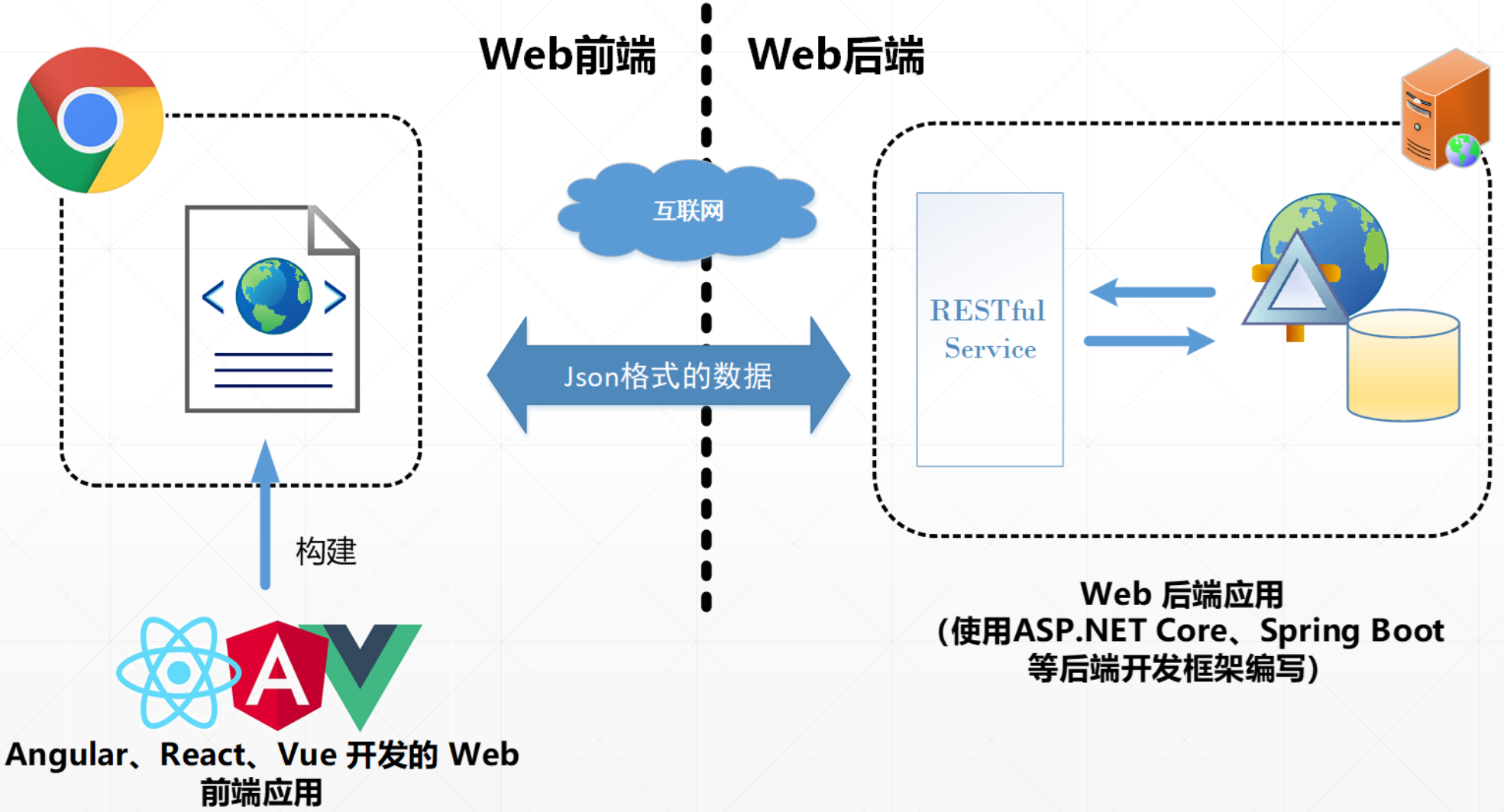
前后端分离的Web应用

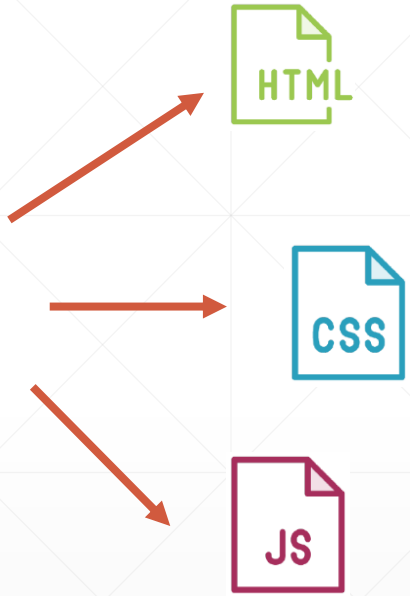
北京理工大学计算机学院
金旭亮

“前后端分离”的移动互联应用



原生态 → 工程化

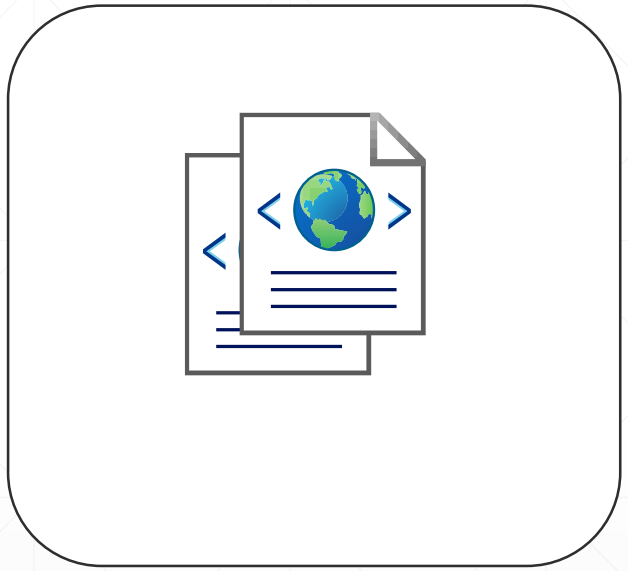
不要怂就是干！



早期网页设计，上来就直接写CSS/JS代码到网页中，立即就可以在浏览器中跑起来……

现在的Web前端应用，需要一个“**构建**”过程——对原材料进行编译和转换之后，浏览器才认识它们……

Web前端开发进入“组件化开发”时代



以网页为中心

过去

现在

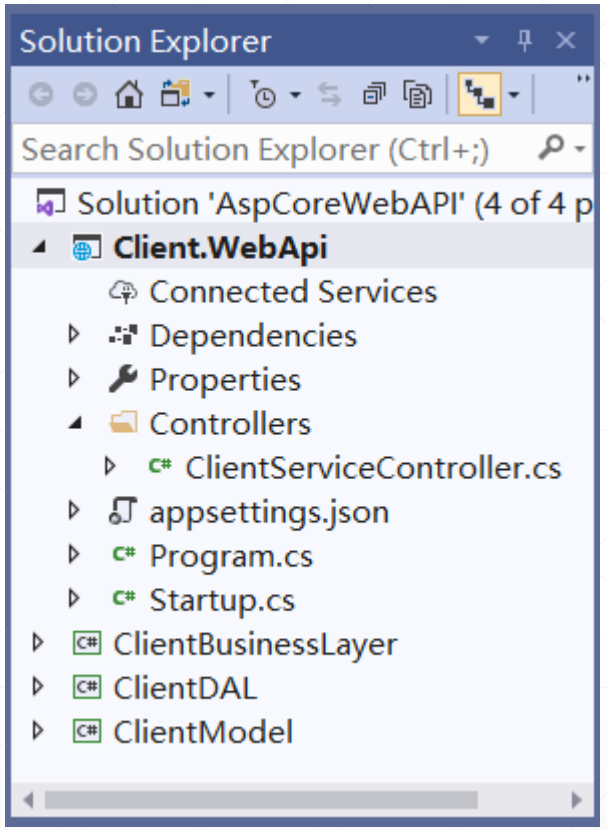
以组件为中心

开发前后端分离的Web应用

示例的总体结构

RESTful Service

客户信息 {Json}



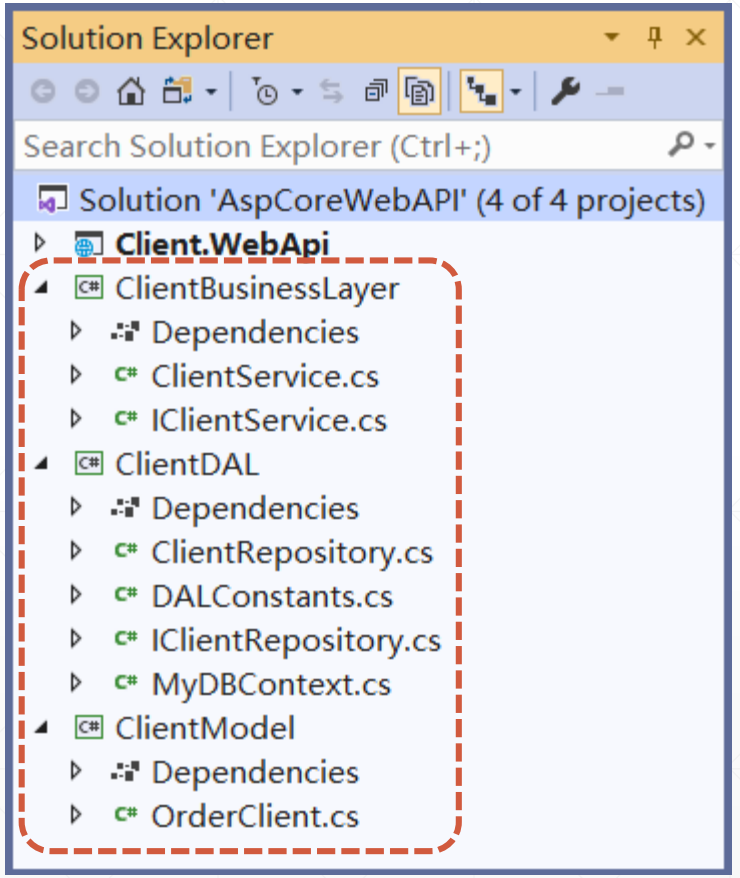
使用Vue CLI创建的独立Web前端项目：client_app

在前一讲AspCoreSPA项目基础上改造而成的Web后端项目

准备工作

1. 示例数据库MyDB已经附加到了本机SQL Server之上。

2. 从前一讲示例AspCoreSPA中复制三个类库项目到本讲示例解决方案中。
再创建一个Web API项目作为后端服务。

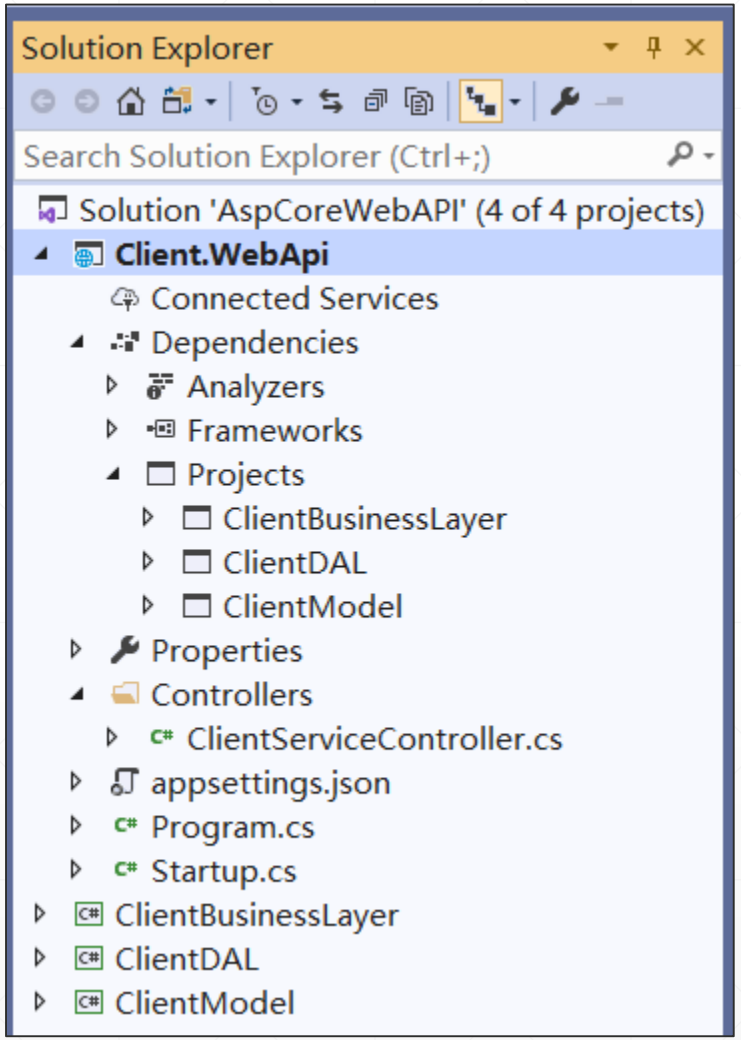


开发Web API项目

