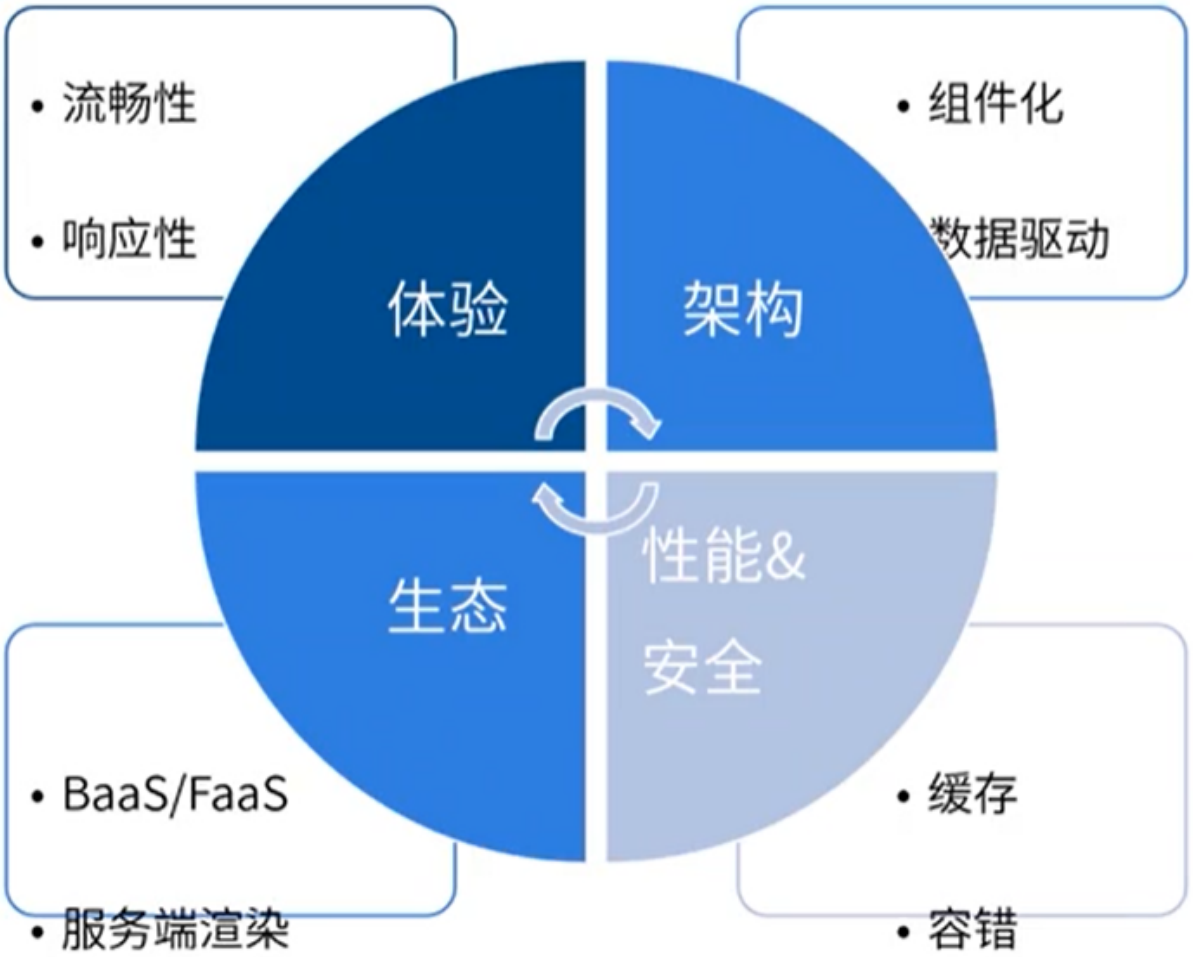
课程目标

单页面应用



单页面优势:

- **体验**: 切换页面不需要刷新,流畅性和响应式更好
- **架构**: 前后端需要分离，前端走api获取数据。页面渲染交给前端，更好的组件化。数据驱动。

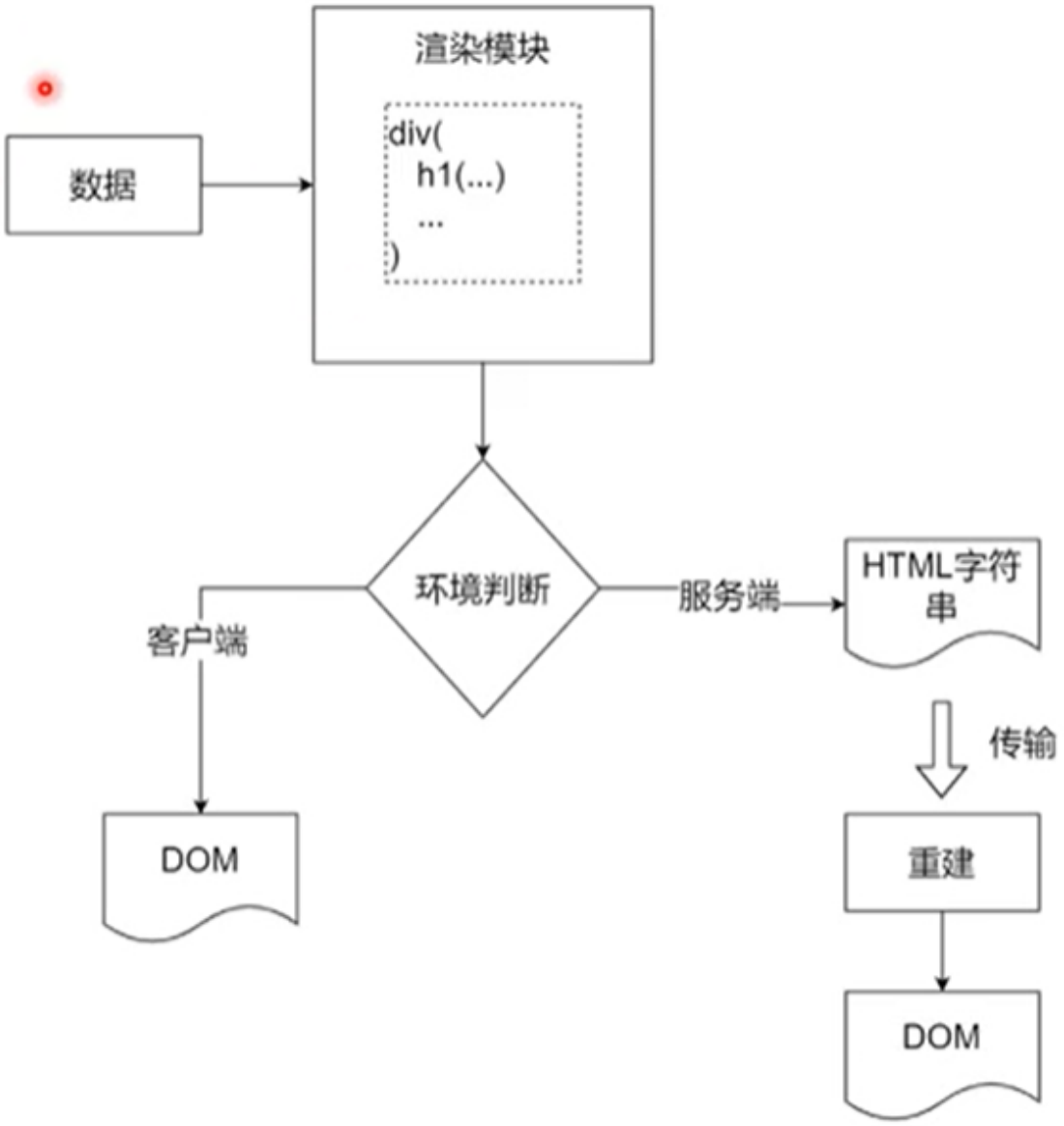
- **生态:**大部分逻辑都搬迁到前端控制，主要是一些跳转啊用户的逻辑。前端做的越来越多就产生了一个生态，就Baas(后台即服务)/FaaS(函数即服务)，这两个可称为一个完整的 Serverless 的实现。把服务端看成一个服务，并不是看作一个接口API。比如我们接了一个地图的应用，我们直接跟百度 高德对接好了，无需经过我们的服务端了。如果中间需要一些行为，简单的处理一些服务，我们可以用node搭建一个Faas服务,由它来作函数的衔接(用这个既就可以减轻大量依赖以后端服务了)。服务端渲染(国内做的比较好的，就是知乎)
- **性能&容错:** 性能方面大量减少I/O，更好做缓存。容错做的更好了

