4-1 直播 Promise

内容介绍：

- 模块问题答疑
- Promise 讲解

直播代码：📎 4-1 直播代码汇总.rar

模块问题答疑

1. 关于监听器中handler中的oldval和val为啥是一样的?
   a. **回顾功能**：vue 的侦听器 watch 用于侦听数据变化，从而进行相应的处理
   b. **明确场景**：侦听的处理函数 handler 可以通过参数接收变化前后的两个值，但侦听的值为复杂类型（数组、对象）时，这两个值相同。
   c. **原因阐述**：本质上，新旧值其实是在数据**修改前后**分别获取的**数据副本（相当于 = 赋值，非拷贝）**，由于数组与对象为引用类型，数据副本与原数据本质就是同一个，导致两个参数在访问时相同。
   d. 如有需求，可设置 computed，在内部克隆要侦听的数据，再通过 watch 侦听计算属性，即可在对象（数组）变化时，克隆数据，并可以在参数得到两个不同引用的对象数据。

```
<script src="https://cdn.jsdelivr.net/npm/vue@2.6.12"></script>
// 示例代码:
const vm = new Vue({
  el: '#app',
```

```
  data: {
    num: 1,
    arr: [2, 3, 4]
  },
  watch: {
    // 参数取值不同
    num (newValue, oldValue) {
      console.log('新值: ',newValue);
      console.log('旧值: ', oldValue);
    },
    // 参数取值相同
    arr (newValue, oldValue) {
      console.log('新值: ',newValue);
      console.log('旧值: ', oldValue);
      console.log(newValue === oldValue); // true 可证明新旧值均为数据副本
    }
  }
})

// c 部分分析示意:
const list = [1, 2, 3]
let listA = list // 修改前获取副本
list.push(4)
let listB = list // 修改后获取副本
console.log(listA, listB)
console.log(listA === listB)
```

2. 关于项目构建过程中功能的实现，该如何思考，比如哪些先实现哪些后实现?
    a. 项目大体流程
        i. 项目制作前首先要先画原型图（产品）
        ii. 然后根据原型图出设计图（设计）
        iii. 技术选型、制作等（技术）
        iv. 测试、发布等等
    b. 得到项目后，要先对项目进行功能划分，从大到小拆解项目。需要**具体情况具体分析**。
    c. 实际情况:
        i. 工作初期，团队负责人会进行任务分配，我们只是负责大项目中的某部分功能。
        ii. 我们只需要将自己负责的部分进行功能划分与实现即可。
        iii. 慢慢掌握熟练了，也可以去尝试参与一些自己不熟悉的功能，一步步走向巅峰。

3. 这个todoMVC的项目其实算是半成品开始制作的，有没有从0开始的项目希望老师给演示一下，如果后续课程有该类的内容，请忽略我的问题。
    a. 问题解答：
        i. 综合案例是为了练习 Vue.js 语法而设置的。
        ii. 半成品其实就是进行了布局与样式处理，且 todoMVC 的结构并不复杂，大体为输入框 + 列表。
        iii. 如需要练习，可自行查看源码仿制，有问题及时与我交流。
4. TodoMVC项目中对于设置全部切换选框状态时怎么添加css样式的
    a. 通过查看代码可发现，是在 :checked 时修改了 label 的背景色。

Promise 讲解

我们将分为以下几步进行讲解：

1. 简介：Promise 是在什么背景下产生的（背景），用来解决什么问题（作用），与操作思路。
2. 使用方式
3. Async 函数
4. try..catch

背景说明

从同步与异步说起：

- JavaScript 中具有同步和异步的概念，最常见的异步操作莫过 Ajax，Ajax 是前后端数据交互的桥梁。

异步操作特点：

- 同步任务总是顺序执行的。与同步任务不同，异步任务**执行结果与书写顺序无关**，例如，定时器结束时间取决于延迟时间；Ajax 的响应成功时间取决于数据量与网速。
- 这就导致，如果要对异步操作的结果进行处理，就必须使用**回调函数**（callback）。

