[ID] idleTaskの一例

Posted by 10 A in アプリケーション自動化総合コミュニティフォーラム (Japan) on Jul 12, 2018 9:01:00 PM

InDesignではタスクを一定時間待った後に実行するというクラスが存在します。これを再帰的に回し続けることで一定時間ごとに同じ処理を繰り返し行う事ができます。
これはアプリケーションの状態監視に利用できる手法で、通常のイベント処理関連ではカバーできない範囲の処理を一定時間毎に状態を監視することでイベントによるトリガを擬似的に実現します。
今回はルビ処理の為のScriptUIを利用したヘルパースクリプトをサンプルに解説します。

> システムディレクトリ
>
> Win(x86): C:\Program Files\Common Files\Adobe\CEP\extensions
>
> Win(x64): C:\Program Files (x86)\Common Files\Adobe\CEP\extensions, and C:\Pr
> Files\Common Files\Adobe\CEP\extensions (since CEP 6.1)
>
> Mac: /Library/Application Sup ns
>
> ユーザー個別のディレクトリ
>
> Win: C:\Users\<USERNAME> \AppData\Roaming\Adobe\CEP/extensions
>
> Mac: ~/Library/Application Support/Adobe/CEP/extensions
>
> デバッグの場合は基本的に各エクステンションプロジェクトを上記ディレクトリにコピーする事にクステンションを認識させることができます。

まず下に示すのはフルコードです。

```
(function(){
  var ps;
  if (app.selection[0] instanceof Character
  ||app.selection[0] instanceof Word
  ||app.selection[0] instanceof InsertionPoint
  ||app.selection[0] instanceof Text) {
  if (app.selection[0].toString().indexOf(InsertionPoint)>-1) {
  ps = app.selection[0].index;
  }
  else {
```

```
    ps = 0;
    }
    if(app.idleTasks.itemByName('rubyPalette')!=null)
    app.idleTasks.itemByName("rubyPalette").remove();
    var idleTask = app.idleTasks.add({name:'rubyPalette', sleep:250})
    .addEventListener(IdleEvent.ON_IDLE, task1);
    var w = new Window ('palette', 'ルビ処理', undefined);
    var tx = w.add('edittext', undefined,'', {multiline:false});
    tx.characters = 20;
    w.submitBtn = w.add('button', undefined, '適　用', {name:'ok'});
    w.closelBtn = w.add('button', undefined, 'とじる', {name:'close'});
    w.closelBtn.onClick = function(){
    app.idleTasks.itemByName("rubyPalette").remove();
    w.close();
    }
    w.submitBtn.onClick = function(){
    app.selection[0].rubyString = tx.text;
    app.selection[0].rubyFlag = true;
    };
    w.show();
    }
    function task1(ev){
    app.idleTasks.itemByName("rubyPalette").remove();
    if (app.selection[0] instanceof Character
    ||app.selection[0] instanceof Word
    ||app.selection[0] instanceof InsertionPoint
    ||app.selection[0] instanceof Text) {
    if (app.selection[0].index!=ps){
    tx.text = app.selection[0].rubyString;
    ps = app.selection[0].index;
    }
    app.idleTasks.add({name:'rubyPalette', sleep:250})
    .addEventListener(IdleEvent.ON_IDLE, task1);
    }
    else {
    w.close();
    return;
    }
    }
    }());
```

```
(function(){
    var ps;
    if (app.selection[0] instanceof Character
    ||app.selection[0] instanceof Word
    ||app.selection[0] instanceof InsertionPoint
    ||app.selection[0] instanceof Text) {
    if (app.selection[0].toString().indexOf(InsertionPoint)>-1) {
    ps = app.selection[0].index;
    }
    else {
    ps = 0;
    }
    if(app.idleTasks.itemByName('rubyPalette')!=null)
    app.idleTasks.itemByName("rubyPalette").remove();
    var idleTask = app.idleTasks.add({name:'rubyPalette', sleep:250})
    .addEventListener(IdleEvent.ON_IDLE, task1);
    var w = new Window ('palette', 'ルビ処理', undefined);
    var tx = w.add('edittext', undefined,'', {multiline:false});
    tx.characters = 20;
    w.submitBtn = w.add('button', undefined, '適　用', {name:'ok'});
```

```
w.closelBtn = w.add('button', undefined, 'とじる', {name:'close'});
w.closelBtn.onClick = function(){
app.idleTasks.itemByName("rubyPalette").remove();
w.close();
}
w.submitBtn.onClick = function(){
app.selection[0].rubyString = tx.text;
app.selection[0].rubyFlag = true;
};
w.show();
}
function task1(ev){
app.idleTasks.itemByName("rubyPalette").remove();
if (app.selection[0] instanceof Character
||app.selection[0] instanceof Word
||app.selection[0] instanceof InsertionPoint
||app.selection[0] instanceof Text) {
if (app.selection[0].index!=ps){
tx.text = app.selection[0].rubyString;
ps = app.selection[0].index;
}
app.idleTasks.add({name:'rubyPalette', sleep:250})
.addEventListener(IdleEvent.ON_IDLE, task1);
}
else {
w.close();
return;
}
}
}());
```

