日日是Oracle APEX

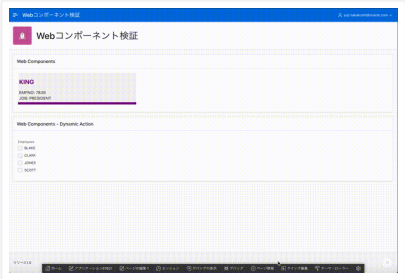
Oracle APEXを使った作業をしていて、気の付いたところを忘れないようにメモをとります。

2021年1月5日火曜日

Oracle APEXでWebコンポーネントを扱う

Oracle IndiaのSrihari RavvaさんがWeb Components in Oracle APEXとして、WebコンポーネントをOracle APEXで扱う方法を紹介しています。面白いトピックなので、自分でも試してみました。

環境に依存せずに作業を試行できるよう、いくつか手順を変えています。特にJavaScriptから呼び出すRESTful APIを独自のものから、Oracle APEXのRESTful APIのサンプルとして提供されているoracle.example.hrを使うようにしています。



確認作業は以下の順番で行います。

1. Oracle APEX Builder Extension by FOSを導入する。
2. RESTfulサービスのサンプルをインストール(またはリセット)する。
3. 空のアプリケーションを作成する。
4. 静的アプリケーション・ファイルに、Webコンポーネントとなる JavaScriptのコードを記述する。
5. 静的リージョンを作成し、動作を確認する。
6. 動的アクションを作成し、動作を確認する。

１、２、３が準備作業、４、５、６が元記事に記載のあるWebコンポーネントにまつわる作業です。

Oracle APEX Builder Extension by FOSを導入する

FOEX GmbHが開発した、Oracle APEXの静的アプリケーション・ファイル、静的ワークスペース・ファイルやプラグインのファイルを、Monaco Editorで直接編集可能にするブラウザの拡張機能です。

こちらよりChromeまたはFirefoxのウェブストアへ移動し、拡張機能を導入します。

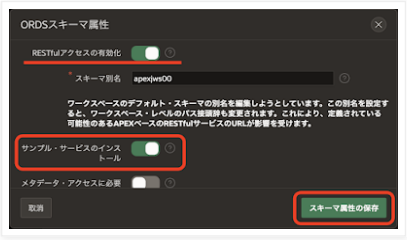
この拡張機能を導入しなくても、手元で編集したファイルを静的アプリケーション・ファイルとしてアップロードすることもできます。

RESTfulサービスのサンプルをインストール(またはリセット)する

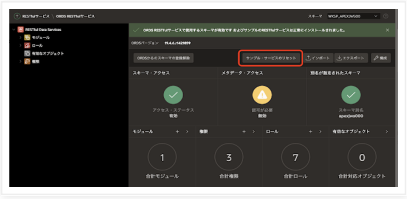
SQLワークショップより**RESTfulサービス**を開きます。

RESTfulサービスは、Oracle REST Data Servicesによって実装されています。Oracle APEXでは、サービスを作成するためのユーザー・インターフェースのみ提供しています。Oracle REST Data Servicesによって、Oracle APEXのワークスペースに紐づいているスキーマが扱えるよう、**ORDSにスキーマを登録**を実行します。

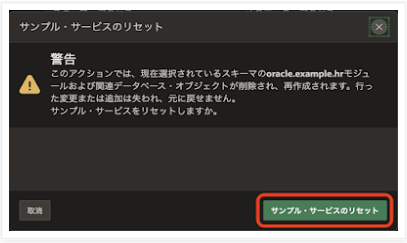
今回はサンプル・サービスをそのまま利用するので、**サンプル・サービスのインストール**を**ON**にし、**スキーマ属性の保存**を実行します。



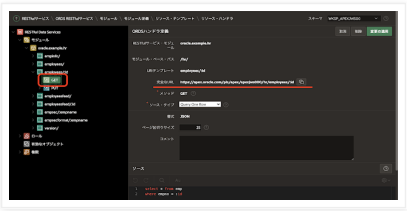
以上でサンプル・サービスが利用可能になっています。すでにRESTfulサービスが有効になっている場合は、**サンプル・サービスのリセット**を実施します。



警告が表示されます。oracle.example.hrモジュールおよび表EMPとDEPTが削除され、再作成されることに注意してください。困る場合は、検証の際に使用する従業員の情報を調整することにして、リセットせずに作業を進めましょう。リセットすることに問題がなければ、**サンプル・サービスのリセット**を実行します。



ORDSハンドラ定義を開いて、**完全なURL**を確認します。今回使用するREST APIは、**モジュール**oracle.example.hrの**URIテンプレート**employees/:id、**メソッド**は**GET**です。



