Red Hat Process Automation Manager 7.6

PMML モデルでのデシジョンサービスの作成

Last Updated: 2020-02-24

Red Hat Process Automation Manager 7.6 PMML モデルでのデシジョンサービスの作成

Red Hat Customer Content Services
brms-docs@redhat.com

法律上の通知

概要

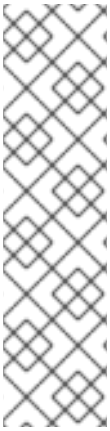
本書では、Red Hat Process Automation Manager 7.6 のデシジョンサービスに Predictive Model Markup Language (PMML) モデルを実装する方法を説明します。

目次

前書き

ビジネスルール開発者は、Predictive Model Markup Language (PMML) を使用して統計またはデータマイニングモデルを定義し、Red Hat Process Automation Manager のデシジョンサービスと統合できます。Red Hat Process Automation Manager には、回帰、スコアカード、ツリー、マイニングモデル向けの PMML 4.2.1 のコンシューマー適合サポートが含まれます。Red Hat Process Automation Manager には、PMML モデルエディターが同梱されていませんが、XML または PMML 固有のオーサリングツールを使用して、PMML モデルを作成してから Red Hat Process Automation Manager プロジェクトに統合できます。

PMML に関する詳細は、DMG PMML specification を参照してください。



注記

Decision Model and Notation (DMN) モデルを使用して独自のデシジョンサービスを設計し、DMN サービスの一部として PMML モデルを追加することもできます。Red Hat Process Automation Manager 7.6 における DMN サポートの詳細は、以下を参照してください。

- 『Getting started with decision services』(DMN デシジョンサービスの例を使用したステップバイステップのチュートリアル)
- 『Designing a decision service using DMN models』(Red Hat Process Automation Manager における DMN のサポートおよび機能の概要)

第1章 PREDICTIVE MODEL MARKUP LANGUAGE (PMML)

Predictive Model Markup Language (PMML) は、統計およびデータマイニングモデルの定義用に、Data Mining Group (DMG) が確立した XML ベースの標準です。PMML モデルは、ビジネスアナリストや開発者が一元的に PMML ベースのアセットやサービスを設計、分析、実装するために、PMML 準拠のプラットフォーム間や組織全体で共有することができます。

PMML の背景およびアプリケーションに関する詳細は、DMG PMML specification を参照してください。

1.1. PMML 適合レベル

PMML 仕様は、PMML モデルが確実に作成され、統合されるように、プロデューサーとコンシューマーの適合レベルを定義します。各適合レベルの正式な定義については、DMG PMML conformance のページを参照してください。

以下の一覧では、PMML 適合レベルをまとめています。

プロデューサーの適合

ツールまたはアプリケーションは、少なくとも 1 種類のモデルに対して有効な PMML ドキュメントが生成されている場合に、プロデューサーの適合があるとされます。PMML のプロデューサー適合要件を満たすことで、モデル定義のドキュメントが構文的に正しく、このドキュメントで定義したモデルインスタンスが、モデル仕様に定義されている意味論の基準と整合性を保てるようにします。

コンシューマーの適合

ツールまたはアプリケーションは、少なくとも 1 種類のモデルに対して有効な PMML ドキュメントが生成されている場合に、コンシューマーの適合があるとされます。コンシューマー適合要件を満たすことで、プロデューサー適合にあわせて作成された PMML モデルが定義どおりに統合され、使用できるようにします。たとえば、アプリケーションが回帰モデルタイプに適合しているコンシューマーである場合には、有効な PMML ドキュメントで、別の適合プロデューサーにより作成されたこのタイプのモデルを定義していれば、アプリケーション内でどちらでも使用できます。

Red Hat Process Automation Manager には、以下の PMML 4.2.1 モデルタイプに対するコンシューマー適合サポートが含まれます。

- 回帰モデル
- スコアカードモデル
- ツリーモデル
- マイニングモデル (サブタイプとして **modelChain**、**selectAll** および **selectFirst** が含まれる)

Red Hat Process Automation Manager でサポートされていないモデルを含め、すべての PMML モデルタイプの一覧については、DMG PMML specification を参照してください。

第2章 PMML モデルの例

PMML は、XML スキーマ を定義しており、このスキーマを使用することで、PMML モデルが 異なる PMML 準拠のプラットフォーム間で使用できるようになります。PMML 仕様では、プロデューサーとコンシューマー適合を満たしていることが前提で、複数のソフトウェアプラットフォームが同じファイルを使用してオーサリング、テスト、実稼働環境での実行を可能にします。

以下は、PMM の回帰、スコアカード、ツリー、マイニングモデルの例です。これらの例では、Red Hat Process Automation Manager のデシジョンサービスと統合可能なモデルでサポートされているタイプを紹介します。

PMML の詳細例については、DMG PMML Sample Files ページを参照してください。

PMML 回帰モデルの例

```
<PMML version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-1/pmml-4-2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.dmg.org/PMML-4_2">
  <Header copyright="JBoss"/>
  <DataDictionary numberOfFields="5">
    <DataField dataType="double" name="fld1" optype="continuous"/>
    <DataField dataType="double" name="fld2" optype="continuous"/>
    <DataField dataType="string" name="fld3" optype="categorical">
      <Value value="x"/>
      <Value value="y"/>
    </DataField>
    <DataField dataType="double" name="fld4" optype="continuous"/>
    <DataField dataType="double" name="fld5" optype="continuous"/>
  </DataDictionary>
  <RegressionModel algorithmName="linearRegression" functionName="regression"
modelName="LinReg" normalizationMethod="logit" targetFieldName="fld4">
    <MiningSchema>
      <MiningField name="fld1"/>
      <MiningField name="fld2"/>
      <MiningField name="fld3"/>
      <MiningField name="fld4" usageType="predicted"/>
      <MiningField name="fld5" usageType="target"/>
    </MiningSchema>
    <RegressionTable intercept="0.5">
      <NumericPredictor coefficient="5" exponent="2" name="fld1"/>
      <NumericPredictor coefficient="2" exponent="1" name="fld2"/>
      <CategoricalPredictor coefficient="-3" name="fld3" value="x"/>
      <CategoricalPredictor coefficient="3" name="fld3" value="y"/>
      <PredictorTerm coefficient="0.4">
        <FieldRef field="fld1"/>
        <FieldRef field="fld2"/>
      </PredictorTerm>
    </RegressionTable>
  </RegressionModel>
</PMML>
```