1. 给Web API项目添加对三个类库项目的引用，修改Startup.cs，向IoC容器注册三个类库项目中的类型。

2. 在appsettings.json中添加MyDB示例数据库的连接字符串。

3. 删除WeatherForecastController，添加ClientServiceController。



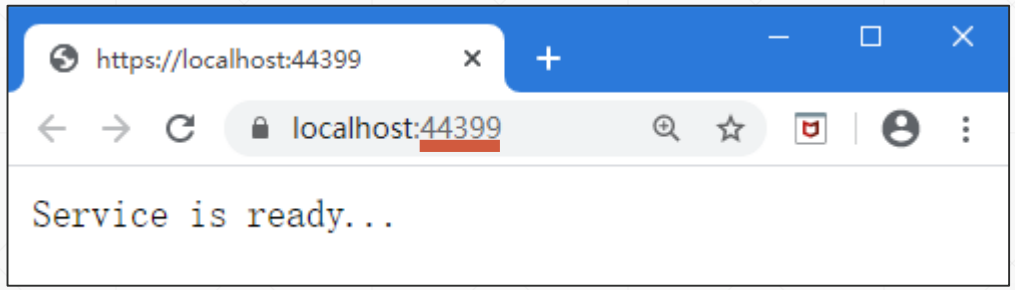
实现Web API控制器

```
[Route("api/[controller]")]
[ApiController]
public class ClientServiceController : ControllerBase
{
    private IClientService clientService = null;
    //将IClientService注入到控制器中
    public ClientServiceController(IClientService clientService)
    {
        this.clientService = clientService;
    }
    //提取所有的客户
    [HttpGet("clients")]
    public async Task<IActionResult> GetClients()
    {
        var clients = await clientService.GetAllClients();
        return Ok(clients);
    }
}
```

