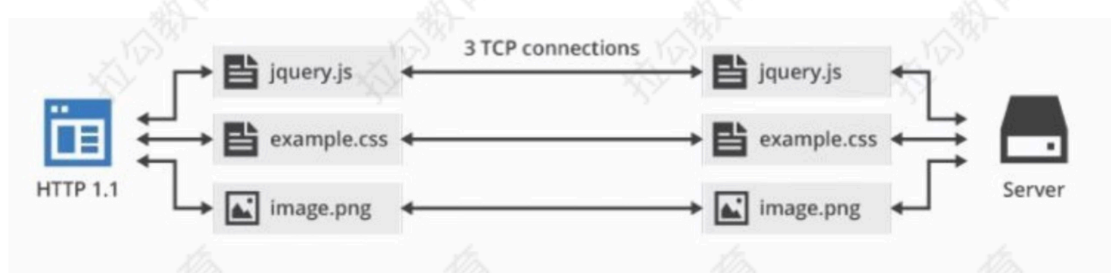
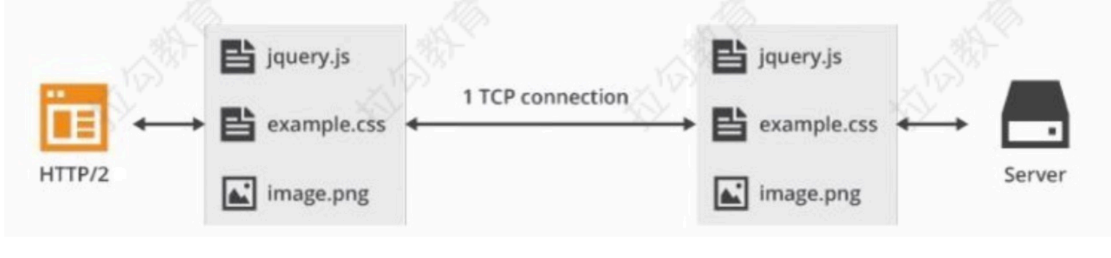


而且每次请求除了请求体中的内容，请求头中也会包含很多数据，大量请求的情况下也会浪费很多资源。

但是这些问题随着 HTTP 2 的出现，也就不复存在了。



关于 HTTP 1.1 与 HTTP 2 之间的差异，可以通过这个链接体验：https://http2.akamai.com/demo，直观感受下 HTTP/2 比 HTTP/1 到底快了多少。

而且不打包也有一个好处，就是可以把按需加载实现到极致。

> *关于 HTTP 2 的详细介绍，可以参考：*
>
> - *https://blog.fundebug.com/2019/03/07/understand-http2-and-http3/*
> - *https://www.digitalocean.com/community/tutorials/http-1-1-vs-http-2-what-s-the-difference*

开箱即用

- TypeScript - 内置支持
- less/sass/stylus/postcss - 内置支持（需要单独安装所对应的编译器）

特性小结

Vite 带来的优势主要体现在提升开发者在开发过程中的体验。

- Dev Server 无需等待，即时启动；
- 几乎实时的模块热更新；
- 所需文件按需编译，避免编译用不到的文件；
- 开箱即用，避免各种 Loader 和 Plugin 的配置；

实现原理

Vite 的核心功能：Static Server + Compile + HMR

核心思路：

1. 将当前项目目录作为静态文件服务器的根目录
2. 拦截部分文件请求
    a. 处理代码中 import node_modules 中的模块
    b. 处理 vue 单文件组件（SFC）的编译
3. 通过 WebSocket 实现 HMR

手写实现

详细参考 my-vite

```
#!/usr/bin/env node

const path = require('path')
const { Readable } = require('stream')
const Koa = require('koa')
const send = require('koa-send')
const compilerSfc = require('@vue/compiler-sfc')

const cwd = process.cwd()

const streamToString = stream => new Promise((resolve, reject) => {
  const chunks = []
  stream.on('data', chunk => chunks.push(chunk))
  stream.on('end', () =>
resolve(Buffer.concat(chunks).toString('utf8')))
  stream.on('error', reject)
})

const app = new Koa()

// 重写请求路径，/@modules/xxx => /node_modules/
app.use(async (ctx, next) => {
  if (ctx.path.startsWith('/@modules/')) {
    const moduleName = ctx.path.substr(10) // => vue
    const modulePkg = require(path.join(cwd, 'node_modules',
moduleName, 'package.json'))
    ctx.path = path.join('/node_modules', moduleName,
modulePkg.module)
  }
```

```
  await next()
})

// 根据请求路径得到相应文件 /index.html
app.use(async (ctx, next) => {
  // ctx.path // http://localhost:3080/
  // ctx.body = 'my-vite'
  await send(ctx, ctx.path, { root: cwd, index: 'index.html' }) // 有可能还需要额外处理相应结果
  await next()
})

// .vue 文件请求的处理，即时编译
app.use(async (ctx, next) => {
  if (ctx.path.endsWith('.vue')) {
    const contents = await streamToString(ctx.body)
    const { descriptor } = compilerSfc.parse(contents)
    let code

    if (ctx.query.type === undefined) {
      code = descriptor.script.content
      code = code.replace(/export\s+default\s+/, 'const __script = ')
      code += `
  import { render as __render } from "${ctx.path}?type=template"
  __script.render = __render
  export default __script`
      // console.log(code)
      ctx.type = 'application/javascript'
      ctx.body = Readable.from(Buffer.from(code))
    } else if (ctx.query.type === 'template') {
      const templateRender = compilerSfc.compileTemplate({ source: descriptor.template.content })
      code = templateRender.code
    }

    ctx.type = 'application/javascript'
    ctx.body = Readable.from(Buffer.from(code))
  }
  await next()
})

// 替换代码中特殊位置
app.use(async (ctx, next) => {
  if (ctx.type === 'application/javascript') {
    const contents = await streamToString(ctx.body)
    ctx.body = contents
      .replace(/(from\s+['"])(?![\.\/])/g, '$1/@modules/')
```

```
      .replace(/process\.env\.NODE_ENV/g, '"production"')
  }
})

app.listen(3080)

console.log('Server running @ http://localhost:3080')
```