PMML スコアカードモデルの例

```
<PMML version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-1/pmml-4-2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xmlns="http://www.dmg.org/PMML-4_2">
  <Header copyright="JBoss"/>
  <DataDictionary numberOfFields="4">
    <DataField name="param1" optype="continuous" dataType="double"/>
    <DataField name="param2" optype="continuous" dataType="double"/>
    <DataField name="overallScore" optype="continuous" dataType="double" />
    <DataField name="finalscore" optype="continuous" dataType="double" />
  </DataDictionary>
  <Scorecard modelName="ScorecardCompoundPredicate" useReasonCodes="true"
isScorable="true" functionName="regression"    baselineScore="15" initialScore="0.8"
reasonCodeAlgorithm="pointsAbove">
    <MiningSchema>
      <MiningField name="param1" usageType="active" invalidValueTreatment="asMissing">
      </MiningField>
      <MiningField name="param2" usageType="active" invalidValueTreatment="asMissing">
      </MiningField>
      <MiningField name="overallScore" usageType="target"/>
      <MiningField name="finalscore" usageType="predicted"/>
    </MiningSchema>
    <Characteristics>
      <Characteristic name="ch1" baselineScore="50" reasonCode="reasonCh1">
        <Attribute partialScore="20">
          <SimplePredicate field="param1" operator="lessThan" value="20"/>
        </Attribute>
        <Attribute partialScore="100">
          <CompoundPredicate booleanOperator="and">
            <SimplePredicate field="param1" operator="greaterOrEqual" value="20"/>
            <SimplePredicate field="param2" operator="lessOrEqual" value="25"/>
          </CompoundPredicate>
        </Attribute>
        <Attribute partialScore="200">
          <CompoundPredicate booleanOperator="and">
            <SimplePredicate field="param1" operator="greaterOrEqual" value="20"/>
            <SimplePredicate field="param2" operator="greaterThan" value="25"/>
          </CompoundPredicate>
        </Attribute>
      </Characteristic>
      <Characteristic name="ch2" reasonCode="reasonCh2">
        <Attribute partialScore="10">
          <CompoundPredicate booleanOperator="or">
            <SimplePredicate field="param2" operator="lessOrEqual" value="-5"/>
            <SimplePredicate field="param2" operator="greaterOrEqual" value="50"/>
          </CompoundPredicate>
        </Attribute>
        <Attribute partialScore="20">
          <CompoundPredicate booleanOperator="and">
            <SimplePredicate field="param2" operator="greaterThan" value="-5"/>
            <SimplePredicate field="param2" operator="lessThan" value="50"/>
          </CompoundPredicate>
        </Attribute>
      </Characteristic>
    </Characteristics>
  </Scorecard>
</PMML>
```

PMML ツリーモデルの例

```
<PMML version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-1/pmml-4-2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.dmg.org/PMML-4_2">
 <Header copyright="JBOSS"/>
 <DataDictionary numberOfFields="5">
  <DataField dataType="double" name="fld1" optype="continuous"/>
  <DataField dataType="double" name="fld2" optype="continuous"/>
  <DataField dataType="string" name="fld3" optype="categorical">
   <Value value="true"/>
   <Value value="false"/>
  </DataField>
  <DataField dataType="string" name="fld4" optype="categorical">
   <Value value="optA"/>
   <Value value="optB"/>
   <Value value="optC"/>
  </DataField>
  <DataField dataType="string" name="fld5" optype="categorical">
   <Value value="tgtX"/>
   <Value value="tgtY"/>
   <Value value="tgtZ"/>
  </DataField>
 </DataDictionary>
 <TreeModel functionName="classification" modelName="TreeTest">
  <MiningSchema>
   <MiningField name="fld1"/>
   <MiningField name="fld2"/>
   <MiningField name="fld3"/>
   <MiningField name="fld4"/>
   <MiningField name="fld5" usageType="predicted"/>
  </MiningSchema>
  <Node score="tgtX">
   <True/>
   <Node score="tgtX">
    <SimplePredicate field="fld4" operator="equal" value="optA"/>
    <Node score="tgtX">
     <CompoundPredicate booleanOperator="surrogate">
      <SimplePredicate field="fld1" operator="lessThan" value="30.0"/>
      <SimplePredicate field="fld2" operator="greaterThan" value="20.0"/>
     </CompoundPredicate>
     <Node score="tgtX">
      <SimplePredicate field="fld2" operator="lessThan" value="40.0"/>
     </Node>
     <Node score="tgtZ">
      <SimplePredicate field="fld2" operator="greaterOrEqual" value="10.0"/>
     </Node>
    </Node>
    <Node score="tgtZ">
     <CompoundPredicate booleanOperator="or">
      <SimplePredicate field="fld1" operator="greaterOrEqual" value="60.0"/>
      <SimplePredicate field="fld1" operator="lessOrEqual" value="70.0"/>
     </CompoundPredicate>
     <Node score="tgtZ">
      <SimpleSetPredicate booleanOperator="isNotIn" field="fld4">
       <Array type="string">optA optB</Array>
      </SimpleSetPredicate>
     </Node>
```

```
      </Node>
     </Node>
     <Node score="tgtY">
      <CompoundPredicate booleanOperator="or">
       <SimplePredicate field="fld4" operator="equal" value="optA"/>
       <SimplePredicate field="fld4" operator="equal" value="optC"/>
      </CompoundPredicate>
      <Node score="tgtY">
       <CompoundPredicate booleanOperator="and">
        <SimplePredicate field="fld1" operator="greaterThan" value="10.0"/>
        <SimplePredicate field="fld1" operator="lessThan" value="50.0"/>
        <SimplePredicate field="fld4" operator="equal" value="optA"/>
        <SimplePredicate field="fld2" operator="lessThan" value="100.0"/>
        <SimplePredicate field="fld3" operator="equal" value="false"/>
       </CompoundPredicate>
      </Node>
      <Node score="tgtZ">
       <CompoundPredicate booleanOperator="and">
        <SimplePredicate field="fld4" operator="equal" value="optC"/>
        <SimplePredicate field="fld2" operator="lessThan" value="30.0"/>
       </CompoundPredicate>
      </Node>
     </Node>
    </Node>
  </TreeModel>
</PMML>
```

