这节课开始以项目形式编写vue前端项目，因此会用到一系列脚手架，项目的结构我们可以采用脚手架工具vue-cli或者vite来创建

安装node

需要提前安装node，没有安装或者不会安装的可以访问官网

```
https://nodejs.org/zh-cn/
```

下载长期维护版本安装即可。安装完成后会自动给你添加环境变量，因此有两个命令

```
C:\Users\Shone>node --version
v12.14.0

C:\Users\Shone>npm --version
6.13.4
```

只要是12版本以上即可,14也可以

此外，npm下载需要设置国内源，不然速度太慢了

```
# 配置指向源
$ npm config set registry http://registry.npm.taobao.org
```

不推荐再安装cnpm, npm配置了源已经解决网络慢的问题了。

npm是node的默认包管理器，除此以外还有个更优秀的包管理器叫yarn

```
https://yarnpkg.com/getting-started/install
```

可以参考官方文档安装使用yarn.

创建项目

使用vue-cli创建项目（首选）

```
## 查看@vue/cli版本，确保@vue/cli版本在4.5.0以上
vue --version
## 安装或者升级你的@vue/cli
npm install -g @vue/cli     #安装或升级       npm update -g @vue/cli
#或者 yarn命令安装
yarn global add @vue/cli    #安装或升级       yarn global upgrade --latest
@vue/cli
#查看版本
vue --version   # @vue/cli 4.5.13
## 创建
vue create vue_test
## 选择手动模式---上下键移动选择，Enter键选中
Vue CLI v4.5.13
? Please pick a preset:
  Default ([Vue 2] babel, eslint)
  Default (Vue 3) ([Vue 3] babel, eslint)
```

松勤songqin

```
> Manually select features
# 选中项---上下键移动选择，Space键选中(以下打*的选中)，选好后Enter
 (*) Choose Vue version
 (*) Babel
 ( ) TypeScript
 ( ) Progressive Web App (PWA) Support
 (*) Router
 (*) Vuex
>(*) CSS Pre-processors
 (*) Linter / Formatter
 ( ) Unit Testing
 ( ) E2E Testing
## 选择Vue版本--3.x
? Choose a version of Vue.js that you want to start the project with
  2.x
> 3.x
## 使用历史模式--否
Use history mode for router? (Requires proper server setup for index fallback in
production) (Y/n)y

## css预处理器模式 --选择Sass/SCSS (with dart-sass)
Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by
default): (Use arrow keys)
> Sass/SCSS (with dart-sass)
  Sass/SCSS (with node-sass)
  Less
  Stylus
# ESlint代码检查--选择第一个
? Pick a linter / formatter config: (Use arrow keys)
> ESLint with error prevention only
  ESLint + Airbnb config
  ESLint + Standard config
  ESLint + Prettier
# 何时检查？ --选择第一个
Pick additional lint features: (Press <space> to select, <a> to toggle all, <i>
to invert selection)
>(*) Lint on save
 ( ) Lint and fix on commit
## 配置文件--选择package.json
Where do you prefer placing config for Babel, ESLint, etc.?
  In dedicated config files
> In package.json
## 是否保存配置？--可以根据你的需求选择，如果是初学者，这里选择N 后面再多创建几次熟练一下
Save this as a preset for future projects? (y/N)

## 启动
cd vue_test
npm run serve 或 yarn serve
```

