p01：Next.js简介和创建项目

Next.js简介

Next.js 是一个轻量级的 React 服务端渲染应用框架。

用一个框架，就要知道它的优点（或者是解决了我们什么问题）：

- 完善的React项目架构，搭建轻松。比如：Webpack配置，服务器启动，路由配置，缓存能力，这些在它内部已经完善的为我们搭建完成了。
- 自带数据同步策略，解决服务端渲染最大难点。把服务端渲染好的数据，拿到客户端重用，这个在没有框架的时候，是非常复杂和困难的。有了Next.js，它为我们提供了非常好的解决方法，让我们轻松的就可以实现这些步骤。
- 丰富的插件帮开发人员增加各种功能。每个项目的需求都是不一样的，包罗万象。无所不有，它为我们提供了插件机制，让我们可以在使用的时候按需使用。你也可以自己写一个插件，让别人来使用。
- 灵活的配置，让开发变的更简单。它提供很多灵活的配置项，可以根据项目要求的不同快速灵活的进行配置。

目前Next.js是React服务端渲染的最佳解决方案，所以如果你想使用React来开发需要SEO的应用，基本上就要使用Next.js。

手动创建Next.js项目

第一步： 建立文件夹

创建一个`Next.js`项目，可以有两种方法进行，一种是手动创建，另一种是用`create-next-app`（脚手架）来创建。这节课我们先来进行手动创建，这样虽然麻烦点，但是可以更容易让新手了解过程和原理。

先在你喜欢的位置新建一个文件夹，名称你也可以自己起，我这里是在D盘里建立了一个叫`NextDemo`文件夹。

```
D:
mkdir NextDemo
npm init
```

这里的`npm init` 是用来把文件夹初始化成可管理的项目的，其实就是在根目录里给你添加了一个`package.json`的文件。

第二步：安装所需要的依赖包

接下来可以使用yarn来安装所需要的项目依赖包，先来安装下面三个`react`、`react-dom`和`next`。

```
yarn add react react-dom next
```

当让你也可以使用npm来进行安装，npm安装时记得要使用--save

```
npm install --save react react-dom next
```

安装完可以打开package.json文件查看一下dependencies的版本。

第三步：增加快捷命令

为了开发时简便的使用Next.js中的操作命令行工具，所以把常用的配置到package.json中，代码如下：

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "dev" : "next" ,
    "build" : " next build",
    "start" : "next start"
 },
```

第四步：创建pages文件夹和文件

在根目录下，创建一个pages文件夹，这个文件夹是Next规定的，在这个文件夹下写入的文件，Next.js会自动创建对应的路由。有了文件夹以后，在文件下面创建一个index.js文件，这就是我们的首页了，然后用React Hooks的写法，写个最简单的Hello World。

```
function Index(){
    return (
        <div>Hello Next.js</div>
    )
}
export default Index
```

写好后在终端中使用yarn dev来打开预览，在浏览器中可以看到输出了正确的结果。这节课就到这里，你可以试着去联系一下这种Next.js项目的搭建方法，因为这会让我们更明白Next项目的来龙去脉。

p02：creact-next-app快速创建Next.js项目

create-next-app可以快速的创建Next.js项目，它就是一个脚手架，有了它只要一句命令就可以把项目需要的依赖包和基本目录都生成，工作中我基本不用手动的形式自己创建，全部使用create-next-app来创建。

安装create-next-app

使用脚手架前，需要先进行全局安装。

```
npm install -g create-next-app
```

安装完成后，就可以通过create-next-app命令来创建一个Next.js的项目了。

创建Next.js项目

目前可以支持三种方式的创建，分别是用npx,yarn和create-next-app命令来进行安装，安装的结构都是完全一样的，所以就给大家演示其中的一种npx的形式。

npx 是Node自带的npm模块，所以你只要安装了Node都是可以直接使用npx命令的。但低版本的Node是不带这个命令的，所以你需要手都安装一下。