PMML マイニングモデルの例 (modelChain)

```
<PMML version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-
1/pmml-4-2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.dmg.org/PMML-4_2">
  <Header>
    <Application name="Drools-PMML" version="7.0.0-SNAPSHOT" />
  </Header>
  <DataDictionary numberOfFields="7">
    <DataField name="age" optype="continuous" dataType="double" />
    <DataField name="occupation" optype="categorical" dataType="string">
      <Value value="SKYDIVER" />
      <Value value="ASTRONAUT" />
      <Value value="PROGRAMMER" />
      <Value value="TEACHER" />
      <Value value="INSTRUCTOR" />
    </DataField>
    <DataField name="residenceState" optype="categorical" dataType="string">
      <Value value="AP" />
      <Value value="KN" />
      <Value value="TN" />
    </DataField>
    <DataField name="validLicense" optype="categorical" dataType="boolean" />
    <DataField name="overallScore" optype="continuous" dataType="double" />
    <DataField name="grade" optype="categorical" dataType="string">
      <Value value="A" />
      <Value value="B" />
      <Value value="C" />
      <Value value="D" />
```

```
      <Value value="F" />
    </DataField>
    <DataField name="qualificationLevel" optype="categorical" dataType="string">
      <Value value="Unqualified" />
      <Value value="Barely" />
      <Value value="Well" />
      <Value value="Over" />
    </DataField>
  </DataDictionary>
  <MiningModel modelName="SampleModelChainMine" functionName="classification">
    <MiningSchema>
      <MiningField name="age" />
      <MiningField name="occupation" />
      <MiningField name="residenceState" />
      <MiningField name="validLicense" />
      <MiningField name="overallScore" />
      <MiningField name="qualificationLevel" usageType="target"/>
    </MiningSchema>
    <Segmentation multipleModelMethod="modelChain">
      <Segment id="1">
        <True />
        <Scorecard modelName="Sample Score 1" useReasonCodes="true" isScorable="true"
functionName="regression"              baselineScore="0.0" initialScore="0.345">
          <MiningSchema>
            <MiningField name="age" usageType="active" invalidValueTreatment="asMissing" />
            <MiningField name="occupation" usageType="active" invalidValueTreatment="asMissing" />
            <MiningField name="residenceState" usageType="active" invalidValueTreatment="asMissing"
/>
            <MiningField name="validLicense" usageType="active" invalidValueTreatment="asMissing" />
            <MiningField name="overallScore" usageType="predicted" />
          </MiningSchema>
          <Output>
            <OutputField name="calculatedScore" displayName="Final Score" dataType="double"
feature="predictedValue"                    targetField="overallScore" />
          </Output>
          <Characteristics>
            <Characteristic name="AgeScore" baselineScore="0.0" reasonCode="ABZ">
              <Extension name="cellRef" value="$B$8" />
              <Attribute partialScore="10.0">
                <Extension name="cellRef" value="$C$10" />
                <SimplePredicate field="age" operator="lessOrEqual" value="5" />
              </Attribute>
              <Attribute partialScore="30.0" reasonCode="CX1">
                <Extension name="cellRef" value="$C$11" />
                <CompoundPredicate booleanOperator="and">
                  <SimplePredicate field="age" operator="greaterOrEqual" value="5" />
                  <SimplePredicate field="age" operator="lessThan" value="12" />
                </CompoundPredicate>
              </Attribute>
              <Attribute partialScore="40.0" reasonCode="CX2">
                <Extension name="cellRef" value="$C$12" />
                <CompoundPredicate booleanOperator="and">
                  <SimplePredicate field="age" operator="greaterOrEqual" value="13" />
                  <SimplePredicate field="age" operator="lessThan" value="44" />
                </CompoundPredicate>
              </Attribute>
```

```
          <Attribute partialScore="25.0">
            <Extension name="cellRef" value="$C$13" />
            <SimplePredicate field="age" operator="greaterOrEqual" value="45" />
          </Attribute>
        </Characteristic>
        <Characteristic name="OccupationScore" baselineScore="0.0">
          <Extension name="cellRef" value="$B$16" />
          <Attribute partialScore="-10.0" reasonCode="CX2">
            <Extension name="description" value="skydiving is a risky occupation" />
            <Extension name="cellRef" value="$C$18" />
            <SimpleSetPredicate field="occupation" booleanOperator="isIn">
              <Array n="2" type="string">SKYDIVER ASTRONAUT</Array>
            </SimpleSetPredicate>
          </Attribute>
          <Attribute partialScore="10.0">
            <Extension name="cellRef" value="$C$19" />
            <SimpleSetPredicate field="occupation" booleanOperator="isIn">
              <Array n="2" type="string">TEACHER INSTRUCTOR</Array>
            </SimpleSetPredicate>
          </Attribute>
          <Attribute partialScore="5.0">
            <Extension name="cellRef" value="$C$20" />
            <SimplePredicate field="occupation" operator="equal" value="PROGRAMMER" />
          </Attribute>
        </Characteristic>
        <Characteristic name="ResidenceStateScore" baselineScore="0.0" reasonCode="RES">
          <Extension name="cellRef" value="$B$22" />
          <Attribute partialScore="-10.0">
            <Extension name="cellRef" value="$C$24" />
            <SimplePredicate field="residenceState" operator="equal" value="AP" />
          </Attribute>
          <Attribute partialScore="10.0">
            <Extension name="cellRef" value="$C$25" />
            <SimplePredicate field="residenceState" operator="equal" value="KN" />
          </Attribute>
          <Attribute partialScore="5.0">
            <Extension name="cellRef" value="$C$26" />
            <SimplePredicate field="residenceState" operator="equal" value="TN" />
          </Attribute>
        </Characteristic>
        <Characteristic name="ValidLicenseScore" baselineScore="0.0">
          <Extension name="cellRef" value="$B$28" />
          <Attribute partialScore="1.0" reasonCode="LX00">
            <Extension name="cellRef" value="$C$30" />
            <SimplePredicate field="validLicense" operator="equal" value="true" />
          </Attribute>
          <Attribute partialScore="-1.0" reasonCode="LX00">
            <Extension name="cellRef" value="$C$31" />
            <SimplePredicate field="validLicense" operator="equal" value="false" />
          </Attribute>
        </Characteristic>
      </Characteristics>
    </Scorecard>
  </Segment>
  <Segment id="2">
    <True />
```

```
      <TreeModel modelName="SampleTree" functionName="classification"
missingValueStrategy="lastPrediction" noTrueChildStrategy="returnLastPrediction">
        <MiningSchema>
          <MiningField name="age" usageType="active" />
          <MiningField name="validLicense" usageType="active" />
          <MiningField name="calculatedScore" usageType="active" />
          <MiningField name="qualificationLevel" usageType="predicted" />
        </MiningSchema>
        <Output>
          <OutputField name="qualification" displayName="Qualification Level" dataType="string"
feature="predictedValue"                    targetField="qualificationLevel" />
        </Output>
        <Node score="Well" id="1">
          <True/>
          <Node score="Barely" id="2">
            <CompoundPredicate booleanOperator="and">
              <SimplePredicate field="age" operator="greaterOrEqual" value="16" />
              <SimplePredicate field="validLicense" operator="equal" value="true" />
            </CompoundPredicate>
            <Node score="Barely" id="3">
              <SimplePredicate field="calculatedScore" operator="lessOrEqual" value="50.0" />
            </Node>
            <Node score="Well" id="4">
              <CompoundPredicate booleanOperator="and">
                <SimplePredicate field="calculatedScore" operator="greaterThan" value="50.0" />
                <SimplePredicate field="calculatedScore" operator="lessOrEqual" value="60.0" />
              </CompoundPredicate>
            </Node>
            <Node score="Over" id="5">
              <SimplePredicate field="calculatedScore" operator="greaterThan" value="60.0" />
            </Node>
          </Node>
          <Node score="Unqualified" id="6">
            <CompoundPredicate booleanOperator="surrogate">
              <SimplePredicate field="age" operator="lessThan" value="16" />
              <SimplePredicate field="calculatedScore" operator="lessOrEqual" value="40.0" />
              <True />
            </CompoundPredicate>
          </Node>
        </Node>
      </TreeModel>
    </Segment>
  </Segmentation>
 </MiningModel>
</PMML>
```