松勤songqin

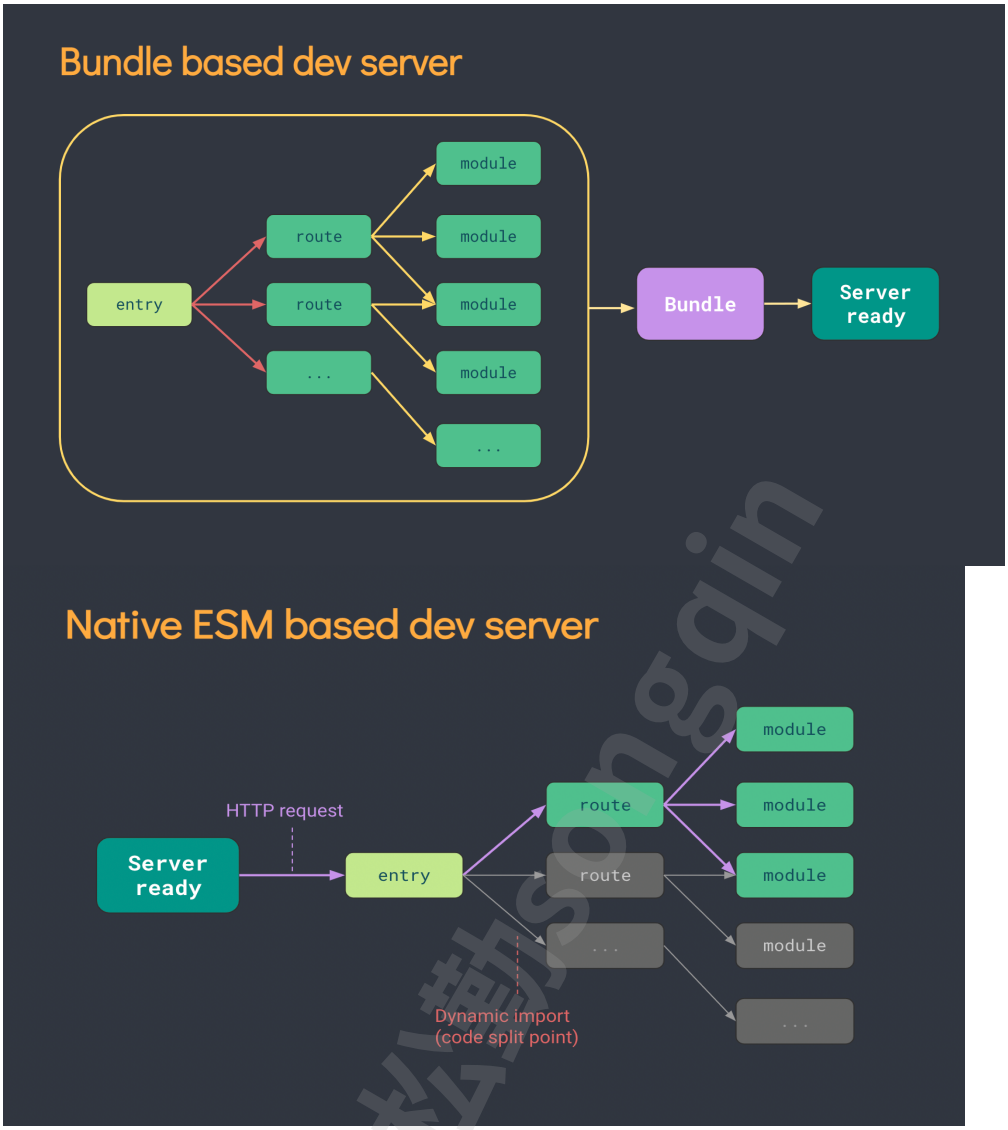
vite快速创建项目（选择2）

官方文档：https://v3.cn.vuejs.org/guide/installation.html#vite

vite官网：https://vitejs.cn

- 什么是vite？——尤雨溪大大亲自为Vue编写的一款打包工具，号称 新一代前端构建工具。

- 优势如下：
  - 开发环境中，无需打包操作，可快速的冷启动。
  - 轻量快速的热重载（HMR）。
  - 真正的按需编译，不再等待整个应用编译完成。
- 传统构建 与 vite构建对比图



通过在终端中运行以下命令，可以使用 Vite 快速构建 Vue 项目。

```
使用 npm：
$ npm init vite <project-name> -- --template vue
$ cd <project-name>
$ npm install  # 安装依赖包
$ npm run dev  # 启动项目

或者 yarn：
$ yarn create vite <project-name> --template vue
$ cd <project-name>
$ yarn         # 安装依赖包
$ yarn dev     # 启动项目
```

