



SAS[®] Viya[®] プラットフォームプログラミング: CAS 用 Java 入門ガイド CAS

2020.1 - 2025.06

このドキュメントは、ソフトウェアの追加バージョンに適用される場合があります。このドキュメントを [SAS Help Center](#) で開き、バナーのバージョンをクリックすると、使用できるすべてのバージョンが表示されます。

要件	2
アクションの実行方法	3
サーバーステータスの表示	3
流暢なインターフェイスプログラミング	6
例: 記述統計量の表示	6
セットアップ	6
summary アクションの実行	8
変更: BY グループ処理	9
変更: 流暢なインターフェイスを使用する	11
結果の操作	12
イベントの操作	14
イベントと CASActionResults クラス	14
ログイベント	14
パフォーマンスイベント	15
配置イベント	16
メッセージタグイベント	16
サーバーイベント	17
ソケットイベント	18
接続とアクション: 詳細	19

TLS (SSL)接続	19
REST 接続	20
authinfo ファイルの詳細	21
ユーザー名とパスワードのプロンプト	22
Kerberos 認証	22
認証方法の強制	23
期限切れのチケットまたは証明書	24
SAS 欠損値の操作	24
アクションパラメーター	24
セッションの詳細	27
サンプルJava プログラム	27
サンプルプログラムについて	27
com.sas.cas.samples	28
com.sas.cas.samples.builtins	29
com.sas.cas.sample.simple	30
com.sas.cas.samples.table	30
システムプロパティ	31
ログ	32
CASClient ログの概要	32
構成ファイルを使用したログ	32
Java システムプロパティを使用したログ	33
ログファイル形式	33
CAS シェルの使用	33
シェルについて	33
サーバーへの接続	34
次のステップ	35
SASHDAT ファイルへの CSV ファイルのインポート	35
CSV から SASHDAT ユーティリティについて	35
コマンドラインの使用	36
例	36
レコードの追加に関する考慮事項	37
コマンドライン引数	37
グラフィカルユーザーインターフェイスの使用	40

要件

SAS Cloud Analytic Services で Java を使用するには、Java を実行するクライアントマシンが Java 11 の次の要件を満たしている必要があります。

パッケージ	バージョン
cas-client	4.7.7 以降
antlr-runtime	3.5.2

パッケージ	バージョン
protobuf-java	2.6.1
json	20230227
commons-crypto	1.0.14
jakarta.xml.bind-api	2.3.2

cas-client パッケージ(関連する Javadoc パッケージを含む)と commons-crypto パッケージは、次の URL からダウンロードできます。

<https://support.sas.com/downloads/package.htm?pid=1976>

残りのパッケージは Maven リポジトリからダウンロードできます。

Java アプリケーションは、SAS Cloud Analytic Services に対して認証する必要がありますユーザー名とパスワードによる認証の場合、.authinfo ファイルを作成して資格情報を提供できます。詳細については、[Authinfo ファイルを使用したクライアント認証](#)を参照してください。

注: CAS クライアント JAR の 3.10.x バージョン以降では、**com.sas.cas.authinfo.enabled** システムプロパティを true に設定して、.authinfo ファイルからの資格情報の読み取りを有効にする必要があります。以前のリリースでは、デフォルト値は true でした。

アクションの実行方法

サーバーステータスの表示

次のコードは、serverStatus アクションの実行方法を示しています。

注: この例およびこのドキュメントのすべての例では、SAS Cloud Analytic Services の実行中インスタンスのアクセス権が必要です。ドキュメント規則として、ホストは cloud.example.com として提示され、ネットワークポートは 5570 です。置き換える値については、システム管理者に問い合わせてください。

例のコード 1 ServerStatusExample1.java

```
import com.sas.cas.CASActionResults;
import com.sas.cas.CASClient;
import com.sas.cas.CASClientInterface;
import com.sas.cas.CASValue;
import com.sas.cas.actions.builtins.ServerStatusOptions;
```

```

public class ServerStatusExample1 {
    public static void main(String[] args) throws Exception {
        try (CASClientInterface client =
            new CASClient("cloud.example.com", 5570)) {           // 1

            ServerStatusOptions options = new ServerStatusOptions(); // 2

            CASActionResults<CASValue> results = client.invoke(options);

            for (int i = 0; i < results.getResultsCount(); i++) {
                System.out.println(results.getResult(i));
            }

            // The CASClientInterface class implements java.lang.AutoCloseable 3
            // that was introduced in Java 1.7. The client.close() method is
            // automatically called to end the session and close the network socket.
        }
    }
}

```

- 1 CASClient クラスのインスタンスを使用して、SAS Cloud Analytic Services に接続し、セッションを開始します。サーバーのホスト名とポートを置き換えます。
- 2 ServerStatusOptions クラスのインスタンスは、実行するアクションを識別します。この場合、ServerStatusOptions クラスは serverStatus アクションに対応します。
- 3 com.sas.cas.session.close Java システムプロパティが true に設定されている場合、セッションは終了します。デフォルト値は false です。デフォルトの動作では、接続を閉じますが、タイムアウトになるか、再度接続するまで、セッションを実行したままにします。

アウトプット 1 serverStatus アクションの結果

```
{
  About= {
    CAS="Cloud Analytic Services"
    Version="4.00"
    VersionLong="V.04.00M0P10052023"
    Copyright="Copyright 2014-2023 SAS Institute Inc. All Rights Reserved."
    ServerTime="2023-10-06T17:43:25Z"
    System= {
      Hostname="cloud"
      OS Name="Linux"
      OS Family="LIN X64"
      OS Release="2.6.32-573.el6.x86_64"
      OS Version="#1 SMP Wed Jul 1 18:23:37 EDT 2015"
      Model Number="x86_64"
      Linux Distribution="Red Hat Enterprise Linux Server release 6.7 (Santiago)"
    }
    license= {
      site="SAS Institute Inc."
      siteNum=1
      expires="30Nov2023:00:00:00"
      gracePeriod=62
      warningPeriod=31
    }
    CASHostAccountRequired="OPTIONAL"
    Transferred="NO"
    CASCacheLocation="CAS Disk Cache"
  }
}
{
  server=server
  Node Count Total Actions
  -----
           8           1
  1 row

  {
    nodestatus=nodes
    Node Name      Role      Uptime (Sec) Running Stalled
    -----
    cloud1.unx.sas.com  backup ctl    0.044      0      0
    cloud2.unx.sas.com  worker        0.042      0      0
    cloud3.unx.sas.com  worker        0.045      0      0
    cloud4.unx.sas.com  worker        0.042      0      0
    cloud5.unx.sas.com  worker        0.045      0      0
    cloud6.unx.sas.com  worker        0.043      0      0
    cloud7.unx.sas.com  worker        0.045      0      0
    cloud.unx.sas.com   controller    0.041      0      0
    8 rows
  }
}
```

出力に示されているように、serverStatus アクションの結果には、About、server、nodestatus の3つのキーが含まれています。この例では、nodestatus は8つのマシン(1つのコントローラー、1つのバックアップコントローラー、6つのワーカーノード)を使用する分散サーバーを示しています。単一マシンサーバーの場合、結果にはコントローラーノードの行のみが含まれます。

キーで結果をフィルタリングできます。

```
...
CASActionResults<CASValue> results = client.invoke(options);
```

```

if (null != results) {
    System.out.println(CASValue.getValue(results, "server"));
}
...

```

注: CASValue オブジェクトの詳細については、“[結果の操作](#)” (12 ページ)を参照してください。

出力には Server キーの結果のみが含まれます。

```

{
  server=server
Node Count Total Actions
-----
          8          1
1 row
}

```

流暢なインターフェイスプログラミング

API は、流暢なプログラミングインターフェイスを提供します。たとえば、次のスニペットは、クライアントオブジェクトの `getActionSets` メソッドを使用して `builtins.serverStatus` アクションを呼び出します。

```

CASActionResults<CASValue> result = client.getActionSets()
    .builtins()
    .serverStatus()
    .invoke();

```

別の方法は、クライアントコンテキストのみを使用してアクションを呼び出すことです(これは、インスタンス化または使用された最後のクライアントを使用します)。

```

CASActionResults<CASValue> result = new ServerStatusOptions().invoke();

```

関連項目

“[変更: 流暢なインターフェイスを使用する](#)” (11 ページ)を参照してください。

例: 記述統計量の表示

セットアップ

この例では、分析するインメモリデータが必要です。cas-client JAR ファイルに含まれているサンプルプログラムを使用して CSV ファイルを読み取り、データをサーバーに転送できます。

summary アクションの実行

Orsales データがメモリにロードされた後、インメモリテーブルにアクセスし、データに対して summary アクションを実行するプログラムを実行できます。

例のコード 2 SummaryExample1.java

```
import com.sas.cas.CASActionResults;
import com.sas.cas.CASClient;
import com.sas.cas.CASClientInterface;
import com.sas.cas.CASValue;

import com.sas.cas.actions.simple.SummaryOptions;
import static com.sas.cas.actions.simple.SummaryOptions.SUBSET.*;
import com.sas.cas.actions.Castable;

public class SummaryExample1 {
    public static void main(String[] args) throws Exception {
        try (CASClientInterface client = new CASClient("cloud.example.com", 5570)) {
            SummaryOptions so = new SummaryOptions();
            so.setSummarySubset(                // 1
                new SummaryOptions.SUBSET[]{MIN, MAX, MEAN, N, NMISS,
                    STD, STDERR}
            );

            Castable table = new Castable();    // 2
            table.setName("orsales");
            table.setCaslib("casuser");
            so.setTable(table);

            CASActionResults<CASValue> results = client.invoke(so);

            for (int i = 0; i < results.getResultsCount(); i++) {
                System.out.println(results.getResult(i));
            }
        }
    }
}
```