单页面缺点:

- **CPU开销大**(组件 vs 时间　页面都是用js去渲染,js更消耗CPU，这个开销主要在浏览器上　早期服务端渲染的话，服务端本来就把页面绘制好了，开销不在浏览器上。)
- **流量开销大**(一次性加载多张页面，所有的页面都是放在一个js文件中,或者说一次性加载过啦来的，即便是异步加载，有一些组件需要一起加载的)
- **需要更好的系统架构**
- **搜索引擎不太友好**(搜索引擎到一个网页，用的不是渲染，它是直接把这个页面的HTML

拿过来，而spa在未渲染前就是一个div，这个div还没有被渲染。)

服务端渲染



spa client渲染，讲究的是数据驱动，做很多控制器。二服务端渲染讲究的数据映射。服务端渲染是如何把数据通过渲染模块，映射成相关的html，然后判断环境，如果是浏览器就直接渲染dom，如果是服务端的话，会先转变成的html字符串。而且服务端传回HTML字符串，一些onclick事件肯定是失效的，所以肯定还需要客户端重建把事件合进去。比如：一个客户端一个按钮绑定事件，简单就可以触发。但是服务端的话，返回html的按钮，但是按钮绑定的事件，就渲染不了，还得重新绑定相关事件。

SSR示例

- 开发一个生成html节点并且管理事件的函数render
- render(meta,renderAs=dom|html|rehydrate) renderAs三种 一种是直接dom 一种是html字符串 一种是rehydrate，html不渲染，合事件进去

- 实现render函数的服务端渲染能力和客户端渲染能力

相关代码：

node主ssr.js

```
const app = require('express')()
const render = require('./render')
const path = require('path')
//html数据模板
const html = render({
 name:'div',
 props:{
 onclick:(e)=>{
 e.preventDefault()
  window.alert('这是外层div')
 }
 },
 child:[
 {
 name:'ul',
 child:[
 {
 name:'li',
 props:{
 onclick:function () {
 console.log('我是ul的li---1')
 },
 style:{
 background:'red'
 }
 },
 child:'apple'
 },
 {
 name:'li',
 props:{
 onclick:function () {
 console.log('我是ul的li---2')
 },
 style:{
```

```
 background:'blue'
 }
 },
 child:'apple1'
 },
 {
 name:'li',
 props:{
 onclick:function () {
 console.log('我是ul的li----3')
 },
 style:{
 background:'yellow'
 }
 },
 child:'apple2'
 }
 ]
 }
 ]
},'html')

console.log(html);

//请求静态js page-ssr.js
app.get('/page-ssr.js',(req,res)=>{
 res.sendFile(path.resolve(__dirname,'page-ssr.js'))
})

// 请求静态js page-ssr.js
app.get('/page.js',(req,res)=>{
 res.sendFile(path.resolve(__dirname,'page.js'))
})


//请求html page
app.get('/page',(req,res)=>{
 const htmlStr = `<div id="root"></div>
 <script src="page.js"></script>
 `
```

```
 res.send(htmlStr)
})

// 请求html page-ssr
app.get('/page-ssr',(req,res)=>{
 const htmlStr = `<div id="root">
 ${html}
 <script src="page-ssr.js"></script>
 </div>`
 res.send(htmlStr)
})

app.listen('9001',()=>{
 console.log('进入9001服务');
})

