WebPerformer-NX

WebPerformer-NX
処理を共通化するための参考資料

Canon キヤノン ITソリューションズ株式会社

目次

1. 本資料について

2. 基本概念
    1. 共通化
    2. NX で共通化を実現する手法
    3. どのような関数・定数を共通化するか

3. 各手法の詳細
    1. ユーザ関数
    2. 定数
    3. プロパティファイル

4. 各手法のメリット、デメリット

1. 本資料の目的

本資料はWebPerformer-NXによるアプリケーション開発における
「処理を共通化する方法」について解説しています。

実現方法の例と各手法のメリット/デメリットをご紹介しておりますので、
システム開発を進める際のご参考としてください。

◆ 注意事項

- WebPerformer-NX バージョン 3.0.0時点での情報です。
- 本資料は共通化手法の一例をご紹介するものであり、
  この方法で実装しなければならない、というわけではございません。

2-1. 共通化

共通化とは
プログラムの中で繰り返し使われるコードや処理を一つにまとめて、共通のプログラムとして実装することです。

共通化するメリットとして、下記が挙げられます。
- 複数の画面やアクションの重複コードが無くなり、可読性が上がる
- バグ修正や機能追加を一箇所で行うだけで済むため、メンテナンスがしやすくなる
- 共通化することで関数に切り出された状態となる。関数には名前が付けられているため、
  コードの抽象化と見やすさに繋がる

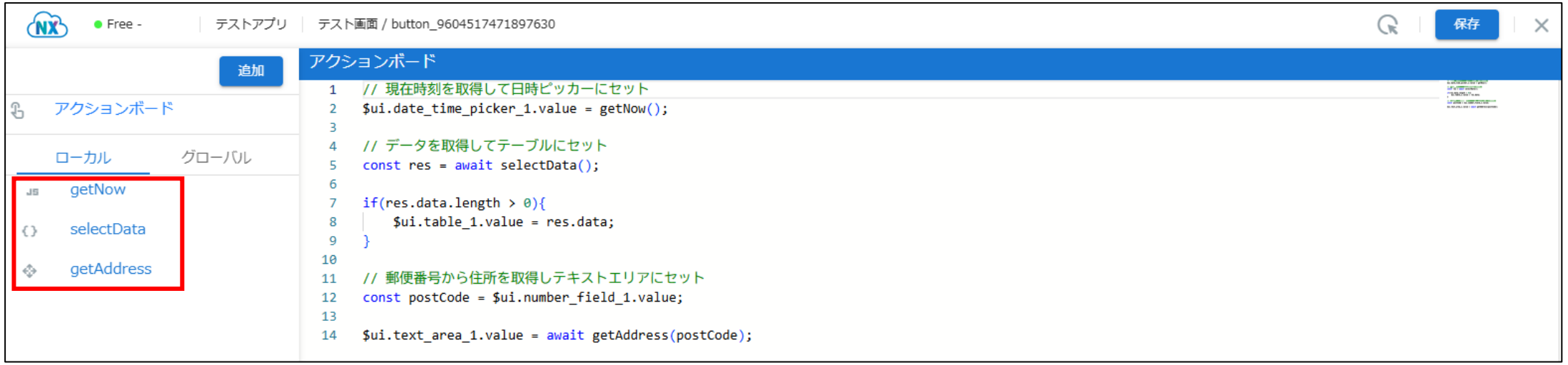
コードを共通化することで、システム全体の構造が複雑になることもあり、注意する必要もありますが、
各チームのニーズやプロジェクトの特性に応じて適切に設計/使用することで、
より良いソフトウェア開発を実現することができます。

WebPerformer-NXによる開発時に共通化を行う方法として、下記3種類の手法がございます。
- **ユーザ関数（ローカル関数/グローバル関数）**
- **定数**
- **プロパティファイル**