```
$ npm install -g npx
```

打开命令提示符工具，然后进入D盘，然后直接用下面的npx命令创建项目。

```
$ npx create-next-appnext-create
```

输入后按回车，就会自动给我们进行安装项目需要的依赖。并且会给我们添加好命令。稍等一会，全部安装完成后，可以进入项目母乳，执行yarn dev来测试项目。
在浏览器中输入http://localhost:3000/,看到下面的内容，说明项目生成成功.

项目结构介绍

看到结果后，用VSCode打开目录，可以看到已经有了很多自动建立好的文件和文件夹，下面就简单的介绍一下这些它们的用处：

- components文件夹:这里是专门放置自己写的组件的，这里的组件不包括页面，指公用的或者有专门用途的组件。
- node_modules文件夹：Next项目的所有依赖包都在这里，一般我们不会修改和编辑这里的内容。
- pages文件夹：这里是放置页面的，这里边的内容会自动生成路由，并在服务器端渲染，渲染好后进行数据同步。
- static文件夹： 这个是静态文件夹，比如项目需要的图片、图标和静态资源都可以放到这里。
- .gitignore文件： 这个主要是控制git提交和上传文件的，简称就是忽略提交。
- package.json文件：定义了项目所需要的文件和项目的配置信息（名称、版本和许可证），最主要的是使用npm install 就可以下载项目所需要的所有包。

当你了解项目目录和文件后就可以试着修改一下项目，简单的尝试一下了。这节课就到这里了，主要讲解的就是利用create-next-app来创建项目和生成项目的基本结构介绍。

p03：Next.js的Page和Component的使用

上节课已经利用create-next-app创建了项目，也简单的介绍了一下创建后的项目结构。这节课就来看看如何新建页面和新建组件。

新建页面和访问路径

直接在根目录下的pages文件夹下，新建一个jspang.js页面。然后写入下面的代码：

```
function Jspang(){
    return (<button>技术胖</button>)
}
```

```
4
5 export default  Jspang;
```

只要写完上面的代码，Next框架就自动作好了路由，这个也算是Next的一个重要优点，给我们节省了大量的时间。
现在要作一个更深的页面，比如把有关博客的界面都放在这样的路径下http://localhost:3000/blog/nextBlog,其实只要在pages文件夹下再建立一个新的文件夹blog，然后进入blog文件夹，新建一个nextBlog.js文件，就可以实现了。
nextBlog.js文件内容,我们这里就用最简单的写法了

```
1 export default ()=><div>nextBlog page</div>
```

写完后，就可以直接在浏览器中访问了，是不是发现Next框架真的减轻了我们大量的工作。

Component组件的制作

制作组件也同样方便，比如要建立一个jspang组件，直接在components目录下建立一个文件jspang.js，然后写入下面代码:

```
1 export default ({children})=><button>{children}button>
```

组件写完后需要先引入，比如我们在Index页面里进行引入：

```
1 import Jspang from'../components/jspang'
```

使用就非常简单了，直接写入标签就可以。

```
1 <Jspang>按钮Jspang>
```

一个自定义组件的创建和使用也是这么简单， 如果你React的基础很好，那这节课的内容对你来说就更加简单了。也就是说Next框架并没有给我们带来太多的学习成本，但是为我们减轻了很多配置工作。

p04：路由-基础和基本跳转

学会编写组件和页面后，下一步应该了解的就是路由体系，每个框架都有着不同的路由体系，这节先学习最基础的页面如何跳转。页面跳转一般有两种形式，第一种是利用标签<Link>,第二种是用js编程的方式进行跳转，也就是利用Router组件。先来看一下标签的形式如何跳转。

标签式导航

在编写代码之前，先删除index.js中的代码，保证代码的最小化。使用标签式导航需要先进行引入，代码如下:

```
1 import Link from'next/link'
```

然后新建两个页面`jspangA.js`和`jspangB.js`，新建后写个最简单的页面，能标识出来A、B两个页面就好。

```
//jspangA.js
import Link from 'next/link'

export default ()=>(
    <>
        <div>Jspang-A page .  </div>
        <Link href="/"><a>返回首页</a></Link>
    </>
)
```

写完A页面后，可以直接复制A页面的内容，然后修改一下就是B页面。

```
//jspangB.js

import Link from 'next/link'

export default ()=>(
    <>
        <div>Jspang-B page .  </div>
        <Link href="/"><a>返回首页</a></Link>
    </>
)
```

有了两个页面后，可以编写首页的代码，实现跳转了。