```

render渲染函数

```
//node节点是一个数据格式是一个递归
function render(node,renderAs ='dom',path = [] ) {
 const {name,props,child} = node
 const Strateyer = {
 dom: function () {
 //外层创建一个的element
 let element = document.createElement(name)
 // 是否有props属性click事件
 if(props&&props.onclick){
 element.addEventListener('click',props.onclick)
 }
 if(typeof child === 'string'){
 element.innerHTML = child
 }else if(Array.isArray(child)){
 //遍历children,把相关的child的elemnt追加进去
 child.forEach((item,i)=>{
 element.appendChild(render(item,renderAs,path.concat(i))) //需要递归
 })
 }
 return element
```

```
  },
  html: function () {
  let childStr = ''
  if(typeof child === 'string'){
  childStr = child
  }else if( Array.isArray(child)){
  childStr = child.map((item,index)=>{
  return render(item,renderAs,path.concat(index))
  }).join("")
  }
  return `<${name} id='node-${path.join('-')}'>${childStr}</${name}>`
  },
  rehydrate: function () {
  //第外层
  if(props&&props.onclick){
  document.getElementById(`node-${path.join('-')}`).addEventListener('cli
ck',props.onclick)
  }
  if(Array.isArray(child)){
  //递归一下
  child.forEach((item,index)=>{
  render(item,renderAs,path.concat(index))
  })
  }

  },
  }
  if( renderAs in Strateyer){
  return Strateyer[renderAs]()
  }else {
  throw "not support renderAs Type"
  }

}

//node环境的导出 moudule方式
module.exports = render
```

客户端渲染html

```
function render(node,renderAs ='dom',path = [] ) {
```

```
const {name,props,child} = node
const Strateyer = {
dom: function () {
//外层创建一个的element
let element = document.createElement(name)
// 是否有props属性click事件
if(props&&props.onclick){
element.addEventListener('click',props.onclick)
 }
 if(typeof child === 'string'){
 element.innerHTML = child
 }else if(Array.isArray(child)){
 //遍历children,把相关的child的elemnt追加进去
 child.forEach((item,i)=>{
 element.appendChild(render(item,renderAs,path.concat(i))) //需要递归
 })
 }
 return element
 },
 html: function () {
 let childStr = ''
 if(typeof child === 'string'){
 childStr = child
 }else if(Array.isArray(child)){
 childStr = child.map((item,index)=>{
 return render(item,renderAs,path.concat(index))
 }).join("")
 }
 return `<${name} id='node-${path.join('-')}'>${childStr}</${name}>`
 },
 rehydrate: function () {
 //第外层
 if(props&&props.onclick){
 document.getElementById(`node-${path.join('-')}`).addEventListener('cli
ck',props.onclick)
 }
 if(Array.isArray(child)){
 //递归一下
 child.forEach((item,index)=>{
 render(item,renderAs,path.concat(index))
```

```
  })
  }

  },
  }
  if( renderAs in Strateyer){
  return Strateyer[renderAs]()
  }else {
  throw "not support renderAs Type"
  }

}
  const html = render({
  name:'div',
  props:{
  onclick:(e)=>{
  e.preventDefault()
  window.alert('这是外层div')
  }
  },
  child:[
  {
  name:'ul',
  child:[
  {
  name:'li',
  props:{
  onclick:function () {
  console.log('我是ul的li---1')
  },
  style:{
  background:'red'
  }
  },
  child:'apple'
  },
  {
  name:'li',
  props:{
  onclick:function () {
```

```
 console.log('我是ul的li---2')
 },
 style:{
 background:'blue'
 }
 },
 child:'apple1'
 },
 {
 name:'li',
 props:{
 onclick:function () {
 console.log('我是ul的li----3')
 },
 style:{
 background:'yellow'
 }
 },
 child:'apple2'
  }
  ]
  }
  ]
  },'dom')
document.getElementById('root').appendChild(html)
```