まずはUIの方から説明します。以下に抜き出したのはScriptUIで構成された部分です。

```
var w = new Window ('palette', 'ルビ処理', undefined);
var tx = w.add('edittext', undefined,'', {multiline:false});
tx.characters = 20;
w.submitBtn = w.add('button', undefined, '適　用', {name:'ok'});
w.closelBtn = w.add('button', undefined, 'とじる', {name:'close'});
w.closelBtn.onClick = function(){
  app.idleTasks.itemByName("rubyPalette").remove();
  w.close();
  }
w.submitBtn.onClick = function(){
  app.selection[0].rubyString = tx.text;
  app.selection[0].rubyFlag = true;
  };
w.show();
```

```
var w = new Window ('palette', 'ルビ処理', undefined);
var tx = w.add('edittext', undefined,'', {multiline:false});
tx.characters = 20;
w.submitBtn = w.add('button', undefined, '適　用', {name:'ok'});
w.closelBtn = w.add('button', undefined, 'とじる', {name:'close'});
w.closelBtn.onClick = function(){
  app.idleTasks.itemByName("rubyPalette").remove();
  w.close();
  }
w.submitBtn.onClick = function(){
  app.selection[0].rubyString = tx.text;
  app.selection[0].rubyFlag = true;
```

```
  };
w.show();
```

フローティングかつノンブロッキングなpaletteで作られています。これは、編集作業とUIパネルでの入力を両立させるための物です。UIの構成としてはedittextとボタンが２点の シンプルな構成です。各要素も位置関連情報をundefinedに設定してあるために自動配置となります。各ボタンにはonClickでのイベントを設定してあります。submitBtnの方 はテstory内の選択したcharacterに対してrubyStringとrubyFlagを設定します。このUI部分はこれだけでルビ入力用のフローティングパレットとして利用可能な物 となっています。

続いては先頭部分のドキュメント上で選ばれているオブジェクトによる振り分けです。

```
if (app.selection[0] instanceof Character
  ||app.selection[0] instanceof Word
  ||app.selection[0] instanceof InsertionPoint
  ||app.selection[0] instanceof Text) {
  if (app.selection[0].toString().indexOf(InsertionPoint)>-1) {
  ps = app.selection[0].index;
  }
  else {
  ps = 0;
  }
  ・
  ・
  ・
```


```
if (app.selection[0] instanceof Character
  ||app.selection[0] instanceof Word
  ||app.selection[0] instanceof InsertionPoint
  ||app.selection[0] instanceof Text) {
  if (app.selection[0].toString().indexOf(InsertionPoint)>-1) {
  ps = app.selection[0].index;
  }
  else {
  ps = 0;
  }
  ・
  ・
  ・
```


