# Promise

### Promise是什么？

如何实现Promise （Promise A+）？

class MyPromise {  
 constructor(fn) {  
 this.callbacks = [];  
 this.state = "PENDING";  
 this.value = null;  
  
 fn(this.\_resolve.bind(this), this.\_reject.bind(this));  
 }  
  
 then(onFulfilled, onRejected) {  
 return new MyPromise((resolve, reject) =>  
 this.\_handle({  
 onFulfilled: onFulfilled || null,  
 onRejected: onRejected || null,  
 resolve,  
 reject,  
 })  
 );  
 }  
  
 catch(onRejected) {  
 return this.then(null, onRejected);  
 }  
  
 \_handle(callback) {  
 if (this.state === "PENDING") {  
 this.callbacks.push(callback);  
  
 return;  
 }  
  
 let cb =  
 this.state === "FULFILLED" ? callback.onFulfilled : callback.onRejected;  
 if (!cb) {  
 cb = this.state === "FULFILLED" ? callback.resolve : callback.reject;  
 cb(this.value);  
  
 return;  
 }  
  
 let ret;  
  
 try {  
 ret = cb(this.value);  
 cb = this.state === "FULFILLED" ? callback.resolve : callback.reject;  
 } catch (error) {  
 ret = error;  
 cb = callback.reject;  
 } finally {  
 cb(ret);  
 }  
 }  
  
 \_resolve(value) {  
 if (value && (typeof value === "object" || typeof value === "function")) {  
 let then = value.then;  
  
 if (typeof then === "function") {  
 then.call(value, this.\_resolve.bind(this), this.\_reject.bind(this));  
  
 return;  
 }  
 }  
  
 this.state === "FULFILLED";  
 this.value = value;  
 this.callbacks.forEach((fn) => this.\_handle(fn));  
 }  
  
 \_reject(error) {  
 this.state === "REJECTED";  
 this.value = error;  
 this.callbacks.forEach((fn) => this.\_handle(fn));  
 }  
}  
  
const p1 = new Promise(function (resolve, reject) {  
 setTimeout(() => reject(new Error("fail")), 3000);  
});  
  
const p2 = new Promise(function (resolve, reject) {  
 setTimeout(() => resolve(p1), 1000);  
});  
  
p2.then((result) => console.log(result)).catch((error) => console.log(error));

### 聊聊状态机和Promise的关系？

### 历史上异步代码的处理方式有哪些？

### async await和Promise的区别？

### 如何理解generater？

### 如何理解thunk与co函数？

### js脚本加载问题，async、defer问题

* 如果依赖其他脚本和 DOM 结果，使用 defer
* 如果与 DOM 和其他脚本依赖不强时，使用 async

下面代码的执行顺序

Promise.resolve().then(() => {  
 console.log(0);  
 return Promise.resolve(4);  
}).then((res) => {  
 console.log(res)  
})  
  
Promise.resolve().then(() => {  
 console.log(1);  
}).then(() => {  
 console.log(2);  
}).then(() => {  
 console.log(3);  
}).then(() => {  
 console.log(5);  
}).then(() =>{  
 console.log(6);  
})

### 解释一下async/await的运行机制