```
//index.js
import React from 'react'
import Link from 'next/link'


const Home = () => (
  <>
    <div>我是首页</div>
    <div><Link href="/jspangA"><a>去JspangA页面</a></Link></div>
    <div><Link href="/jspangB"><a>去JspangB页面</a></Link></div>

```

```
    </>
)

export default Home
```

用标签进行跳转是非常容易的，但是又一个小坑需要你注意一下，就是他不支持兄弟标签并列的情况。

```
<div>
  <Link href="/jspangA">
    <span>去JspangA页面</span>
    <span>前端博客</span>
  </Link>
</div>
```

如果这样写会直接报错，报错信息如下

```
client pings, but there's no entry for page: /_errorWarning: You're using a strin
```

但是你可以把这两个标签外边套一个父标签，就可以了，比如下面的代码就没有错误。

```
<Link href="/jspangA">
  <a>
    <span>去JspangA页面</span>
    <span>前端博客</span>
  </a>
</Link>
```

通过标签跳转非常的简单，跟使用标签几乎一样。那再来看看如何用编程的方式进行跳转。

Router模块进行跳转

在Next框架中还可以使用Router模块进行编程式的跳转，使用前也需要我们引入Router，代码如下：

```
import Router from'next/router'
```

然后在Index.js页面中加入，直接使用Router进行跳转就可以了。

```
<div>
    <button onClick={()=>{Router.push('/jspangA')}}>去JspangA页面</button>
</div>
```

这样写只是简单，但是还是耦合性太高，跟Link标签没什么区别，你可以修改一下代码，把跳转放到一个方法里，然后调用方法。

```
import React from 'react'
import Link from 'next/link'
import Router from 'next/router'
const Home = () => {
  function gotoA(){
    Router.push('/jspangA')
  }
  return(
    <>
      <div>我是首页</div>
      <div>
        <Link href="/jspangA">
          <a>
            <span>去JspangA页面</span>
            <span>前端博客</span>
          </a>
        </Link>
      </div>
      <div><Link href="/jspangB"><a>去JspangB页面</a></Link></div>
      <div>
        <button onClick={gotoA}>去JspangA页面</button>
      </div>
    </>
  )

}
export default Home
```

这样也是可以实现跳转的，而且耦合性也降低了,所以个人更喜欢这种跳转方式。这节课主要学习了Next的两种跳转方式，第一种是标签式跳转，第二种是编程式跳转。

p05：路由-跳转时用query传递和接受参数

项目开发中一般都不是简单的静态跳转，而是需要动态跳转的。动态跳转就是跳转时需要带一个参数或几个参数过去，然后在到达的页面接受这个传递的参数，并根据参数不同显示不同的内容。比如新闻列表，然后点击一个要看的新闻就会跳转到具体内容。这些类似这样的需求都都是通过传递参数实现的。

只能用query传递参数

这节课作一个“找小姐姐”的例子，通过这个例子来通俗易懂的讲解一下路由带参数的知识。在Next.js中只能通过通过query（?id=1）来传递参数，而不能通过(path:id)的形式传递参数，这个一定要记住，在工作中很容易就容易记混。

现在我们改写一下pages文件夹下的index.js文件。

```
import React from 'react'
import Link from 'next/link'
import Router from 'next/router'
const Home = () => {
  return(
    <>
      <div>我是首页</div>
      <div>
        <Link href="/xiaojiejie?name=波多野结衣"><a>选波多野结衣</a></Link><br/>
        <Link href="/xiaojiejie?name=苍井空"><a>选苍井空</a></Link>
      </div>
    </>
  )

}
export default Home
```

这样编写query参数就可以进行传递过去了，接下来就是要接受参数了。

接收传递过来的参数

现在还没有小姐姐对应的页面，所以我们要创建xiaojiejie.js页面，并写下下面的代码。

```
import { withRouter} from 'next/router'
import Link from 'next/link'

const Xiaojiejie = ({router})=>{
    return (
        <>
            <div>{router.query.name},来为我们服务了 .</div>
            <Link href="/"><a>返回首页</a></Link>
        </>
    )
}
```

```
12
13 export default withRouter(Xiaojiejie)
```

withRouter是Next.js框架的高级组件，用来处理路由用的，这里先学简单用法，以后还会学习的。通过这种方式就获得了参数，并显示在页面上了。

编程式跳转传递参数

回了这种标签式跳转传递参数的形式，那编程式跳转如何传递那，其实也可以简单使用?加参数的形式，代码如下：

