User Defined Aggregate Functions (UDAFs)

Description

User-Defined Aggregate Functions (UDAFs) are user-programmable routines that act on multiple rows at once and return a single aggregated value as a result. This documentation lists the classes that are required for creating and registering UDAFs. It also contains examples that demonstrate how to define and register UDAFs in Scala and invoke them in Spark SQL.

Aggregator[-IN, BUF, OUT]

A base class for user-defined aggregations, which can be used in Dataset operations to take all of the elements of a group and reduce them to a single value.

IN - The input type for the aggregation.

BUF - The type of the intermediate value of the reduction.

OUT - The type of the final output result.

- **bufferEncoder: Encoder[BUF]**

  Specifies the Encoder for the intermediate value type.

- **finish(reduction: BUF): OUT**

  Transform the output of the reduction.

- **merge(b1: BUF, b2: BUF): BUF**

  Merge two intermediate values.

- **outputEncoder: Encoder[OUT]**

  Specifies the Encoder for the final output value type.

- **reduce(b: BUF, a: IN): BUF**

  Aggregate input value `a` into current intermediate value. For performance, the function may modify `b` and return it instead of constructing new object for `b`.

- **zero: BUF**

  The initial value of the intermediate result for this aggregation.

Examples

Type-Safe User-Defined Aggregate Functions

User-defined aggregations for strongly typed Datasets revolve around the Aggregator abstract class. For example, a type-safe user-defined average can look like:

Scala Java

```java
import java.io.Serializable;

import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Encoder;
import org.apache.spark.sql.Encoders;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.TypedColumn;
import org.apache.spark.sql.expressions.Aggregator;

public static class Employee implements Serializable {
  private String name;
  private long salary;

  // Constructors, getters, setters...

}

public static class Average implements Serializable  {
  private long sum;
  private long count;

  // Constructors, getters, setters...

}

public static class MyAverage extends Aggregator<Employee, Average, Double> {
  // A zero value for this aggregation. Should satisfy the property that any b + zero = b
  public Average zero() {
    return new Average(0L, 0L);
  }
  // Combine two values to produce a new value. For performance, the function may modify `buffer`
  // and return it instead of constructing a new object
  public Average reduce(Average buffer, Employee employee) {
    long newSum = buffer.getSum() + employee.getSalary();
    long newCount = buffer.getCount() + 1;
    buffer.setSum(newSum);
    buffer.setCount(newCount);
    return buffer;
  }
  // Merge two intermediate values
  public Average merge(Average b1, Average b2) {
    long mergedSum = b1.getSum() + b2.getSum();
    long mergedCount = b1.getCount() + b2.getCount();
    b1.setSum(mergedSum);
    b1.setCount(mergedCount);
    return b1;
  }
  // Transform the output of the reduction
  public Double finish(Average reduction) {
    return ((double) reduction.getSum()) / reduction.getCount();
  }
  // Specifies the Encoder for the intermediate value type
  public Encoder<Average> bufferEncoder() {
    return Encoders.bean(Average.class);
  }
  // Specifies the Encoder for the final output value type
  public Encoder<Double> outputEncoder() {
    return Encoders.DOUBLE();
  }
}

Encoder<Employee> employeeEncoder = Encoders.bean(Employee.class);
String path = "examples/src/main/resources/employees.json";
Dataset<Employee> ds = spark.read().json(path).as(employeeEncoder);
ds.show();
// +-------+------+
// |   name|salary|
// +-------+------+
// |Michael|  3000|
// |   Andy|  4500|
// | Justin|  3500|
// |  Berta|  4000|
// +-------+------+

MyAverage myAverage = new MyAverage();
// Convert the function to a `TypedColumn` and give it a name
TypedColumn<Employee, Double> averageSalary = myAverage.toColumn().name("average_salary");
Dataset<Double> result = ds.select(averageSalary);
```

```
result.show();
// +--------------+
// |average_salary|
// +--------------+
// |        3750.0|
// +--------------+
```

Find full example code at "examples/src/main/java/org/apache/spark/examples/sql/JavaUserDefinedTypedAggregation.java" in the Spark repo.

Untyped User-Defined Aggregate Functions

Typed aggregations, as described above, may also be registered as untyped aggregating UDFs for use with DataFrames. For example, a user-defined average for untyped DataFrames can look like:

Scala Java SQL

»

```java
import java.io.Serializable;

import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Encoder;
import org.apache.spark.sql.Encoders;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.expressions.Aggregator;
import org.apache.spark.sql.functions;

public static class Average implements Serializable  {
  private long sum;
  private long count;

  // Constructors, getters, setters...
  public Average() {
  }

  public Average(long sum, long count) {
    this.sum = sum;
    this.count = count;
  }

  public long getSum() {
    return sum;
  }

  public void setSum(long sum) {
    this.sum = sum;
  }

  public long getCount() {
    return count;
  }

  public void setCount(long count) {
    this.count = count;
  }
}

public static class MyAverage extends Aggregator<Long, Average, Double> {
  // A zero value for this aggregation. Should satisfy the property that any b + zero = b
  public Average zero() {
    return new Average(0L, 0L);
  }
  // Combine two values to produce a new value. For performance, the function may modify `buffer`
  // and return it instead of constructing a new object
  public Average reduce(Average buffer, Long data) {
    long newSum = buffer.getSum() + data;
    long newCount = buffer.getCount() + 1;
    buffer.setSum(newSum);
    buffer.setCount(newCount);
    return buffer;
  }
  // Merge two intermediate values
  public Average merge(Average b1, Average b2) {
    long mergedSum = b1.getSum() + b2.getSum();
    long mergedCount = b1.getCount() + b2.getCount();
    b1.setSum(mergedSum);
    b1.setCount(mergedCount);
    return b1;
  }
  // Transform the output of the reduction
  public Double finish(Average reduction) {
    return ((double) reduction.getSum()) / reduction.getCount();
  }
  // Specifies the Encoder for the intermediate value type
  public Encoder<Average> bufferEncoder() {
    return Encoders.bean(Average.class);
  }
  // Specifies the Encoder for the final output value type
  public Encoder<Double> outputEncoder() {
    return Encoders.DOUBLE();
  }
}

// Register the function to access it
spark.udf().register("myAverage", functions.udaf(new MyAverage(), Encoders.LONG()));
```

»

```
Dataset<Row> df = spark.read().json("examples/src/main/resources/employees.json");
df.createOrReplaceTempView("employees");
df.show();
// +-------+------+
// |   name|salary|
// +-------+------+
// |Michael|  3000|
// |   Andy|  4500|
// | Justin|  3500|
// |  Berta|  4000|
// +-------+------+

Dataset<Row> result = spark.sql("SELECT myAverage(salary) as average_salary FROM employees");
result.show();
// +--------------+
// |average_salary|
// +--------------+
// |        3750.0|
// +--------------+
```

Find full example code at "examples/src/main/java/org/apache/spark/examples/sql/JavaUserDefinedUntypedAggregation.java" in the Spark repo.

Related Statements

- Scalar User Defined Functions (UDFs)
- Integration with Hive UDFs/UDAFs/UDTFs