databricks / spark-csv

CSV data source for Spark SQL and DataFrames http://databricks.com/

barrybecker committed with **falaki** fix NPE that can occur for null dates ... Latest commit `1fd65ad` 21 days ago

dev	Port PR merge script from Spark	11 months ago
project	Enforce more Scalastyle rules and add MiMa binary compatibility checks	11 months ago
sbt	fix sbt script	a year ago
src	fix NPE that can occur for null dates	21 days ago
.gitignore	Ignoring Desktop Services Store file on Mac	a year ago
.travis.yml	Test against Spark 1.6.0	7 months ago
LICENSE	Everything	2 years ago
README.md	Fix for #308 write using specified dateFormat	a month ago
build.sbt	Preparing for V1.4.0	6 months ago
scalastyle-config.xml	Enforce more Scalastyle rules and add MiMa binary compatibility checks	11 months ago

README.md

CSV Data Source for Apache Spark

A library for parsing and querying CSV data with Apache Spark, for Spark SQL and DataFrames.

Requirements

This library requires Spark 1.3+

Linking

You can link against this library in your program at the following coordinates:

Scala 2.10

```
groupId: com.databricks
artifactId: spark-csv_2.10
version: 1.4.0
```

Scala 2.11

```
groupId: com.databricks
artifactId: spark-csv_2.11
```

```
version: 1.4.0
```

Using with Spark shell

This package can be added to Spark using the `--packages` command line option. For example, to include it when starting the spark shell:

Spark compiled with Scala 2.11

```
$SPARK_HOME/bin/spark-shell --packages com.databricks:spark-csv_2.11:1.4.0
```

Spark compiled with Scala 2.10

```
$SPARK_HOME/bin/spark-shell --packages com.databricks:spark-csv_2.10:1.4.0
```

Features

This package allows reading CSV files in local or distributed filesystem as Spark DataFrames. When reading files the API accepts several options:

- `path` : location of files. Similar to Spark can accept standard Hadoop globbing expressions.
- `header` : when set to true the first line of files will be used to name columns and will not be included in data. All types will be assumed string. Default value is false.
- `delimiter` : by default columns are delimited using `,`, but delimiter can be set to any character
- `quote` : by default the quote character is `"`, but can be set to any character. Delimiters inside quotes are ignored
- `escape` : by default the escape character is `\`, but can be set to any character. Escaped quote characters are ignored

- `parserLib` : by default it is "commons" can be set to "univocity" to use that library for CSV parsing.
- `mode` : determines the parsing mode. By default it is PERMISSIVE. Possible values are:
  - `PERMISSIVE` : tries to parse all lines: nulls are inserted for missing tokens and extra tokens are ignored.
  - `DROPMALFORMED` : drops lines which have fewer or more tokens than expected or tokens which do not match the schema
  - `FAILFAST` : aborts with a RuntimeException if encounters any malformed line
- `charset` : defaults to 'UTF-8' but can be set to other valid charset names
- `inferSchema` : automatically infers column types. It requires one extra pass over the data and is false by default
- `comment` : skip lines beginning with this character. Default is `"#"` . Disable comments by setting this to `null` .
- `nullValue` : specifies a string that indicates a null value, any fields matching this string will be set as nulls in the DataFrame
- `dateFormat` : specifies a string that indicates the date format to use when reading dates or timestamps. Custom date formats follow the formats at `java.text.SimpleDateFormat` . This applies to both `DateType` and `TimestampType` . By default, it is `null` which means trying to parse times and date by `java.sql.Timestamp.valueOf()` and `java.sql.Date.valueOf()` .

The package also supports saving simple (non-nested) DataFrame. When writing files the API accepts several options:

- `path` : location of files.
- `header` : when set to true, the header (from the schema in the DataFrame) will be written at the first line.
- `delimiter` : by default columns are delimited using `,` , but delimiter can be set to any character
- `quote` : by default the quote character is `"` , but can be set to any character. This is written according to `quoteMode` .
- `escape` : by default the escape character is `\` , but can be set to any character. Escaped quote characters are written.
- `nullValue` : specifies a string that indicates a null value, nulls in the DataFrame will be written as this string.
- `dateFormat` : specifies a string that indicates the date format to use writing dates or timestamps. Custom date formats follow the formats at `java.text.SimpleDateFormat` . This applies to both `DateType` and `TimestampType` . If no dateFormat is specified, then "yyyy-MM-dd HH:mm:ss.S".