试运行……

Startup.cs中的代码：

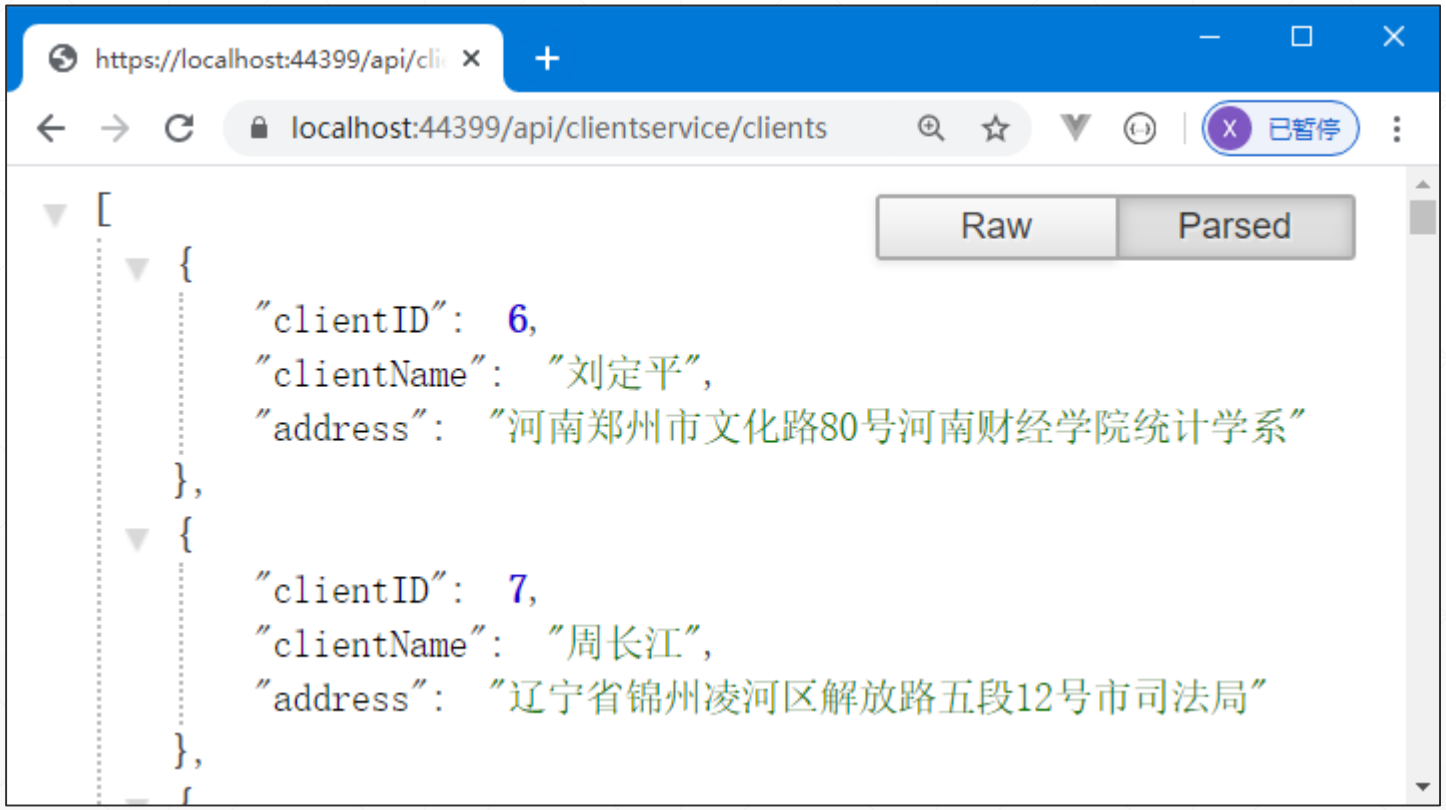
```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
    endpoints.MapGet("/", async context =>
    {
        await context.Response.WriteAsync("Service is ready...");
    });
});
```



注意端口号

测试ClientService控制器工作是否正常

https://localhost:44399/api/clientservice/clients



开发Web前端应用

安装前端工具

https://nodejs.org

https://code.visualstudio.com/

安装yarn

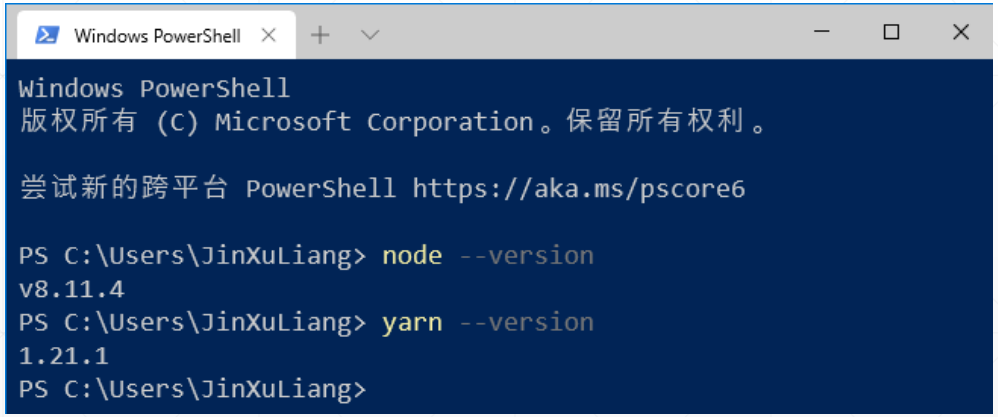
1. 打开“命令提示符窗口（或PowerShell)”，安装yarn:

```
npm install -g yarn
```