- 1 使用可能な記述統計量のサブセットは、SummaryOptions クラスで使用されます。
- 2 Castable クラスは、インメモリテーブルを表すために使用されます。プログラムでは、テーブル名のみが設定されていますが、BY グループ処理で使用するパラメーター、where パラメーターを使用したフィルタリングなど、いくつかのパラメーターを設定できます。

アウトプット 2 summary アクションの結果

```
{
  Summary=Summary Descriptive Statistics for ORSALES
  Analysis Variable  Min    Max      N  Number Missing Mean          Std Dev.      Std Error
  -----
  Year              1999      2002  912              0      2000.5      1.11864745      0.03704212
  Quantity          10      9026  912              0  1465.08552632  1621.72304437      53.70061616
  Profit            209.8  552970.51  912              0  64786.23735197  84128.37618423  2785.76891008
  Total_Retail_Price 422.3  1159837.26  912              0 122090.6840625  166576.07510888  5515.88503487
  4 rows
}
```

変更: BY グループ処理

前述のプログラムで示したように、Castable クラスの `groupBy` パラメーターを指定できます。次のプログラムの変更は、BY グループ処理を実行する方法を示しています。

```
import com.sas.cas.CASActionResults;
import com.sas.cas.CASClient;
import com.sas.cas.CASClientInterface;
import com.sas.cas.CASValue;
import com.sas.cas.actions.Castable; // 1
import com.sas.cas.actions.Casinvardesc;

import com.sas.cas.actions.simple.SummaryOptions;
import static com.sas.cas.actions.simple.SummaryOptions.SUBSET.*;
import com.sas.cas.actions.Castable;

public class SummaryExample1_ByGroup {
    public static void main(String[] args) throws Exception {
        try (CASClientInterface client = new CASClient("cloud.example.com", 5570)) {
            SummaryOptions so = new SummaryOptions();
            so.setSummarySubset(
                new SummaryOptions.SUBSET[]{MIN, MAX, MEAN, N, NMISS, STD, STDERR}
            );

            Castable table = new Castable();
            table.setName("orsales");
            table.setCaslib("casuser");
            table.setGroupBy(new Casinvardesc[] { // 2
                new Casinvardesc().setName("year").setFormattedLength(32),
                new Casinvardesc().setName("product_line")
            });
            so.setTable(table);

            CASActionResults<CASValue> results = client.invoke(so);

            for (int i = 0; i < results.getResultsCount(); i++) {
                System.out.println(results.getResult(i));
            }
        }
    }
}
```

```

    }
}

```

- 1 Castable と Casinvardesc がインポートされます。
- 2 Castable インスタンスの setGroupBy()メソッドは、グループ化に使用する変数を設定するために使用されます。Orsales テーブルの Year および Product_Line 変数のフォーマット値は、一意のグループを形成するために使用されます。summary アクションは各グループで実行され、記述統計量が各グループに対して生成されます。

BY グループ分析を実行すると、結果には ByGroupInfo という名前のテーブルが含まれます。このテーブルは、指定された変数のフォーマット値から形成された一意のグループを示しています。

結果には、各グループの結果テーブルも含まれます。最初の 2 グループの結果(ByGroup1.Summary および ByGroup2.Summary)が、ByGroupInfo 情報の後に表示されます。

アウトプット 3 BY グループ処理による summary アクションの結果

```

{
  ByGroupInfo=ByGroupInfo ByGroupInfo
  Year Year_f      Product_Line      Product_Line_f  _key_
  -----
  1999      1999 Children      Children      1999      Children
  1999      1999 Clothes & Shoes Clothes & Shoes  1999      Clothes & Shoes
  1999      1999 Outdoors      Outdoors      1999      Outdoors
  1999      1999 Sports      Sports      1999      Sports
  2000      2000 Children      Children      2000      Children
  2000      2000 Clothes & Shoes Clothes & Shoes  2000      Clothes & Shoes
  2000      2000 Outdoors      Outdoors      2000      Outdoors
  2000      2000 Sports      Sports      2000      Sports
  2001      2001 Children      Children      2001      Children
  2001      2001 Clothes & Shoes Clothes & Shoes  2001      Clothes & Shoes
  2001      2001 Outdoors      Outdoors      2001      Outdoors
  2001      2001 Sports      Sports      2001      Sports
  2002      2002 Children      Children      2002      Children
  2002      2002 Clothes & Shoes Clothes & Shoes  2002      Clothes & Shoes
  2002      2002 Outdoors      Outdoors      2002      Outdoors
  2002      2002 Sports      Sports      2002      Sports
  16 rows
}
{
  ByGroup1.Summary=Summary Descriptive Statistics for ORSALES
  Analysis Variable  Min      Max      N      Number Missing Mean      Std Dev.      Std Error
  -----
  Year      1999      1999 44      0      1999      0      0
  Quantity      14      2458 44      0      650.29545455      593.2126792      89.43017626
  Profit      288.8 44840.28 44      0      12308.00204545      12121.86661139      1827.4401504
  Total_Retail_Price 580.4 82469.83 44      0      22797.47590909      22253.3548058      3354.81946443
  4 rows
}
{
  ByGroup2.Summary=Summary Descriptive Statistics for ORSALES
  Analysis Variable  Min      Max      N      Number Missing Mean      Std Dev.      Std Error
  -----
  Year      1999      1999 72      0      1999      0      0
  Quantity      25      6853 72      0      1499.33333333      1796.72279913      211.74581253
  Profit      1925.1 340341.14 72      0      58189.28652778      73018.65472409      8605.33098475
  Total_Retail_Price 3433 686906.59 72      0      111268.78444444      145044.2577739      17093.62970735
  4 rows
}
}

```

最初の 2 グループの結果のみが表示されます。ByGroupInfo テーブルの行数で示されているように、合計 16 の結果テーブルを使用できます。

変更: 流暢なインターフェイスを使用する

次のプログラムは、BY グループ処理プログラムを変更したものです。ハイライト行は、流暢なインターフェイススタイルを示しています。

例のコード 3 SummaryExampleFluent.java

```
import com.sas.cas.CASActionResults;
import com.sas.cas.CASClient;
import com.sas.cas.CASClientInterface;
import com.sas.cas.CASValue;

import com.sas.cas.actions.simple.SummaryOptions;
import static com.sas.cas.actions.simple.SummaryOptions.SUBSET.*;
import com.sas.cas.actions.Castable;
import com.sas.cas.actions.Casinvardesc;

public class SummaryExampleFluent {
    public static void main(String[] args) throws Exception {
        try {
            CASClientInterface client = new CASClient("cloud.example.com", 5570);
            Castable table = new Castable();
            table.setName("orsales");
            table.setCaslib("casuser");
            table.setGroupBy(new Casinvardesc[] {
                new Casinvardesc().setName("year").setFormattedLength(32),
                new Casinvardesc().setName("product_line")
            });

            CASActionResults<CASValue> results =
                client.getActionSets().simple().summary().setTable(table).setSubSet(
                    new SummaryOptions.SUBSET[]{MIN, MAX, MEAN, N, NMISS, STD, STDERR}
                ).invoke();

            if (null == results) {
                System.err.println("Failed to run the summary action.");
                return;
            }

            for (int i = 0; i < results.getResultsCount(); i++) {
                System.out.println(results.getResult(i));
            }
            finally {
                // production code should include exception handling
            }
        }
    }
}
```

プログラムの結果は、BY グループ処理プログラムと同じです。

結果の操作

次のリストは、アクションの結果を操作するために一般的に使用されるクラスを示しています。

`com.sas.cas.CASActionResults`

アクションの実行後、結果は `CASActionResults` オブジェクトにラップされます。

`CASActionResults` オブジェクトには、`CASValue` オブジェクト、ログ、およびパフォーマンス情報のリストが含まれています。

`com.sas.cas.CASValue`

`CASValue` クラスは、キーと値のペアです。キーは文字列であり、値は任意のデータ型にすることができます。一般的なデータ型は `CASTable` です。表形式の結果は、`CASTable` オブジェクトとして表されます。その他の可能な値の型は、プリミティブ(整数、長整数、倍精度浮動小数点数、文字列、ブール)と `CASValue` 型のリストです。

`com.sas.cas.CASTable`

`CASTable` クラスは、メタデータをクエリし、結果テーブルのデータをトラバースするメソッドを提供します。

次のコードは、`CASTable` のコンテンツを表示する 1 つの方法を示しています。

```
import com.sas.cas.CASActionResults;
import com.sas.cas.CASClient;
import com.sas.cas.CASClientInterface;
import com.sas.cas.CASValue;
import com.sas.cas.actions.Casouttable;
import com.sas.cas.actions.table.TableActions;
import com.sas.cas.actions.table.FetchOptions;
import com.sas.cas.CASTable; // 1
import com.sas.cas.CASTable.OutputType;

import com.sas.cas.actions.simple.SummaryOptions;
import static com.sas.cas.actions.simple.SummaryOptions.SUBSET.*;
import com.sas.cas.actions.Castable;

public class WorkingWithResults {
    public static void main(String[] args) throws Exception {
        try (CASClientInterface client = new CASClient("cloud.example.com", 5570)) {

            SummaryOptions so = new SummaryOptions();
            so.setSummarySubset(
                new SummaryOptions.SUBSET[]{MIN, MAX, MEAN, N, NMISS,
                    STD, STDERR}
            );

            Castable table = new Castable();
            table.setName("orsales");
            table.setCaslib("casuser");
            so.setTable(table);

            CASActionResults<CASValue> results = client.invoke(so);
```

```

com.sas.cas.CASTable resultTable =           // 2
    (CASTable)results.getResult(0).getValue();

StringBuffer sb = new StringBuffer();         // 3
for (int i = 0; i < resultTable.getColumnCount(); i++) {
    if (i > 0) sb.append(" ");
    sb.append(resultTable.getColumns()[i].getName());
}
System.out.println("\nTable " + resultTable.getName() +
    " columns:\n" + sb);

System.out.println("\nTable " + resultTable.getName() + // 4
    " data:");
for (int row = 0; row < resultTable.getRowCount(); row++) {
    sb = new StringBuffer();
    for (int i = 0; i < resultTable.getColumnCount(); i++) {
        if (i > 0) sb.append(" ");
        sb.append(resultTable.getStringAt(row, i));
    }

    System.out.println(sb);
}
}
}
}

