地，在学习和工作中，人们可以借鉴谓词下推的原则，将注意力集中在关键的事物上，能够更好地管理时间和资源，取得更好的成果。由于在 HBase 中，数据是由 RegionServer 进行管理的，所以过滤器会被 HBase 传输到 RegionServer 执行，此时只有符合过滤器条件的数据才会被传输到客户端。例如，客户端基于过滤器执行 scan 命令查询 HBase 表数据时的执行流程如图 5-1 所示。

在图 5-1 中，过滤器在客户端(Client)执行 scan 命令时被创建，创建完成后会被序列化为网络传输的格式，然后通过 RPC 协议分发到 HBase 集群中管理相关表数据的 RegionServer，在 RegionServer 中过滤器会通过反序列化被还原，并且在 RegionScanner 中基于过滤器进行查询数据的操作，最终将符合过滤器条件的数据返回给客户端。

在客户端查询数据时，常用的操作之一是通过过滤器扫描表。扫描表是指使用 HBase Shell 提供的 scan 命令或 HBase Java API 中 Table 类的 getScanner()方法来查询数据。本章的后续内容将重点介绍如何使用 HBase Java API 在 Java 应用程序中通过过滤器进行扫描表。

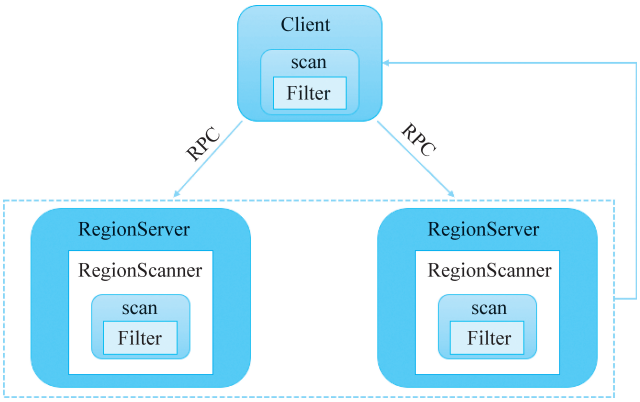

图 5-1　过滤器的执行流程

Scan 类提供了 setFilter()方法用于为扫描表设置过滤器，其程序结构如下。

```
scan.setFilter(Filter);
```

上述程序结构中，scan 为 Scan 类的实例，Filter 用于指定设置的过滤器。

5.2　环境准备

在进行某项事务前，充足的准备能够使我们更好地发挥自己的潜力，提高自身能力和素质。这包括深入学习相关知识，积累相关经验，以及适时地进行规划。通过充分准备，能够为自己创造更多机会，增加成功的可能性，并提高对事务的把控能力和执行效果。

本章后续的内容主要通过在 IntelliJ IDEA 实现 Java 应用程序，演示如何基于过滤器查询数据，为了后续内容讲解的便利，这里事先对相关环境进行准备，包括构建 Java 项目、导入依赖、启动集群环境、连接 HBase、创建命名空间、创建表，以及向表插入数据等，具体操作步骤如下。

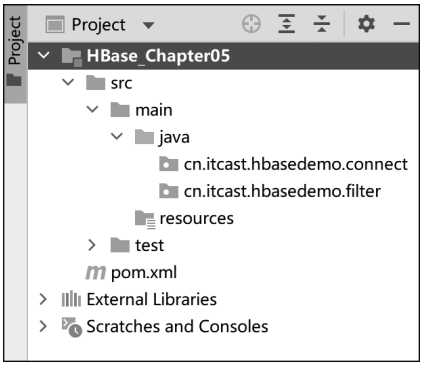

图 5-2　项目构建完成的效果

1. 构建 Java 项目

在 IntelliJ IDEA 基于 Maven 构建 Java 项目 HBase_Chapter05，并且创建包 cn.itcast.hbasedemo.connect 和 cn.itcast.hbasedemo.filter，前者用于存放连接 HBase 相关操作的 Java 文件，后者用于存放过滤器相关操作的 Java 文件，项目构建完成的效果如图 5-2 所示。