问题出现：

- 思考：如果有多个请求，每个请求都依赖与前一个请求的结果，这时会发生什么呢?

```
$.get('http://edufront.lagou.com/front/course/getPurchaseCourse', function (res) {
  console.log('数据处理')
  $.get('http://edufront.lagou.com/front/course/getPurchaseCourse', function (res) {
    console.log(res)
    $.get('http://edufront.lagou.com/front/course/getPurchaseCourse', function (res) {
      console.log(res)
      $.get('http://edufront.lagou.com/front/course/getPurchaseCourse', function (res) {
        console.log(res)
        $.get('http://edufront.lagou.com/front/course/getPurchaseCourse', function (res) {
          console.log(res)
        })
      })
    })
  })
})
```

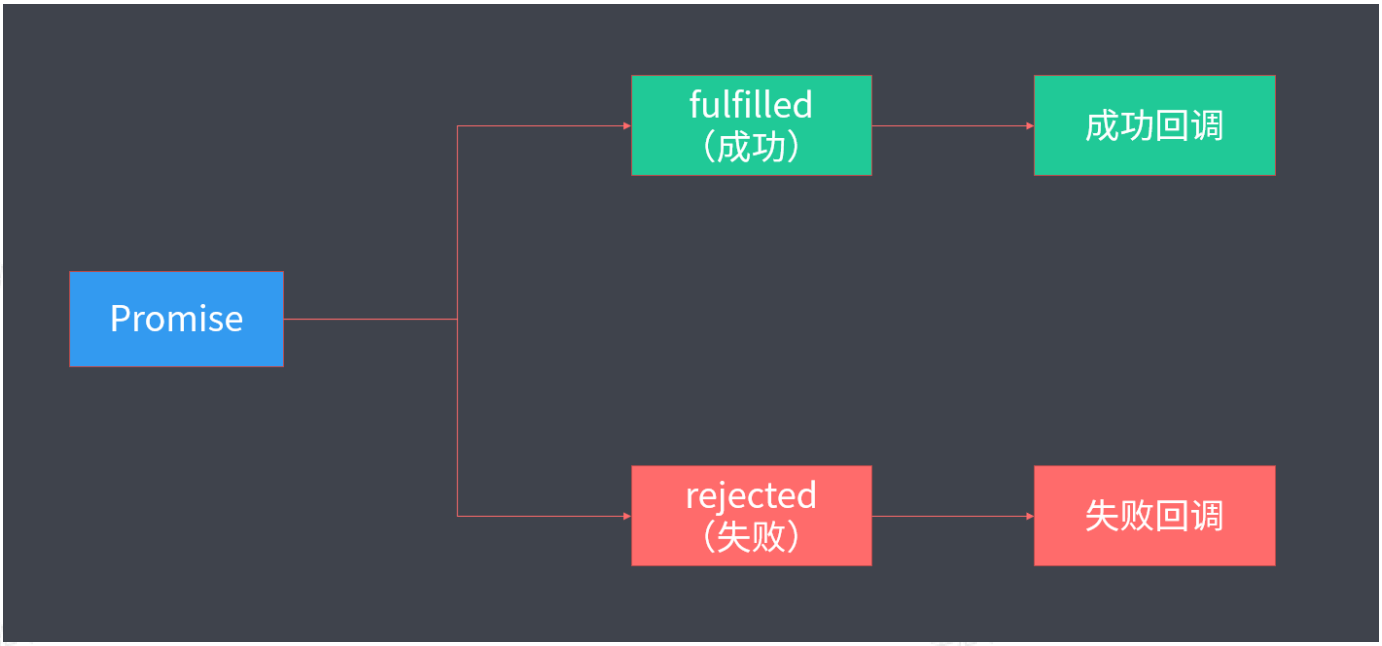
- 这种层层嵌套的回调写法称为“回调地狱”

Promise 的作用：

- Promise 是 ES2015（ES6）中标准化的功能，用于 给异步编程提供一种书写更合理，功能更强的统一的解决方案。

Promise 的操作思路

- Promise 含义为承诺，是一个 ES6 提供的类型，使用时需要创建一个**实例对象**，指代我们要进行的一个操作（异步）
- 创建完毕， 这时 promise 处于一个“待定状态”， 最终结果可能为“成功状态”或“失败状态”。
- 举例增强理解：明天发工资，我要给我媳妇买个苹果，如果发工资了，就买一斤，如果没发就算了，因为没有私房钱。
- **特点：结果明确后无法更改**：例如，发工资后买了一斤苹果，买了就是买了，不可能过一会儿又变成了没买，这不合逻辑。
- **回归操作：**那么对应操作中，利于一个 Ajax 请求，当请求成功或失败时，可以通过相应的“**状态**”进行执行结果的判断，从而进行相应的处理。
- 小结：Promise 对象的操作就是通过状态进行控制，状态变化会自动触发对应的回调。