https://yarn.bootcss.com/

2. 验证yarn的安装是否成功:

```
yarn --version
```



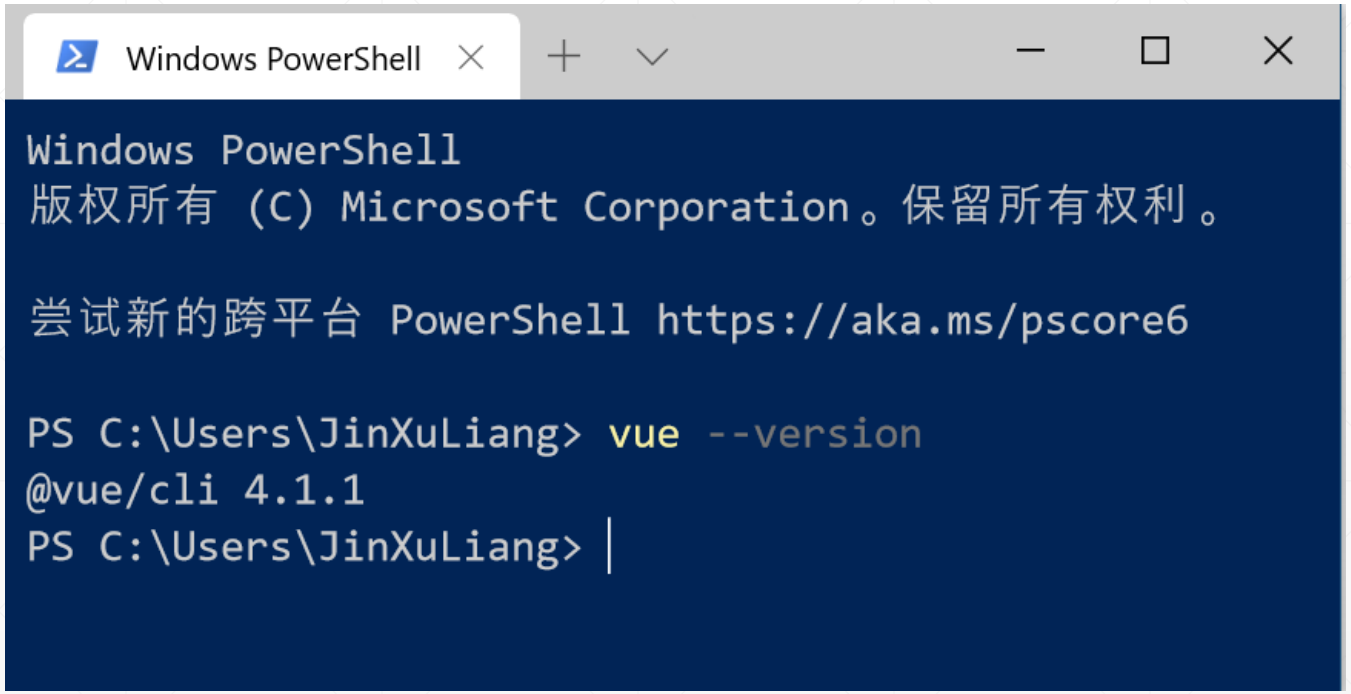
安装Vue CLI

1. 安装Vue-Cli:

```
npm install -g @vue/cli
```

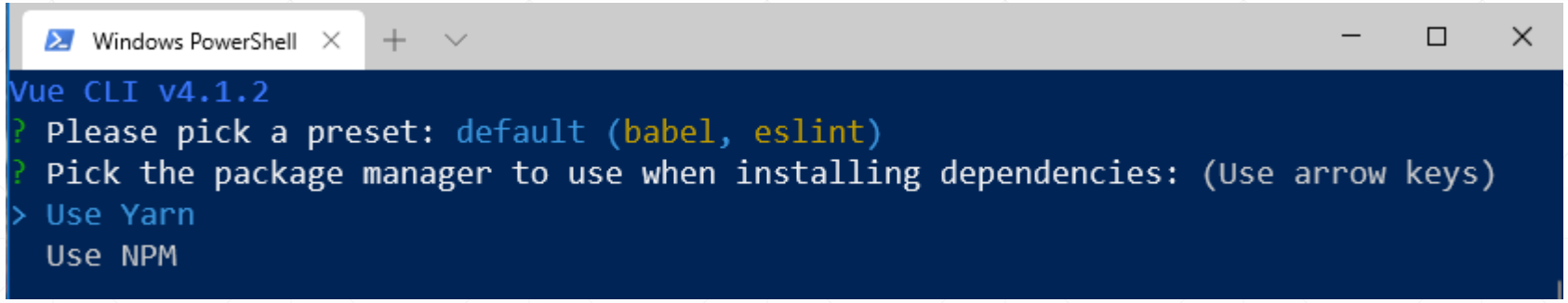
2. 验证安装:

```
vue --version
```

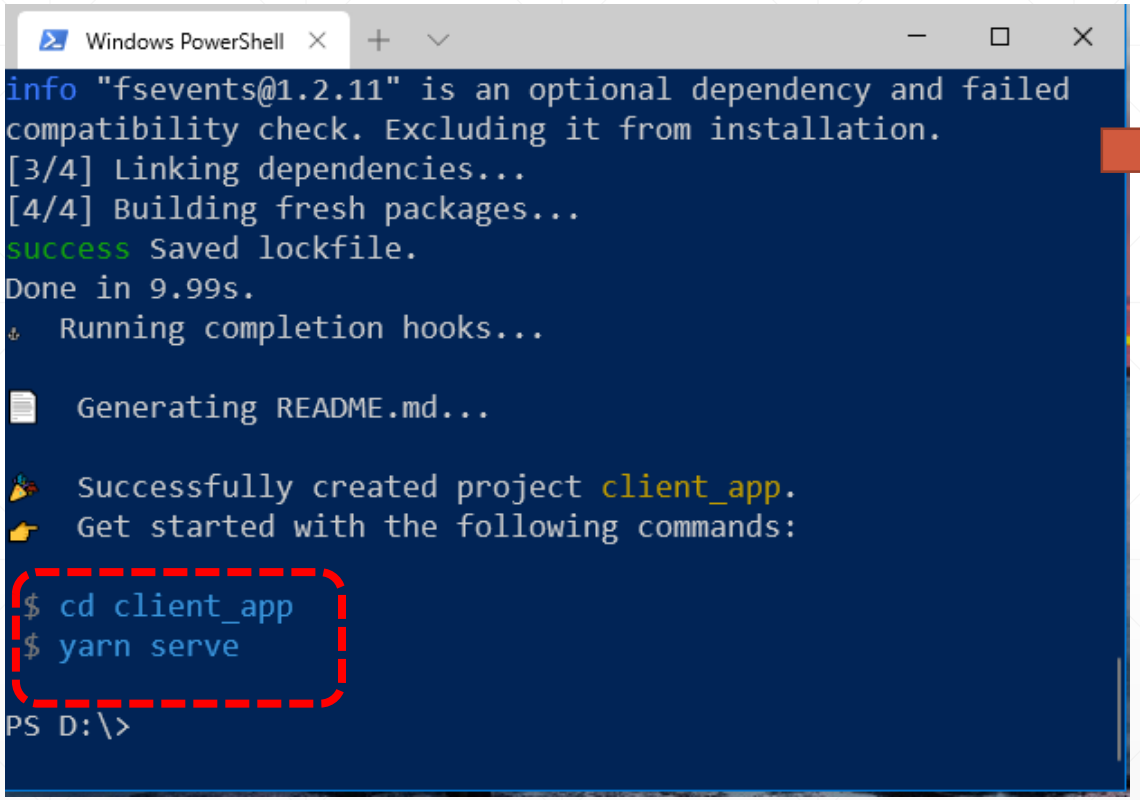


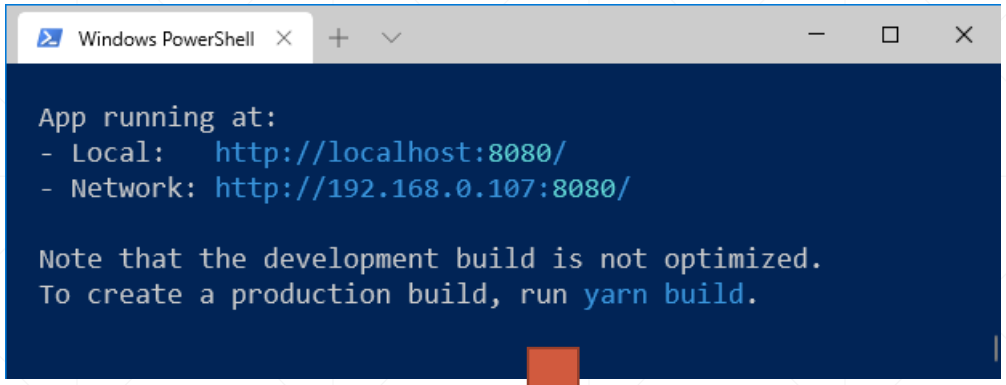