2-2. NX で共通化を実現する手法

ユーザ関数

アクションボード内で、呼び出し可能なJavaScript、SQL、REST関数を指します。



ユーザ関数には、「**グローバル関数**」「**ローカル関数**」の 2種類存在します。
ローカル関数の場合、同じアクション内のみ使用可能で、他のアクションから直接呼び出すことはできません。
グローバル関数の場合、アプリ内で使用可能で、同じアプリ内であれば別のアクションから呼び出すことができます。
スコープが異なるため、下記のような使い分けをすると有効です。

グローバル関数

複数のアクションで同じ処理を行う場合

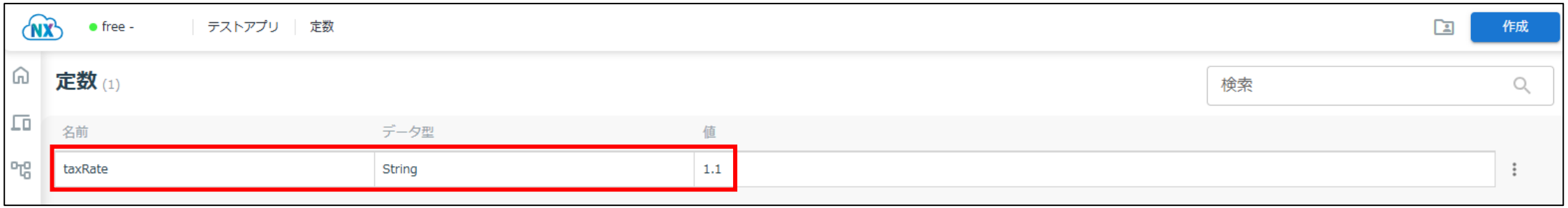
ローカル関数

1つのアクション内のみで繰り返し同じ処理を行う場合

2-2. NX で共通化を実現する手法

定数

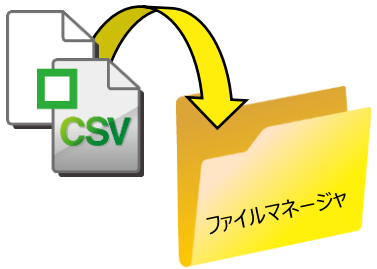
アクションボード内で呼び出し可能な、オブジェクト$constのメソッド を指します。



定数はアプリケーション毎に管理され、一斉に該当箇所に値変更を適用できるようになります。

プロパティファイル

アプリケーションの設定情報や定数を外部ファイルに格納するためのファイル形式の一つです。
ファイルマネージャに格納して呼び出すため、ファイルを読み取るための処理が必要となるものの、
アプリケーションの再デプロイなしに設定を変更できるなどのメリットがあります。



定数とプロパティファイルは、下記のような違いがあります。

定数

更新時はデプロイが必要だが、
アプリ内で定義が完了するため、使いやすい

プロパティファイル

デプロイなく変更可能なため、
利便性は高いものの作りこみが必要（リスクも高い）

2-3. どのような関数・定数を共通化するか

ユーザ関数（グローバル関数/ローカル関数）、定数、プロパティファイルを使い分ける際のポイントは下記の通りです。

ユーザ関数（グローバル関数）

用途：特定の処理や計算を行うための関数。複数のアクションで共通したロジックに使用する
　　グローバル関数/ローカル関数の作成の呼び出し方は異なるため、
　　複数のアクションで使用されることを見据えて、グローバル関数で作成する選択肢もあり

使用例：

- 複数のモジュールやコンポーネントで共通して使用される処理（例：データのフォーマットや計算処理）
- 複雑なロジックやアルゴリズムを持つ場合

ユーザ関数（ローカル関数）

用途：特定のアクション内でのみ必要な共通のロジックに使用する

使用例：

- データの前処理
- 特定の計算を行うための補助的な処理

2-3. どのような関数・定数を共通化するか

定数

用途：変更されることのない値（例：数値、文字列、設定値など）。一括して固定値を変更できるようにする
使用例：
- アプリケーション全体で共通して使用される設定値（例：メッセージ）
- マジックナンバーやマジックストリングを避けるために使用