Promise 使用

- 创建实例用于存储异步操作
- 参数为函数，在创建实例时同步执行
- 函数有**两个参数**，都是函数，调用时可**决定 promise 对象的状态**
  - resolve(data) 传递成功时的结果
  - reject(err) 传递失败时的错误信息
- 注意，由于promise 对象的结果明确后无法更改，所以两个函数在逻辑中只能有一个被执行。

```
1 const promise = new Promise(function () {
```

```
    resolve(100)
    // reject(new Error('error message!'))
})
```

当 promise 对象的状态确定时，会触发对应的回调函数进行处理，我们需要通过 promise 对象的方法指定这些回调函数：

- then() 成功时的处理函数
- catch() 失败时的处理函数

```
const promise = new Promise(function (resolve, reject) {
  resolve(100)
  // reject(new Error('error message!'))
})
// 也可以通过 new Promise().then().catch() 方式进行链式调用。
promise.then(function (value) {
  console.log('成功了，数据为：', value)
}).catch (function (error) {
  console.log('失败了，错误为：', error)
})
```

Promise 练习：封装 Ajax 函数

测试接口地址：

- 成功测试地址：http://edufront.lagou.com/front/course/getPurchaseCourse
- 失败测试地址：http://eduboss.lagou.com/boss/course/changeState (401)

```
function ajax (url) {
  return new Promise(function (resolve, reject) {
    const xhr = new XMLHttpRequest()
    xhr.open('GET', url)
    xhr.responseType = 'json'
    xhr.onload = function () {
      if (this.status === 200) {
        // 接收到响应数据，标记状态为成功，并传递给 then 中的回调处理
        resolve(this.response)
      } else {
        // 失败时更改状态，并传递错误信息
        reject(new Error(this.statusText))
```

```
      }
    }
    // 发送请求
    xhr.send()
  })
}
const successUrl = 'http://edufront.lagou.com/front/course/getPurchaseCourse'
const failUrl = 'http://eduboss.lagou.com/boss/course/changeState'

ajax(failUrl).then(function (res) {
  console.log(res)
}).catch(function (error) {
  console.log(error)
})
```

如果仅仅是这样使用，请求多了，**依然会出现嵌套**的问题，这时要注意使用方式，不要嵌套书写，而是利用 Promise 的链式调用进行操作。

```
// 错误写法
ajax(successUrl)
  .then(function (res) {
    console.log(res)
    ajax(successUrl)
    .then(function (res) {
      console.log(res)
      ajax(successUrl)
      .then(function (res) {
        console.log(res)
      })
    })
  })
```

链式调用

then() 调用的返回值为一个**新的 Promise 对象**，如果要在一次异步操作后再执行下一个异步操作，应当通过这个新的 Promise 对象进行处理，这样就可以避免回调地狱啦。

```
// 执行示例
ajax(successUrl)
  .then(function (res) {
    console.log(1)
  })
  .then(function (res) {
    console.log(2)
  })
  .then(function (res) {
    console.log(3)
  })
```

在 then() 中通过 return 返回 promise 对象会被设置为 then() 的返回值。

```
// 顺序执行异步操作实例
ajax(successUrl)
  .then(function (res) {
    console.log(1, res)
    return ajax(successUrl)
  })
  .then(function (res) {
    console.log(2, res)
    return ajax(successUrl)
  })
  .then(function (res) {
    console.log(3, res)
    return ajax(successUrl)
  })
```

如果返回的是普通值，则会包裹在默认的 promise 对象中，传递给 then 方法，不传时默认为 undefined

```
ajax(successUrl)
  .then(function (res) {
    console.log(1, res)
    return ajax(successUrl)
  })
   .then(function (res) {
    console.log(2, res)
```

```
    // return ajax(successUrl)
    return [1, 2, 3]
  })
   .then(function (res) {
    console.log(3, res) // 3, [1, 2, 3]
    return ajax(successUrl)
  })
```

总结：promise 本质上就是通过回调函数定义异步任务结束后所需要执行的任务，只不过回调是通过 then() 进行设置，由于 then() 可以进行链式调用，所以避免了层层嵌套的书写方式。

