Promise A+ 实现

promise 相信大家都用过，解决回调地狱，更优雅的写代码。

简易版：

```
'use strict'

/**
 * 思路：
 * promise所有两种状态 resolve, reject
 * pending是未完成状态
 * 1. 从 pending 到 resolve 成功
 * 2. 从 pedding 到 reject 失败
 *
 */

function myPromise(constructor) {
  let self = this;
  self.stauts = 'pending'; // 未完成状态
  self.value = undefined
  self.resaon = undefined

  function resolve(value) {
    if (self.stauts === 'pending') {
      self.value = value
      self.stauts = 'resolved'// 成功
    }
  }

  function reject(resaon) {
    if (self.stauts === 'pending') {
      self.resaon = resaon
      self.resaon = 'rejected'// 失败
    }
  }
  try {
    constructor(resolve, reject)
  } catch(e) {
    reject(e)
  }
}

myPromise.prototype.then = function(onFullfiled, onRejected) {
  let self = this
  switch(self.stauts) {
    case 'resolved':
      onFullfiled(self.value)
      break
    case 'rejected':
      onRejected(self.resaon)
    break
    default:
      break
  }
}
```

测试:

```
// test
let promise = new myPromise(function(resolve, reject) {
  resolve(1)
})

promise.then(function(s) {
  console.log(s)
})
```

遵循Promise A+ 规范:

```
// 定义一个promise构造函数
function MyPromise(resolver) {
  // resolver 必须是一个函数
  if (typeof resolver != "function") {
    throw new TypeError("promise resolver if function");
  }

  // 如果当前对象是Promise,则直接返回
  if (!(this instanceof Promise)) return new Promise(resolver);

  // 当前this
  var self = this;
  // 未完成状态
  self.status = "pending";
  // resolve , reject
  self.data = undefined;
  self.callback = [];
```

原文链接：Promise A+ 实现

```
  function resolve(value) {
    setTimeout(() => {
      if (self.status != "pending") return;
      self.data = value;
      self.status = "resolved";

      for (var i = 0; i < self.callback.length; i++) {
        self.callback[i].onResolved(value);
      }
    });

    // // 当从pedning转为resolve时 --> 成功
    // if (self.status === 'pending') {
    //   self.data = value
    //   self.status = 'resolved'
    // }
  }

  function reject(resaon) {
    setTimeout(() => {
      if (self.status != "pending") return;
      self.data = resaon;
      self.status = "rejected";

      for (var i = 0; i < self.callback.length; i++) {
        self.callback[i].onRejected(resaon);
      }
    });

    // // 当从pending转为rejected时 --> 失败
    // if (self.status === 'pending') {
    //   self.data = resaon
    //   self.status = 'rejected'
    // }
  }

  try {
    executor(resolve, reject);
  } catch (e) {
    reject(e);
  }
}

/**
 * resolve promise
 * @param {*} promise
 * @param {*} x
 * @param {*} resolve
 * @param {*} reject
 */

function resolverPromise(promise, x, resolve, reject) {
  var then;
  var thenCalledOrThrow = false;
  if (promise === x) return reject(new TypeError("chend del promise"));

  if ((x != null && typeof x === "function") || typeof x === "object") {
    try {
      x.then = then;
      if (typeof then === "function") {
        // 如果 resolvePromise 以值 y 为参数被调用，则运行 [[Resolve]](promise, y)
        then.call(
          x,
          function rs(y) {
            if (thenCalledOrThrow) return;
            thenCalledOrThrow = true;
            resolverPromise(promise, y, resolve, reject);
          },
          function rj(r) {
            // 如果 rejectPromise 以据因 r 为参数被调用，则以据因 r 拒绝 promise
            if (thenCalledOrThrow) return;
            thenCalledOrThrow = true;
            return reject(r);
          }
        );
      } else {
        reject(x);
      }
    } catch (e) {
      if (thenCalledOrThrow) return;
      thenCalledOrThrow = true;
      return reject(e);
    }
  } else {
    resolve();
  }
}
```

then 实现

```
MyPromise.prototype.then = function(onResolved, onRejected) {
  // 性能优化
  onResolved =
    typeof onResolved === "function"
      ? onResolved
      : function(value) {
          return value;
        };
  onRejected =
    typeof onRejected === "function"
      ? onRejected
      : function(resaon) {
          return resaon;
        };

  // 保存  this
  var self = this;
  var promise2;

  // 当前状态为 resolved
  if (self.status === "resolved") {
    return (promise2 = new Promise(function(resolve, reject) {
      setTimeout(function() {
        try {
          var x = onResolved(self.data);
          resolverPromise(promise2, x, resolve, reject);
        } catch (e) {
          return reject(e);
        }
      });
    }));
  }

  // 当状态改变为 rejected
  if (self.status === "rejected") {
    return (promise2 = new Promise(function(resolve, reject) {
      setTimeout(function() {
        try {
          var x = onRejected(self.data);
          resolverPromise(promise2, x, resolve, reject);
        } catch (e) {
          return reject(e);
        }
      });
    }));
  }

  // 当前状态 pending
  if (self.status === "pending") {
    return (promise2 = new Promise(function(resolve, reject) {
      self.callback.push({
        onResolved: function(value) {
          try {
            var x = onResolved(self.data);
            resolverPromise(promise2, x, resolve, reject);
          } catch (e) {
            return reject(e);
          }
        },
        onRejected: function(resaon) {
          try {
            var x = onRejected(self.data);
            resolverPromise(promise2, x, resolve, reject);
          } catch (e) {
            return reject(e);
          }
        }
      });
    }));
  }
};
```

测试

```
var p = new MyPromise(function(resolve, reject) {
  resolve(1);
});
p.then(function (x) {
  console.log(x)
})
```