```

- 1 CASTable と CASTable.OutputType をインポートします。
- 2 出力テーブルを CASTable.object に保存します。
- 3 テーブルの列名のリストを表示します。
- 4 テーブルデータを表示します。

次に出力を示します。

```

Table Summary columns:
Column Min Max N NMiss Mean Std StdErr

Table Summary data:
Year 1999.0 2002.0 912.0 0.0 2000.5 1.118647450518278 0.037042118608994416
Quantity 10.0 9026.0 912.0 0.0 1465.0855263157894 1621.723044367722 53.70061616157693
Profit 209.8 552970.510000002 912.0 0.0 64786.23735197367 84128.37618423298 2785.76891008392
Total_Retail_Price 422.3 1159837.26 912.0 0.0 122090.68406250014 166576.07510887776 5515.88503486514

```

CASTable クラスには、結果テーブルを文字列値としてフォーマットするのに便利な toString() メソッドもあります。次のコードは、返し方を示しています。

- 位置を揃えた列でフォーマットされた文字列
- 列ラベルを表示する
- 最大 1000 行を表示する
- 数値型の小数点以下 6 桁を表示する

```

String str = resultTable.toString(OutputType.PRETTY, true, 1000, 6);
System.out.println(str);

```

次に出力の例を示します。

Summary Descriptive Statistics for orsales							
Analysis Variable	Minimum	Maximum	N	N Miss	Mean	Std Dev	Std Error
Year	1999	2002	912	0	2000.5	1.118647	0.037042
Quantity	10	9026	912	0	1465.085526	1621.723044	53.700616
Profit	209.8	552970.51	912	0	64786.237352	84128.376184	2785.76891
Total_Retail_Price	422.3	1159837.26	912	0	122090.684063	166576.075109	5515.885035
4 rows							

次のコードは、返し方を示しています。

- CSV (カンマ区切り値)としてフォーマットされた文字列
- 列ラベルを表示する
- 最大 500 行を表示する
- 数値型の小数点以下 8 桁を表示する

```
String csv = resultTable.toString(OutputType.CSV, true, 500, 8);
System.out.println(csv);
```

次に出力の例を示します。

```
"Summary","Descriptive Statistics for orsales"
"Analysis Variable","Minimum","Maximum","N","N Miss","Mean","Std Dev","Std Error"
"Year",1999,2002,912,0,2000.5,1.11864745,0.03704212
"Quantity",10,9026,912,0,1465.08552632,1621.72304437,53.70061616
"Profit",209.8,552970.51,912,0,64786.23735197,84128.37618423,2785.76891008
"Total_Retail_Price",422.3,1159837.26,912,0,122090.6840625,166576.07510888,5515.88503487
4 rows
```

イベントの操作

イベントと CASActionResults クラス

デフォルトでは、すべてのサーバーイベントは、`invoke()`メソッドによって返される `CASActionResults` オブジェクトの一部として返されます。これは、アクションの完了後にすべてのイベントにアクセスできることを意味します。または、サーバーからストリーミングされるイベントを受信するために、イベントの種類ごとにリスナーを登録できます。

ログイベント

ログイベントは、情報がログに送信されるときにサーバーによって生成されます。ログイベントは、種類(WARN や ERROR など)とメッセージで構成されます。

サーバーからストリーミングされるときにログイベントを受信する場合は、アクションを呼び出す前に、アクションオプションオブジェクトに CASLogEventListener を登録する必要があります。

```
// Set the log event listener. This handles the log events as they
// come from the server instead of accessing them from the results
// after the action is complete.
options.setLogEventListener(new CASLogEventListener() {

    @Override
    public boolean handleLogEvent(CASActionOptions options, CASLogEvent logEvent) {

        System.out.println("# " + logEvent);

        // Return the propagate flag. If true, the log event
        // is propagated to the result object. Otherwise nothing
        // else will be done with the log event.
        return false;
    }
});
```

パフォーマンスイベント

パフォーマンスイベントは、アクションの特定のパフォーマンス特性について呼び出し元に通知するようにアクションが設計されている場合に、サーバーによって生成されます。パフォーマンスイベントには、タイミング(経過時間、CPU 時間など)、メモリ使用量、ノード使用量、および CPU コア使用量を含めることができます。

サーバーからストリーミングされるときにパフォーマンスイベントを受信する場合は、アクションを呼び出す前に、アクションオプションオブジェクトに CASPerformanceEventListener を登録する必要があります。

```
// Set the performance event listener. This handles the performance events as they
// come from the server instead of accessing them from the results
// after the action is complete.
options.setPerformanceEventListener(new CASPerformanceEventListener() {

    @Override
    public boolean handlePerformanceEvent(CASActionOptions options, CASPerformanceEvent
performanceEvent) {

        System.out.println("# " + performanceEvent);

        // Return the propagate flag. If true, the performance event
        // is propagated to the result object. Otherwise nothing
        // else will be done with the performance event.
        return false;
    }
});
```

配置イベント

配置イベントは、アクションが完了または失敗したときにサーバーによって生成されます。配置イベントには、重要度、理由およびメッセージが含まれます。

サーバーからストリーミングされるときに配置イベントを受信する場合は、アクションを呼び出す前に、アクションオプションオブジェクトに `CASDispositionEventListener` を登録する必要があります。

```
// Set the disposition event listener. This handles the disposition events as they
// come from the server instead of accessing them from the results
// after the action is complete.
options.setDispositionEventListener(new CASDispositionEventListener() {

    @Override
    public boolean handleDispositionEvent(CASActionOptions options, CASDispositionEvent dispositionEvent) {

        System.out.println("# " + dispositionEvent);

        // Return the propagate flag. If true, the disposition event
        // is propagated to the result object. Otherwise nothing
        // else will be done with the disposition event.
        return false;
    }
});
```

メッセージタグイベント

メッセージタグイベントは他のイベントタイプとは異なります。いくつかのアクションでは、追加の情報を要求するアクションの呼び出し中にクライアントにメッセージを送り返すことができます。これはメッセージタグで行われます。タグのリストについては、`com.sas.cas.message.CASMessageHeader` を参照してください。

テーブルアクションセットの `addTable` アクションは、メッセージタグイベントを使用するアクションの一例です。`addTable` アクションはメッセージをクライアントに送り返すので、クライアントはいつサーバーにデータ行を送信してテーブルにデータを入力するかを決定できます。DATA タグを持つこのメッセージを処理するには、アクションオプションに(タイプ `CASMessageTagHandler` の)メッセージタグハンドラーを登録する必要があります。

次のコードは、単純にデータを返さない `CASMessageTagHandler` の例を示しています。

```
@Override
public boolean handleMessageTag(CASMessageTagEvent event) throws CASException, IOException {

    // Just create the table; don't send any rows
    CASDataAppender.sendZeroRows(event);

    // Do not propagate the response
    return false;
}
```

このメッセージタグハンドラーを使用するには、アクションを呼び出す前に、アクションオプションにハンドラーを登録する必要があります。

```
options.setMessageTagHandler(CASMessageHeader.TAG_DATA, <the handler>);
```

データ行をサーバーに送信するには(テーブルにデータを追加するには)、最初の例よりも高度な次のメッセージハンドラーを使用します。

```
@Override
public boolean handleMessageTag(CASMessageTagEvent event) throws CASException, IOException {

    // Get the variable list
    Addtablevariable[] vars = (Addtablevariable[]) event.getOptions().get(AddTableOptions.KEY_VARS);
    Integer reclen = (Integer) event.getOptions().get(AddTableOptions.KEY_RECLLEN);
    if (reclen == null) {
        // This shouldn't happen. The server should have verified.
        throw new CASException("Missing reclen");
    }

    // Create our data appender
    CASDataAppender appender = new CASDataAppender(event, vars, reclen, bufSize);

    // Creates a new random generator with the given seed
    Random r = new Random(0);

    for (int i = 0; i < nrows; i++) {
        appender.setDouble(0, i);
        appender.setString(1, "Some String at " + i);
        appender.setDouble(2, r.nextDouble());
        appender.appendRecord();
    }

    appender.close();

    // Do not propagate the response
    return false;
}
```

前記のコードは、addTable アクション(`com.sas.actions.table.AddTableOptions` クラス)からのメッセージタグイベントの処理に適しています。メッセージタグイベントを使用する別のアクションである upload アクション(特殊な `com.sas.cas.io.UploadDataTagHandler` クラス)も使用可能です。ファイル名または `InputStream` が入力として受け入れられます。

サーバーイベント

サーバーによって生成されるいくつかのタイプのイベントに登録できます。このイベントは次のとおりです。

- caslib リスト変更済み
- テーブルリスト変更済み
- アクションセット追加済み
- データソースリスト変更済み

■ アクセス許可変更済み

これらのイベントの 1 つ以上に登録するには、イベントリスナーを追加します。

```
client.addListener(CASConstants.EVENT_FLAG_CASLIBS, new CASEventListener() {
    @Override
    public void handleCASEvent(long flag) {
        // Do something with the event
    }
});
```

