在线实验，请到PC端体验

一、实验介绍

1.1 实验内容

本课程编写一个Node.js端口扫描模块，主要用到Node.js的net模块，同时，我们会使用一些ES6的新语法，以前的一些Node.js课程已经介绍过一些ES6语法，这个实验主要会用到Promise。

1.2 实验知识点

- Node.js net 模块

1.3 实验环境

- Node.js 6.x

1.4 适合人群

本课程难度为一般，属于初级级别课程，适合具有 JavaScript 基础的用户学习 Node.js net 模块。

二、开发准备

本课程主要用到 net 模块，net 模块包含如下内容：

```
Class: net.Server
Class: net.Socket
net.connect(options[, connectListener])
net.connect(path[, connectListener])
net.connect(port[, host][, connectListener])
net.createConnection(options[, connectListener])
net.createConnection(path[, connectListener])
net.createConnection(port[, host][, connectListener])
net.createServer([options][, connectionListener])
net.isIP()
net.isIPv4()
net.isIPv6()
```

`net.createServer()` 用于创建socket服务，返回 `net.Server` 的实例。 `net.connection()` 和 `net.createConnection()` 是工厂函数，返回 `net.Socket` 实例并自动连接到传入参数指定的socket。 `net.isIP()` 、 `net.isIPv4()` 及 `net.isIPv6()` 用于检测IP。如：

创建一个socket服务：

```
// demoServer.js

const net = require('net');

const server = net.createServer((c) => {
    // 'connection' listener
    console.log('client connected');
    c.on('end', () => {
        console.log('client disconnected');
    });
    c.write('hello\r\n');
    c.pipe(c);
});
server.on('error', (err) => {
    throw err;
});
server.listen(8124, () => {
    console.log('server bound');
});
```

连接上面创建的**socket**服务：

```
// demoClient.js

const net = require('net');

const client = net.connect({port: 8124}, () => {
    // 'connect' listener
    console.log('connected to server!');
    client.write('world!\r\n');
});
client.on('data', (data) => {
    console.log(data.toString());
    client.end();
});
client.on('end', () => {
    console.log('disconnected from server');
});
```

三、实验步骤

3.1 编写端口扫描模块

既然是端口扫描，我们就不需要创建**socket**服务了，直接去连接需要扫描的端口，看能否连接上，能连上说明此端口是打开的，否则此端口没有打开。

新建 `scanPorts.js` 文件，代码如下：

```
'use strict';

 // 引入 net 模块
const net = require('net');
// 进度条模块
// 此模块需要安装：npm install progress
// 用于显示扫描端口完成进度
const ProgressBar = require('progress');

 /**
  * 端口扫描函数
  *
  * @param host {String} 扫描端口的IP/URL地址
  * @param start {Number} 起始端口
  * @param end {Number} 结束端口
  * @return {Promise} 返回一个Promise对象
  */
function checkPorts(host, start, end) {
    // 返回Promise
    return new Promise((resolve, reject) => {
        let counts = end - start + 1; // 需要扫描的IP数量
        let ports = []; // 保存可连接的IP

        // 创建进度条
        let bar = new ProgressBar(' scanning [:bar] :percent :etas', {
            complete: '=',
            incomplete: ' ',
            width: 50,
            total: counts,
        });

        // 循环扫描所有IP
        for (let i = start; i <= end; ++i) {
            // 使用 net.connect() 尝试连接端口
            let check = net.connect({
                host: host,
                port: i,
            }, () => {
                // 连接成功，表示此端口是开发的
                // 保存此端口
                ports.push(i);
                // 检测完毕，断开此连接
                check.destroy();
            });

            check.on('close', () => {
                // check.destroy() 会触发 close 事件
                // 尝试连接端口也会触发 close 事件断开连接

                // 每断开一个连接，说明就检测完成了一个端口
                counts--;

                // 显示进度条
                bar.tick(1);

                // 此时检测完了所以端口
                if (counts === 0) {
                    if (ports.length) {
                        resolve(ports);
                    } else {
                        reject('no port is open');
                    }
                }
            });

            check.on('error', (err) => {
                // 端口未开发时，连接会失败
                // 此时会触发 error 事件
                // 然后会触发 close 事件
            });
        }
    });
}

/**
 * 导出端口扫描包装函数
 *
```

```
 * @param host {String} 扫描端口的IP/URL地址
 * @param start {Number} 起始端口
 * @param end {Number} 结束端口
 * @param callback {function} 回调函数
 */
module.exports = (host, start, end, callback) => {
    // 检测参数
    // 如果只传了三个参数，并且end是一个函数
    // 那么自动作参数调换
    if (typeof end === 'function' && callback === undefined ) {
        callback = end;
        end = start;
    }

    // 调用函数扫描端口
    checkPorts(host, start, end).then((ports) => {
        callback(ports);
    }).catch((err) => {
        console.log(err);
    });
}
```

使用Promise可以避免多层回调函数，使得代码更加清晰，非常方便。

下面我们来测试一下吧，创建 test.js 文件：

```
'use strict';

const scanPorts = require('./scanPorts');

scanPorts('127.0.0.1', 1, 65535, (ports) => {
    console.log('open ports: ', ports);
});
```

运行代码：

```
$ node test.js
```

可以看到终端打印出了开放的端口。

一个简单的端口扫描程序就实现了。

四、总结

本课程基于 Node.js 的 net 模块编写一个Node.js端口扫描模块，同时通过这个简单的项目学习 ES6 的新语法。

课程教师

hojas

共发布过**2**门课程

查看老师的所有课程 > (/teacher/23105)

前置课程

Node.js 教程 (/courses/44)

进阶课程

Node.js 经典项目实战 (/courses/455)