2. 导入依赖

在 Java 项目的 pom.xml 文件中添加 HBase 的客户端依赖，依赖添加完成的效果如文件 5-1 所示。

文件 5-1　pom.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
```

```
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>cn.itcast</groupId>
    <artifactId>HBase_Chapter03</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
    </properties>
    <dependencies>
        //HBase 的客户端依赖
        <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase-shaded-client</artifactId>
            <version>2.0.0</version>
        </dependency>
    </dependencies>
</project>
```

3. 启动集群环境

在虚拟机 HBase01、HBase02 和 HBase03 按照 ZooKeeper、Hadoop 和 HBase 的顺序启动相关集群，本章使用基于完全分布式模式部署的 HBase 集群。

4. 连接 HBase

在 Java 项目的 cn.itcast.hbasedemo.connect 包中创建 HBaseConnect 类，该类用于实现连接 HBase 的功能，如文件 5-2 所示。

文件 5-2　HBaseConnect.java

```
public class HBaseConnect {
    public static Connection getConnect() {
      Connection conn =null;
      try {
          Configuration config =HBaseConfiguration.create();
          config.set("hbase.zookeeper.quorum","hbase01,hbase02,hbase03");
          config.set("zookeeper.znode.parent","/hbase-fully");
          conn =ConnectionFactory.createConnection(config);
      }
      catch (Exception e){
          e.printStackTrace();
      }
      return conn;
    }
    public static void closeConn() {
      Connection connect =getConnect();
      try {
          if (connect.isClosed() ==false){
                System.out.println("关闭 HBase 连接!!!!!!");
                connect.close();
          }
      }catch (Exception e){
          e.printStackTrace();
```

```
        }
    }
}
```

文件 5-2 的内容与 4.2 节中文件 4-2 的内容一致，这里不再赘述。

5. 创建命名空间

创建命名空间 commodity，该命名空间用于存放本章相关操作的表。在 HBase Shell 执行如下命令。

```
>create_namespace 'commodity'
```

6. 创建表

在命名空间 commodity 中创建表 fruit_table，并指定列族 fruit_info 和 sale_info，其中列族 fruit_info 用于存储水果信息，列族 sale_info 用于存放销售信息。在 HBase Shell 执行如下命令。