Async 函数

Promise 的出现解决了异步操作回调地狱的问题，但书写方式中函数出现了**大量的回调**，虽然没有嵌套，但可读性远不及同步代码清晰。

为了让异步代码**书写更加简洁、可读性更强**，在 ES2017 中提供了 Async 函数（异步函数），用于简化 Promise 的书写方式，让代码格式更加接近同步代码写法。

Async 是异步编程的新标准，是一种用于改进异步书写的语法糖，本质还是 promise 对象。

使用方式：

- 在任意函数前设置 async 关键字，将函数设置为 async 函数（异步函数）。
- 在 async 函数内调用**返回了 promise 对象的函数**，并在调用前设置 await。
  - 此时，调用的返回值会被 await 设置为 resolve 传递的数据，既**成功**的结果
  - 失败时报错（需通过 try...catch 处理，见下一小节）。
  - 注意：**await 只能出现在 async 函数中**

```
async function fn () {
  const data1 = await ajax(successUrl)
  console.log(1, data1)
  const data2 = await ajax(successUrl)
  console.log(2, data2)
  const data3 = await ajax(failUrl)
  console.log(3, data3)
}
fn()
```

try..catch

try..catch 用于进行**异常捕获**（报错处理），当程序代码可能出现报错时，可以通过 try...catch 进行异常捕获。

```
// try 块的代码会先执行，如果未出现报错，则执行结果，catch 不会触发
// 如果 try 块中出现异常（报错），则会执行 catch 内的代码，同时异常不会被抛出（报错不会真正出现）。
try {
  // 把可能会出现报错的代码放到 try 中
  fn() // 会报错
} catch (err) {
  // 当 try 中出现任意的报错（异常），catch 会将异常捕获，并进行补救处理
  console.log(err)
}
console.log('这句代码可以执行')
```

为了程序的**健壮性**，应当利用 try..catch 对 async/await 进行处理。

```
function ajax (url) {
  return new Promise(function (resolve, reject) {
    const xhr = new XMLHttpRequest()
    xhr.open('GET', url)
    xhr.responseType = 'json'
    xhr.onload = function () {
      if (this.status === 200) {
        resolve(this.response)
      } else {
        reject(new Error(this.statusText))
      }
    }
    xhr.send()
  })
}


async function fn () {
  const data1 = await ajax(successUrl)
  console.log(1, data1)
  const data2 = await ajax(successUrl)
  console.log(2, data2)

  // 例如：未避免请求3失败导致后续代码无法执行，可通过 try...catch 处理
```

```
  try {
    const data3 = await ajax(failUrl)
    console.log(3, data3)
  } catch (error) {
    console.log('进行补救处理')
  }

    // 即使请求3失败，后续请求也可以继续执行。
  const data4 = await ajax(successUrl)
  console.log(4, data4)

}
fn()
```

