promise思路分析

定义解释

一个 Promise 就是一个代表了异步操作最终完成或者失败的对象。大多数人都在使用由其他函数创建并返回的promise。
promise本质上是一个绑定了回调的对象，而不是将回调传进函数内部。

example

使用规约

- 在JavaScript事件队列的当前运行完成之前，回调函数永远不会被调用
- 通过 .then 形式添加的回调函数，甚至都在异步操作完成之后才被添加的函数，都会被调用，如上所示
- 通过多次调用 .then，可以添加多个回调函数，它们会按照插入顺序并且独立运行。

链式调用

一个常见的需求就是连续执行两个或者多个异步操作，这种情况下，每一个后来的操作都在前面的操作执行成功之后，带着上一步操作所返回的结果开始执行。我们可以通过创造一个promise chain来完成这种需求。

```
const promise = doSomething();
const promise2 = promise.then(successCallback, failureCallback);
```

第二个promise不仅代表doSomething()函数的完成，也代表了你传入的 successCallback 或者 failureCallback 的完成，这也可能是其他异步函数返回的promise。这样的话，任何被添加给 promise2 的回调函数都会被排在 successCallback 或 failureCallback 返回的promise后面。基本上，每一个promise代表了链式中另一个异步过程的完成。

在过去，做多重的异步操作，会导致经典的回调地狱：

```
doSomething(function(result) {
  doSomethingElse(result, function(newResult) {
    doThirdThing(newResult, function(finalResult) {
      console.log('Got the final result: ' + finalResult);
    }, failureCallback);
  }, failureCallback);
}, failureCallback);
```

通过现代的函数，我们把回调附加到被返回的promise上代替以往的做法，形成一个promise 链：

```
doSomething().then(function(result) {
  return doSomethingElse(result);
})
.then(function(newResult) {
  return doThirdThing(newResult);
})
.then(function(finalResult) {
  console.log('Got the final result: ' + finalResult);
})
.catch(failureCallback);
```