プログラマブル レポートのガイドライン

前提条件

計画

1. レポートがファブリック全体またはデバイス全体に対して実行されることを意図しているかどうかを判断します。
2. 必要なデータを収集するために、どの **[show]** コマンドをスイッチで実行する必要があるかを調べます。
   - CLI 出力が xml または json をサポートしているか、どちらもサポートしていないかを調べます。
   - どちらでもない場合、スイッチはプレーンな CLI 出力を返します。
   - 実行されたコマンドを含む CLI 応答を elasticsearch に保存する必要があるかどうかを判断します。応答を保存すると、ストレージが大幅に増加する可能性があるため、注意が必要です。
3. 繰り返し、期間などのレポート作成入力を事前検証する必要があるかどうかを判断します。たとえば、レポート ジョブは定期的なレポートをサポートしていますか。また、ジョブはどのくらいの頻度で実行する必要がありますか。

レポートのプレゼンテーション

1. **サマリー**が必要な場合は、データを表示する形式を選択します。
   - キーと値のペア

- 表
- チャート（縦棒、円、折れ線）

2. **セクション**（詳細ビュー）で、データを表示する形式を選択します。
   - キーと値のペア
   - JSON オブジェクトの配列
   - チャート（縦棒、円、折れ線）
3. **フォーマッタ**の場合、以下が適用されます。
   - UI に表示される値にフォーマットを追加する
   - サポートされているマーカー：ERROR、SUCCESS、WARNING、INFO

レポート間のデータ比較

1. レポートで現在のレポートと古いレポートのデータを比較する必要があるかどうかを判断します。
2. 「はい」の場合、レポートインフラ API を使用すると、次のような以前のレポートを取得できます。
   - 1 つまたは複数の以前に生成されたレポート
   - レポート ジョブの最も古いレポート
   - 特定のレポートのサマリ
   - reportCLI 出力プロセスの特定のセクション

CLI 出力プロセス

XML形式

CLI 出力が XML 形式でデータを返す場合、レポートインフラストラクチャで提供される XML ユーティリティを使用して、XML データを読み取ることができます。

From reportlib.preport import *:

- getxmltree(xml_string, tag)
- getxmlrows(xml_tree, tag_xpath)
- getnodevalue(xml_tree, node_xpath)
- has_tag(xml_tree, tag)

例については、レポート テンプレート **switch_inventory** を参照してください。

[JSON 形式（JSON Format）]

CLI 出力が JSON 形式でデータを返す場合は、Python の json モジュールをインポートし、**json.loads()** メソッドを使用して JSON 文字列を解析します。

import json

json_string =<CLI response>

json_obj = json.loads(json_string)

例については、レポート テンプレート **fabric_nve_vni_counter** を参照してください。

プレーンな CLI 出力

CLI 出力が CLI UI に表示されるのと同じ形式でデータを返す場合、独自の解析メソッドを記述して、CLI 応答でデータを読み取る必要があります。

ロガー

Logger を使用すると、レポート テンプレートからメッセージをログに記録できます。ログに記録されたメッセージは、/usr/local/cisco/dcm/fm/logs/preport_jython.log に書き込まれます。

レポート テンプレート

テンプレートのプロパティ

次の必須テンプレート プロパティを指定します。

```
name = <template-name>;
tags = fabric or device;
userDefined = true or false;
templateType = REPORT;
templateSubType = GENERIC;
contentType = PYTHON;
```

（注）

- ファブリックに対してレポートが実行される場合は、**tags = Fabric** を設定します。デバイスに対してレポートが実行される場合は、**tags = device** を設定します。
- テンプレートがお客様によって作成された場合は、**userDefined = true** を設定します。テンプレートが DCNM 開発者によって作成された場合は、**userDefined = false** を設定します。

テンプレート変数

```
Specify the following template variables:
##template variables
@(IsInternal=true)
```

```
string fabric_name or serial_number;
string user_input;
```

（注）
- **tags = fabric** の場合、変数 *fabric_name* を構成します
- **tags = device** の場合、変数 *serial_number* を構成します
- ユーザー変数はオプションです。DCNM テンプレート インフラでサポートされているすべてのデータ タイプと注釈を使用できます。