使用Vue命令行工具创建单独的Web前端项目

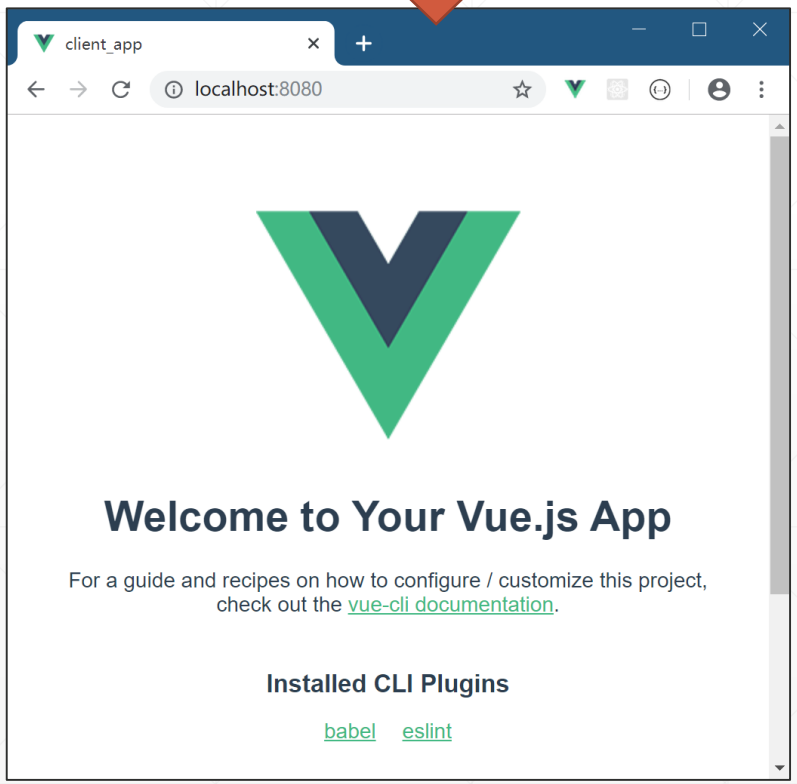
```
vue create client_app
```



启动Web前端应用







启动运行：

```
yarn serve
```

安装bootstrap

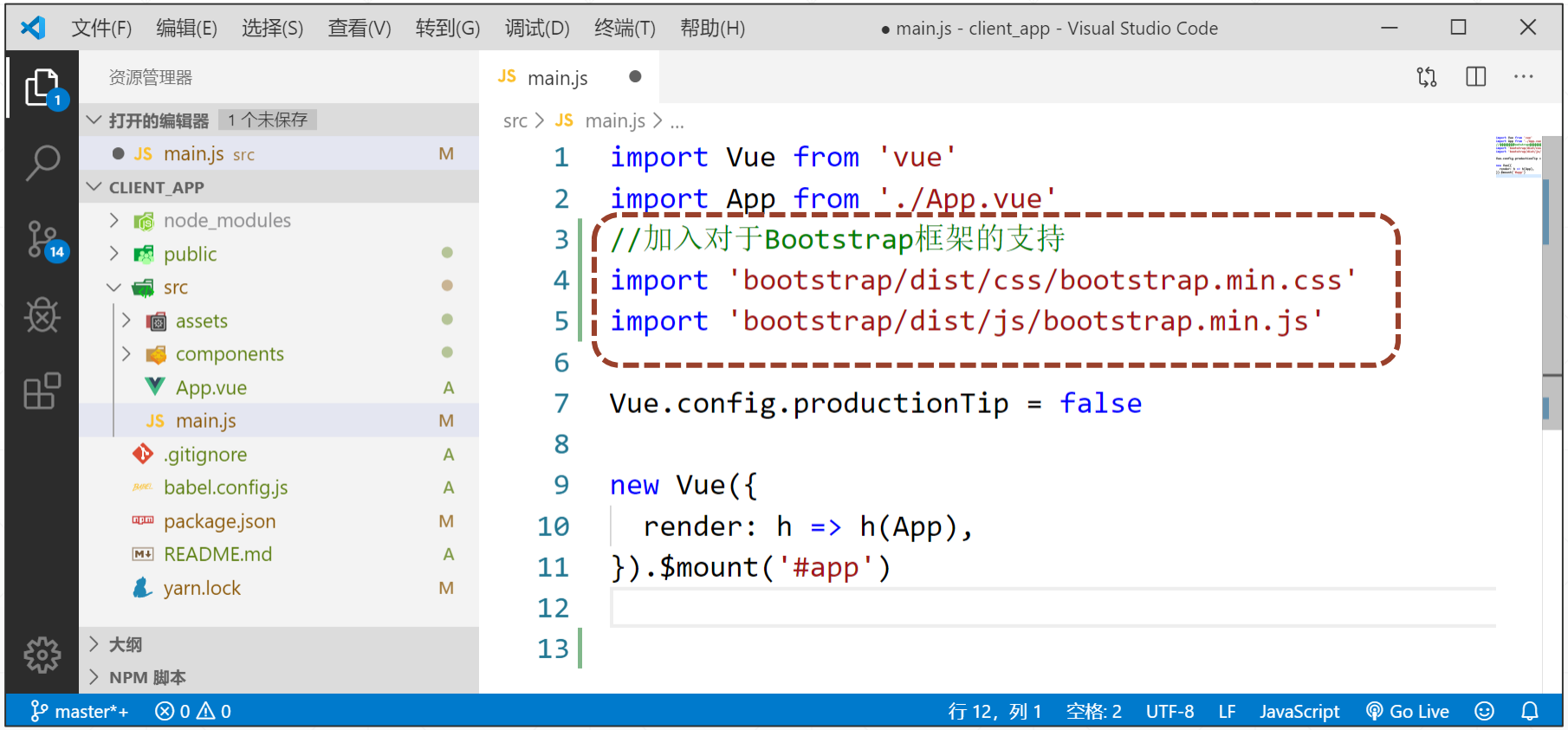
1. 先安装jQuery和propper.js:

```
yarn add jquery --save
yarn add popper.js --save
```

2. 再安装bootstrap:

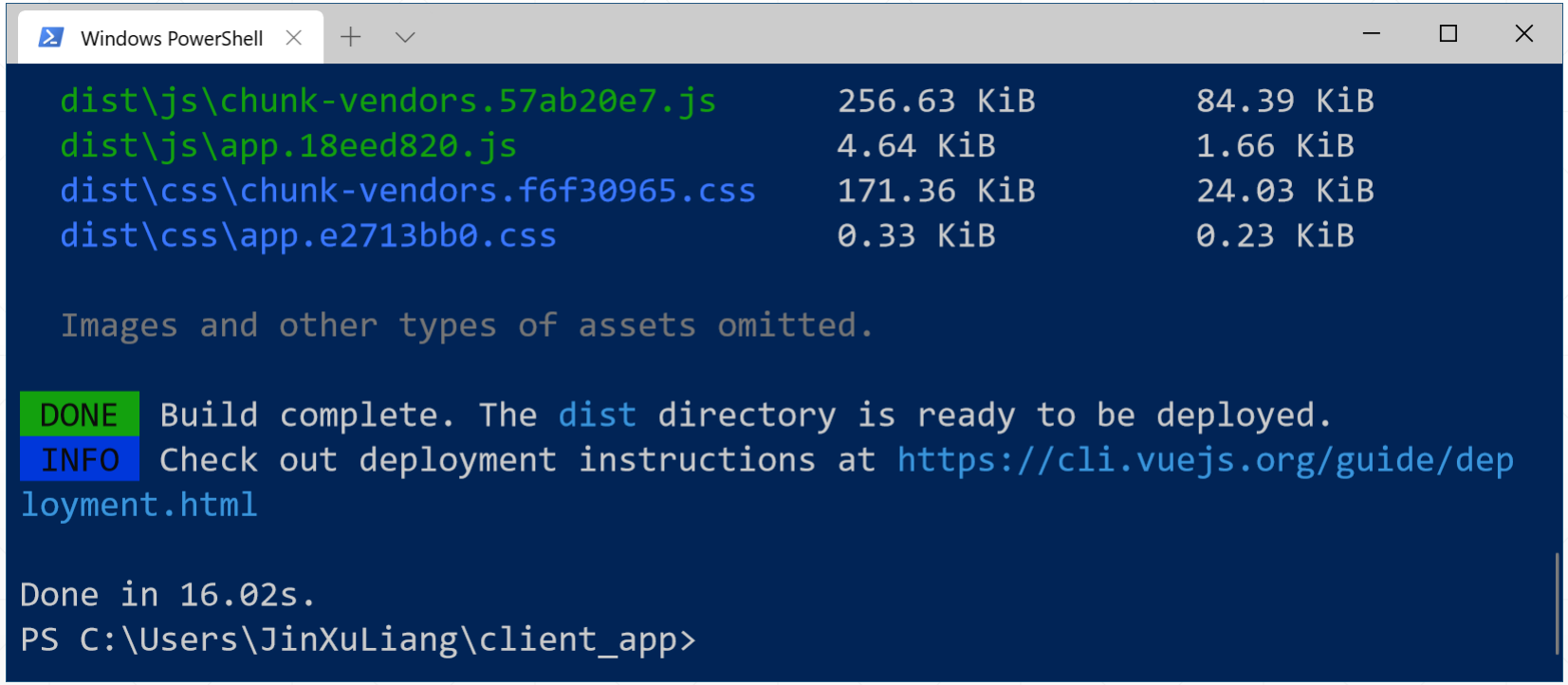
```
yarn add bootstrap --save
```

使用VS Code打开项目



尝试编译，看看有无错误……

```
yarn build
```



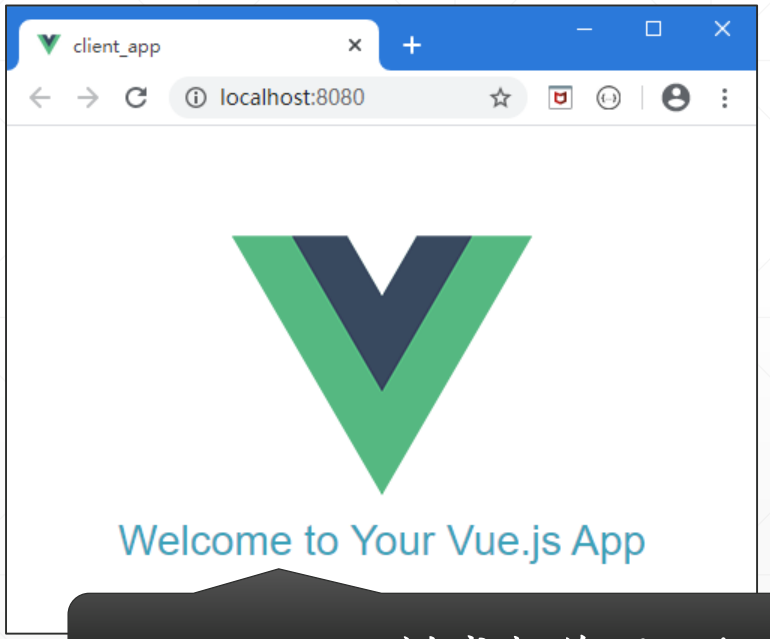
测试使用bootstrap样式

这里使用BootStrap提供的样式

HelloWorld.vue

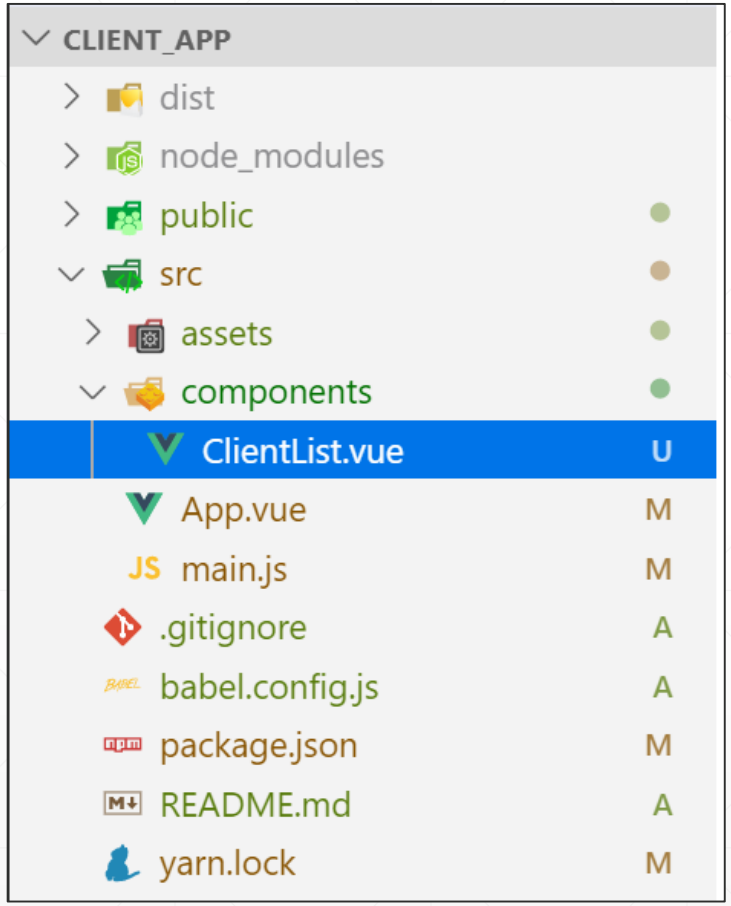
```vue
<template>
  <div class="text-center">
    <h3 class="text-info">{{ msg }}</h3>
  </div>
</template>

<script>
export default {
  name: 'HelloWorld',
  props: {
    msg: String
  }
}
</script>

<style scoped>
</style>
```

Ok，一切正常。



BootStrap样式起作用了！

创建自己的Vue组件



ClientList.vue

src > components > ClientList.vue > {} "ClientList.vue"

```vue
<template>
  <div class="text-center">
    <h3 class="text-info">{{ message }}</h3>
  </div>
</template>

<script>
export default {
  name: 'ClientList',
  data:function(){
      return {
          message:'客户清单'
      }
  }
}
</script>

<style scoped>
</style>
```