`<project-name>` 换成项目名称

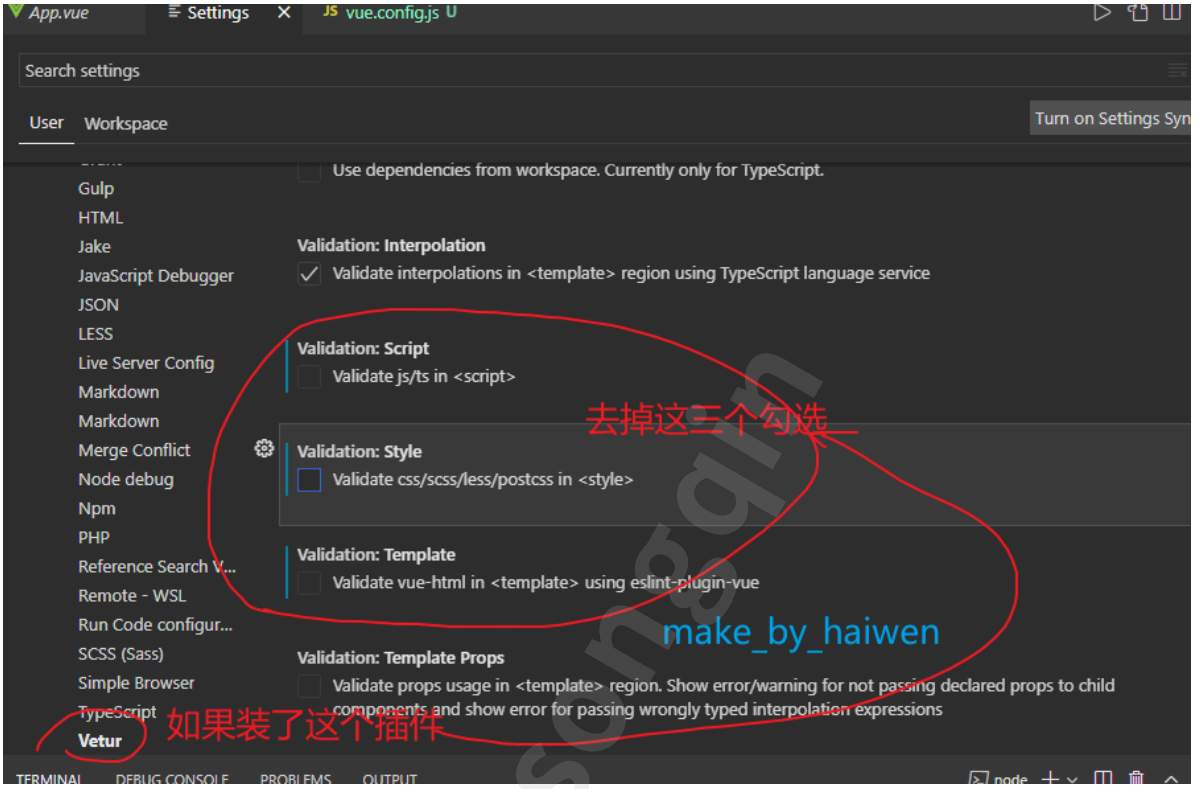
项目初始化

项目根目录创建 vue.config.js文件

```
module.exports={
    lintOnSave:false
}
```

关闭语法检查，不然写一半就要用来纠错不必要的语法问题。

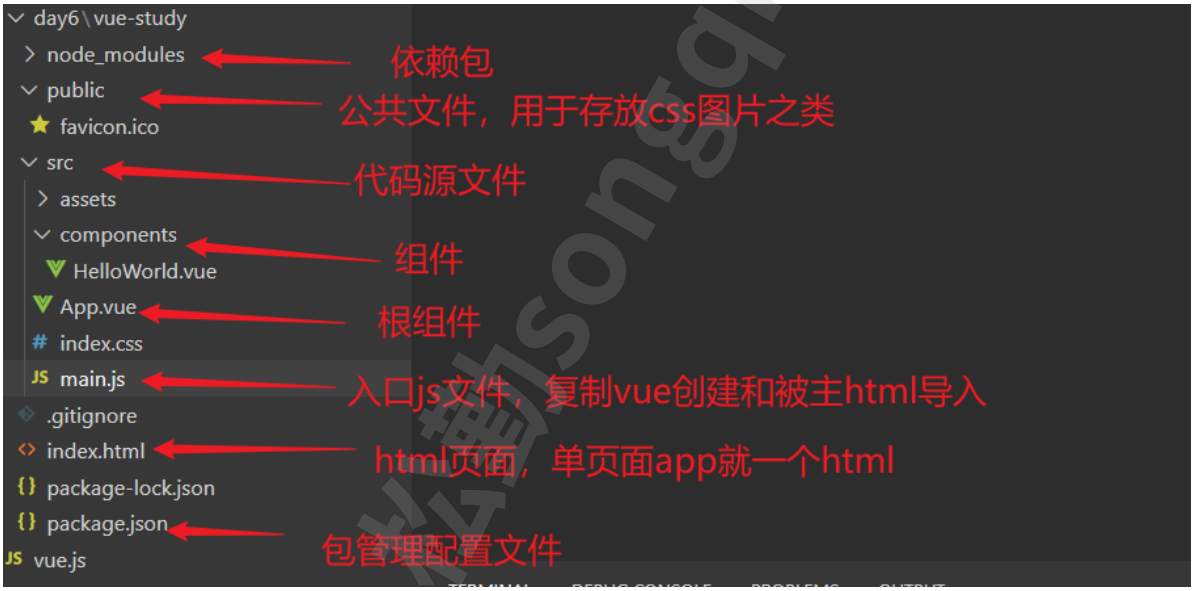
关闭Vetur插件的模板语法检测（如果安装了此插件）消除模板上莫名其妙的报错。



目录结构介绍

Vue-Cli初始化的项目结构：

Vite初始化的项目结构：



基本上大同小异，项目结构并不是一成不变，后期随着项目发展，可以根据实际情况不断更新目录结构。

组件文件

我们可以看到这种带.vue后置名的文件，这个文件是干嘛的呢?

点进去看下Home.vue

```
<template>
  <div class="home">
    <img alt="Vue logo" src="../assets/logo.png">
    <HelloWorld msg="Welcome to Your Vue.js App"/>
  </div>
</template>

<script>
// @ is an alias to /src
```

```
import HelloWorld from '@/components/HelloWorld.vue'

export default {
  name: 'Home',
  components: {
    HelloWorld
  }
}
</script>
```

发现这个其实就是vue组件，之前我们是写在一起的，但是发现这样不方便管理和扩展。于是在工程化的目录中，组件从代码独立出来了，通过文件形式来管理。这样更加便于我们写代码。

关于组件的文件规范可以查看官方文档 https://v3.cn.vuejs.org/api/sfc-spec.html

组件的基础就不多做介绍了，这里说明下注意点。

首选，HTML模板部分通过 `<template>` 标签定义。

其次，代码写在script标签中

```
// @ is an alias to /src