このように選択範囲のインスタンスをチェックしてテキストに関連するものでない場合は何もせず終了するように構成されています。ここで利用されているinstanceof演算子は対象オブジェ クトの基となるプロトタイプにコンストラクタprototypeプロパティが存在するかを見ます。ややこしいことを書いていますが、該当オブジェクトがどのコンストラクタを基に作られたかを見 る物です。
その次に続くのは現在選択されているテキストの位置の記録です。これはidleTaskがスリープした後で選択範囲は変化していないかどうかをチェックする際に利用します。
続いてidleTaskに関連する部分に入ります。

```
if(app.idleTasks.itemByName('rubyPalette')!=null)
  app.idleTasks.itemByName("rubyPalette").remove();
var idleTask = app.idleTasks.add({name:'rubyPalette', sleep:250})
  .addEventListener(IdleEvent.ON_IDLE, task1);
```


```
if(app.idleTasks.itemByName('rubyPalette')!=null)
  app.idleTasks.itemByName("rubyPalette").remove();
var idleTask = app.idleTasks.add({name:'rubyPalette', sleep:250})
```

```
04.    .addEventListener(IdleEvent.ON_IDLE, task1);
```

先ず予防措置としてこれから登録するidleTaskが既に存在するかどうかをチェックします。これは予期せぬ重複などの好ましくない状況を未然に防ぐためのものです。もし存在する場合はそれ らを一旦破棄します。
idleTaskはapplication下のクラスでaddメソッドを利用します。上の例ではnameとsleepの２つのプロパティをaddメソッドの引数として与えて生成しています。こ の例ではaddEventListenerメソッドをチェーンしてidleTaskがsleep終了後に実行するtask1（function）を指定しています。
このidleTaskの最大の特徴はノンブロッキングである点です。類似の機能に$globalに存在するsleepメソッドがありますが、このメソッドはスリープしている間アプリケーション 全体をブロックします。一方**idleTaskはスリープの間はバックグランドに沈んでアプリケーションの動作を阻害することはありません。**
さて、スリープタイマーが指定時間になると以下の関数を実行します。

```
01. function task1(ev){
02.   app.idleTasks.itemByName("rubyPalette").remove();
03.   if (app.selection[0] instanceof Character
04.   ||app.selection[0] instanceof Word
05.   ||app.selection[0] instanceof InsertionPoint
06.   ||app.selection[0] instanceof Text) {
07.   if (app.selection[0].index!=ps){
08.   tx.text = app.selection[0].rubyString;
09.   ps = app.selection[0].index;
10.   }
11.   app.idleTasks.add({name:'rubyPalette', sleep:250})
12.   .addEventListener(IdleEvent.ON_IDLE, task1);
13.   }
14.   else {
15.   w.close();
16.   return;
17.   }
18.   }
```

```
01. function task1(ev){
02.   app.idleTasks.itemByName("rubyPalette").remove();
03.   if (app.selection[0] instanceof Character
04.   ||app.selection[0] instanceof Word
05.   ||app.selection[0] instanceof InsertionPoint
06.   ||app.selection[0] instanceof Text) {
07.   if (app.selection[0].index!=ps){
08.   tx.text = app.selection[0].rubyString;
09.   ps = app.selection[0].index;
10.   }
11.   app.idleTasks.add({name:'rubyPalette', sleep:250})
12.   .addEventListener(IdleEvent.ON_IDLE, task1);
13.   }
14.   else {
15.   w.close();
16.   return;
17.   }
18.   }
```

ここで行われるのは選択範囲の比較です。アイドルする前のテキストポジションと現在のテキストポジションを比較して変化がある場合に選択文字のルビ情報を抽出してUIに返します。
その後再び現在選択されているテキストの位置を記録して再度同じ設定でidleTaskを実行します。
この様に**再帰的にidleTaskを実行し続けることで擬似的に選択範囲の変化をトリガすることができます。**

654 Views Tags : javascript, indesign, event, idletask

0 Comments