



データベースの設定

概要

この章では、データベースと連動させるための Cisco Service Control Management Suite (SCMS) Collection Manager (CM) の設定方法、CM の機能を拡張させるためのデータベース インフラストラクチャの使用方法について説明します。

- 「[クイック スタート ガイド](#)」 (P.6-1)
- 「[Velocity Template Language を使用した SQL コードの生成](#)」 (P.6-2)
- 「[データベース コンフィギュレーション ファイル](#)」 (P.6-2)
- 「[作業サンプル](#)」 (P.6-6)
- 「[テストおよびデバッグ](#)」 (P.6-8)
- 「[スクリプトでの JDBC フレームワークの使用方法](#)」 (P.6-9)
- 「[Oracle のスケーラビリティに関するヒント](#)」 (P.6-11)

クイック スタート ガイド

CM で外部データベースを使用するには、データベースが配置されている IP アドレスやポートなどの基本接続パラメータを変更する必要があります。これらのパラメータを変更するには、`dbconf.sh` スクリプトを使用します。「[データベースの設定](#)」 (P.4-5) を参照してください。

Velocity Template Language を使用した SQL コードの生成

JDBC アダプタ フレームワークは Velocity Template Language (VTL) で記述されたマクロを使用して、データベース サーバに渡されるすべての SQL コードを生成します。次のセクションでは、生成プロセスを制御するために使用されるコンフィギュレーション ファイルについて説明します。

VTL (Apache Jakarta Project の一部) に関する詳細については、<http://jakarta.apache.org/velocity/vtl-reference-guide.html> を参照してください。

表 6-1 では VTL 構造について説明しています。

表 6-1 VTL 構造の概要

ディレクティブ	構文例	目的
#foreach	#foreach (\$item in \$collection) item is \$item #end	コレクション、配列、またはマップに対して処理を反復します。
#if ... #else ... #elseif	#if (\$order.total == 0) No charge #end	条件文。
#parse	#parse("header.vm")	指定されたテンプレートをロードおよび解析して、生成された出力に取り込みます。
#macro	#macro(currency \$amount) \${formatter.currency(\$amount)} #end	新しいディレクティブおよび必要なすべてのパラメータを定義します。結果は、後でこのテンプレートで使用する場合に解釈されます。
#include	#include("disclaimer.txt")	指定されたファイルをそのまま、生成された出力に含めます。
#set	#set (\$customer = \${order.customer})	コンテキスト オブジェクトに値を割り当てます。コンテキスト オブジェクトが存在しない場合は、値が追加されます。コンテキスト オブジェクトが存在する場合は、値が置き換えられます。
#stop	#if (\$debug) #stop #end	テンプレート処理を停止します。

データベース コンフィギュレーション ファイル

データベース アクセス フレームワークを初期化する場合、データベース アクセス フレームワークによって最初に検索されるファイルは **main.vm** です。このファイルには、定義または必要なすべてのデータベース SQL 定義に対するポインタが格納されています。このファイルの検索に使用する場所は、CM で使用する dbpack によって異なります。dbpack は特定のデータベース インストールに関連するコンフィギュレーション ファイルの集合です。(このコンフィギュレーション ファイルに関連づけられた) アダプタが dbpack を選択します。たとえば、次に示す **jdbcadapter.conf** ファイルのコードフラグメントは、Oracle dbpack と連携するようにアダプタを設定します。

```
db_template_dir = dbpacks/oracle/9204e/  
db_template_file = main.vm
```



(注) ディレクトリの場所は、メインの CM 設定ディレクトリ (通常は **~scmscm/cm/config**) に対して相対的に解釈されます

設定をモジュール化するには、一般的に、**main.vm** ファイルがその他のファイルをポイントします。ただし、このポイントが厳密に必要なわけではありません。このファイルには任意の定義を格納し、スクリプト内などで、後で使用できます。一部の定義は JDBC アダプタの動作に使用されるため、必須です。これらの定義は表 6-2 でリスト表示されています。