import HelloWorld from '@/components/HelloWorld.vue'    //从其他文件导入组件

export default {      // 导出组件内容 其他组件才能导入（引用）这个组件
  name: 'Home',       // 组件名称，其他组件导入时使用
  components: {       // 引用其他组件
    HelloWorld
  }
}
```

组件小案例

定义组件文件

`src/components/Haiwen.vue`

```
<template>
  <div>
    <h3>hello {{name}}</h3>
  </div>
</template>

<script>
export default {
  name: 'Haiwen',
  data(){
    return{
      name: 'haiwen'
    }
  }
}
</script>
```

松勤songqin

使用组件文件

根组件下 `src/App.vue`

```
<template>
  <h3>
    这里是主页
  </h3>
  <haiwen/>
</template>

<script>
import Haiwen from '@/components/Haiwen'  //使用模块语法引入组件
export default {
  name: 'App',
  components:{
    Haiwen
  }
}
</script>
```

保存，刷新页面。

组合式API

拉开序幕的setup

回顾下之前的Vue创建实例写法，这种通过选项定义组件的方式叫做选项式API（Options API）

```
    const app = Vue.createApp({
      data(){
        ...
      },
      methods:{
        ...
      },
      computed:{
       ...
      },
      watch:{
       ...
      }
    })
    const vm = app.mount('#haiwen')
```