ClientList.vue

修改App组件

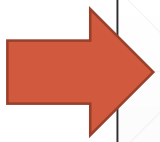
App.vue

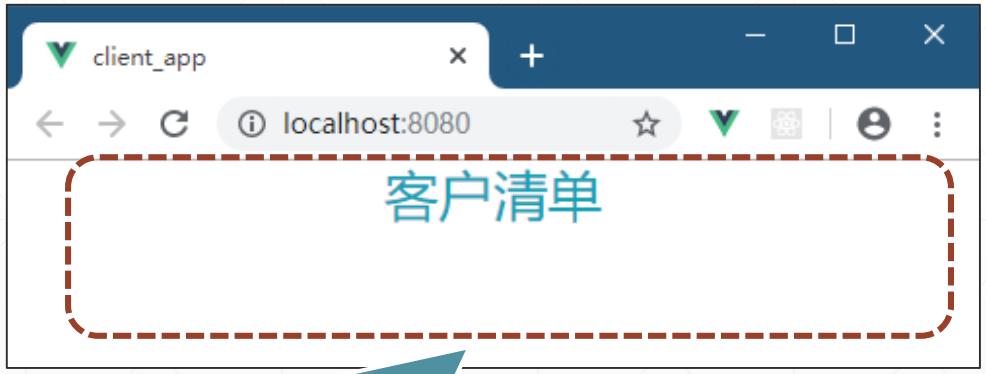
src > App.vue > { } "App.vue" > style

```vue
<template>
  <div id="app">
    <client-list/>
  </div>
</template>

<script>
import ClientList from './components/ClientList.vue'

export default {
  name: 'app',
  components: {
    ClientList
  }
}
</script>

<style>
</style>
```

App.vue

将ClientList组件插入到这里……

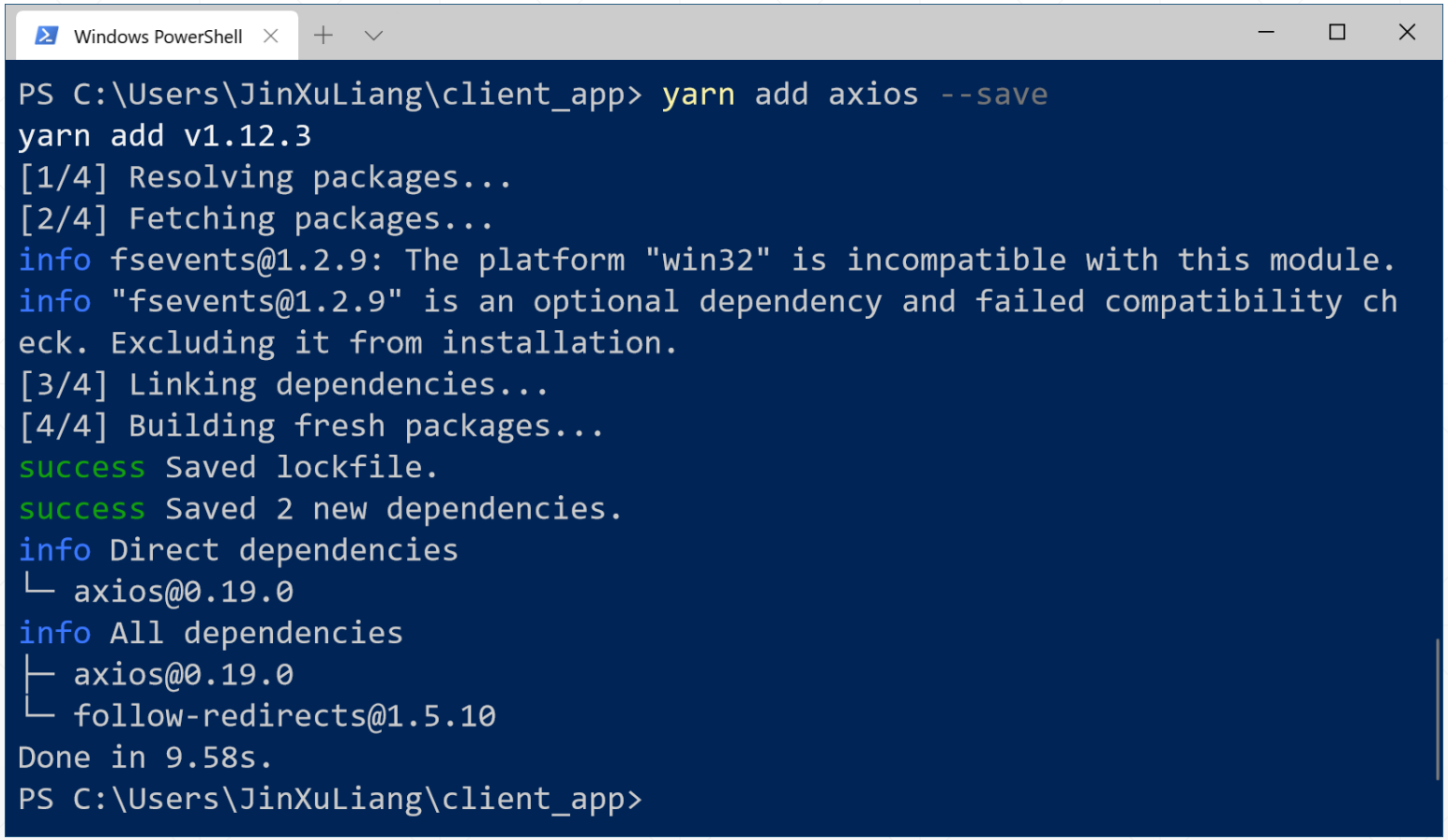




ClientList组件所包容的内容

安装axios

```
yarn add axios --save
```



导入axios

main.js

src > main.js > ...

```
1  import Vue from 'vue'
2  import App from './App.vue'
3  //加入对于Bootstrap框架的支持
4  import 'bootstrap/dist/css/bootstrap.min.css'
5  import 'bootstrap/dist/js/bootstrap.min.js'
6  //导入axios
7  import axios from 'axios'
8
9  Vue.prototype.axios = axios
10 Vue.config.productionTip = false
11
12 new Vue({
13   render: h => h(App),
14 }).$mount('#app')
```

main.js

设计页面

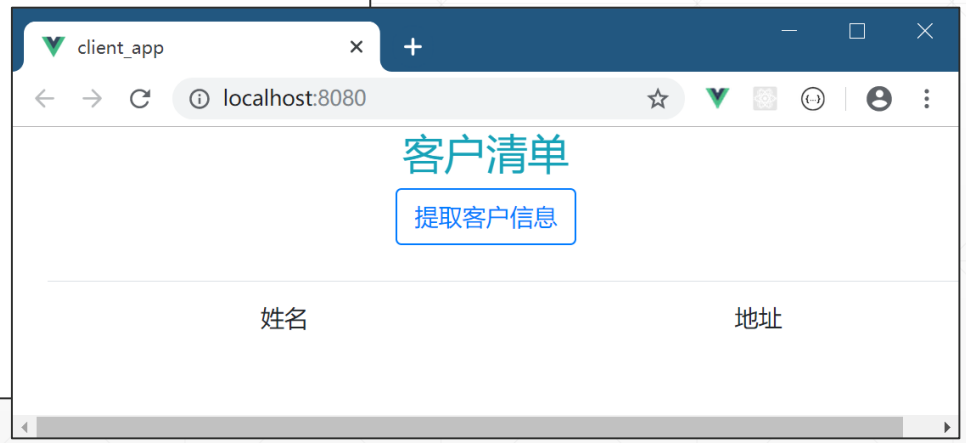
用户点击按钮之后，使用表格显示客户清单。

ClientList.vue

ClientList.vue

src > components > ClientList.vue > { } "ClientList.vue" > template > div.text-center

```vue
<template>
  <div class="text-center">
    <h3 class="text-info">{{ message }}</h3>
    <button class="btn btn-outline-primary" @click="getClient">提取客户信息</button>
    <table class="table">
      <thead>
        <td>姓名</td>
        <td>地址</td>
      </thead>
      <tr v-for="client in clients" :key="client.ClientID">
        <td>{{client.clientName}}</td>
        <td>{{client.address}}</td>
      </tr>
    </table>
  </div>
</template>
```