- `codec` : compression codec to use when saving to file. Should be the fully qualified name of a class implementing `org.apache.hadoop.io.compress.CompressionCodec` or one of case-insensitive shorten names (`bzip2` , `gzip` , `lz4` , and `snappy`). Defaults to no compression when a codec is not specified.
- `quoteMode` : when to quote fields (`ALL` , `MINIMAL` (default), `NON_NUMERIC` , `NONE`), see Quote Modes

These examples use a CSV file available for download here:

```
$ wget https://github.com/databricks/spark-csv/raw/master/src/test/resources/cars.csv
```

SQL API

CSV data source for Spark can infer data types:

```
CREATE TABLE cars
USING com.databricks.spark.csv
OPTIONS (path "cars.csv", header "true", inferSchema "true")
```

You can also specify column names and types in DDL.

```
CREATE TABLE cars (yearMade double, carMake string, carModel string, comments string, blank string)
USING com.databricks.spark.csv
OPTIONS (path "cars.csv", header "true")
```

Scala API

Spark 1.4+:

Automatically infer schema (data types), otherwise everything is assumed string:

```
import org.apache.spark.sql.SQLContext

val sqlContext = new SQLContext(sc)
val df = sqlContext.read
    .format("com.databricks.spark.csv")
    .option("header", "true") // Use first line of all files as header
    .option("inferSchema", "true") // Automatically infer data types
    .load("cars.csv")

val selectedData = df.select("year", "model")
selectedData.write
    .format("com.databricks.spark.csv")
    .option("header", "true")
    .save("newcars.csv")
```

You can manually specify the schema when reading data:

```
import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.types.{StructType, StructField, StringType, IntegerType};

val sqlContext = new SQLContext(sc)
val customSchema = StructType(Array(
    StructField("year", IntegerType, true),
    StructField("make", StringType, true),
    StructField("model", StringType, true),
    StructField("comment", StringType, true),
    StructField("blank", StringType, true)))

val df = sqlContext.read
    .format("com.databricks.spark.csv")
    .option("header", "true") // Use first line of all files as header
    .schema(customSchema)
    .load("cars.csv")

val selectedData = df.select("year", "model")
```

```
selectedData.write
    .format("com.databricks.spark.csv")
    .option("header", "true")
    .save("newcars.csv")
```

You can save with compressed output:

```
import org.apache.spark.sql.SQLContext

val sqlContext = new SQLContext(sc)
val df = sqlContext.read
    .format("com.databricks.spark.csv")
    .option("header", "true") // Use first line of all files as header
    .option("inferSchema", "true") // Automatically infer data types
    .load("cars.csv")

val selectedData = df.select("year", "model")
selectedData.write
    .format("com.databricks.spark.csv")
    .option("header", "true")
    .option("codec", "org.apache.hadoop.io.compress.GzipCodec")
    .save("newcars.csv.gz")
```

Spark 1.3:

Automatically infer schema (data types), otherwise everything is assumed string:

```
import org.apache.spark.sql.SQLContext

val sqlContext = new SQLContext(sc)
val df = sqlContext.load(
    "com.databricks.spark.csv",
    Map("path" -> "cars.csv", "header" -> "true", "inferSchema" -> "true"))
```

```
val selectedData = df.select("year", "model")
selectedData.save("newcars.csv", "com.databricks.spark.csv")
```

You can manually specify the schema when reading data:

```
import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.types.{StructType, StructField, StringType, IntegerType};

val sqlContext = new SQLContext(sc)
val customSchema = StructType(Array(
    StructField("year", IntegerType, true),
    StructField("make", StringType, true),
    StructField("model", StringType, true),
    StructField("comment", StringType, true),
    StructField("blank", StringType, true)))

val df = sqlContext.load(
    "com.databricks.spark.csv",
    schema = customSchema,
    Map("path" -> "cars.csv", "header" -> "true"))

val selectedData = df.select("year", "model")
selectedData.save("newcars.csv", "com.databricks.spark.csv")
```