相对来说组合式API（composition Api)是Vue3对比Vue2的一个最大区别。

从OptionAPI到CompositionAPI的过度就是从选项式编程到函数编程的过度。

后面大型项目想更好的管理代码，提高改代码的复用性，推荐使用组合式API（composition Api)。

组合式API的定义就在Setup()中。

我们先来将一个简单选项式API改造成组合式API

template:

松勤songqin

```
<template>

  <button @click="chilema">吃饭/消化</button>
  <h3 v-if="ate">吃饱了</h3>
</template>
```

选项式API

```
export default {
  name: 'App',
  data(){
    let ate=false
    return{
      ate
    }
  },
  methods:{
    chilema(){
      this.ate=!this.ate
    }
  }
}
```

组合式API改造

```
import {ref} from 'vue'
export default {
  name: 'App',

  setup(props) {
    let ate= ref(ate) //响应式数据需要通过ref包裹

    function chilema(){
      ate.value=!ate.value    //数据类型需要通过value
    }
    return{
      ate,
      chilema
    }
  }
}
```

内部的响应式

ref

将简单数据类型转成响应式数据

松勤songqin

```
import {ref} from 'vue'

let ate= ref(ate) //响应式数据需要通过ref包裹
console.log(ate.value)  //数据内容需要通过value访问或赋值

const person = ref({
    name:'xiaoming',
    age:18
})
console.log(person.value.name) //访问复杂数据也是如此--通过value
```

reactive

若复杂数据还要通过value访问，这个着实比较麻烦,用reactive就可以了。ref底层在包裹复杂结构数据时也是调用的reactive。

```
import {ref,reactive} from 'vue'
const person = reactive({
    name:'xiaoming',
    age:18
})
console.log(person.name) //直接访问 不需要调用value了
```

聪明的你肯定会想，reactive这么方便，那我简单数据类型也用reactive? 好，试试看。

```
let ate= reactive(false)
console.log(ate)
```

也确实能打印，但是没有响应式效果了。并且vue给出了警告

```
value cannot be made reactive: false
```

组合式API的this问题

组合式api没有this

模板引用ref

Vue获取元素的手段：ref

用法：模板

```
  <div ref="root">
    <button @click="chilema">吃饭/消化</button>
    <h3 v-if="ate">吃饱了</h3>
    <h3>ate:{{ate}}</h3>
  </div>
```

脚本：

松勤songqin

```
setup() {
  const root = ref(null) //获取ele的dom引用，变量名需要和模板中的ref属性值相同
  onMounted(() => {
    console.log(root.value.tagName) //加载之后才能看到数据
  })
 return {
  root  //一定要返回ref数据，这样模板才能引用到
}
```

关于组合式API中的生命周期

你可以通过在生命周期钩子前面加上 “on” 来访问组件的生命周期钩子。

下表包含如何在 setup () 内部调用生命周期钩子，即选项式API与组合式API生命周期钩子的对应关系。

选项式 API	`setup` 内的生命周期钩子
`beforeCreate`	Not needed*
`created`	Not needed*
`beforeMount`	`onBeforeMount`
`mounted`	`onMounted`
`beforeUpdate`	`onBeforeUpdate`
`updated`	`onUpdated`
`beforeUnmount`	`onBeforeUnmount`
`unmounted`	`onUnmounted`
`errorCaptured`	`onErrorCaptured`
`renderTracked`	`onRenderTracked`
`renderTriggered`	`onRenderTriggered`
`activated`	`onActivated`
`deactivated`	`onDeactivated`

使用方式

1. 导入生命周期钩子

```
import {ref,reactive,onMounted} from 'vue'
```

2. 在setup内使用生命周期钩子，内部传入回调函数

```
onMounted(() => {
  console.log(ele.value)
})
```

vscode 命令行提示：因为在此系统上禁止运行脚本(node命令)

1. 管理员身份打开powerShell
2. 输入`set-ExecutionPolicy RemoteSigned`，回车
3. 输入Y，回车

松勤songqin