```
1 <div>
2   <button onClick={gotoXiaojiejie}>选井空</button>
3 </div>
```

```
1 // gotoXiaojiejie
2 function gotoXiaojiejie(){
3     Router.push('/xiaojiejie?name=井空')
4   }
```

这种形式跳转和传递参数是完全没有问题的，但是不太优雅（优雅这东西很难界定，其实你完全可以看成一种装X，这太简单了，我需要装个X），所以也可以写成Object的形式。

```
1  function gotoXiaojiejie(){
2     Router.push({
3       pathname:'/xiaojiejie',
4       query:{
5         name:'井空'
6       }
7     })
8   }
```

嗯，这样写确实优雅很多(我们一定要面向对象编程，有对象比没对象要好)。
其实标签也可以写成这种形式，比如我们把第一个修改成这种形式.

```
1 <Linkhref={{pathname:'/xiaojiejie',query:{name:'结衣'}}}><a>选结衣a>Link><br/>
```

在浏览器中预览一下，如果一切正常是可以顺利进行跳转，并接收到传递的值。这节课主要讲解了Next框架的路由跳转时带参数过去，然后用withRouter进行接收。

p06：路由-六个钩子事件的讲解

路由的钩子事件，也就是当路由发生变化时，可以监听到这些变化事件，执行对应的函数。它一共有六个钩子事件，这节课就学习一下。

routerChangeStart路由发生变化时

在监听路由发生变化时，我们需要用Router组件，然后用on方法来进行监听,在pages文件夹下的index.js，然后写入下面的监听事件，这里我们只打印一句话，就不作其他的事情了。代码如下：

```
1    Router.events.on('routeChangeStart',(...args)=>{
2      console.log('1.routeChangeStart->路由开始变化,参数为:',...args)
3    })
```

这个时路由发生变化时，时间第一时间被监听到，并执行了里边的方法。

routerChangeComplete路由结束变化时

路由变化开始时可以监听到，那结束时也时可以监听到的，这时候监听的事件是routerChangeComplete。

```
1    Router.events.on('routeChangeComplete',(...args)=>{
2      console.log('routeChangeComplete->路由结束变化,参数为:',...args)
3    })
```

beforeHistoryChange浏览器history触发前

history就是HTML中的API，如果这个不了解可以百度了解一下，Next.js路由变化默认都是通过history进行的，所以每次都会调用。不适用history的话，也可以通过hash

```
1    Router.events.on('beforeHistoryChange',(...args)=>{
2      console.log('3,beforeHistoryChange->在改变浏览器 history之前触发,参数为:',...args
3    })
```

routeChangeError路由跳转发生错误时

```
1  Router.events.on('routeChangeError',(...args)=>{
2      console.log('4,routeChangeError->跳转发生错误,参数为:',...args)
3    })
```

需要注意的是404找不到路由页面不算错误，这个我们就不演示了。

转变成hash路由模式

还有两种事件，都是针对hash的，所以现在要转变成hash模式。hash模式下的两个事件hashChangeStart和hashChangeComplete,就都在这里进行编写了。

```
  Router.events.on('hashChangeStart',(...args)=>{
    console.log('5,hashChangeStart->hash跳转开始时执行,参数为:',...args)
  })

  Router.events.on('hashChangeComplete',(...args)=>{
    console.log('6,hashChangeComplete->hash跳转完成时,参数为:',...args)
  })
```

在下面的jsx语法部分，再增加一个链接,使用hash来进行跳转，代码如下：

```
<div>
    <Link href="#jspang"><a>选JSPang</a></Link>
</div>
```

为了方便你学习，我这里给出index.js的全部代码，你可以在练习时进行参考。

```
import React from 'react'
import Link from 'next/link'
import Router from 'next/router'


const Home = () => {

  function gotoXiaojiejie(){
    Router.push({
      pathname:'/xiaojiejie',
      query:{
        name:'井空'
      }
    })
  }



  Router.events.on('routeChangeStart',(...args)=>{
    console.log('1.routeChangeStart->路由开始变化,参数为:',...args)
  })

  Router.events.on('routeChangeComplete',(...args)=>{
```

```
    console.log('2.routeChangeComplete->路由结束变化,参数为:',...args)
  })

