App.vue这个东西，就是嵌套在index.html中。

App.vue里其实也差不多是空的，我们暂时只关注最上边的<template>，有一行是<router-view></router-view>

**App.vue调用了Hello.vue这个组件，是通过router来控制的显示**

打开router文件夹下的index.js:

1. 上述的小实验已经充分证明了，index.html, App.vue, Hello.vue之间的关系。一般来说，index.html是不用动的（顶多增加一些css头文件）。所以我们开始修改的其实文件就是App.vue，然后鉴于我们想要添加文件，所以需要修改router里边的index.js文件，最后鉴于我们要添加新页面和组件，我们还需要增加一些components文件夹里边的东西。
2. 如果按照目前的index.js的结构，如果你怎么修改那个Hello.vue文件，或者增加其他组件，它们只有一个归宿：就是躺在App.vue里的那个Logo图像的下面。因为App.vue是父组件，而Hello.vue只不过是它的子组件。但是这并不意味着谁重要谁难写，因为如果你把App.vue里边那个logo那一行注释掉，App.vue就什么都无法呈现，仅仅是为Hello.vue以及未来的一些页面提供了一个承载框架而已。
3. 未来开发完的网站里，用户感觉上可能是一个页面和一个url的单独对应，就像是曾经的每一个PHP文件对应每一个url一样，但实际上，可能很多页面，都像Hello.vue一样，仅仅是App.vue的子组件而已。
4. 目前在App.vue里，有Hello组件；以后我们在Hello组件里，还可以插入其他的组件，当然不一定是这种使用router的办法，因为用这个模式，每一个组件都会有一个对应的url，如果你的页面上有三个tab，可能没有必要每一个tab对应一个url吧？但目前要理解的就是：**我个人觉得Vue最核心的设计理念，就是把一个看起来很完整的页面，分散到了每一个组件，这个组件可能大道一个页面，Hello.vue，也可能小到一个按钮，最终我们看到的页面，是这一群组件的堆砌的结果，这样做的一大好处就是，每一个组件都可以单独进行与数据库和用户的交互，开发更为清晰。**

安装elementUI

npm i element-ui -S