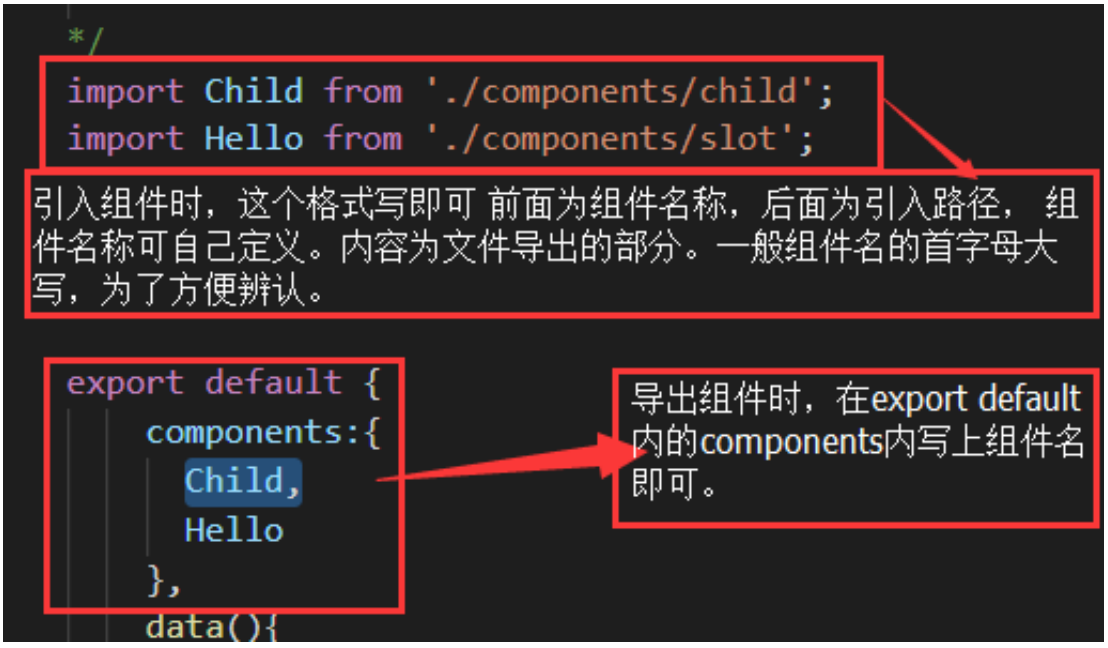
1/8

vue项目

引入组件，不用Vue.comonent( )，而是在导出的对象上挂一个components的属性，属性里面为一个对象，key值和value一样的情况下，写一个即可。
在Vue/cli中引入的子组件名称可以首字母大写，方便认识为组件。



木偶组件（为了接受数据、渲染数据、基本上是没有逻辑的，越往下越木偶）
功能组件（更多的是控制数据，有大量的逻辑。越往顶层越功能 ）

ref（相当于id）

ref：
在Vue中 拿DOM也可以通过 定义 ref 相当于id
当做给元素或者组件加了一个id，可以通过this.$refs去找它们（此处就是多一个”s”）。
使用技巧：如果在操作数据之后，还要操作ref的DOM元素，可以使用nextTick，方便操作。

```
<!--例如在template中写：一个ref-->
<input type="text" ref="txt">


<!--然后在script中的 export default 中的methods下面的方法里面的函数事件中：-->
console.log(this.$refs.txt)
```

```
<!-- 这样打印出来的是  ref="txt" 这个标签 -->
```