第3章 RED HAT PROCESS AUTOMATION MANAGER における PMML サポート

Red Hat Process Automation Manager には、以下の PMML 4.2.1 モデルタイプに対するコンシューマー適合サポートが含まれます。

- 回帰モデル
- スコアカードモデル
- ツリーモデル
- マイニングモデル (サブタイプとして **modelChain**、**selectAll** および **selectFirst** が含まれる)

Red Hat Process Automation Manager でサポートされていないモデルを含め、すべての PMML モデルタイプの一覧については、DMG PMML specification を参照してください。

Red Hat Process Automation Manager には、PMML モデルエディターが同梱されていますが、XML または PMML 固有のオーサリングツールを使用して、Red Hat Process Automation Manager のデシジョンサービスで PMML モデルを作成することができます。Business Central (**Menu → Design → Projects → Import Asset**) でプロジェクトに PMML ファイルをインポートするか、Business Central なしにナレッジ JAR (KJAR) ファイルの一部として PMML ファイルをパッケージ化できます。

Red Hat Process Automation Manager でプロジェクトに PMML ファイルを追加する場合には複数のアセットが生成されます。PMML モデルのタイプごとに、異なるアセットのセットが生成されますが、すべての PMML モデルタイプは少なくとも以下のアセットセットを生成します。

- PMML モデルに関連する全ルールを含む DRL ファイル
- 少なくとも以下の Java クラス 2 つが含まれていること
  - モデルタイプのデフォルトオブエジェクトタイプとして使用するデータクラス
  - データソースとルール実行の管理に使用する **RuleUnit** クラス

PMML ファイルに root モデルとして **MiningModel** が含まれている場合には、これらのファイルごとに複数のインスタンスが生成されます。

プロジェクトのパッケージ化およびデプロイメントの方法に、PMML ファイルなどのアセットを追加する方法の詳細は、『Packaging and deploying a Red Hat Process Automation Manager project』を参照してください

3.1. RED HAT PROCESS AUTOMATION MANAGER の PMML 命名規則

以下は、生成された PMML パッケージ、クラス、ルールの命名規則です。

- PMML モデルファイルにパッケージ名が指定されていない場合には、**"org.kie.pmml.pmml_4_2"+modelName** の形式で、生成されたルールのモデル名に、デフォルトのパッケージ名 **org.kie.pmml.pmml_4_2** がプリフィックスとして追加されます。
- 生成された **RuleUnit** Java クラスのパッケージ名は、生成されたルールのパッケージ名と同じです。
- 生成された **RuleUnit** Java クラス名は、**RuleUnit** にモデル名を追加して **modelName+"RuleUnit"** の形式で指定します。

- PMML モデルにはそれぞれ、最低でも生成されたデータクラスが 1 つ含まれます。これらのクラスのパッケージ名は **org.kie.pmml.pmml_4_2.model** です。
- 生成されたデータクラス名は、モデルタイプにより決まり、プリフィックスとしてモデル名を指定します。
  - 回帰モデル: **modelName+"RegressionData"** という名前のデータクラス 1 つ
  - スコアカードモデル: **modelName+"ScoreCardData"** という名前のデータクラス 1 つ
  - ツリーモデル: 1 つ目は **modelName+"TreeNode"** という名前で、2 つ目は **modelName+"TreeToken"** という名前の合計 2 つのデータクラス
  - マイニングモデル: **modelName+"MiningModelData"** という名前のデータクラス 1 つ

注記

マイニングモデルは、各セグメントに含まれるルールとクラスすべても生成します。

3.2. RED HAT PROCESS AUTOMATION MANAGER における PMML 拡張

PMML 仕様は、PMML モデルのコンテンツを拡張する **Extension** 要素をサポートします。モデルの主要要素の最初と最後の子として、PMML モデル定義のほぼすべてのレベルで拡張を使用し、最大限に柔軟性をもたせることができます。PMML 拡張の詳細は DMG PMML Extension Mechanism を参照してください。

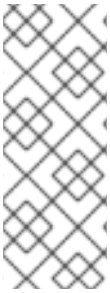
Red Hat Process Automation Manager は、PMML 統合を最適化するために以下の追加の PMML 拡張をサポートします。

- **modelPackage**: 生成されたルールと Java クラスのパッケージ名を指定します。PMML モデルファイルの **Header** セクションにこの拡張を追加します。
- **adapter**: ルールの入出力データを含めるのに使用するコンストラクトタイプ (**bean** または **trait**) を指定します。PMML モデルファイルの **MiningSchema** または **Output** セクション (または両方) にこの拡張を挿入します。
- **externalClass**: **adapter** 拡張と連携して使用し、**MiningField** または **OutputField** を定義します。この拡張には、**MiningField** または **OutputField** 要素名と一致する要素名のクラスが含まれます。

第4章 PMML モデルの実行

Business Central を使用して Red Hat Process Automation Manager に PMML ファイルをインポートする (**Menu → Design → Projects → Import Asset**) か、Business Central を使用しないでプロジェクトのナレッジ JAR (KJAR) ファイルの一部として DMN ファイルをパッケージ化できます。Red Hat Process Automation Manager プロジェクトに PMML ファイルに実装した後に、Java アプリケーションに直接 PMML 呼び出しを埋め込み、PMML ベースのデシジョンサービスを実行できます。