```
>create 'commodity:fruit_table','fruit_info','sale_info'
```

7. 向表插入数据

将数据文件 fruit_info.csv 存储的水果销售数据插入命名空间 commodity 的表 fruit_table，有关数据文件 fruit_info.csv 的内容如图 5-3 所示。

	A	B	C	D	E	F	G
1	fruit001	Banana	Tropical and subtropical fruits	Yunnan	5.5	157	863.5
2	fruit002	Apple	Kernel fruits	Shandong	6.8	284	1931.2
3	fruit003	Pear	Kernel fruits	Anhui	4.6	452	2079.2
4	fruit004	Grape	Berries	Xinjiang	10.5	313	3286.5
5	fruit005	Durian	Tropical and subtropical fruits	Hainan	45	156	7020
6	fruit006	Strawberry	Berries	Hebei	28	429	12012
7	fruit007	Jujube	Stone fruits	Shandong	13	249	3237
8	fruit008	Orange	Citrus fruits	Jiangxi	7.5	471	3532.5
9	fruit009	Peach	Stone fruits	Beijing	6.1	143	872.3
10	fruit010	Grapefruit	Citrus fruits	Guangxi	6.5	293	1904.5
11	fruit011	Persimmon	Kernel fruits	Liaoning	6.7	273	1829.1
12	fruit012	Watermelon	Melon fruits	Henan	2.4	415	996
13	fruit013	Hamimelon	Melon fruits	Xinjiang	6.5	286	1859
14	fruit014	Pitaya	Tropical and subtropical fruits	Guangxi	7	371	2597
15	fruit015	Lichee	Tropical and subtropical fruits	Hainan	15	425	6375
16	fruit016	Lemon	Citrus fruits	Yunnan	17	388	6596
17	fruit017	Coconut	Tropical and subtropical fruits	Hainan	11	349	3839
18	fruit018	Kiwifruit	Berries	Sichuan	14.5	355	5147.5
19	fruit019	Hawthorn	Kernel fruits	Shanxi	6.6	476	3141.6
20	fruit020	Cherry	Stone fruits	Shandong	35	229	8015

fruit_info

图 5-3 数据文件 fruit_info.csv 的内容

在图 5-3 中，每行数据的字段按照从左到右的顺序依次表示水果编号、水果名称、水果类型、水果产地、单价（单位是元/千克）、销售量（单位是千克）和总销售额（单位是元）。

在插入数据时，如果使用 HBase Shell 的方式实现，数字类型的数据（如总销售额和单价）会被转换为字符串类型进行存储，这将导致后续无法使用过滤器基于数字类型的数据进行过滤。因此，这里通过编写 Java 应用程序的方式实现插入数据的操作。

在 Java 项目的 cn.itcast.hbasedemo.connect 包中创建 LoadCsvData 类，该类用于将数据文件 fruit_info.csv 的数据插入命名空间 commodity 的表 fruit_table，具体代码如文件 5-3 所示。

文件 5-3　LoadCsvData.java

```
public class LoadCsvData {
  private static Connection conn;
  private static Table table;
  public static void main(String[] args) throws IOException {
      //连接 HBase
      conn =HBaseConnect.getConnect();
      //向命名空间 commodity 的表 fruit_table 插入数据
      table =conn.getTable(TableName.valueOf("commodity:fruit_table"));
      //指定数据文件 fruit_info.csv 的存储路径
      String path ="D:\\Data\\fruit_info.csv";
      //水果编号
      String fruitNo ="";
      //水果名称
      String fruitName ="";
      //水果类型
      String fruitType ="";
      //水果产地
      String fruitOrigin ="";
      //单价
      BigDecimal unitPrice =null;
      //销售量
      Long quantity =0L;
      //总销售额
      BigDecimal totalSale =null;
      //通过定义的 readCsvByCsvReader()方法获取数据文件的数据
      ArrayList<String[]>csvData =readCsvByCsvReader(path);
      for (int i =0;i <csvData.size();i++){
          ArrayList<Put>puts =new ArrayList<>();
          fruitNo =csvData.get(i)[0];
          fruitName =csvData.get(i)[1];
          fruitType =csvData.get(i)[2];
          fruitOrigin =csvData.get(i)[3];
          unitPrice =new BigDecimal(csvData.get(i)[4]);
          quantity =Long.valueOf(csvData.get(i)[5]);
          totalSale =new BigDecimal(csvData.get(i)[6]);
          Put put =new Put(Bytes.toBytes(fruitNo));
          Put put1 =put.addColumn(
                  Bytes.toBytes("fruit_info"),
                  Bytes.toBytes("fruitName"),
                  Bytes.toBytes(fruitName));
          Put put2 =put.addColumn(
                  Bytes.toBytes("fruit_info"),
                  Bytes.toBytes("fruitType"),
                  Bytes.toBytes(fruitType));
          Put put3 =put.addColumn(
                  Bytes.toBytes("fruit_info"),
                  Bytes.toBytes("fruitOrigin"),
                  Bytes.toBytes(fruitOrigin));
          Put put4 =put.addColumn(
                  Bytes.toBytes("sale_info"),
                  Bytes.toBytes("unitPrice"),
```