プロパティファイル

用途：設定情報や環境依存の値を外部ファイルとして管理。アプリケーション設定を柔軟に変更できるようにする
使用例：
- 環境毎の設定（例：開発環境、テスト環境、本番環境で異なるファイルパスなど設定値があれば）
- お客様側から頻繁に変更の要望があるもの（例：メール文面）

3-1. ユーザ関数

本章では、各手法の詳細をご紹介します。
<ユーザ関数の作成>
①アクションボードを開き、追加ボタンを選択

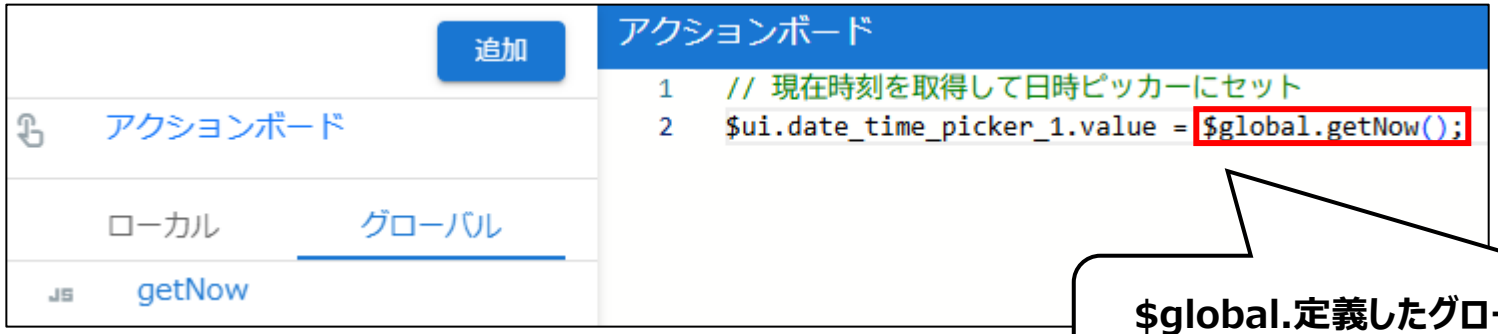
②関数範囲[ローカル][グローバル] どちらか使いたい方を選択した上で
関数タイプに作成したい関数、任意の関数名を記入し追加ボタンを押下