表 6-2 必須 VM 定義

オブジェクト名	必須の定義
\$table.sql.dropTable	これらの設定は、テーブルごとに、指定の処理に対する SQL の生成方法を制御します。
\$table.sql.createTable	
\$table.sql.createIndexes	
\$table.sql.insert	
\$table.sql.metaDataQuery	
\$dbinfo.driverjarfile	JDBC ドライバの場所およびクラス名。
\$dbinfo.driver	
\$dbinfo.cmdSeparator	複数の SQL 文を区切るために使用されるパターン。
\$dbinfo.url	データベースに接続するための URL、およびさまざまな接続プロパティ。
\$dbinfo.connOptions	

VTL 解析コンテキストに含まれる CM 設定を表すオブジェクトの一部は、テンプレートで使用できるようになっています。これらのオブジェクトについては、次のセクションで説明します。

- 「コンテキスト オブジェクト」 (P.6-3)
- 「アプリケーションの設定」 (P.6-5)

コンテキスト オブジェクト

VM テンプレートをロードして、CM コンポーネント (TA または JDBC アダプタ、スクリプトなど) で解析する前に、次の Java オブジェクトを使用して解析コンテキストを初期化します。

- **tables** オブジェクト
- **dbinfo** オブジェクト
- **tools** オブジェクト

tables オブジェクト

tables オブジェクトは、データベースに格納する必要がある RDR の構造、データベース テーブルの構造とその格納場所、および CM で使用できるその他のデータベース テーブルの構造など、アプリケーションに関連するデータベース設定について説明します。このオブジェクトは配列であり、各行は CM が使用するデータベース テーブルの 1 つを表しています。各テーブルの行には、次の情報が含まれる場合があります（すべての項目がすべてのテーブルに存在するわけではありません）。

- 論理名
- 物理名
- このテーブルに関連付けられた RDR タグ
- それぞれ次の属性を持つ、このテーブル内のフィールドまたはカラムのリスト
 - フィールド ID
 - フィールド名
 - フィールド固有のタイプ
 - 自由形式のフィールド オプション
- それぞれ次の属性を持つ、このテーブルのインデックス リスト
 - インデックス名
 - インデックス付きカラム名
 - 自由形式のインデックス オプション

tables オブジェクトの内容は、テンプレートをロードするときに調べたり、操作したりできます。

tables オブジェクトは、アプリケーション固有の XML コンフィギュレーション ファイルを使用して初期化されます。「[アプリケーションの設定](#)」(P.6-5) を参照してください。

dbinfo オブジェクト

dbinfo オブジェクトは、データベース接続を開くときに使用されるパラメータ、使用される SID やスキーマなど、データベース固有の設定について説明します。このオブジェクトには、データベース固有の設定オプションが保持されます。保持される情報は、次のとおりです。

- このデータベースのドライバとして使用される JDBC クラス名
- ドライバが格納された JAR ファイルの名前
- JDBC URL として表されたデータベースの場所
- 認証データ（ユーザおよびパスワード）などの自由形式の JDBC 接続オプション

tools オブジェクト

tools オブジェクトは、テンプレートを開発する場合や、コンテキスト データ構造を操作する場合に役立つ複数のユーティリティ メソッドを含むコンテナです。

オブジェクトのメソッドを **\$tools.method(arg1, ..., argN)** として呼び出すことができます。ここでは、**method** はメソッド名です。

含まれるメソッドは表 6-3 にリスト表示されています。

表 6-3 tools オブジェクトのメソッド

メソッド名と引数	機能
<code>getTableByName (allTables, name)</code>	論理名が name に対応するデータベース テーブル オブジェクトを検出します。
<code>getTableByDbTabName (allTables, name)</code>	物理名が name に対応するデータベース テーブル オブジェクトを検出します。
<code>assignParams (sql, list_of_args)</code>	sql 文字列内の疑問符文字を、文字列で表される list_of_args パラメータ内の連続要素で置き換えます。このメソッドは、ベースとして JDBC Prepared Statement 文字列を使用する SQL 挿入文を作成するテンプレートで作業している場合に役立ちます。
<code>collapseWhitespace ()</code>	複数の連続スペース文字からなるすべてのインスタンスを 1 つのスペースに変換して、先頭および末尾のスペースを削除します。このメソッドは、少なくとも 1 つの新規行と他のスペース文字が含まれている SQL が必要なデータベースで便利です (Sybase と Oracle はこれが不要です)。