テンプレートの内容

インポート済みライブラリ

次の 2 つの Python ライブラリが必要です。**reportlib.preport** には、すべてのレポート インフラストラクチャ API が含まれていることに注意してください。

```
##template content
from com.cisco.dcbu.vinci.rest.services.jython import WrappersResp
from reportlib.preport import *
```

テンプレート関数

generateReport()

generateReport() は入力関数であり、レポートの生成中に呼び出されます。すべてのレポート導入ロジックをここに提供する必要があります。この関数は、コンテキスト オブジェクトを受け取ります。「コンテキスト」パラメータは、レポート ジョブの作成時にレポート インフラストラクチャによって作成されます。

```
def generateReport(context):

    report = Report("Report title")   ## Create a report object
     ## Gather data and fill in content for the report

    respObj = WrappersResp.getRespObj()
    respObj.setSuccessRetCode()
    respObj.setValue(report)
    return respObj
```

（注）
- この関数は WrappersResp オブジェクトを返す必要があります。
- レポートの生成にエラーがない場合、この関数内で作成されたレポート オブジェクトは、**WrappersResp** オブジェクトが返される前に **WrappersResp.setValue()** で設定する必要があります。

CLI を実行し、CLI 応答を処理する

以下は、1 つまたは複数のデバイスに **show** コマンドを送信する方法、およびデバイスからの応答を処理する方法に関するサンプル コードです。

```
show_cmd1 = 'show xxx'
show_cmd2 = 'show yyy'
device_list = [device1,device2]
## run the command(s) on each device in the device_list


cli_responses = show(device_list, show_cmd1, show_cmd2)
## run the command(s) on each device in the device_list and store the CLI response(s)

cli_responses = show_and_store(device_list, show_cmd1, show_cmd2)
```

For resp in cli_responses:

```
command = resp['command'].strip()

    if show_cmd1 in command:
        cmd1_response = resp['response'].strip()
        ## process show_cmd1 response

    elif show_cmd2 in command:
        cmd2_response = resp['response'].strip()
        ## process show_cmd1 response
```

validate()

validate() 関数はオプションの関数であり、繰り返し、期間などのレポート作成入力の事前検証を実行するために使用されます。この関数が定義されている場合、レポートジョブの作成中に呼び出されます。レポート ジョブは、この関数が SuccessRetCode で WrappersResp を返す場合にのみ作成されます。検証が失敗した場合、エラーを含むFailureRetCodeを持つWrappersRespが返されます。

```
def validate(context):
    respObj = WrappersResp.getRespObj()
    ## Validation content

    if validation_failed:
        respObj.addErrorReport(…)
        respObj.setFailureRetCode()
    else:
        respObj.setSuccessRetCode()
    return respObj
```

report.add_summary

各レポートには1つのサマリを含めることができ、コンテンツはPythonディクショナリです。

```
summary = report.add_summary()
summary[key] = value
summary.add_message(msg)
## Present the summary in a table format

table = summary.add_table(title,_id)   ## _id must be a unique id for the table
table.append(value,_id)   ## adding rows to table
## Present the summary in a chart format

chart = summary.add_chart(ChartType,_id)
## ChartTypes: ChartTypes.COLUMN_CHART, ChartTypes.PIE_CHART, ChartTypes.LINE_CHART
```

report.add_section

セクションは、レポート コンテンツの論理グループです。セクションの詳細は、**[詳細の表示（View Details）]** に表示されます。

```
section = report.add_section(title,_id)   ## _id must be a unique id for the section
section[key] = value
section.append(key, json_obj, _id)   ## adding rows of json objects to section
## Present the section details in a chart format

chart = section.add_chart(ChartType,_id)
## ChartTypes: ChartTypes.COLUMN_CHART, ChartTypes.PIE_CHART, ChartTypes.LINE_CHART
```

翻訳について
このドキュメントは、米国シスコ発行ドキュメントの参考和訳です。リンク情報につきましては、日本語版掲載時点で、英語版にアップデートがあり、リンク先のページが移動/変更されている場合がありますことをご了承ください。あくまでも参考和訳となりますので、正式な内容については米国サイトのドキュメントを参照ください。