结语

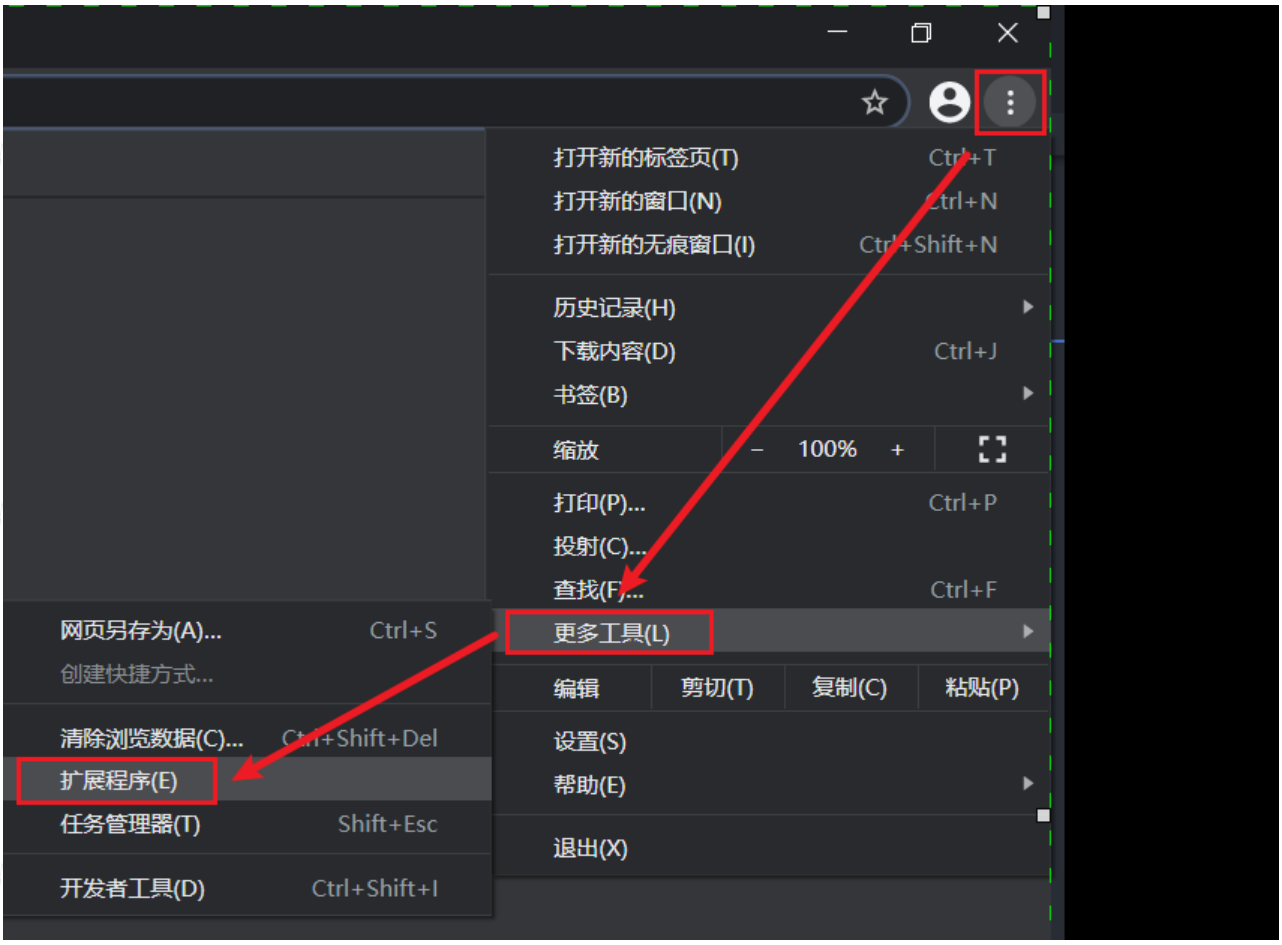
技术更新导致环境变化，环境变化又促使技术的不断变革。对旧代码的分析与深入思考可以让我们获得更多，古人云：温故而知新。望诸君能在技术探索的道路上坚持不懈，砥砺前行！

Good Hunting!