これらのツールの使用例については、「スクリプトでの JDBC フレームワークの使用方法」(P.6-9) を参照してください。

アプリケーションの設定

アプリケーションに関連するすべての設定は、次の項目が格納された特定のファイル (**tables.xml**) 内で行います。

- アプリケーションの名前およびバージョン
- 各データベース テーブルの名前とプロパティ、特に、データベース テーブルに格納されるアプリケーション RDR の構造
- 各データベース テーブル関連：
 - テーブルおよび RDR フィールドの名前および固有のタイプ
 - テーブル インデックスの名前およびプロパティ

この情報は、主にコンテキストを解析するテンプレートに **tables** オブジェクトを読み込む場合に使用されます。「**tables オブジェクト**」(P.6-4) を参照してください。

作業サンプル

main.vm ファイルには、モジュール化をサポートするために他の VM ファイルへの参照が含まれています（「データベース コンフィギュレーション ファイル」(P.6-2) を参照）。名前が事前に決められている **VM_global_library.vm** ファイルを除き、これらの他のファイル名は任意です。定義が必要なマクロをこのファイルに格納して、マクロが正しいタイミングでロードされるようにする必要があります。この特別なファイルの詳細については、『*Velocity User Guide*』を参照してください。

次のサンプルは、Oracle セットアップの **main.vm** のコンテンツを示したものです。

```
#parse ('dbinfo.vm')

#foreach ($table in $tables)
  #set ($table.sql.dropTable = "#parse ('drop_table.vm')")
  #set ($table.sql.createTable = "#parse ('create_table.vm')")
  #set ($table.sql.createIndexes = "#parse ('create_indexes.vm')")
  #set ($table.sql.insert = "#parse ('insert.vm')")
  #set ($table.sql.metadataQuery = "#parse ('metadata.vm')")
#end
```

このサンプルでは、必須のデータベースおよび SQL 定義（表 6-2 を参照）が別々のファイルに移動されていて、**#parse** ディレクティブを使用してロードおよび解析されます。

次のセクションでは、Oracle dbpack 内の各ファイルの内容をリスト表示します。定義の一部では、**VM_global_library.vm** ファイルに定義されたマクロを使用します。このファイルには、すべてのテンプレートで使用されるすべてのマクロ定義を格納する必要があります。

- 「マクロの定義」(P.6-6)
- 「dbinfo の設定」(P.6-7)
- 「SQL の定義」(P.6-7)

マクロの定義

次に、固有のタイプと SQL タイプ間のマッピングの定義例、およびリスト内の連続する要素間にカンマを挿入する **optcomma** マクロなどのユーティリティ マクロの定義サンプルを示します。

```
#macro (optcomma)#if ($velocityCount >1),#end#end
#macro (sqltype $field)
#set ($maxLength = 2000)
#if ($field.type == "INT8") integer
#elseif ($field.type == "INT16") integer
#elseif ($field.type == "INT32") integer
#elseif ($field.type == "UINT8") integer
#elseif ($field.type == "UINT16") integer
#elseif ($field.type == "UINT32") integer
#elseif ($field.type == "REAL") real
#elseif ($field.type == "BOOLEAN") char(1)
#elseif ($field.type == "STRING") varchar2(#{if($field.size <= $maxLength)$field.size
#else $maxLength #end})
#elseif ($field.type == "TEXT") long
#elseif ($field.type == "TIMESTAMP") date
#end
#end
```

dbinfo の設定

次のコード サンプルでは、URL および接続オプション（認証用）が唯一の必須フィールドです。

コードの空白行は、読みやすくし、また後で設定変更をしやすくするために、コードを個別のフィールドに分離するものです。

```
#set ($dbinfo.driver = "oracle.jdbc.OracleDriver")
#set ($dbinfo.driverjarfile = "ojdbc14.jar")
#set ($dbinfo.options.host = "localhost")
#set ($dbinfo.options.port = "1521")
#set ($dbinfo.options.user = "pqb_admin")
#set ($dbinfo.options.password = "pqb_admin")
#set ($dbinfo.options.sid = "apricot")
#set ($dbinfo.url =
"jdbc:oracle:thin:@$dbinfo.options.host:$dbinfo.options.port:$dbinfo.options.sid")
#set ($dbinfo.connOptions.user = $dbinfo.options.user)
#set ($dbinfo.connOptions.password = $dbinfo.options.password)
## the vendor-specific piece of SQL that will return the current
## date and time:
#set ($dbinfo.options.getdate = "sysdate")
```