プロジェクトのパッケージ化およびデプロイメントの方法に PMML アセットを追加する方法の詳細は、『Packaging and deploying a Red Hat Process Automation Manager project』を参照してください



注記

Business Central の Decision Model and Notation (DMN) サービスの一部として、PMML モデルを追加することもできます。DMN ファイル内に PMML モデルを追加すると、その PMML モデルを DMN デシジョンノードまたはビジネスナレッジモデルノードのボックス関数式として呼び出すことができます。DMN サービスへの PMML モデルの追加に関する詳細は、『Designing a decision service using DMN models』を参照してください。

4.1. PMML 呼び出しの JAVA アプリケーションへの直接組み込み

KIE コンテナーは、呼び出しプログラムにナレッジアセットを直接組み込む場合や、KJAR 用 Maven 依存関係を使用して物理的にプルする場合は、ローカルとみなされます。コードのバージョンと、PMML 定義のバージョンとの間に密接な関係がある場合は、通常、ナレッジアセットをプロジェクトに直接組み込みます。意思決定への変更は、アプリケーションをアップデートして再デプロイしないと有効になりません。このアプローチに対する利点は、適切なオペレーションがランタイムへの外部の依存関係に依存していないことですが、ロックされた環境の場合は制限になる可能性があります。

Maven の依存関係を使用すると、システムプロパティーを使用して、アップデートを定期的にスキャンして自動的にアップデートするなど、特定バージョンの意思決定が動的に変更するため、柔軟性が高まります。これにより、外部の依存関係がサービスのデプロイ時間に影響を及ぼしますが、意思決定はローカルで実行されるため、ランタイム時に利用可能な外部サービスに対する信頼が低くなります。

前提条件

- 実行する PMML モデルを含む KJAR が作成されている。プロジェクトのパッケージ化に関する詳細は、『Packaging and deploying a Red Hat Process Automation Manager project』を参照してください。

手順

1. クライアントアプリケーションで、Java プロジェクトの関連クラスパスに以下の依存関係を追加します。

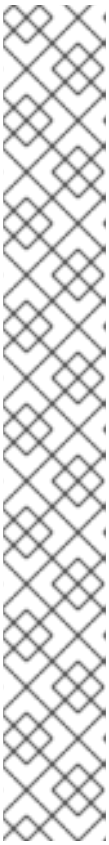
```
<!-- Required for the PMML compiler -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>kie-pmml</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required for the KIE public API -->
```

```
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-api</artifactId>
  <version>${rhpam.version}</version>
</dependencies>

<!-- Required if not using classpath KIE container -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

<version> は、プロジェクトで現在使用する Red Hat Process Automation Manager の Maven アーティファクトバージョンです (例: 7.30.0.Final-redhat-00002)。



注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation 部品表 (BOM) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Process Decision Manager と Red Hat Process Automation Manager の両方に適用します。BOM ファイルを追加すると、指定の Maven リポジトリーからの一時的な依存関係の内、正しいバージョンが、このプロジェクトに追加されます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.6.0.redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) についての詳細情報は、What is the mapping between RHPAM product and maven library version? を参照してください。

2. **classpath** または **ReleaseId** から KIE コンテナーを作成します。

```
KieServices kieServices = KieServices.Factory.get();

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "my-kjar", "1.0.0" );
KieContainer kieContainer = kieServices.newKieContainer( releaseId );
```

または、以下のオプションを使用します。

```
KieServices kieServices = KieServices.Factory.get();

KieContainer kieContainer = kieServices.getKieClasspathContainer();
```

3. **PMMLRequestData** クラスのインスタンスを作成し、PMML モデルをデータセットに適用します。

```
public class PMMLRequestData {
    private String correlationId; (1)
    private String modelName; (2)
    private String source; (3)
    private List<ParameterInfo<?>> requestParams; (4)
    ...
}
```

(1) 特定の要求または結果に関連のあるデータを特定します。

(2) 要求データに適用する必要のあるモデル名

(3) 要求を生成したセグメントを特定するために、内部で生成された **PMMLRequestData** オブジェクトが使用します。

(4) 入力データポイントを送信するデフォルトのメカニズム

4. **PMML4Result** クラスのインスタンスを作成します。このクラスでは、入力データに PMML ベースルールを適用した結果の出力情報を保持します。

```
public class PMML4Result {
    private String correlationId;
    private String segmentationId; (1)
    private String segmentId; (2)
    private int segmentIndex; (3)
    private String resultCode; (4)
    private Map<String, Object> resultVariables; (5)
    ...
}
```

(1) モデルタイプが **MiningModel** の場合に使用します。**segmentationId** は、複数のセグメントを区別化するために使用します。

(2) **segmentationId** と併用して、結果を生成したセグメントを特定します。

(3) セグメントの順番を維持するのに使用します。

(4) モデルが正常に適用されたかどうかを判断するのに使用します。**OK** は成功を示します。

(5) 結果の値として代入される変数とそれに関連する値の名前が含まれます。

通常の getter メソッドに加え、**PMML4Result** クラスも、結果となる変数の値を直接取得する以下の方法をサポートします。

```
public <T> Optional<T> getResultValue(String objName, String objField, Class<T> clazz, Object...params)

public Object getResultValue(String objName, String objField, Object...params)
```

5. **ParameterInfo** クラスのインスタンスを作成します。このクラスは、**PMMLRequestData** クラスの一部として使用する、基本的なデータタイプオブジェクトのラッパーとして機能します。

```
public class ParameterInfo<T> { (1)
   private String correlationId;
   private String name; (2)
   private String capitalizedName;
   private Class<T> type; (3)
   private T value; (4)
   ...
}
```

(1) 多数の異なるタイプを処理するパラメーター化されたクラス

(2) モデルの入力として想定される変数の名前

(3) 変数の実際のタイプとなるクラス

(4) 変数の実際の値

6. 先ほど作成した対象の PMML クラスインスタンスをもとに PMML モデルを実行します。

```
public void executeModel(KieBase kbase,
                         Map<String,Object> variables,
                         String modelName,
                         String correlationId,
                         String modelPkgName) {
   RuleUnitExecutor executor = RuleUnitExecutor.create().bind(kbase);
   PMMLRequestData request = new PMMLRequestData(correlationId, modelName);
   PMML4Result resultHolder = new PMML4Result(correlationId);
   variables.entrySet().forEach( es -> {
      request.addRequestParam(es.getKey(), es.getValue());
   });