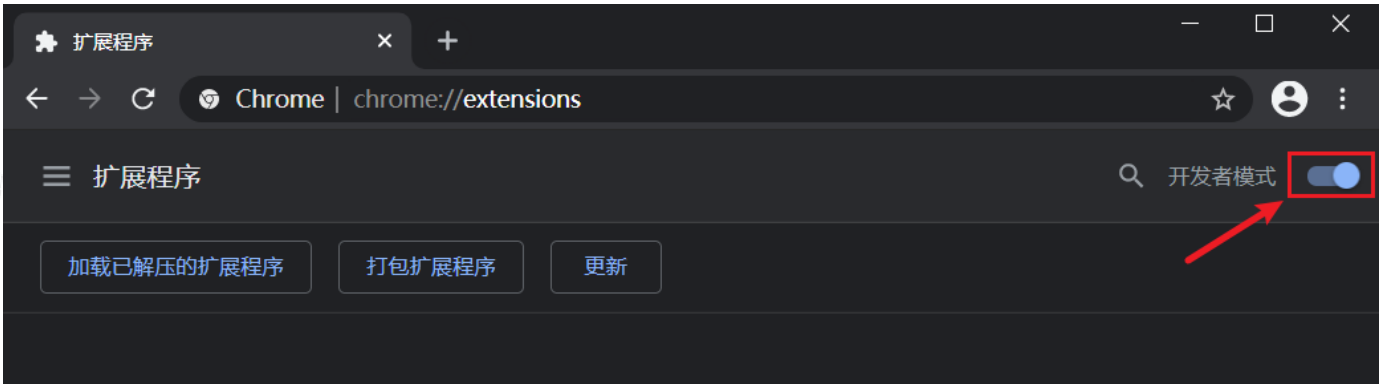
附：DevTools 安装

由于 Google 商店在国内无法正常访问，所以需要需要**手动安装** DevTools。

1. 打开谷歌浏览器，按照如下步骤打开**扩展程序**。



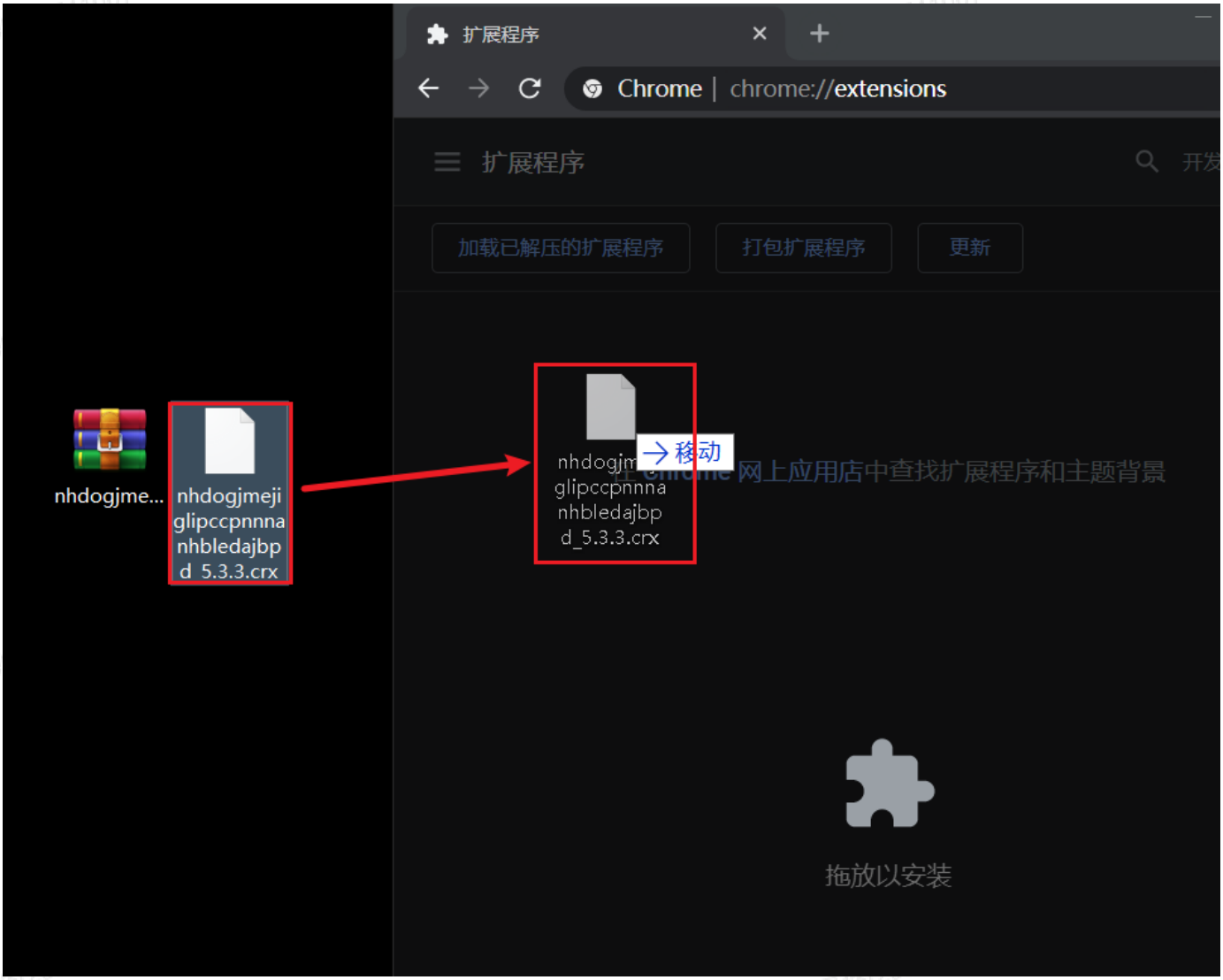