SQL の定義

次のセクションでは、SQL 定義について説明します。

- 「drop table のコード」 (P.6-7)
- 「create table のコード」 (P.6-7)
- 「create indexes のコード」 (P.6-8)
- 「insert のコード」 (P.6-8)
- 「メタデータ クエリーのコード」 (P.6-8)

drop table のコード

次のサンプル コードは、通常の SQL 構文を使用してテーブルを削除します。

```
drop table $table.dbtablename
```

create table のコード

次のサンプル コードは、通常の SQL 構文を使用してテーブルを作成します。テーブル作成用の特殊なディレクティブが必要となる任意のカスタマイズ済みデータベース設定は、この定義を使用して実装できます。たとえば、何らかの一意的なテーブルスペース内にテーブルを作成したり、テーブルをパーティション化するように、この定義を変更することができます。

```
create table $table.dbtablename (
#foreach ($field in $table.fields)
  #optcomma()$field.name #sqltype($field)
  #if ("${!field.options.notNull}" == "true")
    not null
  #end
#end)
```

create indexes のコード

次のコードは、通常の SQL 構文を使用してインデックスを作成します。インデックス作成用の特殊なディレクティブが必要となるカスタマイズ済みデータベース設定は、この定義を使用して実装できません。たとえば、何らかの一意的なテーブルスペース内にインデックスを作成するように、この定義を変更することができます。

```
#foreach ($index in $table.indexes)
  create index $index.name on $table.dbtablename ($index.columns)
#end
```

insert のコード

次のコードは、テーブル構造に対応する JDBC Prepared Statement を作成します。

```
insert into ${table.dbtablename} (
#foreach ($field in $table.fields)
  #optcomma() ${field.name}
#end)
values (
#foreach ($field in $table.fields)
  #optcomma() ?
#end)
```

メタデータ クエリーのコード

次のコードは、テーブルのメタデータ（カラム名およびタイプ）を取得するために使用されるシンプルなクエリーを定義します。空の結果セットを戻す任意のクエリーを使用できます。

```
select * from ${table.dbtablename} where 1=0
```

テストおよびデバッグ

データベースの一連のテンプレートを作成する場合に、解析結果を直接参照できると便利です。この機能を実現するために、JDBC アダプタは CM メイン スクリプト `~scmscm/cm/bin/cm` による直接呼び出しをサポートしています。

このような呼び出しの一般構文は、次のとおりです。

```
~/cm/bin/cm invoke com.cisco.scmscm.adapters.jdbc.JDBCAdapter argument
```

argument は、次のセクションで説明されるフラグの 1 つです。このメカニズムは、CM が稼動しているかどうかに関係なく使用することができます。

また、次のセクションに記載されたクエリーおよび更新の実行メソッドを使用すると、動作中のデータベースに対するテンプレート結果をテストできます。

- 「文字列の解析」(P.6-9)
- 「完全デバッグ情報の取得」(P.6-9)

文字列の解析

すべての文字列は、コンテキスト全体が適切な場合、VTL テンプレートとして解析できます。解析結果は、標準出力に表示されます。文字列を解析するには、`-parse` フラグを指定して、アダプタを呼び出します。次に例を示します（応答は太字で表記されています）。

```
$ ~/cm/bin/cm invoke com.cisco.scmscm.adapters.jdbc.JDBCAdapter -parse 'xxx'
xxx
$ ~/cm/bin/cm invoke com.cisco.scmscm.adapters.jdbc.JDBCAdapter -parse '$dbinfo.url'
jdbc:oracle:thin:@localhost:1521:apricot
$ ~/cm/bin/cm invoke com.cisco.scmscm.adapters.jdbc.JDBCAdapter -parse
'$tools.getTableByName($tables, "LUR").sql.createTable'
create table RPT_LUR (
  TIME_STAMP      date
,RECORD_SOURCE   integer
,LINK_ID         integer
,GENERATOR_ID    integer
,SERVICE_ID     integer
,CONFIGURED_DURATION  integer
,DURATION        integer
,END_TIME        integer
,UPSTREAM_VOLUME integer
,DOWNSTREAM_VOLUME integer
,SESSIONS       integer
)
```