如果在mounted下，不能准确的拿到DOM元素（例如一开始没出来，等触发什么条件时才会出现），那么用nextTick（加载完成后）

```
// template 部分：
<input
    type="text"
    ref="txt"
    v-show="val"  // 当val这个数据为真时，这个便签会显现出来。
>


//export default 部分：

data () {
    return{
        val:''    // 数据默认为空，空时为false
    }
}

mounted(){
    this.val='1';  // 数据有内容时，则为true
    this.$nextTick(()=>{  // 如果没有$nextTick 包裹里面的操作不会执行，
        this.$refs.txt.focus();  // 这个标签自动聚焦，但是因为这个标签一开始是隐藏的（val为空），所以此处无法找到这个标签执行，只有外面包裹this.$nextTick 才能找到这个标签，并在上面进行操作。
    })
}

// 如果给val赋值，使它为true，和不包裹this.$nextTick()直接进行自动聚焦的操作时，   不管是在mounted还是在methods的事件函数内。都不会拿到这个标签。如果是在点击事件函数内，那么第一次点击时，会显现出来，第二次点击会聚焦（用户体验不好）

//所以如果两者都在事件函数内，也是需要用this.$nextTick()包裹的。

// 但是如果只有给val赋值的在mounted（初始化）里面，自动聚焦的操作在事件函数内，或者定时器内，则不用this.$nextTick()包裹也能自动聚焦。
```

slot 插槽

slot：插槽
定义：引入子组件，如果不定义某个默认的结构或者样式，那么直接写子组件的名称即可

如果要自定义：（当场总结的有待验证）

- 1、 子组件必须为双标签
  - 如果是说 `<slot name="content"><div>中间的内容</div></slot>` 这样的双标签的话，那么这句话不一定对，去掉div标签 也可以的，如果内容不多，可以直接放在slot标签内，如果内容多，而且多层的话，最好另用标签包裹。
- 2、子组件内写上自定义的结构或者样式，并且加上slot的属性。而这个属性名要对应默认slot的名字。（如果要设置样式，slot标签的元素内不能直接写类名，需要在下面用标签包裹然后写类名，类名必须为slot的名字）
- 3、在子组件中，包一个slot的元素，可以任意修改，在slot元素上定义一个叫做name的属性，为了好对应。

slot：自我总结：

- 作用：可以看成是**替换**，body（innerHTML）里面的内容替换掉组件（component）内的默认内容。
- 通俗点说：就是组件（component）的slot标签就是一个个卡槽。里面有默认内容，如果不插卡，就是显示（执行）默认内容，如果插卡就根据插的卡的内容（body里的内容）来执行。默认内容就会被插入的内容替换掉。
- 书面的说法就是：**引入组件之后 行内（body）再写的东西可以用子组件的模板内的插槽接收，插槽内的内容被覆盖。**
- 那么怎么确定是插入那个插槽呢： 那就需要给每个插槽设置一个名字：name= “a”–> `<slot name="a"><slot>` ；想插入那个插槽时就在行内写上slot：”插槽名” –> `<div slot="a"></div>` 即可。
- 页面显示的顺序还是按组件里面的顺序来。只是被替换内容，不替换前后顺序。

```
<!-- 简单写一下，只写主要的部分，其他部分还是原来的操作-->

<div id="app">
    <ppa>
        <div slot="a">配置</div>
    </ppa>
</div>

<script>
    Vue.component("ppa"{
        template:`
            <div>
                <slot name="a">默认</slot>
            </div>`
    })
</script>
```

```
<body>
    <div id="app">
```

```
        <ppa>
            <div slot="content">修身</div>
            <div slot="footer">修脚</div>
            <!-- <div class="xob" slot="box"></div> -->
        </ppa>
    </div>
<script src="vue.js"></script>
<script>
    Vue.component('ppa',{
        template:`
            <div>
                <slot name="box">
                    <div class="box" ></div>
                </slot>
                <slot name="content">
                    <div>中间的内容</div>
                </slot>
                <slot name="footer">底部的内容</slot>
            </div>
        `
    });

    new Vue({
        el:'#app'
    });
</script>
```