複数のイベントに登録できます。

```
client.addListener(
    CASConstants.EVENT_FLAG_CASLIBS | CASConstants.EVENT_FLAG_TABLES,
    new CASEventListener() {
        @Override
        public void handleCASEvent(long flag) {
            // Do something with the event
            if (flag == CASConstants.EVENT_FLAG_CASLIBS) {
                // Do something - the caslibs have changed
            }
            else if (flag == CASConstants.EVENT_FLAG_TABLES) {
                // Do something - the tables have changed
            }
        }
    });
```

サーバーイベントはアクション応答の一部としてパッケージ化されているため、非同期イベントではありません。イベント通知は、アクションが完了すると発生します。一部の種類のクライアントは、イベントの"ポーリング"を望む場合があります。これは、com.sas.cas.actions.builtins.PingOptions などのアクションを呼び出すだけで実行できます。

ソケットイベント

CASSocketEventListener クラスは、新しいソケット接続が確立されたとき、およびソケット接続が閉じられたときのコールバックを提供します。これにより、タイムアウトなどのソケット設定をカスタマイズできます。

```
// Register our socket event listener
client.setSocketEventListener(new CASSocketEventListener() {

    @Override
    public void handleSocketConnectionEvent(CASClientInterface client, Socket socket) {
        System.out.println("Socket opened");
    }

    @Override
    public void handleSocketClosedEvent(CASClientInterface client, Socket socket) {
        System.out.println("Socket closed");
    }
});
```

接続とアクション: 詳細

TLS (SSL)接続

概要

SAS Cloud Analytic Services は、サーバーと Java アプリケーションなどのクライアント間の暗号化された接続をサポートします。サーバーがクライアントとサーバー間で暗号化を使用するように構成されている場合、クライアントとサーバー間のすべてのユーザーデータとすべての認証データは、TLS を使用して暗号化されます。

サーバーが暗号化を使用するように構成されている場合、サーバーは認証フェーズ中にクライアントに通知し、CASClient クラスは標準 TLS を使用して接続を試みます。

TLS と SSL の詳細については、[SAS Viya プラットフォームの暗号化: 移動中のデータ](#)を参照してください。

TLS を使用した REST 接続

完全配置では、SAS Viya プラットフォームは HTTP サーバーを使用してサービスへの要求をプロキシ経由にします。デフォルトでは、HTTP サーバーは TLS を使用するように構成されています。HTTP サーバーは自動的に生成された自己署名証明書を使用できますが、ユーザーが証明書を提供することもできます。

CASRestClient クラスを使用して、HTTP サーバーに接続できます。例については、[例のコード 5 \(20 ページ\)](#)を参照してください。

CASRestClient クラスを使用しておらず、HTTP サーバーが既知の証明機関(CA)によって署名された証明書を使用していない限り、trustedcerts.jks ファイルか、または証明書の署名に使用された CA 証明書を含む Java トラストストアにアクセスする必要があります。

TLS を使用したバイナリ接続

完全配置では、CAS は TLS を使用するようにも構成されます。CAS は HashiCorp Vault によって自動的に生成される自己署名証明書を使用します。CAS へのバイナリ接続(通常はポート 5570)を使用する場合は、サーバーが TLS を使用するように構成されていても、CASClient クラスを使用できます。

この場合、証明書は自己署名されているため、Java アプリケーションは CAS コントローラーホストにある trustedcerts.jks ファイルにアクセスする必要があります。

証明書トラストストアおよび TLS 関連のシステムプロパティ

Java アプリケーションが証明書トラストストアへのアクセスを必要とする場合、CAS コントローラーホストのデフォルトの場所は次のとおりです。

/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts/trustedcerts.jks

ヒント コントローラーからトラストストアにアクセスするかわりに、keytool コマンドを使用して、キーストア内の信頼されている証明書を管理できます。

使用するトラストストアを示すために一般的に使用されるのは、次の 2 つのシステムプロパティです。

-Djavax.net.ssl.trustStore=*path-to-trustedcerts.jks*

HashiCorp Vault によって作成される trustedcerts.jks ファイルへのローカルパス、または証明書に署名した CA 証明書を使用して作成するトラストストアへのパスを指定します。

-Djavax.net.ssl.trustStorePass=*changeit*

トラストストアのパスワードを指定します。

REST 接続

デフォルトでは、CASClient クラスはソケットを使用してサーバーと通信します。このドキュメントの例では、デフォルトの動作を使用しています。HTTP または HTTPS を使用する REST 実装を使用して、CAS コントローラーで使用可能な REST インターフェイスと通信することもできます。REST インターフェイスは、ユーザー名とパスワードの認証をサポートしています。これらの値を指定することも、.authinfo ファイルから取得することもできます(次のセクションを参照)。

ユーザー名とパスワード認証で REST インターフェイスを使用するには、CASClient ではなく CASRestClient クラスをインスタンス化するだけです。

例のコード 4 REST 接続のサンプル

```
// import the following classes
import com.sas.cas.rest.CASRestClient;
import java.util.URL;

// Instantiate a new REST client. Set the URL and optionally, user name and password.
CASClientInterface client =
    new CASRestClient(new URL("http://cloud.example.com:8777"));
```

例のコード 5 TLS を使用した REST 接続のサンプル

```
// import the following classes
import com.sas.cas.rest.CASRestClient;
import java.util.URL;

// For full deployments of the SAS Viya platform that use Apache HTTP Server with TLS, connect to
// the web server instead of the CAS server.
CASClientInterface client =
    new CASRestClient(new URL("https://webserver.example.com
```

```
/cas-shared-default-http/"));
```

完全配置と TLS を使用する HTTP サーバーを使用して、HTTP サーバー経由で CAS に接続します。URL の cas-shared-default-http 部分は、一般的な SAS Viya プラットフォームの配置に適用されます。配置インスタンスが配置名にデフォルトを使用しなかった場合、URL のその部分は異なります。

CASClient クラスと同様に、ユーザー名とパスワードの引数を指定しない場合、CASRestClient クラスは authinfo ファイルで資格情報を検索します。

注: JSON はサーバーとの通信に使用されます。CLASSPATH には、json-20230227.jar (またはそれ以降)を含める必要があります。メッセージタグイベントを使用してサーバーとデータを交換するアクションは、REST インターフェイスではサポートされていません。これには addTable アクションと upload アクションが含まれますが、これらに限定されるものではありません。

ヒント CASRestClient クラスを使用する場合、CASClient クラスと比較して特別なコーディングは必要ありません。ただし、JSON メッセージを表示する場合は、コマンドラインで次のプロパティを設定します。

```
-Dcom.sas.cas.rest.debug=true
```

authinfo ファイルの詳細

ユーザー名またはユーザー名とパスワードが指定されていない場合、CASClient クラスは authinfo ファイルを検索します。authinfo ファイルの検索順序は次のとおりです。

- 1 AUTHINFO 環境変数(設定されている場合)
- 2 NETRC 環境変数(設定されている場合)
- 3 %HOMEDRIVE%%HOMEPATH%_authinfo (Windows のみ)
- 4 \$HOME/.authinfo
- 5 %HOMEDRIVE%%HOMEPATH%_netrc (Window のみ)
- 6 \$HOME/.netrc

authinfo ファイルに関するルール:

- 現在のユーザーはファイルの所有者である必要があります。
- 他のユーザー/グループは、ファイルへの読み取り/書き込みアクセス権を持つことができません (UNIX のみ)。

authinfo ファイルによる認証を無効にするには、システムプロパティ com.sas.cas.authinfo.enabled を false に設定します。

関連項目

- [Authinfo ファイルを使用したクライアント認証](#)

ユーザー名とパスワードのプロンプト

ユーザー名とパスワードのプロンプトができるトークンプロバイダーを使用できます。コードがスタンドアロンのコンソールアプリケーションとして実行される場合、次のコードはトークンプロバイダーを登録してプロンプトを実行します。

```
client.setSecurityTokenProvider(new CASPasswordTokenProvider());
```

アプリケーションの実行時に他の資格情報が提供されない場合、次のプロンプトが表示されます。

```
User name:
```

```
Password:
```

アプリケーションが Eclipse などの IDE 内で実行されている場合は、かわりに Swing ポップアップダイアログボックスが表示されます。

Kerberos 認証

ユーザー名またはパスワードの資格情報が(指定または authinfo ファイルから)使用できない場合、CASClient は現在のユーザーの Kerberos チケットを取得しようとします。システムプロパティ `java.security.auth.login.config` を使用して、有効な JAAS 構成ファイルを指定する必要があります。

```
-Djava.security.auth.login.config=./jaas.conf
```

構成ファイル内のデフォルトの JAAS 構成名は、`com.sun.security.jgss.krb5.initiate` です。デフォルトの JAAS 構成名をオーバーライドするには、プロパティ `com.sas.cas.kerberos.config.name` に名前を設定します。

例のコード 6 JAAS 構成エントリのサンプル

```
com.sun.security.jgss.krb5.initiate {
    com.sun.security.auth.module.Krb5LoginModule
    required
    useTicketCache=true
    renewTGT=true
    doNotPrompt=true;
};
```

Kerberos コンテキストを作成するときは、サービス名を指定する必要があります。デフォルトのサービス名は `sascas` です。デフォルトのサービス名をオーバーライドするには、プロパティ `com.sas.cas.kerberos.service.name` に名前を設定します。システムプロパティ `com.sas.cas.kerberos.realm.name` を使用して、オプションのレルム名を指定することもできます。サービスプリンシパル名(SPN)は、サービス名、ターゲットホスト名、およびオプションのレルム名を使用して構成されます。

```
service-name
/hostname@realm-name
```