服务端渲染html

```
function render(node,renderAs ='dom',path = [] ) {
 const {name,props,child} = node
 const Strateyer = {
 dom: function () {
 //外层创建一个的element
 let element = document.createElement(name)
 // 是否有props属性click事件
 if(props&&props.onclick){
 element.addEventListener('click',props.onclick)
  }
  if(typeof child === 'string'){
  element.innerHTML = child
  }else if(Array.isArray(child)){
```

```
14  //遍历children,把相关的child的elemnt追加进去
15  child.forEach((item,i)=>{
16  element.appendChild(render(item,renderAs,path.concat(i))) //需要递归
17  })
18  }
19  return element
20  },
21  html: function () {
22  let childStr = ''
23  if(typeof child === 'string'){
24  childStr = child
25  }else if( Array.isArray(child)){
26  childStr = child.map((item,index)=>{
27  return render(name,renderAs,path.concat(index))
28  }).join("")
29  }
30  console.log(path.join('-'));
31  return `<${name} id='node-${path.join('-')}'>${childStr}</${name}>`
32  },
33  rehydrate: function () {
34  //第外层
35  if(props&&props.onclick){
36  document.getElementById(`node-${path.join('-')}`).addEventListener('cli
ck',props.onclick)
37  }
38  if(Array.isArray(child)){
39  //递归一下
40  child.forEach((item,index)=>{
41  render(item,renderAs,path.concat(index))
42  })
43  }
44
45  },
46  }
47  if( renderAs in Strateyer){
48  return Strateyer[renderAs]()
49  }else {
50  throw "not support renderAs Type"
51  }
52
```

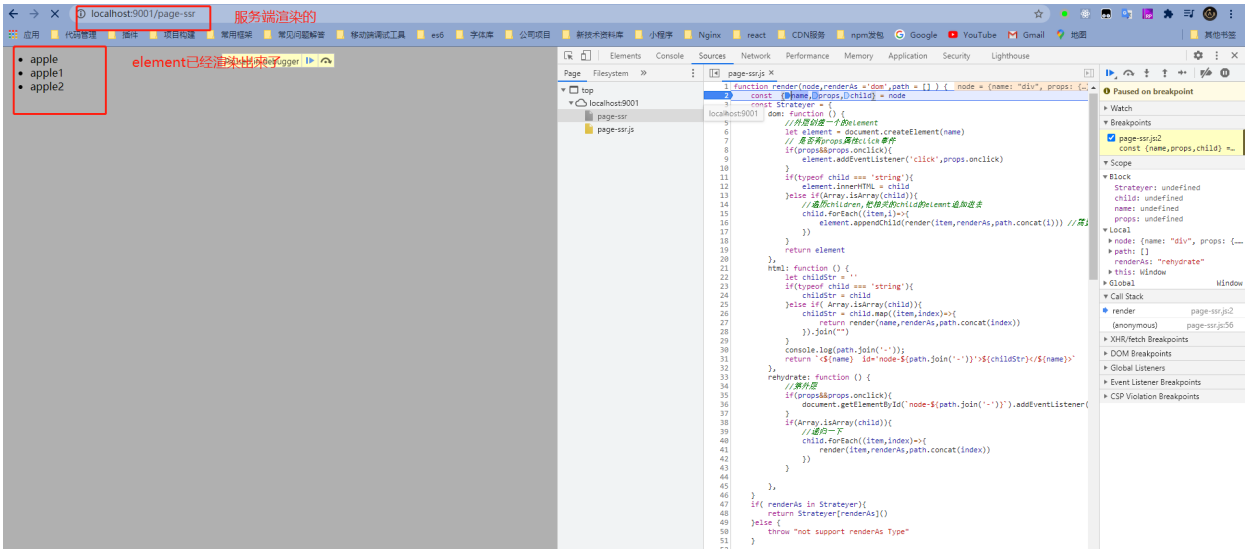
```
}


render({
 name:'div',
 props:{
 onclick:(e)=>{
 e.preventDefault()
 window.alert('这是外层div')
 }
 },
 child:[
 {
 name:'ul',
 child:[
 {
 name:'li',
 props:{
 onclick:function () {
 console.log('我是ul的li---1')
 },
 style:{
 background:'red'
 }
 },
 child:'apple'
 },
 {
 name:'li',
 props:{
 onclick:function () {
 console.log('我是ul的li---2')
 },
 style:{
 background:'blue'
 }
 },
 child:'apple1'
 },
 {
```