完全デバッグ情報の取得

テンプレートによって作成された **tables** および **dbinfo** 構造のすべての内容のダンプを表示するには、`-debug` フラグを使用します。このフラグを使用すると、これらの構造のすべてのフィールド、プロパティ、およびオプションの詳細が標準出力に表示されます。

スクリプトでの JDBC フレームワークの使用法

任意の SQL コマンドをデータベースに送信して実行したり、作成されたデータを表示したりできます。この方法は、定期的にデータベースをメンテナンスしたり、データベース テーブルの内容をモニタしたり、追加のデータベース テーブルを管理する場合などに役立つことがあります。

`update` 処理を実行するには、`-executeUpdate` フラグを指定してアダプタを呼び出します。クエリーを実行して、結果を表示するには、`-executeQuery` フラグを指定してアダプタを呼び出します。

SCE 時間帯オフセットの表示および設定

次の更新処理のサンプルは、Service Control Engine (SCE) 時間帯オフセット設定を保持するデータベース テーブル内の値をプログラムで変更する方法を示しています。このテーブルの名前は、通常 **JCONF_SE_TZ_OFFSET** です。テーブルに別の名前が割り当てられている可能性があるため、ここでは論理名 **TZ** を使用します。「[tables.xml ファイル](#)」(P.A-1) のリストを参照してください。

テーブルの有無を最初に確認してから更新する必要がないように、テーブルを削除してから（テーブルが存在しない場合のエラー ステータスは無視します）、テーブルを再作成し、適切な値を挿入します。テーブルにはタイムスタンプ カラムが含まれているため、データベース内の現在日付を取得する必要があります。この処理は各データベース ベンダーで固有なため、この例ではテンプレート内で定義された設定済みの `getdate` 処理を呼び出しています。

ツール **assignParams** および **getTableByName** の使用方法に注意して、SQL を生成してください。

```
#!/bin/bash

this=$0
tableName=TZ

usage () {
    cat <<EOF
Usage:
    $this --status      - show currently configured TZ offset
    $this --offset=N   - set the offset to N minutes (-1440 <= N <= 1440)
    $this --help       - print this message
EOF
}

query () {
    ~/cm/bin/cm invoke com.cisco.scmscm.adapters.jdbc.JDBCAdapter -executeQuery "$*"
}

update () {
    ~/cm/bin/cm invoke com.cisco.scmscm.adapters.jdbc.JDBCAdapter -executeUpdate "$*"
}

get_tz () {
    query 'select * from $tools.getTableByName($tables, "TZ").dbtabname'
}

set_tz () {
    update '$tools.getTableByName($tables, "TZ").sql.dropTable'
    update '$tools.getTableByName($tables, "TZ").sql.createTable'
    update '$tools.assignParams($tools.getTableByName($tables, "TZ").sql.insert,
[$dbinfo.options.getdate, '$1'])'
}

case $1 in
    --status)
        get_tz
        ;;
    --help)
        usage
        exit 0
        ;;
    --offset=*)
        n=$(echo $1 | egrep 'offset=[-]?[0-9]+' | sed 's/.*=//')
        if [ "$n" ]; then
            if [ "$n" -ge -1440 -a "$n" -le 1440 ]; then
                set_tz $n &>/dev/null
                ok=1
            fi
        fi
        if [ ! "$ok" ]; then
            usage
            exit 2
        fi
        get_tz
        ;;
    *)
        usage
        exit 3
        ;;
esac
```

実行したクエリーによって返された一連の結果は、適切なカラム ヘッダーを使用して、表形式で標準出力に表示されます。

Oracle のスケラビリティに関するヒント

次のセクションでは、CM でのデータベース処理のスケラビリティを高める方法を示します。これらの方法は Oracle 特有であり、この機能が実現可能であることを示す単なるヒントとして示されています。