   DataSource<PMMLRequestData> requestData = executor.newDataSource("request");
   DataSource<PMML4Result> resultData = executor.newDataSource("results");
   DataSource<PMMLData> internalData = executor.newDataSource("pmmlData");

   requestData.insert(request);
   resultData.insert(resultHolder);

   List<String> possiblePackageNames = calculatePossiblePackageNames(modelName,
                                                                     modelPkgName);
   Class<? extends RuleUnit> ruleUnitClass = getStartingRuleUnit("RuleUnitIndicator",
                                                                 (InternalKnowledgeBase)kbase,
                                                                 possiblePackageNames);

   if (ruleUnitClass != null) {
      executor.run(ruleUnitClass);
      if ( "OK".equals(resultHolder.getResultCode()) ) {
        // extract result variables here
      }
   }
}
```

```
protected Class<? extends RuleUnit> getStartingRuleUnit(String startingRule,
InternalKnowledgeBase ikb, List<String> possiblePackages) {
    RuleUnitRegistry unitRegistry = ikb.getRuleUnitRegistry();
    Map<String,InternalKnowledgePackage> pkgs = ikb.getPackagesMap();
    RuleImpl ruleImpl = null;
    for (String pkgName: possiblePackages) {
      if (pkgs.containsKey(pkgName)) {
          InternalKnowledgePackage pkg = pkgs.get(pkgName);
          ruleImpl = pkg.getRule(startingRule);
          if (ruleImpl != null) {
              RuleUnitDescr descr = unitRegistry.getRuleUnitFor(ruleImpl).orElse(null);
              if (descr != null) {
                  return descr.getRuleUnitClass();
              }
          }
      }
    }
    return null;
}

protected List<String> calculatePossiblePackageNames(String modelId,
String...knownPackageNames) {
    List<String> packageNames = new ArrayList<>();
    String javaModelId = modelId.replaceAll("\\s","");
    if (knownPackageNames != null && knownPackageNames.length > 0) {
        for (String knownPkgName: knownPackageNames) {
            packageNames.add(knownPkgName + "." + javaModelId);
        }
    }
    String basePkgName = PMML4UnitImpl.DEFAULT_ROOT_PACKAGE+"."+javaModelId;
    packageNames.add(basePkgName);
    return packageNames;
}
```

ルールは **RuleUnitExecutor** クラスにより実行されます。**RuleUnitExecutor** クラスは KIE セッションを作成し、必要な **DataSource** オブジェクトをこれらのセッションに追加してから、**run()** メソッドへのパラメーターとして渡される **RuleUnit** をもとに、ルールを実行します。**calculatePossiblePackageNames** と **getStartingRuleUnit** メソッドは、**run()** メソッドへのパラメーターとして渡される **RuleUnit** クラスの完全修飾名を決定します。

PMML モデル実行をスムーズに行うため、Red Hat Process Automation Manager でサポートされている **PMML4ExecutionHelper** クラスも使用できます。PMML ヘルパークラスに関する詳細は、「PMML 実行ヘルパークラス」を参照してください。

4.1.1. PMML 実行ヘルパークラス

Red Hat Process Automation Manager には、PMML モデル実行に必要な **PMMLRequestData** クラスの作成や **RuleUnitExecutor** クラスを使用したルールの実行をサポートする **PMML4ExecutionHelper** クラスがあります。

以下では、比較の目的で **PMML4ExecutionHelper** クラスのある場合とない場合の PMML モデル実行の例を紹介しています。

PMML4ExecutionHelper の使用なしの PMML モデル実行例

```
public void executeModel(KieBase kbase,
                         Map<String,Object> variables,
                         String modelName,
                         String correlationId,
                         String modelPkgName) {
    RuleUnitExecutor executor = RuleUnitExecutor.create().bind(kbase);
    PMMLRequestData request = new PMMLRequestData(correlationId, modelName);
    PMML4Result resultHolder = new PMML4Result(correlationId);
    variables.entrySet().forEach( es -> {
        request.addRequestParam(es.getKey(), es.getValue());
    });

    DataSource<PMMLRequestData> requestData = executor.newDataSource("request");
    DataSource<PMML4Result> resultData = executor.newDataSource("results");
    DataSource<PMMLData> internalData = executor.newDataSource("pmmlData");

    requestData.insert(request);
    resultData.insert(resultHolder);

    List<String> possiblePackageNames = calculatePossiblePackageNames(modelName,
                                                    modelPkgName);
    Class<? extends RuleUnit> ruleUnitClass = getStartingRuleUnit("RuleUnitIndicator",
                                                (InternalKnowledgeBase)kbase,
                                                possiblePackageNames);

    if (ruleUnitClass != null) {
        executor.run(ruleUnitClass);
        if ( "OK".equals(resultHolder.getResultCode()) ) {
          // extract result variables here
        }
    }
}

protected Class<? extends RuleUnit> getStartingRuleUnit(String startingRule,
InternalKnowledgeBase ikb, List<String> possiblePackages) {
    RuleUnitRegistry unitRegistry = ikb.getRuleUnitRegistry();
    Map<String,InternalKnowledgePackage> pkgs = ikb.getPackagesMap();
    RuleImpl ruleImpl = null;
    for (String pkgName: possiblePackages) {
      if (pkgs.containsKey(pkgName)) {
          InternalKnowledgePackage pkg = pkgs.get(pkgName);
          ruleImpl = pkg.getRule(startingRule);
          if (ruleImpl != null) {
              RuleUnitDescr descr = unitRegistry.getRuleUnitFor(ruleImpl).orElse(null);
              if (descr != null) {
                  return descr.getRuleUnitClass();
              }
          }
      }
    }
    return null;
}

protected List<String> calculatePossiblePackageNames(String modelId,
String...knownPackageNames) {
    List<String> packageNames = new ArrayList<>();
```

```
    String javaModelId = modelId.replaceAll("\\s","");
    if (knownPackageNames != null && knownPackageNames.length > 0) {
        for (String knownPkgName: knownPackageNames) {
            packageNames.add(knownPkgName + "." + javaModelId);
        }
    }
    String basePkgName = PMML4UnitImpl.DEFAULT_ROOT_PACKAGE+"."+javaModelId;
    packageNames.add(basePkgName);
    return packageNames;
}
```

PMML4ExecutionHelper を使用した PMML モデル実行例

```
public void executeModel(KieBase kbase,
                         Map<String,Object> variables,
                         String modelName,
                         String modelPkgName,
                         String correlationId) {
  PMML4ExecutionHelper helper = PMML4ExecutionHelperFactory.getExecutionHelper(modelName,
kbase);
  helper.addPossiblePackageName(modelPkgName);