  Router.events.on('beforeHistoryChange',(...args)=>{
    console.log('3,beforeHistoryChange->在改变浏览器 history之前触发,参数为:',...args)
  })

  Router.events.on('routeChangeError',(...args)=>{
    console.log('4,routeChangeError->跳转发生错误,参数为:',...args)
  })

  Router.events.on('hashChangeStart',(...args)=>{
    console.log('5,hashChangeStart->hash跳转开始时执行,参数为:',...args)
  })

  Router.events.on('hashChangeComplete',(...args)=>{
    console.log('6,hashChangeComplete->hash跳转完成时,参数为:',...args)
  })




  return(
    <>
      <div>我是首页</div>
      <div>
        <Link href={{pathname:'/xiaojiejie',query:{name:'结衣'}}}><a>选结衣</a></Li
        <Link href="/xiaojiejie?name=井空"><a>选井空</a></Link>
      </div>
      <div>
        <button onClick={gotoXiaojiejie}>选井空</button>
      </div>
      <div>
         <Link href="#jspang"><a>选JSPang</a></Link>
      </div>
    </>
  )

```

```
62 }
63 export default Home
```

这节主要学习了路由的钩子事件，利用钩子事件是可以作很多事情的，比如转换时的加载动画，关掉页面的一些资源计数器.....。

p07：在getInitialProps中用Axios获取远端数据

在Next.js框架中提供了getInitialProps静态方法用来获取远端数据，这个是框架的约定，所以你也只能在这个方法里获取远端数据。不要再试图在声明周期里获得，虽然也可以在ComponentDidMount中获得，但是用了别人的框架，就要遵守别人的约定。

安装和引入Axios插件

Axios是目前最或的前端获取数据的插件了，也是由大神首推的数据接口请求插件，我在工作中也是一直在使用它，所以这里依然使用Axios来进行远端数据请求。在请求前需要先安装Axios插件。
打开终端，直接使用yarn命令进行安装。

```
1 yarn add axios
```

我使用的版本是0.19.0,可能你学习的时候会稍有变化。安装完成后，在需要的页面中用import引入axios，代码如下：

```
1 import axios from'axios'
```

引入后，就可以使用getInitialProps进行获取后端接口数据了。

getInitialProps中获取数据

在xiaojiejie.js页面中使用getInitialProps，因为是远程获取数据，所以我们采用异步请求的方式。数据存在了Easy Mock中，地址如下：

```
1 https://www.easy-mock.com/mock/5cfcce489dc7c36bd6da2c99/xiaojiejie/getList
```

(你可以自己作一个数据源，因为这个可能也不稳定，不过半年内应该是可以的)

```
1 Xiaojiejie.getInitialProps = async ()=>{
2     const promise =new Promise((resolve)=>{
3             axios('https://www.easy-mock.com/mock/5cfcce489dc7c36bd6da2c99/xiaoji
4                 (res)=>{
5                     console.log('远程数据结果: ',res)
6                     resolve(res.data.data)
7                 }
8             )
9     })
```

```
    return await promise
}
```

获得数据后，我们需要把得到的数据传递给页面组件，用{}显示出来就可以了。

```
const Xiaojiejie = ({router,list})=>{
    return (
        <>
            <div>{router.query.name},来为我们服务了 .<br/>{list}</div>
            <Link href="/"><a>返回首页</a></Link>
        </>
    )
}
```

这样我们就利用Axios从远端获取了数据，为了方便你学习，这里给出xiaojiejie.js的所有代码。

```
import { withRouter} from 'next/router'
import Link from 'next/link'
import axios from 'axios'

const Xiaojiejie = ({router,list})=>{
    return (
        <>
            <div>{router.query.name},来为我们服务了 .<br/>{list}</div>
            <Link href="/"><a>返回首页</a></Link>
        </>
    )
}

Xiaojiejie.getInitialProps = async ()=>{
    const promise =new Promise((resolve)=>{
            axios('https://www.easy-mock.com/mock/5cfcce489dc7c36bd6da2c99/xiaoji
                (res)=>{
                    console.log('远程数据结果: ',res)
                    resolve(res.data.data)
                }
            )
    })
    return await promise
```

