promise 实现

作者：gmw-zjw
原文链接：https://ld246.com/article/1567518648666
来源网站：链滴

思路：

(1) 一个promise的当前状态只能是pending、fulfilled和rejected三种之一。状态改变只能是pendin到fulfilled或者pending到rejected。状态改变不可逆。

(2) promise的then方法接收两个可选参数，表示该promise状态改变时的回调(promise.then(onFulfilled, onRejected))。then方法返回一个promise。then 方法可以被同一个 promise 调用多次。

```
// 示例代码
var Promise = (function () {
    // promise 实例 构造函数
    function Promise(resolver) {

        // resolver 必须是一个函数
        if (typeof resolver != 'function') {
            throw new TypeError('Promise resolver is function');
        }

        if (!(this instanceof Promise)) return new Promise(resolver);

        var self = this;
        self.status = 'pending'; // promise 当前状态
        self.data = undefined; // promisr 的值
        // self.onResolveCallback = []; // resolve
        // self.onRejectCallback = []; // reject
        self.callbacks = []; // resolve and reject

        // resolve
        function resolve(value) {
            setTimeout(() => {
                // 当前状态为pending时
                if (self.status !== 'pending') {
                    return;
```

```
            }
            self.status = 'resolved';
            self.sata = value;

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

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

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


    // 错误拦截处理
    try {
        executor(resolve, reject);
    } catch(err) {
        reject(err);
    }

}

function resolvePromise(promise, x, resolve, reject) {
    var then;
    var thenCalledOrThrow = false;

    if (promise === x) {
        return reject(new TypeError('Chaining cycle deleted for promise'));
    }

    if ((x != null) && (typeof x === 'object') || (typeof x === 'function')) {
        try {
            then = x.then;
            if (typeof then === 'function') {
                then.call(x, function rs(y) {
                    if (thenCalledOrThrow) return;
                    thenCalledOrThrow = true;
                    return resolvePromise(promise, y, resolve, reject);
                }, function rj(r) {
                    if (thenCalledOrThrow) return;
                    thenCalledOrThrow = true;
```

```
                    return reject(r);
                });
            } else {
                return resolve(x);
            }
        } catch(err) {
            if (thenCalledOrThrow) return;
            thenCalledOrThrow = true;
            return reject(err);
        }
    } else {
        return resolve(x);
    }
}

// 原型方法
// then
Promise.prototype.then = function (onResolved, onRejected) {
    // 性能处理
    onResolved = typeof onResolved === 'function' ? onResolved : function (value) { return
alue };
    onRejected = typeof onRejected === 'function' ? onRejected : function (resaon) { throw
esaon };

    var self = this;
    var promise2;


    // 将当前的状态变为resolved
    if (self.status === 'resolved') {
        return promise2 = new Promise(function (resolve, reject) {
            setTimeout(function () {
                try {
                    var x = onResolved(self.data);
                    resolvePromise(promise2, x, resolve, reject);
                } catch(err) {
                    return reject(err);
                }
            });
        });
    };

    // 将当前状态变为rejected
    if (self.status === 'rejected') {
        return promise2 = new Promise(function (resolve, reject) {
            setTimeout(function() {
                try {
                    var x = onRejected(self.data);
                    resolvePromise(promise2, x, resolve, reject);
                } catch(err) {
                    return reject(err);
                }
            })
        });
```

```
    };

    // promise状态为pending
    // 需要等待promise的状态完成
    if (self.status === 'pending') {
        return promise2 = new Promise(function (resolve, reject) {
            self.callbacks.push({
                onResolved: function (value) {
                    try {
                        var x = onResolved(value);
                        resolvePromise(promise2, x, resolve, reject);
                    } catch (err) {
                        return reject(err);
                    }
                },
                onRejected: function (resaon) {
                    try {
                        var x = onRejected(resaon);
                        resolvePromise(promise2, x, resolve, reject);
                    } catch (err) {
                        return reject(err);
                    }
                }
            });
        });
    }
}

// 将当前promise的值传递
Promise.prototype.valueOf = function () {
    return this.data;
}

// catch
Promise.prototype.catch = function (onRejected) {
    return this.then(null, onRejected);
}

// finally
Promise.prototype.finally = function (fn) {
    return this.then(function (value) {
        setTimeout(fn);
        return value;
    }, function (resaon) {
        setTimeout(fn);
        throw resaon;
    })
};

Promise.prototype.spread = function (fn, onRejected) {
    return this.then(function (values) {
        return fn.apply(null, values);
    }, onRejected);
};
```

```
Promise.prototype.inject = function (fn, onRejected) {
    return this.then(function (v) {
        return fn.apply(null, fn.toString().match(/\((.*?)\)/)[1].split(',').map(function (key) {
            return v[key];
        }))
    }, onRejected);
};

Promise.prototype.delay = function (duration) {
    return this.then(function (value) {
        return new Promise(function (resolve, reject) {
            setTimeout(function () {
                resolve(value);
            }, duration);
        });
    }, function (resaon) {
        return new Promise(function (resolve, reject) {
            setTimeout(function () {
                reject(resaon);
            }, duration);
        });
    });
};

// 静态方法
Promise.all = function (promises) {
    return new Promise(function (resolve, reject) {
        var resolvedCounter = 0;
        var promisesNum = promises.length;
        var resolveValues = new Array(promisesNum);
        for (var i = 0; i < promisesNum; i++) {
            (function (i) {
                Promise.resolve(function (promises) {
                    resolvedCounter ++;
                    resolveValues[i] = value;
                    if (resolvedCounter == promisesNum) {
                        return resolve(resolveValues);
                    }
                }, function (resaon) {
                    reject(resaon);
                })
            })(i);
        };
    });
}


Promise.resolve = function (value) {
    return new Promise(function (resolve, reject) {
        resolvePromise(promise, value, resolve, reject);
    });
};
```

```
    Promise.reject = function (resaon) {
        return new Promise(function (resolve, reject) {
            reject(resaon);
        });
    }

    Promise.race = function (promises) {
        return new Promise(function (resolve, reject) {
            for (var i = 0; i < promises.length; i++) {
                Promise.resolve(function () {
                    promises[i]
                }).then(function (value) {
                    return resolve(value);
                }, function (resaon) {
                    return reject(resaon);
                });
            }
        });
    };

    Promise.fcall = function (fn) {
        return Promise.resolve().then(fn);
    }

    Promise.done = Promise.stop = function () {
        return new Promise(function () {});
    }

    Promise.deferred = Promise.defer = function () {
        var dfd = {};
        dfd.promise = new Promise(function (resolve, reject) {
            dfd.resolve = resolve;
            dfd.reject = reject;
        });
        return dfd;
    }

    try {
        module.exports = Promise;
    } catch(err) { }

    return Promise;
});
```