SPN の構成をオーバーライドするには、システムプロパティ `com.sas.cas.kerberos.spn` を使用して完全な SPN を指定します。

表 1 トラブルシューティング

エラーメッセージ	提案
"Unable to obtain Principal Name for authentication"例外	Java 6 を使用している場合、この例外が発生する可能性があります。これは、Java 6 のバグが原因である可能性があります。 kinit -R を実行してチケットを更新してください。
"unsupported key type found the default TGT: 18"のようなデバッグ出力	このメッセージは、Java インストールが必要な暗号化の種類すべてをサポートしていないことを示しています。Java Cryptography Extension (JCE) Unlimited Strength ポリシーファイルをインストールする必要があります。 現在インストールされているセキュリティプロバイダーを表示するには、 <code>com.sas.cas.samples.ShowSecurityProviders</code> サンプルを実行します。AES キーの最大長は無制限(または 2147483647)にする必要があります。
"javax.security.auth.login.LoginException: No LoginModules configured for com.sun.security.jgss.krb5.initiate"例外	この例外は、Kerberos が環境に対して適切に構成されていないことを示している可能性があります。

関連項目

["システムプロパティ" \(31 ページ\)](#)

認証方法の強制

Java アプリケーションに特定の認証メカニズムを強制できます。CAS_AUTH_METHOD 環境変数は、次のいずれかに設定できます。

- authinfo (または netrc)
- OAuth
- userpass (または password)
- Kerberos

指定した認証メカニズムに十分な情報を提供すると、デフォルトのプロセスに関係なく、そのメカニズムが使用されます。提供された情報に基づいて不可能な認証方法を指定すると('password'を指定してパスワードを指定しないなど)、認証は失敗します。

期限切れのチケットまたは証明書

場合によっては、OAuth チケットが期限切れになることがあります。これが発生した場合、CASClient クラスは登録された securityTokenProvider でコールバックしようとします。プロバイダーは、次のようにクライアントに登録する必要があります。

```
client.setSecurityTokenProvider(new CASSecurityTokenProvider() {
    public CASSecurityToken getSecurityToken(CASClientInterface client, REQUEST_TYPE type)
        throws CASException, IOException {

        CASSecurityToken token = ...

        return token;
    }
});
```

- OAuth チケットの有効期限が切れている場合、getSecurityToken 呼び出しの REQUEST_TYPE は EXPIRATION に設定されます。この場合、トークンフィールドに新しい OAuth チケットを含む新しい CASSecurityToken を返す必要があります。
- サーバーがクライアントの認証中に Kerberos に切り替える必要がある場合、getSecurityToken 呼び出しの REQUEST_TYPE は EXPIRED_TICKET に設定されます。この場合、GSSCredential を使用して新しい CASSecurityToken を返す必要があります。

SAS 欠損値の操作

SAS には欠損値という概念があります。これは null 値と比較されることが多いのですが、それとは多少異なります。

CAS アクションは欠損値タイプを返す可能性があります。この欠損値の場合、ビットは標準の double NaN 値に設定されます。欠損値タイプを取得するには、com.sas.cas.io.SASMissingValue クラスを使用してデコードします。

```
if (Double.isNaN(value)) {
    String type = SASMissingValue.getMissingValueType(value);
    ...
}
```

欠損値タイプは、CAS アクションによって作成された対応する欠損タイプを表す文字("I"など)です。

アクションパラメーター

このドキュメントに示されているコード例では、CAS アクションに対応するクラスに "setter" メソッドを使用しています。たとえば、次のコードでは、SummaryOptions クラスの setTable() メソッドを使用して、分析で使用するテーブルを識別しています。

```
SummaryOptions so = new SummaryOptions();
```

```
Castable table = new Castable();
table.setName("orsales");
so.setTable(table);
```

前の例で示したように、通常、CAS アクションのクラスをインスタンス化してから、名前付きパラメーター値を設定します。アクションオブジェクトは、基本的にキーと値を持つ Map です。Map とのこの類似性により、パラメーターを指定してアクションを呼び出す別の方法が可能になります。厳密に言えば、必要な唯一の引数は CASActionOptions オブジェクトであり、これによりアクションを呼び出す別の方法が可能になります。

一部の CAS アクションは、引数としてパラメーターリストを取ります。一般的な例は、table パラメーターです。パラメーターリストはパラメーターのサブリストであり、Map としてか、アクションセットまたはアクション名のない CASActionOptions インスタンスとして表すことができます。次のコードは、[例のコード 2 \(8 ページ\)](#)を書き直したもので、アクションとパラメーターを基本的に Maps および CASActionOptions インスタンスとする方法を示しています。

例のコード 7 UseOptions.java

```
import com.sas.cas.CASActionResults;
import com.sas.cas.CASClient;
import com.sas.cas.CASClientInterface;
import com.sas.cas.CASValue;

import com.sas.cas.CASActionOptions;

import java.util.Map;
import java.util.HashMap;

public class UseOptions {
    public static void main(String[] args) throws Exception {
        try {
            CASClientInterface client = new CASClient("cloud.example.com", 5570);

            // Set up the simple.summary action options
            CASActionOptions summaryOptions =
                new CASActionOptions("simple", "summary");           // 1
            summaryOptions.setParameter("summarySubset",
                new String[] {"MIN", "MAX", "MEAN", "N", "NMISS", "STD", "STDERR"}
            );

            // A table is a parameter list which can be represented as Map or
            // an instance of CASActionOptions (which is a Map)
            Map<String, Object> tableParams = new HashMap<String, Object>(); // 2

            // The table name
            tableParams.put("name", "orsales");

            // The table caslib
            tableParams.put("casLib", "casuser");

            // An array of parameter names
            tableParams.put("vars",
                new String[] {"year", "quarter", "quantity", "profit",
                    "total_retail_price"} // 3
            );
```

```

// Set the table parameter list
summaryOptions.setParameter("table", tableParams);

// Perform the analysis - generate descriptive statistics
CASActionResults<CASValue> results = client.invoke(summaryOptions);

for (int i = 0; i < results.getResultsCount(); i++) {
    System.out.println(results.getResult(i));
}
}
catch(Exception e) {
    // Include appropriate exception handling.
    System.out.println("An error occurred:\n" + e);
}
finally {
    // Include any finally code that is needed.
}
}
}

```

- 1 CASActionOptions("simple", "summary")クラスのインスタンスでは、アクションセットとして simple、アクションとして summary が指定されます。このインスタンスは、com.sas.cas.actions.simple.SummaryOptions クラスのインスタンスのインスタンス化と同等です。
- 2 HashMap インスタンスは Castable クラスと同等です。
- 3 変数の指定は、SummaryExample1.java コードと同等ではありません。ただし、これは、Map を使用して summary アクションの vars パラメーターを指定する方法を示しています。

vars パラメーターで変数名を指定する別の方法は、List クラスのインスタンスを使用することです。

```

List<String> variableNames = new ArrayList<String>();
variableNames.add("year");
variableNames.add("quarter");
variableNames.add("quantity");
variableNames.add("profit");
variableNames.add("total_retail_price");
tableParams.put("vars", variableNames);

```

結果は、SummaryExample1.java コードの結果と同じです。

```

{
  Summary=Summary Descriptive Statistics for ORSALES
  Analysis Variable  Min    Max      N   Number Missing Mean          Std Dev.      Std Error
  -----
  Year              1999    2002  912           0      2000.5      1.11864745    0.03704212
  Quantity           10    9026  912           0  1465.08552632  1621.72304437  53.70061616
  Profit            209.8  552970.51  912           0  64786.23735197  84128.37618423  2785.76891008
  Total_Retail_Price 422.3  1159837.26  912           0 122090.6840625  166576.07510888  5515.88503487
  4 rows
}

```

セッションの詳細

CASClient インスタンスが作成されても、クライアントはすぐにサーバーに接続するわけではありません。クライアント/サーバー接続は、クライアントがサーバー側アクションを初めて呼び出すときに確立されます。CASClient インスタンスの作成時にクライアント側でセッション ID が指定されていない場合、新しいセッションが作成されます。

CASClient クラスは、既存のセッションからセッション ID を取得する getSessionID() メソッドを提供します。既存のセッションにアタッチする場合は、CASClient インスタンスで setSessionID() メソッドを使用して、セッションの ID を指定します。クライアントがどのセッションにもアタッチされていない場合、getSessionID() は null を返します。

CASClient クラスは、java.lang.AutoCloseable インターフェイスを実装します。その結果、クライアントインスタンスが try ブロックで作成された場合、セッションは終了し、try ブロックの最後にソケット接続が自動的に閉じられます。セッションの終了と接続を閉じる処理をより詳細に制御したい場合は、これらのタスクを実行するコードを記述します。

ここで示唆されているように、セッションの終了(EndSession アクションの呼び出し)と接続を閉じること(CASClient.close() メソッドの呼び出し)は別物です。EndSession アクションは、サーバー上の現在のセッションを終了します。CASClient.close() メソッドは、クライアントとサーバー間のソケット接続を閉じます。

CASClient.close() を呼び出すと、接続は閉じられますが、セッションはまだ存在し、サーバー上で引き続き実行されます。これにより、後で同じセッションに再接続できます。セッションはタイムアウトの対象であるため、切断されたセッションは、タイムアウトになるまで実行されます。切断されたセッションは無期限実行されません。

セッションを終了して接続を閉じるには、EndSession アクションの次に CASClient.close() という順序で呼び出す必要があります。または、接続を閉じる前に EndSession アクションを呼び出す CASClient.close(true) を呼び出します。