```
}

export default withRouter(Xiaojiejie)
```

这节课主要学习了在Next.js框架下在getInitialProps方法中利用Axios来获取远端数据的的操作，这个在实际项目中是必备技能，所以需要多练习几遍。

p08：使用Style JSX编写页面的CSS样式

在Next.js中引入一个CSS样式是不可以用的，如果想用，需要作额外的配置。因为框架为我们提供了一个style jsx特性，也就是把CSS用JSX的语法写出来。如果你以前学过Vue，那这种写法你是非常熟悉的。

初识Style JSX语法 把字体设成蓝色

在pages文件夹下，新建一个jspang.js文件。然后写入下面的代码：

```
//jspang.js
function Jspang(){
    return (
        <>
            <div>技术胖免费前端教程</div>
        </>
    )
}
export default Jspang
```

这个是一个最简单的页面，只在层中写了一句话。这时候我们想把页面中字的颜色变成蓝色，就可以使用Style JSX语法。直接在<>之间写下如下的代码:

```
<style jsx>
    {`
        div{color:blue;}
    `}
</style>
```

主要所有的css样式需要用{}进行包裹，否则就会报错。这时候你打开浏览器进行预览，字体的颜色就变成了蓝色。

自动加随机类名 不会污染全局CSS

加入了Style jsx代码后，Next.js会自动加入一个随机类名，这样就防止了CSS的全局污染。比如我们把代码写成下面这样，然后在浏览器的控制台中进行查看，你会发现自动给我们加入了类名，类似jsx-xxxxxxxx。

```
function Jspang(){
    return (
        <>
            <div>技术胖免费前端教程</div>
            <div className="jspang">技术胖免费前端教程</div>

            <style jsx>
                {`
                    div { color:blue;}
                    .jspang {color:red;}
                `}
            </style>
        </>
    )
}
export default Jspang
```

动态显示样式

Next.js使用了Style jsx,所以定义动态的CSS样式就非常简单，比如现在要作一个按钮，点击一下，字体颜色就由蓝色变成了红色。下面是实现代码。

```
import React, {useState} from 'react'

function Jspang(){
    //关键代码----------start-------
    const [color,setColor] = useState('blue')

    const changeColor=()=>{

        setColor(color=='blue'?'red':'blue')
    }
     //关键代码----------end-------

    return (
        <>
            <div>技术胖免费前端教程</div>
            <div><button onClick={changeColor}>改变颜色</button></div>
            <style jsx>
```

```
                {`
                    div { color:${color};}
                `}
            </style>
        </>
    )
}
export default Jspang
```

这样就完成了CSS的动态显示，是不是非常容易。这节课主要学习了Style jsx的一些知识，有了这些知识，可以让我们的页面开始漂亮起来了。

p09：Lazy Loading实现模块懒加载

当项目越来越大的时候，模块的加载是需要管理的，如果不管理会出现首次打开过慢，页面长时间没有反应一系列问题。这时候可用Next.js提供的LazyLoading来解决这类问题。让模块和组件只有在用到的时候在进行加载，一般我把这种东西叫做“懒加载”.它一般分为两种情况，一种是懒加载（或者说是异步加载）模块，另一种是异步加载组件。他们使用的方法也稍有不同，下面我们就来分别学习一下。

懒加载模块

这里使用一个在开发中常用的模块Moment.js，它是一个JavaScript日期处理类库，使用前需要先进行安装，这里使用yarn来进行安装。

```
yarn add momnet
```

然后在pages文件夹下，新建立一个time.js文件，并使用刚才的moment库来格式化时间，代码如下:

```
import React, {useState} from 'react'
import moment from 'moment'

function Time(){

    const [nowTime,setTime] = useState(Date.now())

    const changeTime=()=>{
        setTime(moment(Date.now()).format())
    }
    return (
        <>
            <div>显示时间为:{nowTime}</div>
```

```
            <div><button onClick={changeTime}>改变时间格式</button></div>
        </>
    )
}
export default Time
```

这个看起来很简单和清晰的案例，缺存在着一个潜在的风险，就是如何有半数以上页面使用了这个momnet的库，那它就会以公共库的形式进行打包发布，就算项目第一个页面不使用moment也会进行加载，这就是资源浪费，对于我这样有代码洁癖的良好程序员是绝对不允许的。下面我们就通过Lazy Loading来进行改造代码。

