Javascript 入門

Promise

鄭祥飛@2021/07/08

同期

一般的にはプログラムは順番に実行する。例:

```
console.log("A");
console.log("B");
console.log("C");
```

上記のプログラムの出力は: A→B→C

同期実行の問題

プログラムにある操作非常に時間のかかる処理があるう場合、別の操作を影響することににになります。

例１：

あるアプリケーションに、サイズは大きなファイルのダウンロードする機能を実行すると、アプリケーションの別の機能も操作できなくなる。

例２：

ユーザ情報をDBに保存する前に、ユーザーの携帯にSMSで通知メッセージを送る事が必要です。SMSメッセージの送信がかかるので、保存処理も時間かかることになる

非同期

- コンピュータCPUは複数スレッドはあります。
- プログラム主体は、主要スレッドで実行する。
- 時間のかかる操作は、子スレッドを生成して実行する。

ユーザー情報保存開始→→→→→→→情報をDBに保存→→→→→→プログラムを終了
↓
↓
↓
→→→→→SMSメッセージを送信

JSのコールバック関数

現状JSは多スレッドを対応していない。ただ、「コールバック関数」手段で同じ効果を実現している。

```
setTimeout(() => {
    console.log("aaaa");
}, 5000);
console.log("bbbb");
```

出力：bbbb→aaaa

setTimeout関数は、指定の時間の後に、なにか関数を実行する。

```
setInterval(() => {
    console.log("aaaa");
}, 5000);
console.log("bbbb");
```

出力：bbbb→aaaa→aaaa→aaaa→aaaa

setTimeout関数は、指定の時間の毎に、なにか関数を実行する。

コールバック関数地獄

```
$.get('/get-user-name',function(name){
    $.get(`/get-user-phone-number?${name}`,function(phone){
        $.get(`/send-msg?${phone}`,function(){
            // メッセージの送信処理
        })
    })
});
```

- サーバからユーザの名前を取得する
- ユーザの名前で、ユーザーの携帯電話を取得する
- ユーザの携帯電話でメッセージを送信する

コールバック関数の問題

- コールバック循環でコールバック関数を読み出し、ソースコードが読みにくいなる。
- 対応方法：Promiseを使用する

Promiseとは

- JSに既存する対象です。
- new Promise()で作成できる
- 下記の実例関数がある
  - then()
  - catch()
- 下記の静態関数がある
  - resolve()
  - all()
  - race()

実例関数

```
let staff = new StaffSalary();
staff.calculateNoTaxTotal();
```

- `staff` は `StaffSalary` の実例です。
- `calculateNoTaxTotal()` は実例から読み出しので、実例関数と呼ばれる

静態関数

```
// StaffSalaryの定義
function StaffSalary(name,basic,position,tax) {
    // 中身省略
}

// StaffSalary対象funという静態関数を追加する
StaffSalary.func = function() { //在构造函数的原型上添加方法
  console.log("This is an static method.");
}

// StaffSalary対象fun1という実例関数を追加する
StaffSalary.prototype.func1 = function() { //在构造函数的原型上添加方法
  console.log("This is an instance method.");
}

// 静態関数の読み出し
StaffSalary.fun();
```

javaの静態関数

```
public class StaffSalary {
    public StaffSalary() {

    }
    public static void test() {

    }
    public void test2() {

    }
}
```

`static` キーワードで定義した関数は静態関数です。

質問

既存のJS対象に、静態関数を思い出しできますか?
ヒント： Math,Object対象など

Promiseの作成

```
let namePromise = new Promise(function(resolve,reject){
    $.get('/get-user-name',function(name){
        resolve(name);
    });
});
```

- Promiseの作成時に、一つ関数をパラメータとして入力する必要です。
- 該当関数は、２つパラメータが必要：resolve,reject。
- resolve,rejectはPromise対象に既存する関数です。
- rejectは必須ではないです。

then 関数

- コールバック関数は「then」関数で設定する
- then関数は２つパラメータが必要です。

```
namePromise.then((value)=>{
    console.log(value);
},(value)=>{
    console.log(value);
});
```

resolve とreject関数

- resolveとrejectの作用は、コールバック関数に発信する。
- Then()パラメータの１番目関数は、resolveに呼び出しされる。
- Then()パラメータの２番目関数は、rejectに呼び出しされる。

例１：

- resolve或いはrejectからthen上の関数の読み出しは非同期です。 `setTimeout()` で実行する

```
let p = new Promise((resolve) => {
    console.log("ccc");
    resolve("aaa");
});

p.then((value) => {
    console.log(value);
});
console.log("bbb");
```