サンプル Java プログラム

サンプルプログラムについて

いくつかのサンプルプログラムをソースコードとしてダウンロードできます。これらのサンプルでは、このドキュメントに含まれているプログラム例よりも幅広いクラスの使用方法が示されています。また、クラスのインスタンスのインスタンス化、適度な例外処理など、より現実的なプログラミング方法も示されます。プログラムは次の URL からダウンロードできます。

<http://support.sas.com/downloads/package.htm?pid=1976>

プログラムは、cas-client JAR ファイル内のコンパイル済みクラスとしても使用できます。コマンドラインからサンプルプログラムを実行できます。com.sas.cas.samples.table.AddCSVSample は、このドキュメントの別のセクションの [セットアップ \(6 ページ\)](#) に含まれています。

-help コマンドラインオプションを使用して各プログラムを実行し、key=value ペアとして指定するパラメーターを確認できます。

```
java com.sas.cas.samples.FileInfoSample -help
```

```
Usage:
    FileInfoSample [caslib=<caslib>] [disablessl=<true|false>] host=<host> [n=<n>]
[nodes=<nodes>] [password=<password>] [path=<path>] [pause=<true|false>] [port=<port>]
[rest=<true|false>] [sessionid=<sessionid>] [username=<username>]
Where:
    caslib      CAS library name
    disablessl  disablessl (default is false)
    host        Host name (or host:port)
    n           The number of times to execute the action (default is 1)
    nodes       Number of nodes to use when creating a session (default is 0)
    password    Password
    path        The path of the file or table (default is /)
    pause       'true' to pause before invoking the sample (default is false)
    port        Port number
    rest        'true' to use the REST interface (default is false)
    sessionid   Session ID to join
    username    User name
```

FileInfoSample プログラムを実行するには、コマンドラインは次の例のようになります。

```
java -Dcom.sas.cas.authinfo.enabled=true com.sas.cas.samples.FileInfoSample \
host="cloud.example.com" port=5570
```

ヒント サンプルの実行を簡単にするために、CASClient.properties という名前のファイルを現在の作業ディレクトリに作成し、ホストやポートなどの頻繁に使用される引数を指定できます。Java プロパティファイルには、標準の key=value 構文を使用します。プロパティファイルでユーザー名キーとパスワードキーを指定できますが、authinfo ファイルに保存する方が安全です。

com.sas.cas.samples

これらのサンプルは、さまざまな機能をカバーしています。これにより、アクションの実行とイベントの操作の基本が示されます。

com.sas.cas.samples.ActionEventSample

アクションイベントリスナーを登録する方法を示します。アクションイベントは、アクション呼び出しがサーバーに送信される前に発生します。

com.sas.cas.samples.DispositionEventSample

アクションの完了後に結果から配置イベントにアクセスするのではなく、配置イベントが発生したときに処理する配置イベントリスナーを登録する方法を示します。

com.sas.cas.samples.FetchSample

インメモリテーブルからデータの基本的なフェッチを実行する方法を示します。

com.sas.cas.samples.FileInfoSample

caslib のデータソースからファイル情報を取得する方法を示します。

`com.sas.cas.samples.LoadActionSetSample`

アクションセットをセッションにロードする方法を示します。

`com.sas.cas.samples.LogEventSample`

アクションの完了後に結果からログイベントにアクセスするのではなく、ログイベントが発生したときに処理するログイベントリスナーを登録する方法を示します。

`com.sas.cas.samples.PerformanceEventSample`

アクションの完了後に結果からパフォーマンスイベントにアクセスするのではなく、パフォーマンスイベントが発生したときに処理するパフォーマンスイベントリスナーを登録する方法を示します。

`com.sas.cas.samples.PingSample`

`builtins.ping` アクションの実行方法を示します。

`com.sas.cas.samples.QueryActionSetSample`

アクションセットがロードされているかどうかを判別するためにサーバーにクエリする方法を示しています。

`com.sas.cas.samples.ReflectionSample`

ロードされたアクションセットとアクションを反映する方法を示します。

`com.sas.cas.samples.ServerStatusSample`

サーバーステータスアクションを呼び出す方法を示します。

`com.sas.cas.samples.ShutdownSample`

サーバーをシャットダウンする方法を示します。

`com.sas.cas.samples.SocketEventSample`

ソケットイベントリスナーを登録して、ソケットの開く/閉じるイベントを処理する方法を示しています。

`com.sas.cas.samples.TableAttributesSample`

テーブル属性を取得する方法を示します。

`com.sas.cas.samples.TableEventSample`

テーブルイベントリスナーを登録して、CASTable 作成イベントを処理する方法を示します。

`com.sas.cas.samples.builtins`

これらのサンプルは、ビルトインアクションセットからの 2 つのシステムレベルアクションを示しています。

`com.sas.cas.samples.builtins.HelpSample`

サーバーからヘルプを取得する方法を示します。

`com.sas.cas.samples.builtins.ListNodesSample`

サーバーからノードのリストを取得する方法を示します。

com.sas.cas.sample.simple

これらのサンプルは、simple アクションセットで分析アクションのいくつかを実行する方法を示しています。

com.sas.cas.samples.simple.CrosstabSample
simple.crosstab アクションの実行方法を示します。

com.sas.cas.samples.simple.NumrowsSample
テーブルの行数を取得する方法を示します。

com.sas.cas.samples.simple.SummarySample
simple.summary アクションの実行方法を示します。

com.sas.cas.samples.table

これらのサンプルは、データを操作してテーブルを作業に追加し、インメモリテーブルに関する情報を取得するための基本を示しています。

com.sas.cas.samples.table.AddCaslibSample
caslib をサーバーに追加し、サーバーイベントリスナーを登録する方法を示します。

com.sas.cas.samples.table.AddCSVSample
ローカル CSV ファイルをサーバーに追加する方法を示します。

com.sas.cas.samples.table.AddCSVWithTransformSample
データトランスフォーマーを使用してローカル CSV ファイルをサーバーに追加し、さまざまな日付と時刻の形式を適切な CAS データ型に変換する方法を示します。

com.sas.cas.samples.table.AddTableSample
サーバーにテーブルを追加して行を追加する方法を示します。

com.sas.cas.samples.table.ColumnInfoSample
テーブルから列情報を取得する方法を示します。

com.sas.cas.samples.table.EmptyTableSample
空のテーブルをサーバーに追加する方法を示します。

com.sas.cas.samples.table.FetchByCursorSample
インメモリテーブルからデータをフェッチし、CASTable.next()を使用してカーソルで CASTable を処理する方法を示します。

com.sas.cas.samples.table.TableExistsSample
サーバーにテーブルが存在するかどうかを確認する方法を示します。

com.sas.cas.samples.table.TableInfoSample
サーバーからテーブル情報を取得する方法を示します。

システムプロパティ

システムプロパティを設定して、cas-client JAR ファイル内のクラスの動作に影響を与えることができます。たとえば、次のコマンドは、Kerberos による認証の試みをすべて防止します。

```
java -Dcom.sas.cas.kerberos.enabled=false YourApplication
```

次のリストでは、いくつかのシステムプロパティについて説明しています。

com.sas.cas.append.buffer.size

設定されている場合、このプロパティは、データをテーブルに追加するために使用される初期行バッファサイズ(バイト単位)を定義します。デフォルトは 131072 (128K) バイトです。

com.sas.cas.authinfo.enabled

設定されている場合、このプロパティは、他のユーザー資格情報が提供されていない場合に資格情報を .authinfo から読み取るかどうかを定義します。有効な値は true または false です。デフォルトは false です。

com.sas.cas.backup.retries

設定されている場合、このプロパティはバックアップコントローラーへの接続試行回数を指定します。デフォルト値は 20 です。

com.sas.cas.connection.timeout

設定されている場合、このプロパティは接続タイムアウトをミリ秒単位で指定します。デフォルト値は 0 (タイムアウトなし)です。

com.sas.cas.kerberos.enabled

設定されている場合、このプロパティは、他の資格情報が提供されていない場合に、クライアントが現在の環境から Kerberos チケットの取得を試みるかどうかを定義します。有効な値は true または false です。デフォルトは true です。

com.sas.cas.kerberos.realm.name

設定されている場合、このプロパティは Kerberos レalm名を指定します。

com.sas.cas.kerberos.service.name

設定されている場合、このプロパティは Kerberos サービス名を定義します。デフォルトは sascas です。

com.sas.cas.kerberos.spn

設定されている場合、このプロパティはサービスプリンシパル名をオーバーライドします。デフォルトでは、サービスプリンシパル名は、次のように、サービス名、ターゲットホスト名、およびオブションのレalm名で構成されます。

```
service-name /hostname@ realm-name
```

com.sas.cas.max.table.mem.size

設定されている場合、このプロパティは、テーブルをディスクにキャッシュする前の結果 CASTable の最大サイズ(バイト単位)を定義します。デフォルトは指定されていません。つまり、最大数はありません。

com.sas.cas.preserve.vchar.padding

設定されている場合、このプロパティは、CASTable VARCHAR 列が空白スペースによるパディングを保持するかどうかを定義します。デフォルト値は false です。false は、VARCHAR 列から先頭と末尾の空白文字を削除することを示します。

com.sas.cas.protobuf.size.limit

設定されている場合、このプロパティは protobuf メッセージのサイズ制限を定義します。デフォルトは 67108864 (64 MB) バイトです。

com.sas.cas.rest.debug

設定されている場合、このプロパティは REST クライアントから stdout への JSON 応答の出力を制御します。有効な値は true または false です。デフォルトは false です。

com.sas.cas.session.close

設定されている場合、このプロパティは、CASClient オブジェクトが閉じられたときにセッションを終了するかどうかを定義します。有効な値は true または false です。デフォルトは false です。