```
import React, {useState} from 'react'
//删除import moment
function Time(){

    const [nowTime,setTime] = useState(Date.now())

    const changeTime= async ()=>{ //把方法变成异步模式
        const moment = await import('moment') //等待moment加载完成
        setTime(moment.default(Date.now()).format()) //注意使用defalut
    }
    return (
        <>
            <div>显示时间为:{nowTime}</div>
            <div><button onClick={changeTime}>改变时间格式</button></div>
        </>
    )
}
export default Time
```

这时候就就是懒加载了，可以在浏览器中按F12，看一下Network标签，当我们点击按钮时，才会加载1.js,它就是momnet.js的内容。

懒加载自定义组件

懒加载组件也是非常容易的，我们先来写一个最简单的组件，在components文件夹下建立一个one.js文件，然后编写如下代码：

```
export default ()=><div>Lazy Loading Component</div>
```

有了自定义组件后，先要在懒加载这个组件的文件中引入dynamic,我们这个就在上边新建的time.js文件中编写了。

```
import dynamic from 'next/dynamic'
```

引入后就可以懒加载自定义模块了，代码如下：

```
import React, {useState} from 'react'
import dynamic from 'next/dynamic'

const One = dynamic(import('../components/one'))

function Time(){

    const [nowTime,setTime] = useState(Date.now())

    const changeTime= async ()=>{
        const moment = await import('moment')

        setTime(moment.default(Date.now()).format())
    }
    return (
        <>
            <div>显示时间为:{nowTime}</div>
            <One/>
            <div><button onClick={changeTime}>改变时间格式</button></div>
        </>
    )
}
export default Time
```

写完代码后，可以看到自定义组件是懒加载的，只有在jsx里用到时，才会被加载进来，如果不使用就不会被加载。
当我们作的应用存在首页打开过慢和某个页面加载过慢时，就可以采用Lazy Loading的形式，用懒加载解决这些问题。

p10：自定义Head 更加友好的SEO操作

既然用了Next.js框架，你就是希望服务端渲染，进行SEO操作。那为了更好的进行SEO优化，可以自己定制标签，定义一般有两种方式，这节课都学习一下。

方法1：在各个页面加上标签

先在/pages文件夹下面建立一个header.js文件，然后写一个最简单的Hooks页面，代码如下:

```
function Header(){
    return (<div>JSPang.com</div>)
}
export default Header
```

写完后到浏览器中预览一下，可以发现title部分并没有任何内容，显示的是localhost:3000/header,接下来就自定义下。自定义需要先进行引入next/head。

```
import Head from 'next/head'
```

引入后你就可以写一些列的头部标签了，全部代码如下:

```
import Head from 'next/head'
function Header(){
    return (
        <>
            <Head>
                <title>技术胖是最胖的! </title>
                <meta charSet='utf-8' />
            </Head>
            <div>JSPang.com</div>

        </>
    )
}
export default Header
```

这时候再打开浏览器预览，你发现已经有了title。

方法2：定义全局的

这种方法相当于自定义了一个组件，然后把在组件里定义好，以后每个页面都使用这个组件,其实这种方法用处不大，也不灵活。因为Next.js已经把封装好了，本身就是一个组件，我们再次封装的意义不大。

比如在components文件夹下面新建立一个myheader.js,然后写入下面的代码:

```
import Head from 'next/head'

```

```
const MyHeader = ()=>{
    return (
        <>
            <Head>
                <title> jspang.com </title>
            </Head>
        </>
    )
}

export default MyHeader
```

这时候把刚才编写的header.js页面改写一下，引入自定义的myheader，在页面里进行使用，最后在浏览器中预览，也是可以得到title的。

```
import Myheader from '../components/myheader'
function Header(){
    return (
        <>
            <Myheader />
            <div>JSPang.com</div>

        </>
    )
}
export default Header
```

这节课讲解了一下Next.js的标签如何使用和自定义，这个在工作中的每个页面都会用到，所以你一定要学会哦。

p11：Next.js框架下使用Ant Design UI

让Next.js支持CSS文件

在前面的课程中我讲过Next.js默认是不支持CSS文件的，它用的是style jsx，也就是说它是不支持直接用import进行引入css的。

比如在根目录下新建一个文件夹static（其实正常情况下你应该已经有这个文件了），然后在文件夹下建立一个test.css文件，写入一些CSS Style。