```
52                  Bytes.toBytes(unitPrice));
53          Put put5 =put.addColumn(
54                  Bytes.toBytes("sale_info"),
55                  Bytes.toBytes("quantity"),
56                  Bytes.toBytes(quantity));
57          Put put6 =put.addColumn(
58                  Bytes.toBytes("sale_info"),
59                  Bytes.toBytes("totalSale"),
60                  Bytes.toBytes(totalSale));
61          puts.add(put1);
62          puts.add(put2);
63          puts.add(put3);
64          puts.add(put4);
65          puts.add(put5);
66          puts.add(put6);
67          table.put(puts);
68        }
69    HBaseConnect.closeConn();
70  }
71  public static ArrayList<String[]>readCsvByCsvReader(String filePath) {
72      ArrayList<String[]>arrList =new ArrayList<String[]>();
73      try {
74          CsvReader reader =new CsvReader(
75                  filePath,
76                  ',',
77                  Charset.forName("UTF-8"));
78          while (reader.readRecord()) {
79               arrList.add(reader.getValues());
80          }
81          reader.close();
82      } catch (Exception e) {
83          e.printStackTrace();
84      }
85      return arrList;
86    }
87 }
```

在文件 5-3 中，第 27～68 行代码用于遍历集合，获取数据文件的每行数据。然后，从当前行数据中提取水果编号、水果名称、水果类型、水果产地、单价、销售量和总销售额，并将它们赋值给相应的变量。最后，使用水果编号作为行键，将水果名称、水果类型、水果产地、单价、销售量和总销售额插入命名空间 commodity 的表 fruit_table 中相应的列。

第 71～86 行代码定义的 readCsvByCsvReader()方法用于读取 CSV 文件，并将文件中的每行数据存储为集合 csvData 的元素。

8. 验证数据是否插入成功

文件 5-3 运行完成后，执行“scan 'commodity:fruit_table',{LIMIT ＝＞ 2}”命令查询命名空间 commodity 中表 fruit_table 的前两行数据，如图 5-4 所示。

从图 5-4 可以看出，数据文件 fruit_info.csv 的数据成功插入命名空间 commodity 的表 fruit_table 中。

需要说明的是，HBase Shell 只能正常显示字符串类型的数据，而列 sale_info:unitPrice、sale_info:quantity 和 sale_info:totalSale 的数据是数字类型，因此以二进制形式

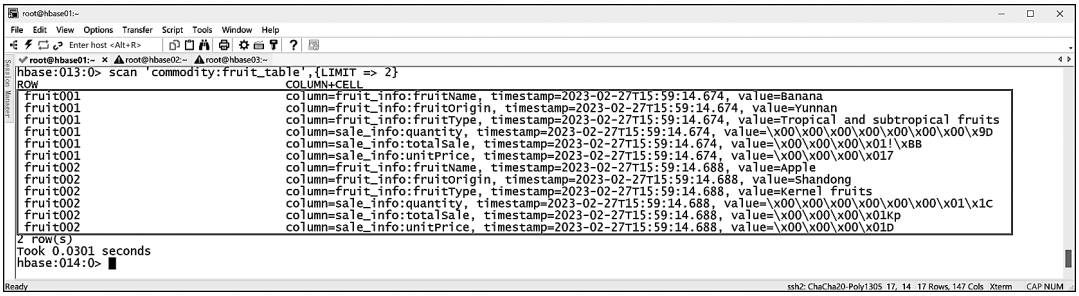

图 5-4　查询命名空间 commodity 中表 fruit_table 的前两行数据

进行显示。

5.3　值过滤器

值过滤器(ValueFilter)基于数据进行过滤。在使用值过滤器扫描表时，HBase 会逐条检查指定表中的数据，并将其与用户定义的条件进行比较。如果数据满足用户定义的条件，HBase 会将该数据所在的单元格返回给客户端。

HBase Java API 提供了 ValueFilter 类用于创建值过滤器，其程序结构如下。

