Promise对象

Promise是JavaScript中一种用于处理异步操作的对象。它用于表示一个可能尚未完成并且需要等待结果的操作，并且可以使用Promise实例来处理操作完成后的结果。

一个Promise对象有三种状态：pending（等待中）、fulfilled（已成功）或rejected（已失败）。当Promise对象处于pending状态时，它表示尚未完成，但可能会在未来某个时间完成。如果Promise成功地解决了其值，它将进入已成功状态，并将返回一个值。如果Promise无法解决其值，则会进入已失败状态，并返回一个错误。

Promise可以使用`then()`和`catch()`方法来处理操作完成后的结果。`then()`方法用于处理已成功的Promise对象，它接受一个回调函数作为参数，当Promise对象成功时会调用该函数并传递解决的值。`catch()`方法用于处理已失败的Promise对象，它接受一个回调函数作为参数，当Promise对象被拒绝时会调用该函数并传递拒绝的原因。

Promise可以嵌套使用，以便处理多个异步操作。在这种情况下，可以使用`Promise.all()`方法来等待所有Promise对象完成，并在它们都完成后执行一些操作。

使用Promise可以更好地处理异步操作，例如网络请求，文件读取等。它避免了回调地狱（callback hell）的问题，使得代码更加容易理解和维护。

基本语法

```
// 创建一个返回Promise对象的函数
function fetchData() {
    // 创建一个新的Promise对象
    return new Promise(function (resolve, reject) {
        // 模拟异步操作，例如加载数据
        setTimeout(function () {
            var data = {code: 1000, data: {name: 'John', age: 30}};

            if (data.code === 1001) {
                // 如果成功，则解决Promise并返回数据
                resolve(data);
            } else {
                // 如果出现错误，则拒绝Promise并返回错误对象
                reject(new Error('Failed to load data'));
            }
        }, 2000);
    });
}

// 调用fetchData()函数并处理Promise对象的结果
fetchData().then(function (data) {
    // 如果Promise对象成功解决，执行这里的代码
    console.log('Data loaded:', data);
}).catch(function (error) {
    // 如果Promise对象被拒绝，执行这里的代码
    console.log('Error loading data:', error);
});
```

回调地狱

```
// 使用回调函数处理异步操作
function getData(callback) {
    // 异步操作
    setTimeout(function () {
        var data = {name: 'John', age: 30};
        callback(data);
        // callback(new Error('Failed to load data'));
    }, 2000);
}

// 另一个使用回调函数处理异步操作的函数
function getUserDetails(name, callback) {
    // 异步操作
    setTimeout(function () {
        var details = {name: name, email: 'john@example.com'};
        callback(details);
        // callback(new Error('Failed to load user details'));
    }, 2000);
}

// 回调函数c1
function c1(data) {
    // 第一个异步操作完成后执行此处的代码
    console.log('Data loaded:', data);
    getUserDetails(data.name, c2);
}

function c2(details) {
    // 第二个异步操作完成后执行此处的代码
    console.log('User details:', details);
}

// 使用回调地狱
getData(c1);
```

```
  // 使用Promise对象处理异步操作
  function getData() {
      return new Promise(function (resolve, reject) {
          // 异步操作
          setTimeout(function () {
              var data = {name: 'John', age: 30};
              resolve(data);
              // reject(new Error('Failed to load data'));
          }, 2000);
      });
  }

  // 另一个使用Promise对象处理异步操作的函数
  function getUserDetails(name) {
      return new Promise(function (resolve, reject) {
          // 异步操作
          setTimeout(function () {
```

```
            var details = {name: name, email: 'john@example.com'};
            resolve(details);
            // reject(new Error('Failed to load user details'));
        }, 2000);
    });
}

// 使用Promise对象避免回调地狱
getData()
    .then(function (data) {
        // 第一个异步操作完成后执行此处的代码
        console.log('Data loaded:', data);
        return getUserDetails(data.name);
    })
    .then(function (details) {
        // 第二个异步操作完成后执行此处的代码
        console.log('User details:', details);
    })
    .catch(function (error) {
        // 如果发生错误，则执行此处的代码
        console.log('Error:', error);
    });
```

Promise与ajax请求

```
// 创建一个返回Promise对象的函数，该函数使用Ajax发送请求并在响应可用时解决Promise对象
function fetchData(url) {
    return new Promise(function (resolve, reject) {
        var xhr = new XMLHttpRequest();
        xhr.open('GET', url);
        xhr.onload = function () {
            if (xhr.status === 200) {
                resolve(xhr.responseText);
            } else {
                reject(Error(xhr.statusText));
            }
        };
        xhr.onerror = function () {
            reject(Error('Network Error'));
        };
        xhr.send();
    });
}

// 调用fetchData()函数并处理Promise对象的结果
fetchData('https://v0.yiketianqi.com/api?
unescape=1&version=v9&appid=47284135&appsecret=jlmX3A6s').then(function
(response) {
    // 如果Promise对象成功解决，执行这里的代码
    console.log('Data loaded:', response);
}).catch(function (error) {
    // 如果Promise对象被拒绝，执行这里的代码
    console.log('Error loading data:', error);
});
```

#