  PMMLRequestData request = new PMMLRequestData(correlationId, modelName);
  variables.entrySet().forEach(entry -> {
    request.addRequestParam(entry.getKey(), entry.getValue);
  });

  PMML4Result resultHolder = helper.submitRequest(request);
  if ("OK".equals(resultHolder.getResultCode)) {
    // extract result variables here
  }
}
```

PMML4ExecutionHelper を使用する場合は、一般的な PMML モデル実行で行うように、考えられる名前または **RuleUnit** クラスを指定する必要はありません。

PMML4ExecutionHelper クラスを構築するには、**PMML4ExecutionHelperFactory** クラスを使用して、**PMML4ExecutionHelper** の取得方法を決定します。

以下は、**PMML4ExecutionHelper** を構築するのに利用可能な **PMML4ExecutionHelperFactory** クラスメソッドです。

KIE ベースにある PMML アセット向けの PMML4ExecutionHelperFactory メソッド

PMML アセットがすでにコンパイルされており、既存の KIE ベースで使用されている場合には、これらのメソッドを使用します。

```
public static PMML4ExecutionHelper getExecutionHelper(String modelName, KieBase kbase)

public static PMML4ExecutionHelper getExecutionHelper(String modelName, KieBase kbase,
boolean includeMiningDataSources)
```

プロジェクトクラスパスにある PMML アセット向けの PMML4ExecutionHelperFactory メソッド

PMML アセットがプロジェクトクラスパスにある場合にこれらのメソッドを使用します。**classPath** の引数は、PMML ファイルのプロジェクトクラスパスの場所を指します。

```
public static PMML4ExecutionHelper getExecutionHelper(String modelName,  String classPath,
KieBaseConfiguration kieBaseConf)

public static PMML4ExecutionHelper getExecutionHelper(String modelName,String classPath,
KieBaseConfiguration kieBaseConf, boolean includeMiningDataSources)
```

バイトアレイ形式の PMML アセット向けの PMML4ExecutionHelperFactory メソッド

PMML アセットがバイトアレイ形式の場合にこれらのメソッドを使用します。

```
public static PMML4ExecutionHelper getExecutionHelper(String modelName, byte[] content,
KieBaseConfiguration kieBaseConf)

public static PMML4ExecutionHelper getExecutionHelper(String modelName, byte[] content,
KieBaseConfiguration kieBaseConf, boolean includeMiningDataSources)
```

Resource にある PMML アセット向けの PMML4ExecutionHelperFactory メソッド

PMML アセットが **org.kie.api.io.Resource** オブジェクトの形式の場合に、これらのメソッドを使用します。

```
public static PMML4ExecutionHelper getExecutionHelper(String modelName, Resource resource,
KieBaseConfiguration kieBaseConf)

public static PMML4ExecutionHelper getExecutionHelper(String modelName, Resource resource,
KieBaseConfiguration kieBaseConf, boolean includeMiningDataSources)
```

注記

クラスパス、バイトアレイ、リソース **PMML4ExecutionHelperFactory** メソッドは、生成されたルールおよび Java クラスの KIE コンテナーを作成します。このコンテナーは、**RuleUnitExecutor** が使用する KIE ベースのソースとして使用します。ただし、このコンテナーには永続性はありません。PMML アセットの **PMML4ExecutionHelperFactory** メソッドが KIE ベースにすでにあるには、この方法では KIE コンテナーは作成されません。

4.2. PROCESS SERVER を使用した PMML モデルの実行

ApplyPmmlModelCommand コマンドを設定済みの Process Server に送信して Process Server にデプロイした PMML モデルを実行できます。このコマンドを使用する場合は、**PMMLRequestData** オブジェクトが Process Server に送信され、**PMML4Result** の結果オブジェクトを返信として受信します。設定済みの Java クラスからの Process Server REST API または、REST クライアントから直接 Process Server に PMML 要求を送信できます。

前提条件

- Process Server がインストールされ、設定されている (**kie-server** ロールが割り当てられているユーザーの既知のユーザー名と認証情報を含む)。インストールオプションは、『Planning a Red Hat Process Automation Manager installation』を参照してください。
- KIE コンテナーは、PMML モデルを含む KJAR の形式で Process Server にデプロイしている。プロジェクトのパッケージ化に関する詳細は、『Packaging and deploying a Red Hat Process Automation Manager project』を参照してください。

- PMML モデルを含む KIE コンテナーのコンテナー ID がある。

手順

1. クライアントアプリケーションで、Java プロジェクトの関連クラスパスに以下の依存関係を追加します。

```
<!-- Required for the PMML compiler -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>kie-pmml</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required for the KIE public API -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-api</artifactId>
  <version>${rhpam.version}</version>
</dependencies>

<!-- Required for the Process Server Java client API -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required if not using classpath KIE container -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

<version> は、プロジェクトで現在使用する Red Hat Process Automation Manager の Maven アーティファクトバージョンです (例: 7.30.0.Final-redhat-00002)。

注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation 部品表 (BOM) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Process Decision Manager と Red Hat Process Automation Manager の両方に適用します。BOM ファイルを追加すると、指定の Maven リポジトリーからの一時的な依存関係の内、正しいバージョンが、このプロジェクトに追加されます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.6.0.redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) についての詳細情報は、What is the mapping between RHPAM product and maven library version? を参照してください。

2. **classpath** または **ReleaseId** から KIE コンテナーを作成します。

```
KieServices kieServices = KieServices.Factory.get();

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "my-kjar", "1.0.0" );
KieContainer kieContainer = kieServices.newKieContainer( releaseId );
```

または、以下のオプションを使用します。

```
KieServices kieServices = KieServices.Factory.get();

KieContainer kieContainer = kieServices.getKieClasspathContainer();
```