```
ValueFilter valueFilter =
      new ValueFilter(
          compareoperator,
          comparator
       );
```

上述程序结构中，valueFilter 用于定义值过滤器的名称。compareoperator 用于指定比较关系，如大于、等于、小于等。comparator 用于指定比较器，比较器根据比较关系来确定比较的方式，例如比较字符串是否相等、比较数字的大小关系等。通过将比较关系和比较器结合使用，可以形成完整的条件，用于值过滤器进行数据过滤。

接下来对 HBase Java API 常用的比较关系和比较器进行介绍，如表 5-1 和表 5-2 所示。

表 5-1　HBase Java API 常用的比较关系

比 较 关 系	描　　述
CompareOperator.LESS	用于指定小于的比较关系
CompareOperator.LESS_OR_EQUAL	用于指定小于或等于的比较关系
CompareOperator.EQUAL	用于指定相等的比较关系
CompareOperator.NOT_EQUAL	用于指定不相等的比较关系
CompareOperator.GREATER_OR_EQUAL	用于指定大于或等于的比较关系
CompareOperator.GREATER	用于指定大于的比较关系

表 5-2 HBase Java API 常用的比较器

比 较 器	描 述
new NullComparator()	用于比较数据是否等于 null
new BinaryComparator(value)	用于将行键、列族、数据等信息视为字节数组与字节数组 value 进行比较，如搭配比较关系 CompareOperator.EQUAL 使用，比较数据是否等于 value
new SubstringComparator(value)	用于将行键、列族、数据等信息视为字符串与字符串 value 进行比较，如搭配比较关系 CompareOperator.EQUAL 使用，比较列族是否等于 value
new LongComparator(value)	将数据与 Long 类型的数值 value 进行比较，如搭配比较关系 CompareOperator.LESS 使用，比较数据是否小于 value
new RegexStringComparator(value)	用于将行键、列族、数据等信息视为字符串与正则表达式 value 进行匹配，如搭配比较关系 CompareOperator.EQUAL 使用，比较数据是否匹配正则表达式
new BinaryPrefixComparator(value)	用于将行键、列族、数据等信息的前缀视为字节数组与字节数组 value 进行比较，如搭配比较关系 CompareOperator.EQUAL 使用，比较列族的前缀是否等于 value
new BigDecimalComparator(value)	用于将数据与 BigDecimal 类型的数值 value 进行比较，如搭配比较关系 CompareOperator.LESS 使用，比较数据是否小于 value

接下来演示如何在 Java 应用程序中使用值过滤器查询数据。在 Java 项目的 cn.itcast.hbasedemo.filter 包中创建 ValueFilterDemo 类，该类用于查询命名空间为 commodity 的表 fruit_table 中水果产地为 Hainan 的水果编号，具体代码如文件 5-4 所示。

文件 5-4 ValueFilterDemo.java

```
1  public class ValueFilterDemo {
2    private static Connection conn;
3    private static Table table;
4    public static void main(String[] args) throws IOException {
5        conn =HBaseConnect.getConnect();
6        //指定查询命名空间 commodity 中表 fruit_table 的数据
7        table =conn.getTable(TableName.valueOf("commodity:fruit_table"));
8        Scan scan =new Scan();
9        //指定查询列族 fruit_info 的数据
10       scan.addFamily(Bytes.toBytes("fruit_info"));
11       //创建值过滤器
12       ValueFilter valueFilter =new ValueFilter(
13             CompareOperator.EQUAL,
14             new BinaryComparator(Bytes.toBytes("Hainan"))
15       );
16       //设置值过滤器
17        scan.setFilter(valueFilter);
18        for (Result result : table.getScanner(scan)) {
19           List<Cell> cells =result.listCells();
20           for (Cell cell : cells) {
21                String rowkey =new String(
22                     cell.getRowArray(),
```