Webコンポーネントの中から、この**完全なURL**に従業員番号(EMPNO)を引数として与えて呼び出します。スクリーンショットに表示されている完全なURLは、それぞれの環境で異なります。

空のアプリケーションを作成する

今回の検証を実装するアプリケーションを作成します。**アプリケーション・ビルダー**より**作成**を実行します。

新規アプリケーションをクリックします。

アプリケーションの**名前**は任意ですが、今回は**Webコンポーネント検証**としています。その他は何もせず、**アプリケーションの作成**を実行します。

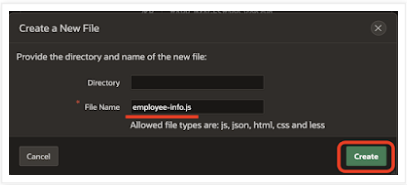
以上でアプリケーションの準備も完了しました。

JavaScriptのコードを記述する

静的アプリケーション・ファイルに、Webコンポーネントとなる**JavaScript**のコードを記述します。**共有コンポーネント**の**静的アプリケーション・ファイル**を開きます。

Oracle APEX Builder Extension by FOSがインストールされていると**Edit Files**の領域が表示されます。**Create File**をクリックします。

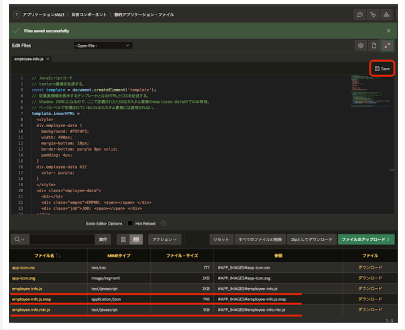
File Nameとして**employee-info.js**を指定し、**Create**を実行します。



以下のコードを貼り付け、Saveを実行します。

```
// JavaScriptコード
// temlate要素を生成する。
const template = document.createElement('template');
// 従業員情報を表示するテンプレートとなるHTMLとCSSを記述する。
// Shadow DOMにになるので、ここで定義されたCSSはカスタム要素のemployee-data内でのみ有効。
// ページレベルで定義されているCSSはカスタム要素には適用されない。
template.innerHTML = `
  <style>
  div.employee-data {
    background: #F5F4F2;
    width: 400px;
    margin-bottom: 10px;
    border-bottom: purple 8px solid;
    padding: 4px;
  }
  div.employee-data h2{
    color: purple;
  }
  </style>
  <div class="employee-data">
    <h2></h2>
    <div class="empno">EMPNO: <span></span> </div>
    <div class="job">JOB: <span></span> </div>
  </div>
`;
// カスタム要素のためのクラスを定義する。
// 一般的にはカスタム要素の名前が custom-element であれば、CustomElement をクラス名とする。
// 今回は employee-info がカスタム要素なので、クラス名は EmployeeInfo となる。
class EmployeeInfo extends HTMLElement {
  constructor() {
    // かならず super をコンストラクタの先頭で呼び出す。HTMLElementとして正しく初期化される。
    super();
    // Shadow DOMをアタッチする。これによりカスタム要素が独立する。
    this.attachShadow({ mode: 'open' });
```

```
    // テンプレートをクローンし、ShadowRootに追加する。
    this.shadowRoot.appendChild(template.content.cloneNode(true));
    // RESTful APIを呼び出し従業員データを取得する。
    // 従業員番号(empno)の属性を引数として、RESTful APIを呼び出す。
    // RESTful APIの応答をテンプレートの穴埋めに使用する。
    fetch('https://apex.oracle.com/pls/apex/your_workspace/hr/employees/' + this.getAttribute('empno')).then(response => response.json()).then(data => {
      this.shadowRoot.querySelector('h2').innerText = data.ename;
      this.shadowRoot.querySelector('div.empno > span').innerText = data.empno;
      this.shadowRoot.querySelector('div.job > span').innerText = data.job;
    });
  }
}
// カスタム要素 employee-info と クラス EmployeeInfo を関連づける。
window.customElements.define('employee-info', EmployeeInfo);
```



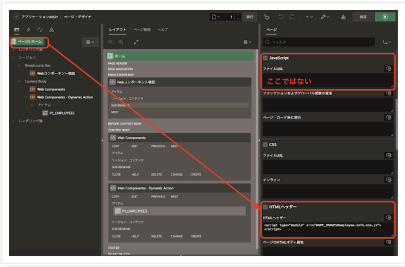
employee-info.jsを保存すると、employee-info.js.mapおよびemployee-info.min.jsも作成されます。