3. Process Server への要求を送信して、応答を受信するクラスを作成します。

```
public class ApplyScorecardModel {
  private static final ReleaseId releaseId =
        new ReleaseId("org.acme","my-kjar","1.0.0");
  private static final String containerId = "SampleModelContainer";
  private static KieCommands commandFactory;
  private static ClassLoader kjarClassLoader; (1)
  private RuleServicesClient serviceClient; (2)

  // Attributes specific to your class instance
  private String rankedFirstCode;
  private Double score;

  // Initialization of non-final static attributes
  static {
    commandFactory = KieServices.Factory.get().getCommands();
```

```
    // Specifications for kjarClassLoader, if used
    KieMavenRepository kmp = KieMavenRepository.getMavenRepository();
    File artifactFile = kmp.resolveArtifact(releaseId).getFile();
    if (artifactFile != null) {
      URL urls[] = new URL[1];
      try {
        urls[0] = artifactFile.toURI().toURL();
        classLoader = new KieURLClassLoader(urls,PMML4Result.class.getClassLoader());
      } catch (MalformedURLException e) {
        logger.error("Error getting classLoader for "+containerId);
        logger.error(e.getMessage());
      }
    } else {
      logger.warn("Did not find the artifact file for "+releaseId.toString());
    }
  }

  public ApplyScorecardModel(KieServicesConfiguration kieConfig) {
    KieServicesClient clientFactory = KieServicesFactory.newKieServicesClient(kieConfig);
    serviceClient = clientFactory.getServicesClient(RuleServicesClient.class);
  }
  ...
  // Getters and setters
  ...

  // Method for executing the PMML model on KIE Server
  public void applyModel(String occupation, int age) {
    PMMLRequestData input = new PMMLRequestData("1234","SampleModelName"); (3)
    input.addRequestParam(new ParameterInfo("1234","occupation",String.class,occupation));
    input.addRequestParam(new ParameterInfo("1234","age",Integer.class,age));

    CommandFactoryServiceImpl cf = (CommandFactoryServiceImpl)commandFactory;
    ApplyPmmlModelCommand command = (ApplyPmmlModelCommand)
cf.newApplyPmmlModel(request); (4)

    ServiceResponse<ExecutionResults> results =
        ruleClient.executeCommandsWithResults(CONTAINER_ID, command); (5)

    if (results != null) { (6)
      PMML4Result resultHolder = (PMML4Result)results.getResult().getValue("results");
      if (resultHolder != null && "OK".equals(resultHolder.getResultCode())) {
        this.score = resultHolder.getResultValue("ScoreCard","score",Double.class).get();
        Map<String,Object> rankingMap =
            (Map<String,Object>)resultHolder.getResultValue("ScoreCard","ranking");
        if (rankingMap != null && !rankingMap.isEmpty()) {
          this.rankedFirstCode = rankingMap.keySet().iterator().next();
        }
      }
    }
  }
}
```

(1) クライアントプロジェクトの依存関係に KJAR を追加しなかった場合には、クラスローダーを定義します。

2. Process Server REST API のアクセス認証情報など、構成設定で定義したサービスクライアントを特定します。
3. **PMMLRequestData** オブジェクトを初期化します。
4. **ApplyPmmlModelCommand** のインスタンスを作成します。
5. サービスクライアントを使用してコマンドを送信します。
6. 実行済みの PMML モデルの結果を取得します。

4. クラスインスタンスを実行して、PMML 呼び出し要求を Process Server に送信します。または JMS および REST インターフェースを使用して、**ApplyPmmlModelCommand** コマンドを Process Server に送信できます。REST 要求については、**ApplyPmmlModelCommand** コマンドを、JSON または JAXB、XStream 要求形式で、**http://SERVER:PORT/kie-server/services/rest/server/containers/instances/{containerId}** への **POST** 要求として使用できます。

POST エンドポイントの例

```
http://localhost:8080/kie-server/services/rest/server/containers/instances/SampleModelContainer
```

JSON 要求ボディーの例

```
{
  "commands": [ {
     "apply-pmml-model-command": {
       "outIdentifier": null,
       "packageName": null,
       "hasMining": false,
       "requestData": {
         "correlationId": "123",
         "modelName": "SimpleScorecard",
         "source": null,
         "requestParams": [
           {
             "correlationId": "123",
             "name": "param1",
             "type": "java.lang.Double",
             "value": "10.0"
           },
           {
             "correlationId": "123",
             "name": "param2",
             "type": "java.lang.Double",
             "value": "15.0"
           }
         ]
       }
     }
   }
  ]
}
```

エンドポイントよびボディーを含む curl 要求の例

```
curl -X POST "http://localhost:8080/kie-server/services/rest/server/containers/instances/SampleModelContainer" -H "accept: application/json" -H "content-type: application/json" -d "{ \"commands\": [ { \"apply-pmml-model-command\": { \"outIdentifier\": null, \"packageName\": null, \"hasMining\": false, \"requestData\": { \"correlationId\": \"123\", \"modelName\": \"SimpleScorecard\", \"source\": null, \"requestParams\": [ { \"correlationId\": \"123\", \"name\": \"param1\", \"type\": \"java.lang.Double\", \"value\": \"10.0\" }, { \"correlationId\": \"123\", \"name\": \"param2\", \"type\": \"java.lang.Double\", \"value\": \"15.0\" } ] } } } ]}"
```

JSON の応答例


```
{
  "results" : [ {
    "value" : {"org.kie.api.pmml.DoubleFieldOutput":{
  "value" : 40.8,
  "correlationId" : "123",
  "segmentationId" : null,
  "segmentId" : null,
  "name" : "OverallScore",
  "displayValue" : "OverallScore",
  "weight" : 1.0
}},
    "key" : "OverallScore"
  }, {
    "value" : {"org.kie.api.pmml.PMML4Result":{
  "resultVariables" : {
    "OverallScore" : {
      "value" : 40.8,
      "correlationId" : "123",
      "segmentationId" : null,
      "segmentId" : null,
      "name" : "OverallScore",
      "displayValue" : "OverallScore",
      "weight" : 1.0
    },
    "ScoreCard" : {
      "modelName" : "SimpleScorecard",
      "score" : 40.8,
      "holder" : {
        "modelName" : "SimpleScorecard",
        "correlationId" : "123",
        "voverallScore" : null,
        "moverallScore" : true,
        "vparam1" : 10.0,
        "mparam1" : false,
        "vparam2" : 15.0,
        "mparam2" : false
      },
      "enableRC" : true,
      "pointsBelow" : true,
      "ranking" : {
        "reasonCh1" : 5.0,
        "reasonCh2" : -6.0
```

```
      }
     }
   },
   "correlationId" : "123",
   "segmentationId" : null,
   "segmentId" : null,
   "segmentIndex" : 0,
   "resultCode" : "OK",
   "resultObjectName" : null
}},
    "key" : "results"
  } ],
  "facts" : [ ]
}
```

第5章 関連資料

- PMML 仕様
- 『Packaging and deploying a Red Hat Process Automation Manager project』
- 『Interacting with Red Hat Process Automation Manager using KIE APIs』

付録A バージョン情報

本書の最終更新日: 2019 年 10 月 31 日 (木)