```
                        cell.getRowOffset(),
                        cell.getRowLength()
                    );
                    System.out.println("水果编号:" +rowkey);
                }
            }
        HBaseConnect.closeConn();
    }
}
```

在文件 5-4 中，第 12～15 行代码创建值过滤器 valueFilter，该过滤器用于比较数据是否等于 Hainan。第 18～28 行代码用于查询数据并遍历查询结果，获取查询结果中每个单元格的行键，即每个水果的水果编号。

文件 5-4 的运行结果如图 5-5 所示。

图 5-5　文件 5-4 的运行结果

从图 5-5 可以看出，水果产地为 Hainan 的水果编号包括 fruit005、fruit015 和 fruit017。

5.4　列值过滤器

列值过滤器（ColumnValueFilter）基于列进行过滤。在使用列值过滤器扫描表时，HBase 会逐条检查指定列的数据，并将其与用户定义的条件进行比较。如果数据满足用户定义的条件，HBase 会将该数据所在的单元格返回给客户端。

HBase Java API 提供了 ColumnValueFilter 类用于创建列值过滤器，其程序结构如下。

```
ColumnValueFilter columnValueFilter =
      new ColumnValueFilter(
          columnfamily,
          qualifier,
          compareoperator,
          comparator
      );
```

上述程序结构中，columnValueFilter 用于定义列值过滤器的名称，columnfamily 用于指定列族，qualifier 用于指定列标识，compareoperator 和 comparator 分别用于指定比较关系和比较器。

接下来演示如何在 Java 应用程序中使用列值过滤器查询数据。在 Java 项目的 cn.itcast.hbasedemo.filter 包中创建 ColumnValueFilterDemo 类，该类用于查询命名空间为 commodity 的表 fruit_table 中总销售额大于或等于 5000 的水果编号及其总销售额，具体代码如文件 5-5 所示。

文件 5-5 ColumnValueFilterDemo.java

```
1  public class ColumnValueFilterDemo {
2    private static Connection conn;
3    private static Table table;
4    public static void main(String[] args) throws IOException {
5        conn =HBaseConnect.getConnect();
6        //指定查询命名空间 commodity 中表 fruit_table 的数据
7        table =conn.getTable(TableName.valueOf("commodity:fruit_table"));
8        Scan scan =new Scan();
9        //创建列值过滤器
10       ColumnValueFilter columnValueFilter =new ColumnValueFilter(
11                Bytes.toBytes("sale_info"),
12                Bytes.toBytes("totalSale"),
13                CompareOperator.GREATER_OR_EQUAL,
14                new BigDecimalComparator(new BigDecimal(5000)));
15       //设置列值过滤器
16       scan.setFilter(columnValueFilter);
17       for (Result result : table.getScanner(scan)) {
18           List<Cell> cells =result.listCells();
19           for (Cell cell : cells) {
20                String rowkey =new String(
21                     cell.getRowArray(),
22                     cell.getRowOffset(),
23                     cell.getRowLength()
24                );
25                BigDecimal value =Bytes.toBigDecimal(
26                     cell.getValueArray(),
27                     cell.getValueOffset(),
28                     cell.getValueLength()
29                );
30                System.out.println("水果编号:" +rowkey
31                     +"\t" +"总销售额:" +value);
32           }
33       }
34       HBaseConnect.closeConn();
35     }
36 }
```

在文件 5-5 中，第 10～14 行代码创建列值过滤器 columnValueFilter，该过滤器用于比较列 sale_info:totalSale 的数据是否大于或等于 5000。第 17～33 行代码用于查询数据并遍历查询结果，获取查询结果中每个单元格的行键和数据，即每个水果的水果编号和总销售额。

文件 5-5 的运行结果如图 5-6 所示。

从图 5-6 可以看出，水果的总销售额大于或等于 5000 的水果编号包括 fruit005、fruit006、fruit015、fruit016、fruit018 和 fruit020，它们的总销售额分别是 7020、12012、6375、6596、5147.5 和 8015。