- 「カスタム テーブルスペースの使用方法」(P.6-11)
- 「テーブルパーティショニングの使用方法」(P.6-12)

カスタム テーブルスペースの使用方法

複数のテーブルスペースを作成し、それらの間で CM テーブルを配信するとします。各テーブルで使用するテーブルスペースを **tables.xml** ファイルに指定します。1 つのテーブルに対して、定義は次のようになります (**太字**のコードに注意)。

```
< rdr name="LUR" dbtabname="RPT_LUR" tag="4042321925" createtable="true">
  < options>
    < option property="tablespace" value="tspacel" />
  < /options>
  < fields>
    < field id="1" name="TIME_STAMP" type="TIMESTAMP">
      <!-- (other field declarations) -->
    < field id="10" name="DOWNSTREAM_VOLUME" type="UINT32"/>
    < field id="11" name="SESSIONS" type="UINT32"/>
  < /fields>
  < indexes>
    < index name="RPT_LUR_I1" columns="END_TIME">
      < options>
        < option property="clustered" value="true"/>
        < option property="allowduprow" value="true"/>
        < option property="tablespace" value="tspace2"/>
      < /options>
    < /index>
  < /indexes>
< /rdr>
```

このサンプルでは、インデックスおよびテーブル用に必要なテーブルスペース (**tspacel** および **tspace2**) が追加されています。CM 内のオプションの **tablespace** に意味は事前設定されていません。任意の新しいオプション名を使用できます。意味は、テンプレートでの今後の使用方法に基づいて決定します。

正しいテーブルスペース内にテーブルを作成するには、**create_table.vm** を次のように変更します。

```
create table $table.dbtabname (
#foreach ($field in $table.fields)
  #optcomma()$field.name #sqltype($field)
  #if ("${field.options.notnull}" == "true")
    not null
  #end
#end)
#if ("${table.options.tablespace}" != "")
  TABLESPACE $table.options.tablespace
#end
```

独自のテーブルスペース内にインデックスを作成するには、**create_indexes.vm** を次のように変更します。

```
#foreach ($index in $table.indexes)
  create index $index.name on $table.dbtabname ($index.columns)
  #if ("${index.options.tablespace}" != "")
    TABLESPACE $index.options.tablespace
  #end
#end
```

テーブルパーティショニングの使用方法

週単位で特定のテーブルにローリングパーティショニングを実装するために、前のセクションの例のような `tables.xml` ファイル内に、テーブル用の `partitioned` オプションを作成できます（「[カスタムテーブルスペースの使用方法](#)」(P.6-11) を参照）。その後、次のように `create_table.vm` コードを追加します（太字のコードに注意）。

```
create table $table.dbtabname (
#foreach ($field in $table.fields)
#optcomma()$field.name #sqltype($field)
#if ("${field.options.notnull}" == "true")
not null
#end
#end)
#if ("${table.options.partitioned}" != "")
partition by range (timestamp)
(partition week_1 values less than (to_date ('01-JAN-2005 00:00:00','DD-MON-YYYY
HH24:MI:SS')) ,
partition week_2 values less than (to_date ('08-JAN-2005 00:00:00','DD-MON-YYYY
HH24:MI:SS'))
partition week_3 values less than (to_date ('15-JAN-2005 00:00:00','DD-MON-YYYY
HH24:MI:SS'))
partition week_4 values less than (to_date ('22-JAN-2005 00:00:00','DD-MON-YYYY
HH24:MI:SS')) ) ;
#end
```

Oracle では時間境界に非定数表現を指定できないため、テーブルを作成するときに値を組み込む必要があります。

cron ジョブを作成して、毎週パーティションをローリング（古いパーティションを削除して、新しいパーティションを作成する）できます。この cron ジョブは、JDBC アダプタのコマンドラインインターフェイスを呼び出し（「[スクリプトでの JDBC フレームワークの使用方法](#)」(P.6-9) を参照）、適切な `alter table drop partition` および `alter table add partition` SQL コマンドを発行するスクリプトを実行します。