アクションボードにてローカル/グローバルタブを選択することで
それぞれ作成したユーザ関数を確認できます。

グローバル関数については対象のアプリを選択した状態で
サイドメニュー「グローバル関数」からも確認ができます。

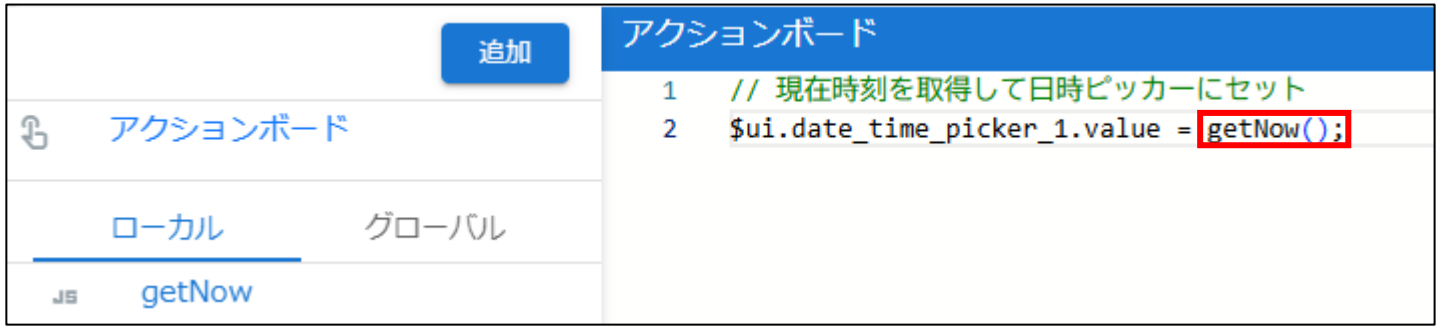
3-1. ユーザ関数

作成したユーザ関数の呼び出し方については、
グローバル関数とローカル関数で定義が異なりますのでそれぞれご紹介します。
<グローバル関数の呼び出し>
アクションボードを開き、呼び出したい箇所で $global オブジェクトを記述する



$global.定義したグローバル関数名(引数) と
定義することで、作成したグローバル関数を呼び出すことが可能

<ローカル関数の呼び出し>
アクションボードを開き、呼び出したい箇所で関数名を記述する

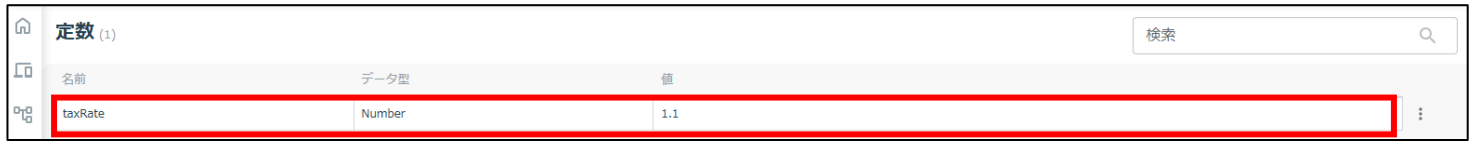


3-2. 定数

利用方法は下記の通りです。
＜定数の作成＞
①該当のアプリを選択した状態で、サイドメニュー「定数」を開く

②画面右上の作成ボタンをクリックし、それぞれの項目を入力し作成ボタンを押下する

一覧に作成した定数が追加される。
定数名の縦三点リーダをクリックすることで、編集/削除も可能



3-2. 定数

呼び出し方法は下記の通りです。
＜定数の呼び出し＞
アクションボードを開き、呼び出したい箇所で $const オブジェクトを定義する

アクションボード

```
1    // 消費税計算を実施する
2    $ui.num_result.value = $ui.num_1.value * $const.taxRate;
```

$const.定義した定数名 と
定義することで、作成した定数を呼び出すことが可能

3-3. プロパティファイル

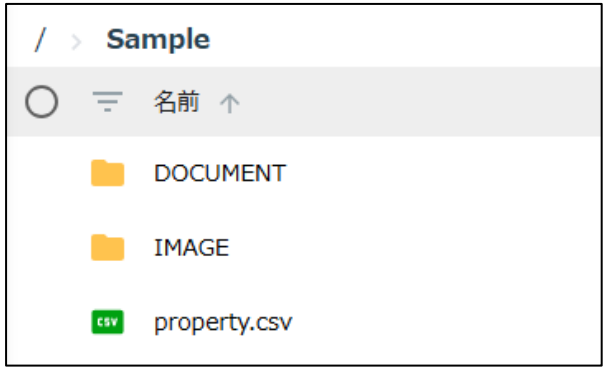
プロパティファイルについては、NX の標準機能ではないため、
ファイルマネージャに格納して読み込む処理をアクションボードに実装する必要があります。
<プロパティファイルの作成>
①設定したいキーと値を記入したファイルを準備する

	A	B	C
1	CSV_FILE_PATH		
2	Sample/CSV/		

本資料では、ファイルの種類を CSV UTF-8 (コンマ区切り) で作成し、
1行目にキー、2行目に値を列ごとに一致するように設定している

```
CSV_FILE_PATH,SYS_ADDRESS,SYS_ADDRESS_NAME
Sample/CSV/,test.mail@example.com,テスト送信専用
```