これらのシステムプロパティは、com.sas.cas.CASConstants のフィールドに対応しています。

ログ

CASClient ログの概要

SAS Cloud Analytic Services に接続してアクションを実行するために使用される CASClient クラスは、java.util.Logger ユーティリティを使用します。標準の java.util.Logger 構成または Java システムプロパティを使用して、Java アプリケーションからのログを有効にできます。

構成ファイルを使用したログ

java.util.Logger 構成ファイルのサンプルは次のとおりです。

例のコード 8 logging.properties ファイルのサンプル

```
handlers=java.util.logging.ConsoleHandler
com.sas.cas.level=FINE
java.util.logging.ConsoleHandler.level=ALL
```

アプリケーションの起動時に、java.util.logging.config.file システムプロパティでファイルへのパスを指定します。

```
-Djava.util.logging.config.file=/path/to/logging.properties
```

Java システムプロパティを使用したログ

ログを有効にするもう 1 つの方法は、アプリケーションの起動時にシステムプロパティとして有効にする部分的または完全なパッケージ名を指定することです。

```
-Dcom.sas.cas.debug=true
```

システムプロパティも指定することで、ログをファイルに書き込むことができます。

```
-Dcom.sas.cas.debug.file=/tmp/my.out
```

ログファイル形式

デフォルトのログ形式は、標準のログシステムプロパティ `java.util.logging.SimpleFormatter.format` を使用してオーバーライドできます。

```
-Djava.util.logging.SimpleFormatter.format="%1$tY-%1$tm-%1$td %1$tH:%1$tM:%1$tS.%1$tL %4$-6s %2$s %5$s%6$s%n"
```

形式文字列の構文については、`java.util.Formatter` クラスの Java ドキュメントを参照してください。Java 11 ドキュメントは次の URL から入手できます。

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Formatter.html>

CAS シェルの使用

シェルについて

`cas-client` JAR ファイルには、`com.sas.cas.Cash` クラスが含まれています。このクラスは、Lua 構文を使用してアクションをサブミットできるようにする SAS Cloud Analytic Services へのシェルのようなインターフェイスを提供します。

CAS シェルを使用すると、Java コードを記述して結果を処理する前に、アクションの結果を判別できます。

CAS シェルを実行するための要件は、Java プログラムを開発するための要件と同じです。[“要件” \(2 ページ\)](#)を参照してください。

サーバーへの接続

CAS シェルを使用するには、CAS で Java アプリケーションを開発および実行するための要件をすべて満たす必要があります。具体的には、cas-client JAR ファイルが CLASSPATH に含まれている必要があります。.authinfo ファイルまたは CASClient.properties ファイルを作成することは、インタラクティブに資格情報を提供するかわりになります。

例のコード 9 CAS シェルの開始

```
java -Dcom.sas.cas.authinfo.enabled=true com.sas.cas.Cash
```

アウトプット 4 CASH プロンプト



CAS シェルが開始されていても、まだ CAS サーバーに接続されておらず、CAS セッションも開始していないことに注意してください。シェルプロンプトで、使用するホスト名とポートを指定できます。

```
host=cloud.example.com:5570
```

.authinfo ファイルも CASClient.properties ファイルもない場合は、host=と同じように、username=および password=の値も指定します。ホストとポートを指定したら、次の例に示すようにアクションを実行できます。

```
s:session_sessionId{}
```

sessionId アクションをサブミットした後、指定されたサーバーへの接続が確立され、セッションが開始されるまで、短い一時停止があります。

アウトプット 5 sessionId アクションの結果

```
{
  Session:Tue Oct 10 14:27:16 2023="7332fc32-afc2-fd4c-be99-fa58ae36b655"
}
[Performance] Elapsed 0.056, CPU 0.131, Nodes 6
[Disposition] (OK)
```

比較のために、このドキュメントの最初の Java プログラムでは、com.sas.cas.actions.builtins.ServerStatusOptions クラスの使用法を示しました。Lua 構文での同等のアクションは、builtins_serverStatus アクションを実行することです。

例のコード 10 serverStatus アクションの実行

```
s:builtins_serverStatus{}
```

```
{
  About= {
    CAS="Cloud Analytic Services"
    Version="4.00"
    VersionLong="V.04.00M0P10092023"
    Copyright="Copyright © 2014-2023 SAS Institute Inc. All Rights Reserved."
    ServerTime="2023-10-10T18:28:39Z"
    System= {
      ...
    }
  }
}
```

次のステップ

- CAS シェルには help コマンドが含まれています。アクションを実行するのではなく、シェルコマンドから、多くの共通タスクを使用できます。

```
cash> help
!!          Executes the last command
!<n>       Executes the specified command number from history
caslib      Sets the caslib
caslibs     Shows the currently defined caslibs
charset     Sets the charset for reading action scripts
...
```

- CAS アクションを実行するための Lua 構文の詳細については、*SAS Viya プラットフォームプログラミング: CAS 用 Lua 入門ガイド*を参照してください。
- CAS セッションの Lua 構文規則は、's'という名前の変数で表されることを理解してください。対応する Java 変数は、CASClient クラスのインスタンスです。
- builtins_serverStatus などのアクションには、com.sas.cas.actions.builtins.ServerStatusOptions という名前の対応する Java クラスがあることを理解してください。

SASHDAT ファイルへの CSV ファイルのインポート

CSV から SASHDAT ユーティリティについて

cas-client JAR ファイルには、com.sas.cas.samples.csv.CSVToHdat クラスが含まれています。このクラスは、CSV ファイルを読み取り、SASHDAT ファイルを作成するためのユーティリティのように機能します。SASHDAT ファイルは、種類 Path または DNFS の caslib からサーバーで読み取ることができます。

コマンドラインインターフェイスを使用するための要件は、Java プログラムを開発するための要件と同じです。[“要件” \(2 ページ\)](#)を参照してください。

このユーティリティは、グラフィカルユーザーインターフェイスも提供します。グラフィカルユーザーインターフェイスは、Windows 上および X Window System サーバーにアクセスできる Linux ホストで使用できます。

コマンドラインの使用

最も簡単な使い方は、path と output の 2 つの必須引数を指定することです。

```
java com.sas.cas.samples.csv.CSVToHdat path="orsales.csv" output="orsales.sashdat"
```

```
Guessing variable info from orsales.csv      1
Found 8 variable(s)
Year (type=SAS, rType=NUMERIC, length=8)
Quarter (type=SAS, rType=CHAR, length=6)
Product_Line (type=SAS, rType=CHAR, length=15)
Product_Category (type=SAS, rType=CHAR, length=24)
Product_Group (type=SAS, rType=CHAR, length=24)
Quantity (type=SAS, rType=NUMERIC, length=8)
Profit (type=SAS, rType=NUMERIC, length=8)
Total_Retail_Price (type=SAS, rType=NUMERIC, length=8)
Creating orsales.sashdat
orsales.sashdat contains 0 bytes, 0 records, 0 large blocks, 0 extents      2
orsales.sashdat written in 00.079
Large block written with 92288 bytes, 912 records, 1 small block           3
```

- 1 デフォルトでは、ユーティリティは変数、変数名、およびデータ型の数を推定します。
- 2 ログ行は、出力ファイルのバイトやレコードなどの初期数を示します。新しいファイルの場合、ゼロが表示されます。ファイルに追加すると、値に SASHDAT ファイルのコンテンツが反映されてから、新しいレコードが追加されます。
- 3 この行は、912 レコードの大きなブロックが 1 つ書き込まれたことを示しています。複数の大きなブロック(デフォルトサイズは 16M)を必要とする大きなファイルでは、この行が大きなブロックごとに繰り返されます。

出力は、<http://support.sas.com/documentation/onlinedoc/viya/examples.htm> からダウンロードできる orsales.csv ファイルに対応しています。

例

例のコード 11 既存のファイルを上書きして新しいファイルを作成する

```
java com.sas.cas.samples.csv.CSVToHdat path=orsales.csv output=orsales.sashdat replace=true
```

または、replace 引数のみを指定することもできます。

```
java com.sas.cas.samples.csv.CSVToHdat path=orsales.csv output=orsales.sashdat replace
```

例のコード 12 レコードの追加

```
java com.sas.cas.samples.csv.CSVToHdat path=orsales.csv output=orsales.sashdat append
```

例のコード 13 特定の変数のみのインポート

```
java com.sas.cas.samples.csv.CSVToHdat path=orsales.csv output=orsales.sashdat \
  keepVars="year,product_line,profit" replace
```

ヒント 変数名のリストの各カンマの後にスペースを入れないでください。

例のコード 14 圧縮 SASHDAT ファイルの作成

```
java com.sas.cas.samples.csv.CSVToHdat path=orsales.csv \
  output=orsales.sashdat compress replace
```

レコードの追加に関する考慮事項

- 欠損値(後続の 2 つのデリミタ)を持つレコードは許容されます。
- 追加の変数を含むレコードは許容されません。コマンドラインから、名前を指定するか、keepVars または dropVars 引数を指定することで、インポートする変数を制御できます。[グラフィカルユーザーインターフェイス \(40 ページ\)](#)を使用すると、**変数を推定**ボタンをクリックした後も保持する変数を指定できます。

次のようなエラーメッセージは、固定幅の文字が使用され、新しいレコードの長さが長いことを示しています。

```
com.sas.cas.CASException: variable-name exceeds the record length of nn
```