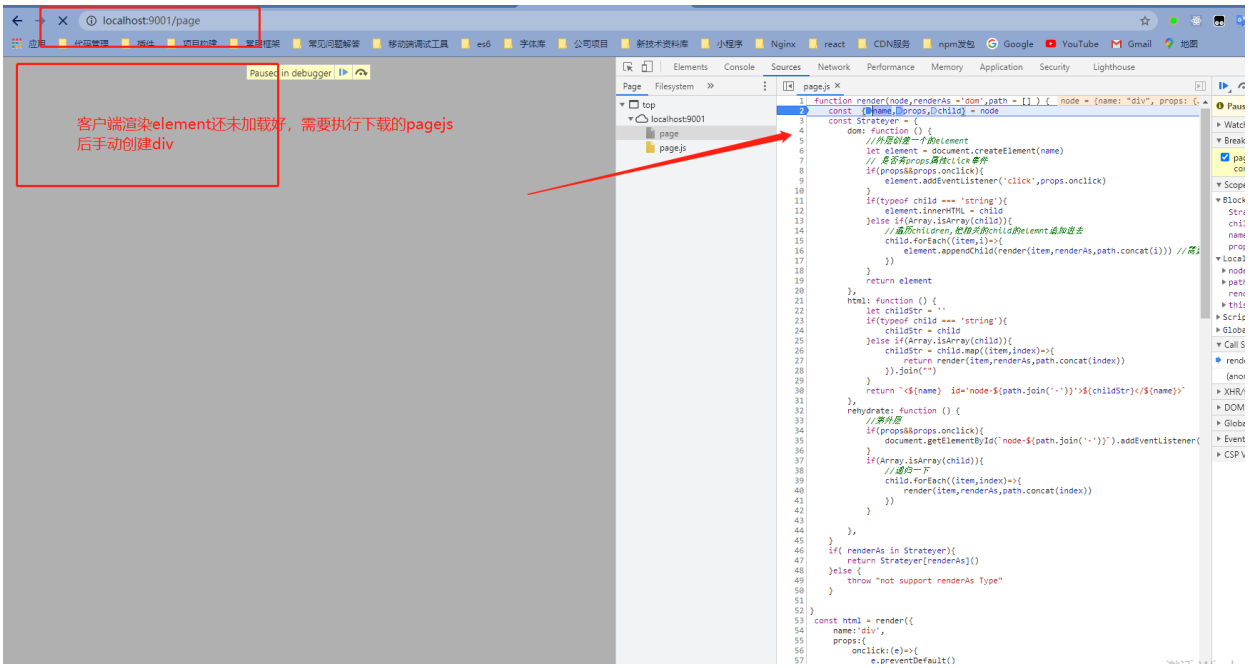
```
93  name:'li',
94  props:{
95  onclick:function () {
96  console.log('我是ul的li----3')
97  },
98  style:{
99  background:'yellow'
100  }
101  },
102  child:'apple2'
103  }
104  ]
105  }
106  ]
107 },'rehydrate')
108
```

查看相关结果:

服务端渲染:



客户端渲染:



SSR优缺点分析

优点	缺点
显著减小TTI(Time To Interactive)	服务器CPU压力大
单页面应用SEO解决方案	CDN不好支持
内网数据组装	开发工作量大
	全局对象失效

优点：

- 显著线上减少TTI(测量页面所有资源加载成功并能够可靠地快速响应用户输入的时间)
- 有利于seo
- 内网数据组装起来，速度更快

缺点：

- 服务器cpu压力大
- CDN不太好支持
- 开发工作量大
- 全局对象失效

课程小结

- 思考：少量页面开发用单页面还是多页面?
- 思考：如果SSR目的仅仅是SEO，该如何做?

1. 这个具体问题具体分析
2. ssr用户体验好，速度快(和客户端渲染比较起来速度快很多)。不单单是SEO