Java API

Spark 1.4+:

Automatically infer schema (data types), otherwise everything is assumed string:

```
import org.apache.spark.sql.SQLContext

SQLContext sqlContext = new SQLContext(sc);
```

```
DataFrame df = sqlContext.read()
    .format("com.databricks.spark.csv")
    .option("inferSchema", "true")
    .option("header", "true")
    .load("cars.csv");

df.select("year", "model").write()
    .format("com.databricks.spark.csv")
    .option("header", "true")
    .save("newcars.csv");
```

You can manually specify schema:

```
import org.apache.spark.sql.SQLContext;
import org.apache.spark.sql.types.*;

SQLContext sqlContext = new SQLContext(sc);
StructType customSchema = new StructType(new StructField[] {
    new StructField("year", DataTypes.IntegerType, true, Metadata.empty()),
    new StructField("make", DataTypes.StringType, true, Metadata.empty()),
    new StructField("model", DataTypes.StringType, true, Metadata.empty()),
    new StructField("comment", DataTypes.StringType, true, Metadata.empty()),
    new StructField("blank", DataTypes.StringType, true, Metadata.empty())
});

DataFrame df = sqlContext.read()
    .format("com.databricks.spark.csv")
    .schema(customSchema)
    .option("header", "true")
    .load("cars.csv");

df.select("year", "model").write()
    .format("com.databricks.spark.csv")
    .option("header", "true")
    .save("newcars.csv");
```

You can save with compressed output:

```
import org.apache.spark.sql.SQLContext

SQLContext sqlContext = new SQLContext(sc);
DataFrame df = sqlContext.read()
    .format("com.databricks.spark.csv")
    .option("inferSchema", "true")
    .option("header", "true")
    .load("cars.csv");

df.select("year", "model").write()
    .format("com.databricks.spark.csv")
    .option("header", "true")
    .option("codec", "org.apache.hadoop.io.compress.GzipCodec")
    .save("newcars.csv");
```

Spark 1.3:

Automatically infer schema (data types), otherwise everything is assumed string:

```
import org.apache.spark.sql.SQLContext

SQLContext sqlContext = new SQLContext(sc);

HashMap<String, String> options = new HashMap<String, String>();
options.put("header", "true");
options.put("path", "cars.csv");
options.put("inferSchema", "true");

DataFrame df = sqlContext.load("com.databricks.spark.csv", options);
df.select("year", "model").save("newcars.csv", "com.databricks.spark.csv");
```

You can manually specify schema:

```
import org.apache.spark.sql.SQLContext;
import org.apache.spark.sql.types.*;

SQLContext sqlContext = new SQLContext(sc);
StructType customSchema = new StructType(new StructField[] {
    new StructField("year", DataTypes.IntegerType, true, Metadata.empty()),
    new StructField("make", DataTypes.StringType, true, Metadata.empty()),
    new StructField("model", DataTypes.StringType, true, Metadata.empty()),
    new StructField("comment", DataTypes.StringType, true, Metadata.empty()),
    new StructField("blank", DataTypes.StringType, true, Metadata.empty())
});

HashMap<String, String> options = new HashMap<String, String>();
options.put("header", "true");
options.put("path", "cars.csv");

DataFrame df = sqlContext.load("com.databricks.spark.csv", customSchema, options);
df.select("year", "model").save("newcars.csv", "com.databricks.spark.csv");
```

You can save with compressed output:

```
import org.apache.spark.sql.SQLContext;
import org.apache.spark.sql.SaveMode;

SQLContext sqlContext = new SQLContext(sc);

HashMap<String, String> options = new HashMap<String, String>();
options.put("header", "true");
options.put("path", "cars.csv");
options.put("inferSchema", "true");

DataFrame df = sqlContext.load("com.databricks.spark.csv", options);

HashMap<String, String> saveOptions = new HashMap<String, String>();
saveOptions.put("header", "true");
```

```
saveOptions.put("path", "newcars.csv");
saveOptions.put("codec", "org.apache.hadoop.io.compress.GzipCodec");

df.select("year", "model").save("com.databricks.spark.csv", SaveMode.Overwrite,
                                saveOptions);
```

Python API

Spark 1.4+:

Automatically infer schema (data types), otherwise everything is assumed string:

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

df = sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').load('cars.csv')
df.select('year', 'model').write.format('com.databricks.spark.csv').save('newcars.csv')
```

You can manually specify schema:

```
from pyspark.sql import SQLContext
from pyspark.sql.types import *

sqlContext = SQLContext(sc)
customSchema = StructType([ \
    StructField("year", IntegerType(), True), \
    StructField("make", StringType(), True), \
    StructField("model", StringType(), True), \
    StructField("comment", StringType(), True), \
    StructField("blank", StringType(), True)])

df = sqlContext.read \
```

```
    .format('com.databricks.spark.csv') \
    .options(header='true') \
    .load('cars.csv', schema = customSchema)

df.select('year', 'model').write \
    .format('com.databricks.spark.csv') \
    .save('newcars.csv')
```

You can save with compressed output:

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

df = sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').load('cars.csv')
df.select('year', 'model').write.format('com.databricks.spark.csv').options(codec="org.apache.hadoop.io.compress.Gzip
```

Spark 1.3:

Automatically infer schema (data types), otherwise everything is assumed string:

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

df = sqlContext.load(source="com.databricks.spark.csv", header = 'true', inferSchema = 'true', path = 'cars.csv')
df.select('year', 'model').save('newcars.csv', 'com.databricks.spark.csv')
```

You can manually specify schema:

```
from pyspark.sql import SQLContext
from pyspark.sql.types import *
```

```
sqlContext = SQLContext(sc)
customSchema = StructType([ \
    StructField("year", IntegerType(), True), \
    StructField("make", StringType(), True), \
    StructField("model", StringType(), True), \
    StructField("comment", StringType(), True), \
    StructField("blank", StringType(), True)])

df = sqlContext.load(source="com.databricks.spark.csv", header = 'true', schema = customSchema, path = 'cars.csv')
df.select('year', 'model').save('newcars.csv', 'com.databricks.spark.csv')
```

You can save with compressed output:

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

df = sqlContext.load(source="com.databricks.spark.csv", header = 'true', inferSchema = 'true', path = 'cars.csv')
df.select('year', 'model').save('newcars.csv', 'com.databricks.spark.csv', codec="org.apache.hadoop.io.compress.GzipC
```

R API

Spark 1.4+:

Automatically infer schema (data types), otherwise everything is assumed string:

```
library(SparkR)

Sys.setenv('SPARKR_SUBMIT_ARGS'='"--packages" "com.databricks:spark-csv_2.10:1.4.0" "sparkr-shell"')
sqlContext <- sparkRSQL.init(sc)

df <- read.df(sqlContext, "cars.csv", source = "com.databricks.spark.csv", inferSchema = "true")
```

```
write.df(df, "newcars.csv", "com.databricks.spark.csv", "overwrite")
```

You can manually specify schema:

```
library(SparkR)

Sys.setenv('SPARKR_SUBMIT_ARGS'='"--packages" "com.databricks:spark-csv_2.10:1.4.0" "sparkr-shell"')
sqlContext <- sparkRSQL.init(sc)
customSchema <- structType(
    structField("year", "integer"),
    structField("make", "string"),
    structField("model", "string"),
    structField("comment", "string"),
    structField("blank", "string"))

df <- read.df(sqlContext, "cars.csv", source = "com.databricks.spark.csv", schema = customSchema)

write.df(df, "newcars.csv", "com.databricks.spark.csv", "overwrite")
```

You can save with compressed output:

```
library(SparkR)

Sys.setenv('SPARKR_SUBMIT_ARGS'='"--packages" "com.databricks:spark-csv_2.10:1.4.0" "sparkr-shell"')
sqlContext <- sparkRSQL.init(sc)

df <- read.df(sqlContext, "cars.csv", source = "com.databricks.spark.csv", inferSchema = "true")

write.df(df, "newcars.csv", "com.databricks.spark.csv", "overwrite", codec="org.apache.hadoop.io.compress.GzipCodec")
```

Building From Source

This library is built with SBT, which is automatically downloaded by the included shell script. To build a JAR file simply run `sbt/sbt package` from the project root. The build configuration includes support for both Scala 2.10 and 2.11.