Promise及其语法糖async和await

更新时间：2019-07-17 13:38:55

> 每个人的生命都是一只小船，理想是小船的风帆。
>
> ——张海迪

TS 在 1.6 版本实验性地支持了 async 函数。在过去的 JavaScript 当中，如果想保证代码的执行顺序，需要使用回调函数，当需要执行的步骤多了时就会陷入当说的“回调地狱”。自从 ES6 增加了 Promise 之后，状况有了缓解，我们先来看个例子，一个简单的多个 ajax 请求的例子：

```
ajax.post( // 这里你可以先忽略ajax的定义，他的post方法用来发送一个post请求
  "/login", // 第一个参数时要请求的url
  {
    data: {
      user_name: "lison",
      password: "xxxxx"
    }
  }, // 第二个参数是这个请求要携带的参数
  function(res) {
    var user_id = res.data.user_id;
    ajax.post( // 这里在/login接口成功返回数据后，再调用一个/user_roles接口，用来获取该登录用户的角色信息
      "/user_roles",
      {
        data: {
          user_id: user_id
        }
      },
      function(res) {
        var role = res.data.role;
        console.log(role);
      }
    );
  } // 第三个参数是接口响应之后的回调函数
);
```

在这个例子中，我们先调用登录的接口发送用户名和密码，然后服务端进行校验之后返回这个用户的一些信息，然后我们可以从信息中拿到用户 id 去获取它的角色用于权限控制。这个过程是有先后顺序的，必须先登录后获取角色，为了保证这个顺序，在过去要使用回调函数，当然一些库也支持链式调用。再来看下使用 ES6 的 Promise 需要怎么写：

```
const loginReq = ({ user_name, password }) => { // 封装一个loginReq函数，用来返回一个Promise，用来调用/login接口
  return new Promise((resolve, reject) => { // Promise接收一个回调函数参数，这个函数有两个参数，两个参数都是回调函数
    ajax.post(
      "/login",
      {
        user_name,
        password
      },
      res => {
        resolve(res); // 第一个参数resolve用来在执行成功后调用，传给他的参数，可以在这个promise的then函数参数中获取到
      },
      error => {
        reject(error); // 第二个参数reject用来在执行出现错误后调用，传给他的错误信息，可以在这个promise的catch函数参数中获取到
      }
    );
  });
};
const getRolesReq = ({ user_id }) => { // 封装一个getRolesReq函数，用来返回一个Promise，用来调用/user_roles接口
  return new Promise((resolve, reject) => {
    ajax.post(
      "/user_roles",
      {
        data: {
          user_id
        }
      },
      res => {
        resolve(res);
      },
      error => {
        reject(error);
      }
    );
  });
};
loginReq({ user_name: "lison", password: "xxxxx" }).then(res => { // 这里在调用loginReq函数后返回一个Promise，在内部当执行到resolve的地方时，这里的then的回调函数就会执行
  getRolesReq({ user_id: res.data.user_id }).then(res => {
    console.log(res.data.role);
  });
});
```

这看起来代码变长了，但是当我们搭配使用诸如 Axios 这类 ajax 请求库和 ES6 语法时，对于一些复用性高的接口调用能够起到很好的封装作用，而且使用起来较为简洁。

ES7 中增加了 async 和 await 的规范，它们其实是 Promise 的语法糖。TypeScript 在 1.6 支持了 async 和 await，下面我们通过 setTimeout 来实现异步过程，看下在 TypeScript 中如何使用 async 和 await：

```ts
interface Res { // 我们定义一个接口，用来定义接口返回结果的结构
  data: {
    [key: string]: any;
  };
}
namespace axios { // 现在我们来定义一个命名空间，用来模拟axios实现接口调用
  export function post(url: string, config: object): Promise<Res> { // 返回值类型是一个Promise，resolve传的参数的类型是Res
    return new Promise((resolve, reject) => { // 然后这里返回一个Promise
      setTimeout(() => { // 通过setTimeout实现异步效果
        let res: Res = { data: {} };
        if (url === "/login") res.data.user_id = 111; // 我们这里通过简单判断，来模拟调用不同接口返回不同数据的效果
        else res.data.role = "admin";
        console.log(2);
        resolve(res); // 在这里传入res结果
      }, 1000);
    });
  }
}
interface Info {
  user_name: string;
  password: string;
}
async function loginReq({ user_name, password }: Info) { // 这里使用async关键字修饰这个函数，那么他内部就可以包含异步逻辑了
  try {
    console.log(1);
    const res = await axios.post("/login", { // 这里调用/login接口
      data: {
        user_name,
        password
      }
    });
    console.log(3);
    return res;
  } catch (error) {
    throw new Error(error);
  }
}
async function getRoleReq(user_id: number) {
  try {
    const res = await axios.post("/user_roles", {
      data: {
        user_id
      }
    });
    return res;
  } catch (error) {
    throw new Error(error);
  }
}
loginReq({ user_name: "lison", password: "123" }).then(res => {
  const {
    data: { user_id }
  } = res;
  getRoleReq(user_id).then(res => {
    const {
      data: { role }
    } = res;
    console.log(role);
  });
});
```