is 解决组件本身和使用时的冲突

is：
有些时候组件本身和使用组件的地方有冲突，
比如：在table中使用div组件，这个时候组件会到table外面去。在这个时候可以通过is的方式吧组件嵌套到这个table中

```
<body>
    <div id="app">
        <button
            v-for="i in 3"
            @click="click(i)"
        >按钮{{i}}</button>
        <div :is="h"></div>
        <table>
            <!-- <ppa></ppa> -->
            <!-- <tr is="ppa"></tr> -->
            <!-- <tr>
```

```
                <td><ppa /></td>
            </tr> -->
        </table>
    </div>

<script src="vue.js"></script>

<script>
    let  ppx = {
        template:`<div>111111111111</div>`
    }
    let  ppx2 = {
        template:`<div>22222222222</div>`
    }
    let  ppx3 = {
        template:`<div>33333333333333</div>`
    }

    new Vue({
        el:'#app',
        data:{
            h:'ppx'
        },
        methods:{
            click(key){
                switch (key) {
                    case 1:
                        this.h = 'ppx';
                        break;
                    case 2:
                        this.h = 'ppx2';
                        break;
                    case 3:
                        this.h = 'ppx3';
                        break;
                    default:
                        this.h = 'ppx';
                        break;
                }
            }
        },
        components:{
            ppx,
            ppx2,
            ppx3
        }
    })
</script>
</body>
```

用express初始化项目目录：

主要步骤：

- 1、npm install express-generator -g ->安装express的脚手架
- 2、express myapp (maapp 是项目的名称)
- 3、npm install 安装
- 运行：npm run start

ejs

- ejs –> js模板（这个知识也是要知道的）
- <%= xxx %> => 后端模板

```
// 逐步发展：
  <%for(var i=0;i<arr.length;i++){%>
        <li><%= arr[i]%></li>
  <%}%>
    ->  {} -> {{}}

// 模板：每一行都要用 " <> " 包裹
    <% for(var i=0;i<arr.length;i++){<li><%=arr[i]%></li>} %>

    <% for(var i=0;i<arr.length;i++){ %>
        <li><%=arr[i]%></li>
    <% } %>
```

- `<%` ‘脚本’ 标签，用于流程控制，无输出。
- `<%_` 删除其前面的空格符
- `<%=` 输出数据到模板（输出是转义 HTML 标签）
- `<%-` 输出非转义的数据到模板
- `<%#` 注释标签，不执行、不输出内容
- `<%%` 输出字符串 ‘<%’
- `%>` 一般结束标签
- `-%>` 删除紧随其后的换行符
- `_%>` 将结束标签后面的空格符删除

ejs 的标签系统非常简单，它只有以下三种标签：

- <% code %>： JavaScript 代码。
- <%= code %>：显示替换过 HTML 特殊字符的内容。
- <%- code %>： 显示原始 HTML 内容。

注意：

<%= code %> 和 <%- code %> 的区别，当变量 code 为普通字符串时，两者没有区别。当 code 比如为 <h1>hello</h1> 这种字符串时， <%= code %> 会原样输出 <h1>hello</h1> ，而 <%- code %> 则会显示 H1 大的 hello 字符串。

vue中的路由：

Vue Router：使用过程vue中的路由管理器

- 1、安装：npm install vue-router

```
// router.js 中的内容

// 第2步 引包
import VueRouter form 'vue-router'
<!-- 在route.js中，先进行组件引入，定义引入的组件的名称，-->
// 第3步 使用（引用）
Vue.use(VuRouter)

// 第4步 需要配置router  ->  route.js
<!-- 在export default new Router中设置router，里面的内容为：path是路由内容，根据路由内容决定使用哪个组件：component：aa  （aa是组件名） -->

//
```

用vue-cli 直接vue create 项目名称就搞定了。

路由

配置路由表：配置什么路径就显示什么组件
‘/’ -> app
‘/ppa’ -> ppa

```
// 创建变量
const router = [
```

```
    {
        path:'/',
        component:app
    },
    {
        path:'./ppa',
        component:ppa
    }
]


// 导出这个变量，方便其他文件引入时使用
export default new VueRouter({
    routes
})
```