```
body{
    color:green;
}
```

然后用import在header.js里引入。

```
import '../static/test.css'
```

写完这些后到浏览器中进行预览，没有任何输出结果而且报错了。这说明Next.js默认是不支持CSS样式引入的，要进行一些必要的设置，才可以完成。

开始进行配置，让Next.js支持CSS文件

先用yarn命令来安装@zeit/next-css包，它的主要功能就是让Next.js可以加载CSS文件，有了这个包才可以进行配置。

```
yarn add @zeit/next-css
```

包安装好以后就可以进行配置文件的编写了，建立一个next.config.js.这个就是Next.js的总配置文件（如果感兴趣可以自学一下）。

```
const withCss = require('@zeit/next-css')

if(typeof require !== 'undefined'){
    require.extensions['.css']=file=>{}
}

module.exports = withCss({})
```

这段代码你有兴趣是可以看看的，其实我对配置文件基本不记忆的，因为配置文件就是别人规定的配置，你写就好。比如要使用CSS就可以把上面这段代码输入到放入到里边的就好了。

修改配置文件需要重新启一下服务，重启服务可以让配置生效，这时候你到浏览器中可以发现CSS文件已经生效了，字变成了绿色。

按需加载Ant Design

加载Ant Design在我们打包的时候会把Ant Design的所有包都打包进来，这样就会产生性能问题，让项目加载变的非常慢。这肯定是不行的，现在的目的是只加载项目中用到的模块，这就需要我们用到一个babel-plugin-import文件。

** 先来安装Ant Design库 **

直接使用yarn来安装就可以。

```
yarn add antd
```

** 安装和配置babel-plugin-import 插件 **

其实babel-plugin-import我讲Vue.js和Webpack.js的时候都一次讲过这个插件，这里我们就再来讲一下，先进行安装。

```
yarn add babel-plugin-import
```

安装完成后，在项目根目录建立.babelrc文件，然后写入如下配置文件。

```
{
    "presets":["next/babel"],  //Next.js的总配置文件，相当于继承了它本身的所有配置
    "plugins":[     //增加新的插件，这个插件就是让antd可以按需引入，包括CSS
        [
            "import",
            {
                "libraryName":"antd",
                "style":"css"
            }
        ]
    ]
}
```

这样配置好了以后，webpack就不会默认把整个Ant Design的包都进行打包到生产环境了，而是我们使用那个组件就打包那个组件,同样CSS也是按需打包的。
通过上面的配置，就可以愉快的在Next.js中使用Ant Desgin，让页面变的好看起来。
可以在header.js里，引入<Button>组件，并进行使用，代码如下。

```
import Myheader from '../components/myheader'
import {Button} from 'antd'


import '../static/test.css'
function Header(){
    return (
        <>
            <Myheader />
            <div>JSPang.com</div>
            <div><Button>我是按钮</Button></div>

        </>
    )
```

```
}
export default Header
```

p12：Next.js生产环境打包（完结）

大部分的Next.js基础知识都作了讲解，我相信你通过11集视频的学习，也一定能入门`Next.js`这个框架了，但就在小伙伴准备进行打包项目时，遇到了问题，所谓这节课讲一下如何进行打包和打包中的一些坑。

其实Next.js大打包时非常简单的，只要一个命令就可以打包成功。但是当你使用了`Ant Desgin`后，在打包的时候会遇到一些坑。

打包 ： next build

运行：next start –p 80

先把这两个命令配置到`package.json`文件里，比如配置成下面的样子。

```
"scripts": {
  "dev": "next dev",
  "build": "next build",
  "start": "next start -p 80"
},
```

然后在终端里运行一下`yarn build`，如果这时候报错，其实是我们在加入`Ant Design`的样式时产生的，这个已经在`Ant Design`的Github上被提出了，但目前还没有被修改，你可以改完全局引入CSS解决问题。

在page目录下，新建一个`_app.js`文件，然后写入下面的代码。

```
import App from 'next/app'

import 'antd/dist/antd.css'

export default App
```

这样配置一下，就可以打包成功了，然后再运行`yarn start`来运行服务器，看一下我们的`header`页面，也是有样式的。说明打包已经成功了。