2. 打开**开发者模式**（重要）。



3. **下载**资源文件，并解压得到内部的 .crx 文件。

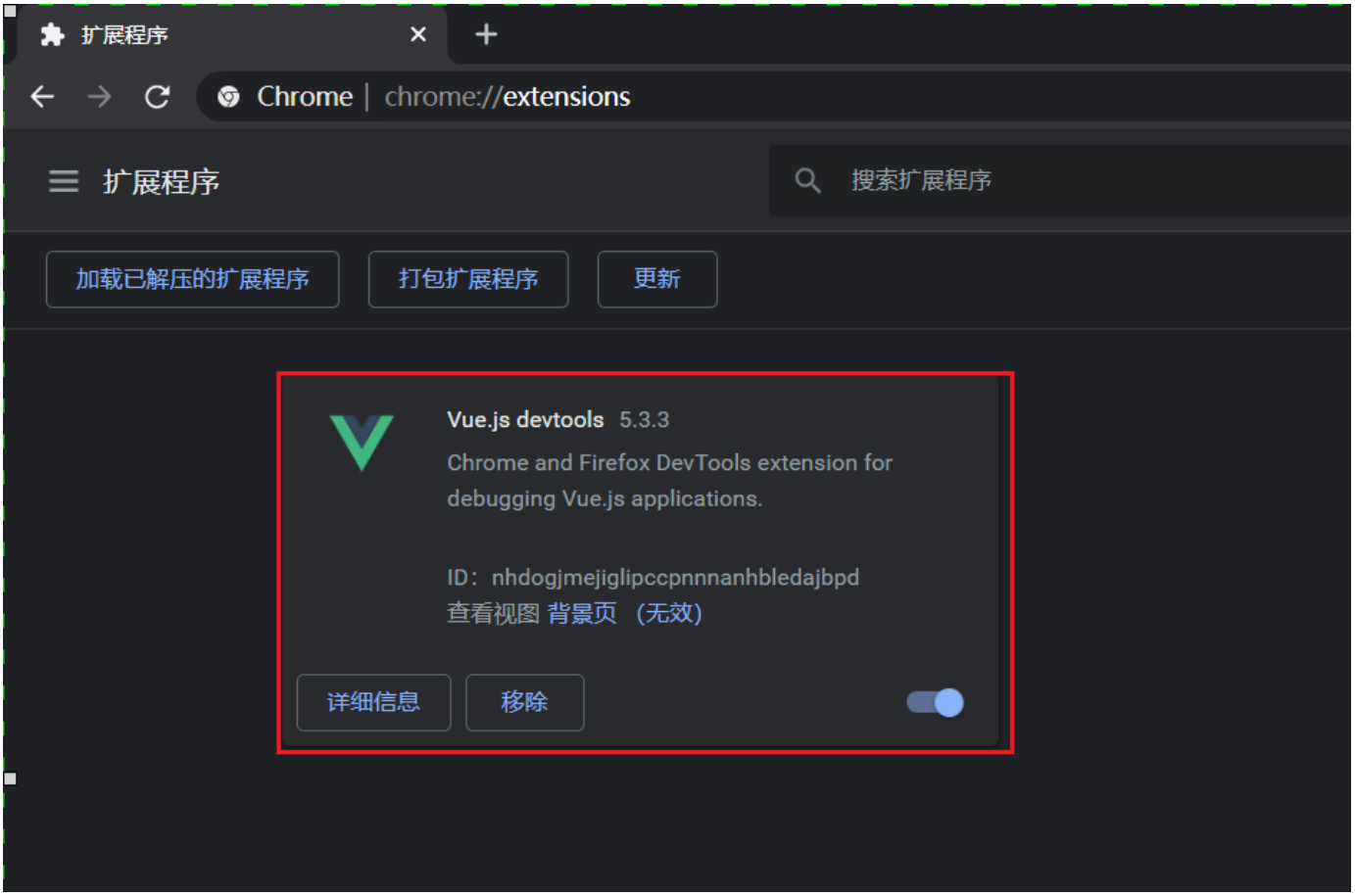
资源地址：📎 nhdogjmejiglipccpnnnanhbledajbpd.zip