如果报告以下错误……

```
yarn serve
```

```
问题  输出  调试控制台  终端                                    1: node

98% after emitting CopyPlugin

ERROR  Failed to compile with 1 errors

error  in ./src/components/ClientList.vue

Module Error (from ./node_modules/eslint-loader/index.js):
error: Unexpected console statement (no-console) at src\components\ClientList.vue:35:11:
  33 |         })
  34 |         .catch(err => {
> 35 |           console.error(err);
     |           ^
  36 |         });
  37 |     }
  38 |   }


1 error found.
```

ESLint

package.json

```
"eslintConfig": {
  "root": true,
  "env": {
    "node": true
  },
  "extends": [
    "plugin:vue/essential",
    "eslint:recommended"
  ],
  "rules": {
    "no-console":"off"
  },
  "parserOptions": {
    "parser": "babel-eslint"
  }
},
```

使用axios提取数据

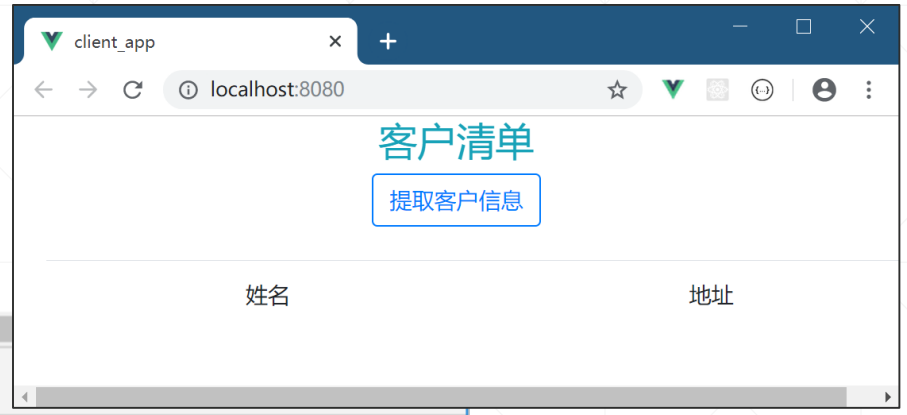
```
<script>
export default {
  name: "ClientList",
  data: function() {
    return {
      message: "客户清单",
      clients: []
    };
  },
  methods: {
    getClient() {
      this.axios
        .get("https://localhost:44399/api/clientservice/clients")
        .then(res => {
          this.clients = res.data;
        })
        .catch(err => {
          console.error(err);
        });
    }
  }
};
</script>
```

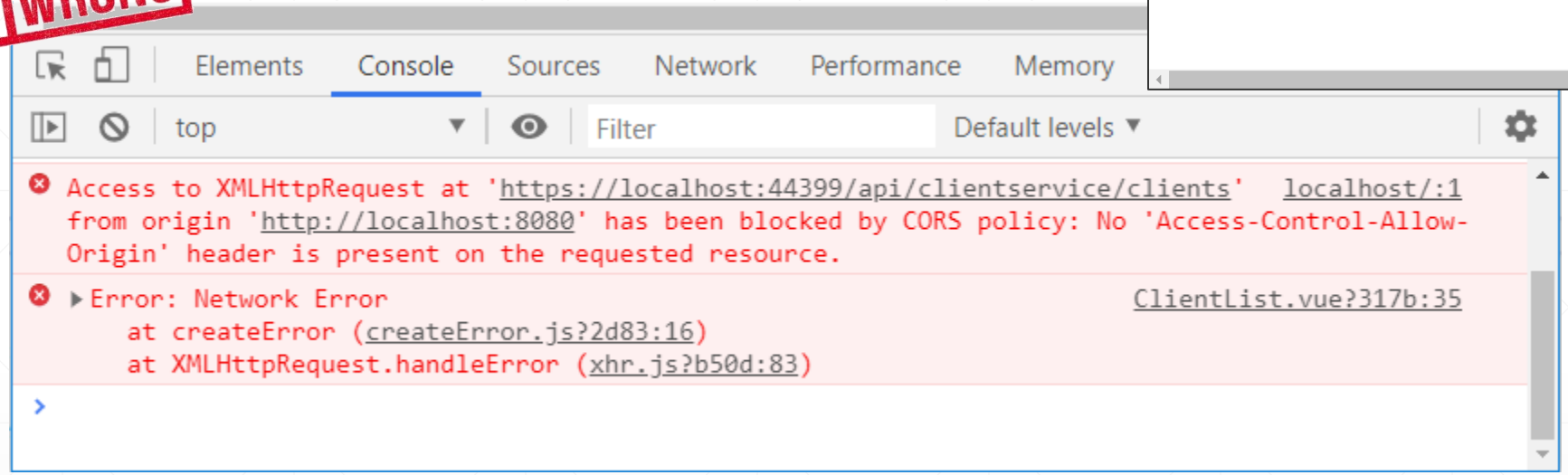
ClientList.vue

使用axios发出HTTP Get请求，提取客户信息。

要求服务端先运行。

提取数据时发生错误……





原因：由于此Web前端App运行于localhost:8080，却尝试访问后端localhost:44399的数据，所以浏览器阻止了这个访问，这种现象称为“**CORS（跨域请求）拦截**”。

在Server端针对前端Web App启用CORS

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
    RegisterMyTypeIoCContainer(services);
    services.AddDbContext<MyDBContext>(options =>...);
    //启用CORS
    services.AddCors(options =>
    {
        //添加一个CORS策略，允许本机8080端口的应用访问
        options.AddPolicy("AllowLocalhost", builder =>
        {
            builder.WithOrigins("http://localhost:8080")
            .AllowAnyMethod()
            .AllowAnyMethod();
        });
    });
}
```

```
public void Configure(IApplicationBuilder app,
    IWebHostEnvironment env)
{
    if (env.IsDevelopment())...
    else...

    //针对所有终结点，启用"AllowLocalhost"策略
    app.UseCors("AllowLocalhost");

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseRouting();
    app.UseAuthorization();
    app.UseEndpoints(endpoints =>...);

}
```

ASP.NET Core的CORS特性官方文档：

https://docs.microsoft.com/en-us/aspnet/core/security/cors?view=aspnetcore-3.1

工作正常的前端应用

由于服务端设置允许8080端口的HTTP请求，所以现在前端应用可以访问到客户清单了。

小结

- 在本讲中我们展示了一个采用了前后端分离架构的Web应用的开发过程。
- 在开发Web前端应用时，CORS是一个需要仔细对待的特性。
- Web安全是一个非常大的话题，本课程对此不作展开，将放在独立的技术专题课程中介绍。

“.NET Core软件开发技术导论与自学指南”

本课程主干内容结束。

.NET Core各种具体领域的相关技术，将作为专题，开设单独的“技术专题”课程介绍。