ES6 PPT1607051: 异步/Async函数

Thursday, November 5, 2015 10:15 AM

异步：

1. JavaScript 下的异步概念，通过回调实现异步，例：

```
myAsyncFunc1(myParam1, function myCallback1() {
  myAsyncFunc2(myParam2, function myCallback2() {
    myAsyncFunc3(myParam3, function myCallback3() {
      // ...
    });
  });
});
```

2. 新的标准化的通过 Promise 进行异步处理方式，允许回调函数的纵向加载，例：

```
myAsyncFunc1(myParam1)
.then(myCallback1)
.then(myCallback2)
.then(myCallback3)
.catch(myErrorCallback);
```

3. 使用 Generator 函数封装异步代码，使之更像同步代码，例：

```
function* myGenerator() {
  var result1 = yield myAsyncWrap(myAsyncFunc1(myParam1), myCallback1);
  var result2 = yield myAsyncWrap(myAsyncFunc2(myParam2), myCallback2);
  var result3 = yield myAsyncWrap(myAsyncFunc3(myParam3), myCallback3);
}
function myAsyncWrap(myAsyncFunc, myCallback, myErrorCallback = function() {}) {
  myAsyncFunc.then(function() {
    myCallback();
    itr.next();
  }).catch(myErrorCallback);
}
var itr = myGenerator();
itr.next();
```

4. JavaScript 下的 Thunk 函数将异步函数的非回调参数与回调参数拆分到两个调用函数传参

```
function Thunk(fn) {
  return function (...args) {
    return function (callback) {
      return fn.call(this, ...args, callback);
    }
  };
};
var $getThunk = Thunk($.get);
getThunk(myURL)(myCallback);
```

5. 使用 Thunk 函数 + Generator 函数重新封装异步代码，例：

```
// 此处仅为针对(...args, callback)形式函数的简单 Thunk 实现，该示例可在 BABEL 在线编辑器下（https://babeljs.io/repl/）运行
function thunkifySimple(fn) {
  return function (...args) {
    return function (callback) {
      return fn.call(this, ...args, callback);
    }
  };
};
function runGeneratorTk(myGenerator) {
  var itr = myGenerator();
  next();
  function next(data) {
    var result = itr.next(data);
    if (result.done) return;
    result.value(next);
  }
}
function* myGeneratorTk() {
  var data = null;
  data = yield $getThunk('https://babeljs.io/favicon-16x16.png');
  console.log('favicon-16x16.png 数据已 get 载入：' + data.length + '字节'); // "favicon-16x16.png 数据已 get 载入：479字节" ->
第一行输出
  data = yield $getThunk('https://babeljs.io/favicon-32x32.png');
  console.log('favicon-32x32.png 数据已 get 载入：' + data.length + '字节'); // "favicon-32x32.png 数据已 get 载入：1037字节" ->
```

```
第二行输出
};
var $getThunk = thunkifySimple($.get);
runGeneratorTk(myGeneratorTk);
```

6. 使用 Generator 函数产出 Promise 对象重新封装异步代码，例：

```
// 仅支持 Promise 对象作为 yield 产出类型，该示例可在 BABEL 在线编辑器下（https://babeljs.io/repl/）运行
function runGeneratorPm(myGenerator) {
  var itr = myGenerator();
  next();
  function next(data) {
    var result = itr.next(data);
    if (result.done) return;
    result.value.then((data) => {
      next(data);
    }, (err) => {
      itr.throw(err);
    });
  }
}
function* myGeneratorPm() {
  var data = null;
  try {
    data = yield $.get('https://babeljs.io/favicon-32x32.png');
    console.log('favicon-32x32.png 数据已 get 载入：' + data.length + '字节'); // "favicon-32x32.png 数据已 get 载入：1037字节" -> 第一行输出
    data = yield $.ajax({url: 'https://babeljs.io/favicon-96x96.png'});
    console.log('favicon-96x96.png 数据已 ajax 载入：' + data.length + '字节'); // "favicon-96x96.png 数据已 ajax 载入：3322字节" -> 第二行输出
  }
  catch(err) {
    console.log('错误：' + err);
  }
};
runGeneratorPm(myGeneratorPm);
```