4. 将 .crx 文件**拖动**到界面中进行安装。

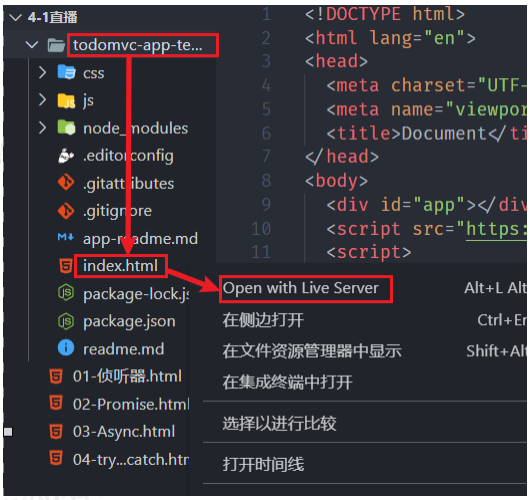


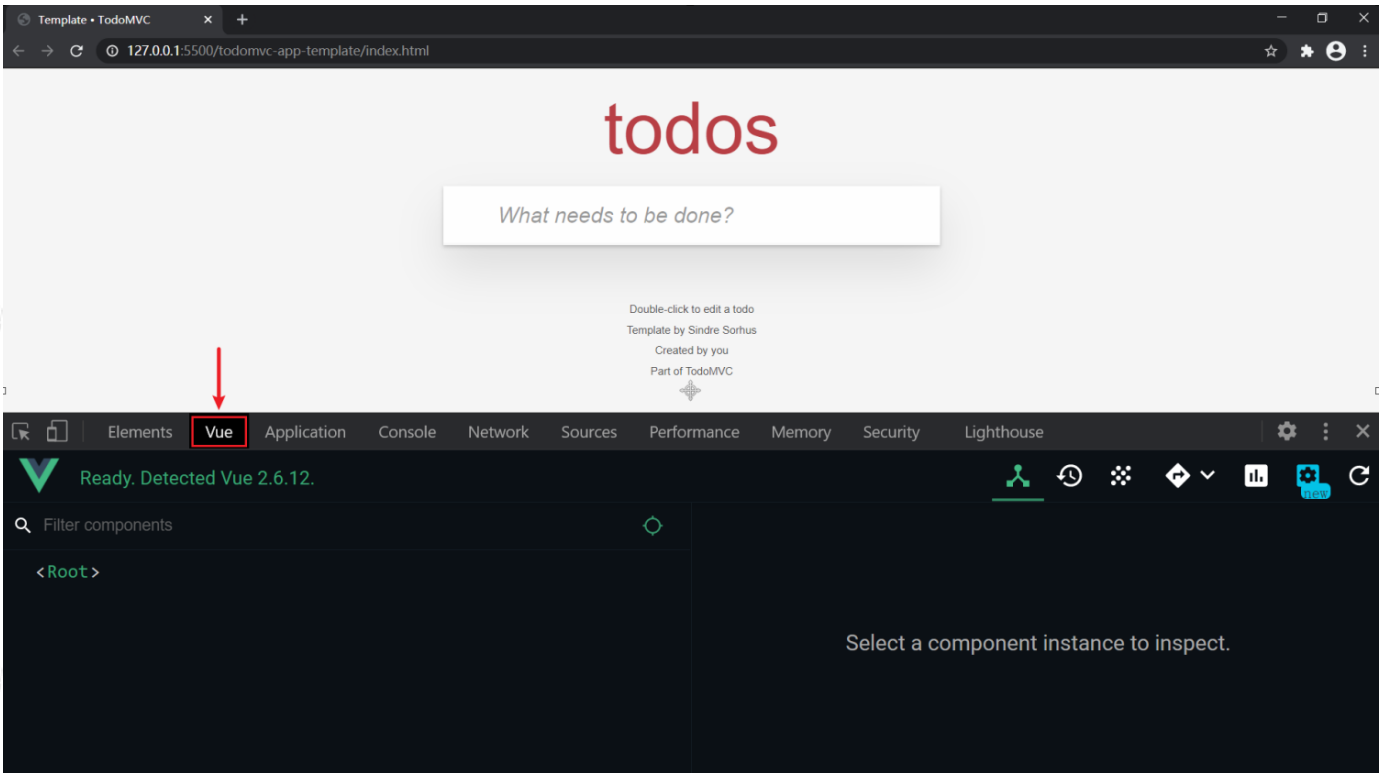
5. 选择**添加扩展程序**。

6. 安装成功



7. 测试是否可以使用，打开示例代码中的 TodoMVC，并通过 **Live Server** 运行。





注意：当自己使用时，要确定代码中引入了 **vue.js** 而不是 vue.min.js，同时要在**服务器环境**中使用哦~