这个例子中用到了很多我们前面学习到的知识，可以帮大家进行复习和实践。

首先我们定义一个命名空间 axios，定义它用来简单模拟 axios 这个 ajax 请求库。我们在命名空间内定义一个 post 方法，第一个参数是要请求的 url，第二个参数是一些配置，这里我们只是定义一个参数，不做具体参数的处理和判断。这个 post 方法返回一个 Promise，这和 axios 库是一样的，使用 setTimeout 来模拟 ajax 请求的异步行为和延迟，当 1 秒后调用 resolve 回调函数，并根据调用的 url 返回不同的结果。post 方法返回的是一个 Promise，所以这个函数返回类型我们使用 TypeScript 内置的条件类型 `Promise<T>` 来指定返回类型，这个 T 的类型就是在 resolve 回调函数中返回的值的类型，我们这里返回的值为一个包含 data 属性的对象，所以我们定义一个接口 Res。

接下来要定义两个发起 ajax 请求的函数了，这里我们使用 async/await 来定义这两个函数。先来看 loginReq 函数。我们在 function 关键字前加 `async` 表明这是一个异步函数，然后就可以在它的函数体内使用 `await` 关键字来让异步代码同步执行了。如果不使用 await，我们可以通过 `.then()` 拿到 axios.post 的结果并且进行后面的操作。我们这里在 axios.post 方法调用前面加上 `await`，这样就可以让这个异步函数同步返回结果，它的返回值就是 Promise 中 resolve 回调函数传入的实际参数。我们在使用 Promise 时，可以使用 `.catch(error => {})` 来捕获异常拿到错误信息，如果使用 await，需要使用 try catch 来捕获异常。

我们再定义一个 getRoleReq 函数来发起获取用户角色的请求，这个请求依赖登录请求返回的用户 id，形式和 loginReq 函数差不多。使用 async/await 要注意，await 只能出现在使用 async 修饰的函数或方法体内。

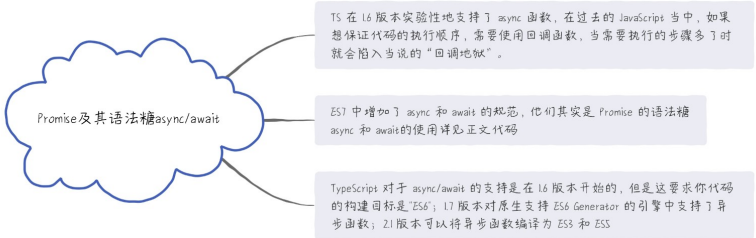
最后我们调用这两个函数，还是使用.then 的方式，这样要比使用 async/await 的形式简单些。这里我们在几个地方打印几个标记，让大家看下执行顺序，可以看到，打印出来的数字，是按 1->2->3 的顺序打印出来的，这是因为代码执行到 console.log(1)后会等 await 修饰的异步代码执行完，才会往后执行，所以 3 在 2 后面执行。

TypeScript 对于 async/await 的支持是在 1.6 版本开始的，但是这要求你代码的构建目标是"ES6"；1.7 版本对原生支持 ES6 Generator 的引擎中支持了异步函数；2.1 版本可以将异步函数编译为 ES3 和 ES5。

本节小结

本小节我们学习了如何使用Promise来保证异步代码的执行顺序，通常我们在调用多个接口，后面接口依赖前面接口返回的数据的时候会用到；还有就是比如confirm弹窗这种，需要在用户点击了“确定”或者“取消”之后才能执行一些逻辑，这种也适用。我们还学习了Promise的语法糖async/await，使用这种语法更为形象清晰，但是不好的地方在于需要使用 `try ... catch` 来获取原本Promise使用catch获取的错误信息。

本章到这里就结束了，下一章我们将学习《项目配置及书写声明文件》，更加偏实战了，所以前面这些基础知识一定要学扎实了，要经常复习运用下，不过我们后面的课程也会对前面部分知识进行巩固的。



混入，兼顾值和类型的合并操作
逐条来看tsconfig.json配置