この問題を回避するには、いくつか方法があります。

- 1 サイズが大きく変化する CHAR 変数の場合は、varcharVars 引数にその名前を指定して、VARCHAR データ型を使用できます。
- 2 CHAR 変数が大きく変化する場合は、"**vars="variable-1 type= data-type-1< , variable-2 type= data-type-2,...>" |vars="variable-1 as new-name-1 type= data-type-1"**" (39 ページ)引数で十分に大きなサイズを指定できます。
- 3 **変数を推定**ボタンをクリックした後、ユーザーインターフェイスを使用して長さを指定します。
- 4 最後に、defaultStringType=varchar を指定して、すべての文字変数に VARCHAR データ型を使用できます。

コマンドライン引数

コマンドライン引数は、?または-help 引数を使用してユーティリティから使用することもできます。

必須引数

path=path

入力ファイルへの相対パスまたは完全修飾パスを指定します。

`output=output`

出力の相対パスまたは完全修飾パスを指定します。 `.sashdat` ファイル名接尾辞を含めます。

オプション引数

`append=<true|false>`

`true` に設定すると、入力ファイルの行が出力ファイルに追加されます。引数を指定すると、`true` に設定されます。

`compress=<true|false>`

`true` に設定すると、出力 SASHDAT ファイルは圧縮を使用します。引数を指定すると、`true` に設定されます。

`compressionLevel=<1-9>`

1 は高速の圧縮を示し、9 は最適な圧縮を示します。

デフォルト 6

`defaultStringLength=integer`

文字変数のデフォルト長を指定します。

デフォルト 1

`defaultStringType=<varchar|fixed>`

文字変数のデフォルトデータ型を指定します。

デフォルト `fixed`

`delimiter=character`

デリミタを指定します。タブ区切りファイルには `\t` を指定できます。

デフォルト `,`

`dropVars="variable-1 <,variable-2, ...>"`

出力ファイルから除外する入力ファイルの変数のカンマ区切りリストを指定します。

`firstrowheader=<true|false>`

`true` に設定すると、変数名は入力ファイルの最初の行から読み取られます。

デフォルト `true`

`fixedVars="variable-1 <,variable-2, ...>"`

固定幅の CHAR データ型を割り当てる入力ファイルの変数のカンマ区切りリストを指定します。

`guess=integer`

変数のデータ型と長さを推定するために読み取る行数を指定します。すべての行を読み取るには、-1 を指定します。

デフォルト 1000

`handleEscapedCommas=<true|false>`

`true` に設定すると、引用符の外側にバックスラッシュを付けてエスケープされたカンマは、フィールド区切り文字とは見なされません。たとえば、`handleEscapedCommas` が `true` に設定されている場合、**Last\First** は 2 つの値ではなく 1 つの値として処理されます。

デフォルト false

`keepVars="variable-1 <,variable-2, ...>"`

出力ファイルに含める入力ファイルの変数のカンマ区切りリストを指定します。デフォルトでは、すべての変数が含まれています。

`locale=locale`

数値を解析するときに使用するロケールを指定します。

`maxLargeBlockSize=integer`

大きなブロックの最大サイズをバイト単位で指定します。大きな値の場合は、K、M または G を接頭辞として付けます。

デフォルト 16M

`maxRows=integer`

入力ファイルからインポートする最大行数を指定します。

デフォルト -1 (すべての行)

`maxSmallBlockSize=integer`

小さなブロックの最大サイズをバイト単位で指定します。大きな値の場合は、K または M を接頭辞として付けます。

デフォルト 256K

`removeDups<=true|false>`

true に設定すると、値が重複する変数(VARCHAR または VARBINARY)が削除されます。引数を指定すると、true に設定されます。

デフォルト false

`replace<=true|false>`

true に設定すると、出力ファイルが置き換えられます。引数を指定すると、true に設定されます。

デフォルト false

`ui<=true|false>`

true に設定すると、グラフィカルユーザーインターフェイスが表示されます。引数を指定すると、true に設定されます。

`varcharVars="variable-1 <,variable-2, ...>"`

VARCHAR データ型で使用する変数のカンマ区切りリストを指定します。

`vars="variable-1 type= data-type-1<, variable-2 type= data-type-2,...>" | vars="variable-1 as new-name-1 type= data-type-1"`

変更された変数名とデータ型のカンマ区切りリストを指定します。

- スペースを含む変数名の場合は、名前を単一引用符で囲みます。
- データ型は次のとおりです。
 - NUMERIC
 - VARCHAR
 - FIXED(length)

- INT32
- INT64
- DATE(*MM/dd/yy*)
- TIME(*HH:mm:ss*)
- DATETIME (*MM/dd/yyyy HH:mm:ss*)

DATE、TIME および DATETIME パターンは、[java.text.SimpleDateFormat](#) クラスの仕様に従います。このパターンは SAS 入力形式として機能し、値は SAS 日付、時刻および日時値に変換されます。

例 `vars=""Product Line' as product_line type=FIXED(128)"`
`vars=""Gross profit' as profit type=NUMERIC,year=DATE('yyyy')"`

入力形式および SAS 日付に関する詳細については、[SAS 出力形式と入力形式 リファレンス](#)および [SAS プログラマガイド: エッセンシャル](#)の>About SAS Date, Time, and Datetime Values"を参照してください。

グラフィカルユーザーインターフェイスの使用

ユーザーインターフェイスの起動

次のコマンドを使用して、グラフィカルユーザーインターフェイスを起動できます。

```
java com.sas.cas.samples.csv.CSVToHdat ui
```

- 他のコマンドライン引数も指定できます。たとえば、ui 引数の前または後に `defaultStringType=varchar` を指定できます。
- ホームディレクトリの `.CSVToHdatDialog.properties` ファイルは、デフォルト値の読み取りに使用されます。
- ユーザーインターフェイスを起動しても、X サーバーがないか、X11 転送を構成していない場合、次のエラーメッセージが表示されます。

No X11 DISPLAY variable was set, but this program performed an operation which requires it.

入力ファイルと出力ファイルの指定

ユーザーインターフェイスを起動した後で、[参照](#)ボタンを使用して、入力ファイルと出力ファイルを指定します。出力ファイルには `.sashdat` ファイル名接尾辞を含めます。

CSV To sashdat

Options **HCS**

csv

Input Path:

Delimiter:

☒ First Row Contains Header

Guess Rows:

Default String Type:

Default String Length:

Maximum Rows:

Data Locale:

sashdat

Output Path:

☐ Replace

☐ Append

☐ Compress

☐ Remove Duplicate Values

Compression Level:

Large Block Size:

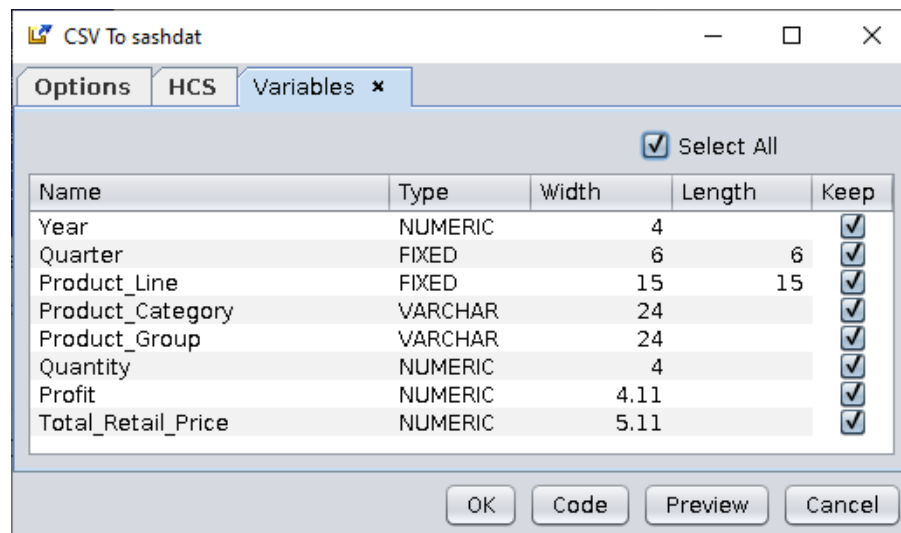
Small Block Size:

すべてのフィールドには、対応するコマンドラインオプションがあります。[“コマンドライン引数”](#)を参照してください。

変数の選択と変更

ユーザーインターフェイスを起動して入力ファイルを指定した後で、**変数を推定**をクリックして変数を選択および変更できます。

図 1 変数タブ



- **すべて選択**および**保持**チェックボックスを使用して、インポートする変数を制御します。
- 変数に別の名前を指定するには、**名前**列の変数名をクリックします。
- データ型を変更するには、**型**列の値をクリックします。FIXED は、固定幅の CHAR データ型を指定します。デフォルトでは、**長さ**列には、その列で検出された最も長い CHAR 値が表示されます。セルをクリックすると、より大きな値を指定できます。詳細については [vars](#) コマンドラインオプションを参照してください。

VARCHAR データ型は、大きく変化する文字データに使用します。VARCHAR には 16 バイトのオーバーヘッドがあり、州の略称など、変数の長さが短い場合、固定幅 CHAR は使用するストレージが少なくなります。
- インポートの追加設定を指定したり**行を推定**の数を増やすには、**オプション**タブをクリックし、**変数を推定**をもう一度クリックします。**変数を推定**をもう一度クリックすると、変数に加えた変更が上書きされることに注意してください。

ファイルのインポート

インポート設定を確認します。

プレビュー

このボタンをクリックすると、最初の 100 行のインポートがプレビューされます。

コード

このボタンをクリックすると、ユーザーインターフェイスでの選択に対応するコマンドラインオプションが表示されます。

ファイルをインポートする準備ができたなら、**OK** をクリックします。インポートが完了すると、次のようなウィンドウが表示されます。