7. 使用第三方导入 Co 模块，支持 yield 类型 Thunk 或 Promise 一站式解决方案：co(myGeneratorPm) / co(MyGeneratorTk)

ES7 提案之 Async 函数：

1. 语法上一比较就会发现，Async 函数就是将 Generator 函数的 function* 替换成 async function，将 yield 替换成 await，并且内置了执行器：

```
async function myAsyncFuncTypical() {
  var data = null;
  try {
    data = await $.get('https://babeljs.io/favicon-96x96.png');
    console.log('favicon-96x96.png 数据已 get 载入：' + data.length + '字节'); // "favicon-96x96.png 数据已 get 载入：3322字节" -> 第一行输出
    data = await $.ajax('https://babeljs.io/favicon-160x160.png');
    console.log('favicon-160x160.png 数据已 ajax 载入：' + data.length + '字节'); // "favicon-160x160.png 数据已 ajax 载入：5912字节" -> 第二行输出
    data = await $.ajax('http://不存在的地址/');
    console.log('不存在的地址数据已 ajax 载入：' + data.length + '字节'); // 此行不会被执行
    data = await $.ajax('https://babeljs.io/favicon-192x192.png'); // 此行不会被执行
    console.log('favicon-192x192.png 数据已 ajax 载入：' + data.length + '字节'); // 此行不会被执行
  }
  catch(err) {
    console.log('错误：', err); // 错误： {"readyState":0,"status":0,"statusText":"error"} -> 第三行输出
  }
}
myAsyncFuncTypical(); // 内置执行器将直接执行 Async 函数内逻辑
```

2. Async 函数可以 yield 产出 Thunk 或 Promise 以外的数据类型，即等同于同步操作

```
async function myAsyncFuncAnyType() {
  return await '我既不是 Promise 也不是 Thunk';
}
myAsyncFuncAnyType().then((val) => {
  console.log('我说：' + val);
});
```

3. Async 函数返回值是 Promise

```
async function myAsyncFuncChainable1() {
  var data = null;
  data = await $.get('https://babeljs.io/favicon-192x192.png');
  console.log('favicon-192x192.png 数据已 get 载入：' + data.length + '字节'); // "favicon-192x192.png 数据已 get 载入：7470字节" -> 第一行输出
  data = await $.ajax({url: 'https://babeljs.io/favicon-160x160.png'});
  console.log('favicon-160x160.png 数据已 ajax 载入：' + data.length + '字节'); // "favicon-160x160.png 数据已 ajax 载入：5912字节" -> 第二行输出
  return '完成第一阶段';
}
async function myAsyncFuncChainable2() {
  var data = null;
  data = await $.get('https://babeljs.io/favicon-96x96.png');
  console.log('favicon-96x96.png 数据已 get 载入：' + data.length + '字节'); // "favicon-96x96.png 数据已 get 载入：3322字节" -> 第四行输出
  data = await $.ajax({url: 'http://不存在的地址/'});
  console.log('不存在的地址数据已 ajax 载入：' + data.length + '字节'); // 此行不会被执行
  return '完成第二阶段';
}
myAsyncFuncChainable1().then((ret) => {
  console.log('结果1：' + ret); // "结果1：完成第一阶段" -> 第三行输出
  return myAsyncFuncChainable2();
}).then((ret) => {
  console.log('结果2：' + ret); // "结果：完成" -> 第三行输出
  return '感谢使用';
}).catch((err) => {
  console.log('错误：', err); // 错误： {"readyState":0,"status":0,"statusText":"error"} -> 第五行输出
  return '仍然感谢使用';
}).then((ret) => {
  console.log('告别语：' + ret); // "告别语：仍然感谢使用" -> 第六行输出
});
```