処理内容については、コードのコメントを参照してください。

静的リージョンで動作確認をする

ページ・デザイナで**ホーム・ページ**(Page 1)を開きます。**HTMLヘッダー**として以下を設定し、ホーム・ページのロード時に、先ほど静的アプリケーション・ファイルとして作成したJavaScriptのファイルを読み込みます。

```
<script type="module" src="#APP_IMAGES#employee-info.min.js"></script>
```



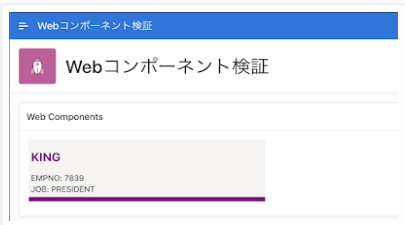
静的アプリケーション・ファイルを標準のJavaScriptのファイルURLではなく、HTMLヘッダーのscriptタグで読み込んでいます。これは、スクリプトをmoduleとして認識させ、グローバルから参照できないようにするためです。

次に**リージョンを作成**し、**名前**は**Web Components**、**タイプ**は**静的コンテンツ**とします。**ソース**の**テキスト**には以下を設定します。

```
<employee-info empno="7839"></employee-info>
```

リージョンの設定ができたら、**ページの保存と実行**を行い、動作を確認します。

以下のように、従業員KINGが表示されたら、手順通り実装できています。

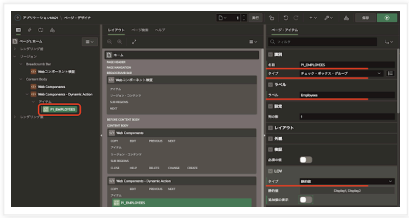


動的アクションを使って動作確認をする

Webコンポーネントの表示に動的アクションを使います。再度、新規にリージョンを作成します。**名前**は**Web Components - Dynamic Action**、**タイプ**は**静的コンテンツ**です。**ソース**の**テキスト**には以下を設定します。

```
<div id="employee_data_container"></div>
```

表示する従業員を指定するため、**ページ・アイテムの作成**を実行します。**名前**は**P1_EMPLOYEES**、**タイプ**は**チェック。ボックス・グループ**、**ラベル**は**Employees**とします。**LOV**の**タイプ**は**静的値**とします。



LOVの**静的値**として、以下のペアを設定します。

表示値	戻り値
BLAKE	7698
CLARK	7782
JONES	7566
SCOTT	7788

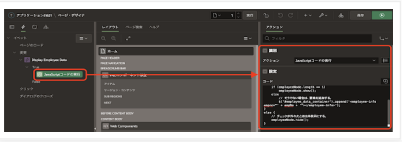
サンプルのRESTfulサービスが初期状態でない場合は、表示値としている従業員が存在しない場合もあります。存在する従業員を探してLOVの値を登録してください。

最後に動的アクションを登録します。左ペインから**動的アクション・ビュー**を開き、動的アクションを作成します。

名前は**Display Employee Data**とし、**タイミング**として、**イベント**は**変更**、**選択タイプ**は**jQueryセレクタ**、**jQueryセレクタ**として**input[name=P1_EMPLOYEES]**を指定します。

実行される**アクション**として、**JavaScriptコードの実行**を指定し、**設定**の**コード**に以下を設定します。

```
// チェック・ボックスより従業員番号を取得する
let empNo = $(this.triggeringElement).val();
// 従業員番号に対応するemployee-infoのタグを探して取得する。
let employeeNode = $("employee-info[empno=" + empNo + "]");
if ($(this.triggeringElement).is(':checked')) {
   // 要素が既に存在するか確認する。存在する場合は表示する。
   if (employeeNode.length == 1)
      employeeNode.show();
   else
      // そうでない場合は、要素を追加する。
      $('#employee_data_container').append('<employee-info empno="' + empNo + '"></employee-info>');
}
else {
   // チェックが外れたときは非表示にする。
   employeeNode.hide();
}
```



動的アクションの設定が完了したら、**ページの保存と実行**を行います。この記事の先頭にあるGIF動画の動作を確認できるはずです。

以上でOracle APEXでのWebコンポーネントの実装についての記事は終了です。

実際にはより複雑な実装になるかと思いますが、元記事を読んで、基本的な実装手順について参考になる内容だと思いました。また、Oracle APEXのプラグインとして実装することで、サーバー側の条件や認可スキームなども活用できるようになり、より便利なものになるでしょう。

完

Yuji N. 時刻: 14:45

共有

‹ ホーム ›

ウェブ バージョンを表示

自己紹介

Yuji N.

日本オラクル株式会社に勤務していて、Oracle APEXのGroundbreaker Advocateを拝命しました。
こちらの記事につきましては、免責事項の参照をお願いいたします。

詳細プロフィールを表示