```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>自定义Promise</title>
</head>
<body>

<script type="text/javascript">

// 自定义 Promise
(function (window) {

  // Promise 构造函数
  // excutor: 内部同步执行的函数 (resolve, reject) => {}
  function Promise(excutor) {
    const self = this;
    self.status = "pending";  // 状态值，初始状态为 pending，成功了变为resolved，失败了变为 rejected
    self.data = undefined;  // 用来保存成功 value 或失败 reason 的属性
    self.callbacks = [];  // 用来保存所有待调用的包含 onResolved 和 onRejected 回调函数的对象的数组

    // 异步处理成功后应该调用的函数
    // value: 将交给 onResolve()的成功数据
    function resolve(value) {
      if (self.status !== "pending") {
        return;
      }
      //立即更新状态，保存数据
      self.status = "resolved";
      self.data ="自定义" + value;
      // 异步调用所有待处理的 onResolved 成功回调函数
      if (self.callbacks.length > 0) {
        setTimeout(() => {
          self.callbacks.forEach(fn => {
            fn.onResolved(value);
          });
        });
      }
    }

    // 异步处理失败后应该调用的函数
    // reason: 将交给 onRejected()的失败数据
    function reject(reason) {
      if (self.status !== "pending") {
        return;
      }
      //立即更新状态，保存数据
      self.status = "rejected";
      self.data = "自定义" +  reason;
```

```
      // 异步调用所有待处理的 onRejected 回调函数
      if (self.callbacks.length > 0) {
        setTimeout(() => {
          self.callbacks.forEach(fn => {
            fn.onRejected(reason);
          });
        });
      }
    }

    try {
      // 立即同步调用 excutor()处理
      excutor(resolve, reject);
    } catch (error) {
      // 如果出了异常，直接失败
      reject(error);
    }
  }

  // 为 promise 指定成功/失败的回调函数
  // 函数的返回值是一个新的 promise 对象
  Promise.prototype.then = function (onResolved, onRejected) {
    const self = this;

    // 如果 onResolved/onRejected 不是函数，可它指定一个默认的函数
    // 指定返回的 promise 为一个成功状态，结果值为 value
    onResolved = typeof onResolved === "function" ? onResolved : value =>
value;
    // 指定返回的 promise 为一个失败状态，结果值为 reason
    onRejected = typeof onRejected === "function" ? onRejected : reason => {
throw reason; };

    // 返回一个新的 promise 对象
    return new Promise((resolve, reject) => {
      //专门抽取的用来处理 promise 成功/失败结果的函数
      // callback: 成功/失败的回调函数
      function handle(callback) {
        // 1. 抛出异常 ===> 返回的 promise 变为 rejected
        try {
          const x = callback(self.data);
          // 2. 返回一个新的 promise ===> 得到新的 promise 的结果值作为返回的
promise 的结果值
          if (x instanceof Promise) {
            // 一旦 x 成功了，resolve(value)，一旦 x失败了: reject(reason)
            x.then(resolve, reject);
          } else {
            // 3. 返回一个一般值(undefined) ===> 将这个值作为返回的 promise 的成功
值
            resolve(x);
          }
        } catch (error) {
          reject(error);
        }
```

```
      }
      if (self.status === "resolved") {
        // 当前 promise 已经成功了
        setTimeout(() => {
          handle(onResolved);
        });
      } else if (self.status === "rejected") {
        // 当前 promise 已经失败了
        setTimeout(() => {
          handle(onRejected);
        });
      } else {
        // 当前 promise 还未确定 pending
        // 将 onResolved 和 onRejected 保存起来
        self.callbacks.push({
          onResolved(value) {
            handle(onResolved);
          },
          onRejected(reason) {
            handle(onRejected);
          }
        });
      }
    });
  }

  // 为 promise 指定失败的回调函数
  // 是 then(null, onRejected)的语法糖
  Promise.prototype.catch = function (onRejected) {
    return this.then(null, onRejected);
  }

  // 返回一个指定了成功 value 的 promise 对象
  //value: 一般数据或 promise
  Promise.resolve = function (value) {
    return new Promise((resolve, reject) => {
      if (value instanceof Promise) {
        value.then(resolve, reject);
      } else {
        resolve(value);
      }
    });
  }

  // 返回一个指定了失败 reason 的 promise 对象
  // reason: 一般数据/error
  Promise.reject = function (reason) {
    return new Promise((resolve, reject) => {
      reject(reason);
    });
  }

  // 返回一个 promise，只有 promises 中所有 promise 都成功时，才最终成功，只要有一
```

个失败就直接失败

```
  Promise.all = function (promises) {
    // 返回一个新的 promise
    return new Promise((resolve, reject) => {
      // 已成功的数量
      let resolvedCount = 0;
      // 待处理的 promises 数组的长度
      const promisesLength = promises.length;
      // 准备一个保存成功值的数组
      const values = new Array(promisesLength);
      // 遍历每个待处理的 promise
      for (let i = 0; i < promisesLength; i++){
        // promises 中元素可能不是一个数组，需要用 resolve 包装一下
        Promise.resolve(promises[i]).then(
          value => {
            // 成功当前 promise 成功的值到对应的下标
            values[i] = value;
            // 成功的数量加 1
            resolvedCount++;
            // 一旦全部成功
            if (resolvedCount === promisesLength) {
              // 将所有成功值的数组作为返回 promise 对象的成功结果值
              resolve(values);
            }
          },
          reason => {
            // 一旦有一个promise 产生了失败结果值，将其作为返回promise 对象的失败结果值
            reject(reason);
          }
        );
      }
    });
  }

  // 返回一个 promise， 一旦某个 promise 解决或拒绝， 返回的 promise 就会解决或拒绝。
  Promise.race = function (promises) {
    // 返回新的 promise 对象
    return new Promise((resolve, reject) => {
      //遍历所有 promise
      for (let i = 0; i < promises.length; i++){
        Promise.resolve(promises[i]).then(
          value => {
            //只要有一个成功了，返回的 promise 就成功了
            resolve(value);
          },
          reason => {
            //只要有一个失败了，返回的结果就失败了
            reject(reason);
          }
        );
      }
```

```
    });
  }

  //返回一个延迟指定时间才确定结果的 promise 对象
  Promise.resolveDelay = function (value, time) {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        if (value instanceof Promise) {
          //如果 value 是一个 promise, 取这个promise 的结果值作为返回的 promise 的
结果值
          //如果 value 成功, 调用resolve(val), 如果 value 失败了, 调用
reject(reason)
          value.then(resolve, reject);
        } else {
          resolve(value);
        }
      }, time);
    });
  }

  //返回一个延迟指定时间才失败的 Promise 对象。
  Promise.rejectDelay = function (reason, time){
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        reject(reason);
      }, time);
    });
  }

  // 暴露构造函数
  window.Promise = Promise;

})(window);

const p = new Promise((resolve, reject) => {
  setTimeout(() => {
    if (Date.now() % 2 === 0) {
      resolve(66)
    } else {
      reject(99)
    }
  }, 100);
});
p.then((value) => {
    // 成功的回调函数onResolved,得到成的value
    console.log("成功的value: " + value);
  }, (reason) => {
    // 失败的回调函数onRejected,得到失败的reason
    console.log("失败的reason: " + reason);
});

</script>
</body>
</html>
```