②ファイルマネージャに格納する
該当のファイルを読み込む処理を実装するため、
格納場所のファイルパス、ファイル名などは事前にルールを決め、指定の場所にファイルを格納する必要がある



※ファイルのフォーマット、格納場所はチームごとに検討・管理する必要がある

3-3. プロパティファイル

<プロパティファイルの読み込み/利用処理の実装>
①最初に表示するUI の初期表示時にプロパティファイルを読み込む処理を実装する

アクションボード

```
1  // プロパティファイル格納用のセッションの値の初期化
2  $session.property = {};
3
4  // プロパティファイルの読み込み、セッションへの格納
5  const file = await $fn.getFile($const.propertyFile);
6  const buffer = file.data;
7  const property = $fn.csvToObject(buffer);
8  $session.property = property[0];
```

左のコードの例でプロパティを読み込んだ場合、
オブジェクト形式でプロパティがセッションに保存される

```
{"CSV_FILE_PATH":"Sample/CSV/"}
```

②使用時に呼び出す処理を実装する
保存したセッションからオブジェクト.プロパティ名にて使用したいデータを指定する

アクションボード

```
1  // WP-NXビルトイン関数：$fn.getUploadFilesでファイル選択ダイアログを表示
2  var uploadfile_array = await $fn.getUploadFiles();
3
4  // 物品番号フォルダ内にアップロードしたファイル分格納する
5  for (let i = 0; i < uploadfile_array.length; i++) {
6      // 指定パスへファイルを保存($fn.putFile)
7      await $fn.putFile($session.property.CSV_FILE_PATH + uploadfile_array[i].filename, { data: uploadfile_array[i].data });
8  }
```

※本資料では、ログイン後の画面初期表示時にセッションに値を保存する方法を解説しましたが、
1か所でしか値が使われない場合は、セッションは使わず使いたい場面で取得する考え方もあります。
またセッションにプロパティファイルの値を保存する場合、セッション破棄のタイミングによってはプロパティの変更が反映されない可能性があります。
変更を即時に適用したい場合は、値が必要なタイミングで都度取得するグローバル関数を準備するなど、ケースによって使い分けることを推奨します。

4. 各手法のメリット、デメリット

各手法を利用するにあたって、それぞれのメリット/デメリットは下記の通りです。

	メリット	デメリット
ユーザ関数 （グローバル関数）	◘ 一つのアプリケーション全体で共通して使用できるため、コードの重複が無くなり、可読性が上がる ◘ 一か所で関数を修正すれば、すべての呼び出しに反映されるため、メンテナンスが容易	◘ グローバル関数に依存するコードが増えると、変更が他の部分に影響を及ぼす可能性がある
ユーザ関数 （ローカル関数）	◘ 特定のアクション内でのみ使用されるため、他の部分に影響を与えない	◘ 他のアクションから呼び出せないため、同じ処理を複数のアクションで行う場合、コードの重複が発生する
定数	◘ 意味のある名前を持つ定数を使用することで、コードの意図が明確になる ◘ 変更時にコード内の全ての箇所を反映でき、マジックナンバーやマジックストリングを避けられる ◘ 型の整合性を保つことができる	◘ 定数の値を変更する場合、アプリケーション全体に影響を与える可能性がある
プロパティファイル	◘ 外部ファイルで管理することで、アプリケーションの再デプロイなしに変更が可能 ◘ 環境ごとに異なる設定をプロパティファイルで管理することで、デプロイ作業が容易になる	◘ プロパティファイルの読み込み/値の呼び出し処理の実装が必要 ◘ プロパティファイルが存在しない、読み込めない場合のエラーハンドリングも必要 ◘ プロパティファイルは通常文字列として管理されるため、型の整合性を保つための追加の処理が必要となる可能性がある