出力は：ccc,bbb,aaa

例２

- resolve或いはreject一回のみ実行する。循環処理要注意、Callbackは一回しか実行しないです。

```
let a =new Promise((resolve, reject) => {
    let total = Math.floor(Math.random() * 100);
    if(total % 2 === 0) {
        resolve(total);
    } else {
        reject(total);
    }
    resolve(8); //無効。
}).then((value) => {
    console.log(`偶数:${value}`);
},(value) => {
    console.log(`奇数:${value}`);
});
```

例 3

- then 関数を読み出さないと、コールバック関数が実施しないです。

```
c = new Promise((resolve, reject) => {
    resolve('aa');
});
console.log("bb");
console.log("cc");
c.then((value) => {console.log(value)});
```

catch関数

- catch関数のパラメータも関数です。
- Promiseパラメータ関数中のソースコードが異常が発生する場合、catchの関数が実行される

```
c = new Promise((resolve, reject) => {
    throw new Error('aa');
}).then((value) => {
    console.log(value);
}).catch((err) => {
    console.log(err);
})
```

resolve静態関数

```
let a = Promise.resolve(1);

// 下記と同じ効果
let b = new Promise((resolve, reject)=>{
    resolve(1);
});
```

all静態関数

- 複数のPromiseのresolve結果を配列に設定して処理する

```
let a = Promise.resolve("aaaa");
let b = Promise.resolve("bbbb");
let c = Promise.resolve("cccc");

Promise.all([a,b,c]).then(([value, value2,value3])=>{
    console.log(value);
    console.log(value2);
    console.log(value3);
},([v1,v2,v3])=>{
    console.log(v1);
    console.log(v2);
    console.log(v3);
});
```

- allで処理するRejectがあれば、結果はRejectの値になる

```
let a = Promise.resolve("aaaa");
let b = Promise.reject("bbbb");
let c = Promise.resolve("cccc");

Promise.all([a,b,c]).then(([value, value2,value3])=>{
    console.log(value);
    console.log(value2);
    console.log(value3);
},([v1,v2,v3])=>{
    console.log(v1);
    console.log(v2);
    console.log(v3);
});
```

race静態関数

- 複数のPromise中に、一つ結果をResovle、その結果を処理する

```
let a = Promise.resolve("aaaa");
let b = Promise.reject("bbbb");
let c = Promise.resolve("cccc");

Promise.race([a,b,c]).then((value)=>{
    console.log(value);
},(v1)=>{
    console.log(v1);
})
```

コールバック関数地獄の回避

```
let namePromise = new Promise((resolve, reject) => {
    $.get('/get-user-name',function(name){
        resolve(name);
    });
}).then((name)=>{
    return new Promise((resolve, reject)=>{
        $.get(`/get-user-phone-number?${name}`,function(phone){
            resolve(phone);
        });
    })
}).then((phone)=>{
   // メッセージの送信処理
})
```

async と　await

```
function getName() {
    return new Promise((resolve, reject) => {
        $.get('/get-user-name',function(name){
            resolve(name);
        });
    });
}

function getPhone(name) {
    return new Promise((resolve, reject)=>{
        $.get(`/get-user-phone-number?${name}`,function(phone){
            resolve(phone);
        });
    })
}
```

```
async function doIt() {
    const name = await getName();
    const phone = await getPhone(name);
    // メール送信
    sendSMS(phone)
}
```

- awaitは、Promiseの返す関数前に使えます。
- awaitは、async定義した関数中しか使えないです。

Promise対象の実現-ご参考

```
function Promise(executor) {
  var _this = this;
  _this.status = 'pending';
  _this.data = undefined ;
  _this.onResolvedCallback = [];
  _this.onRejectedCallback = [];
  //resolve
  function resolve(value){
    setTimeout(function(){
      if(_this.status == 'pending'){
          _this.status = 'resolved';
          _this.data = value;
      }
    for(var i=0;i< _this.onResolvedCallback.length;i++){
      _this. _this.onResolvedCallback[i](value);
    }
   });
  }
```

```
  //reject
  function reject(reason){
    setTimeout(function(){
      if(_this.status == 'pending'){
          _this.status ='rejected';
          _this.data = reason;
      }
      for(var i=0;i< _this.onRejectedCallback.length;i++){
        _this.onRejectedCallback[i](reason);
      }
    });
  }
  try {
      executor(resolve, reject)
  }catch(e){
      reject(e);
  }
}//end
```

宿題

1. ５秒ごとにその時の時間を出力する。 Promiseで実現する。
2. 赤の信号3秒ごとに点滅する，緑の信号が1秒ごとに点滅する，黄色の信号は2秒毎に点滅する一次；Promiseで３つの信号機が交替で点滅する