

## 课程目标

01

工作日常

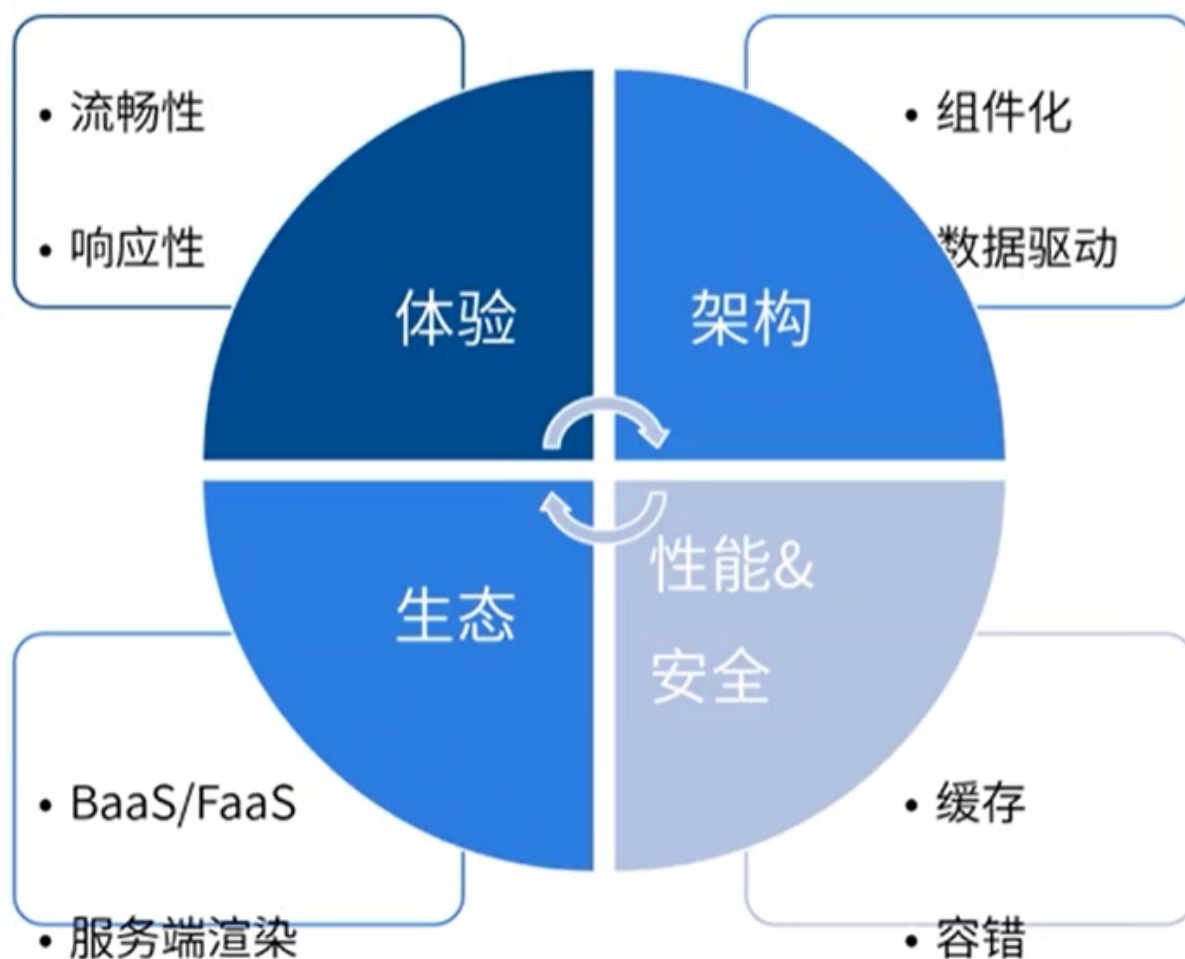
02

面试重点

03

前端架构

## 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。

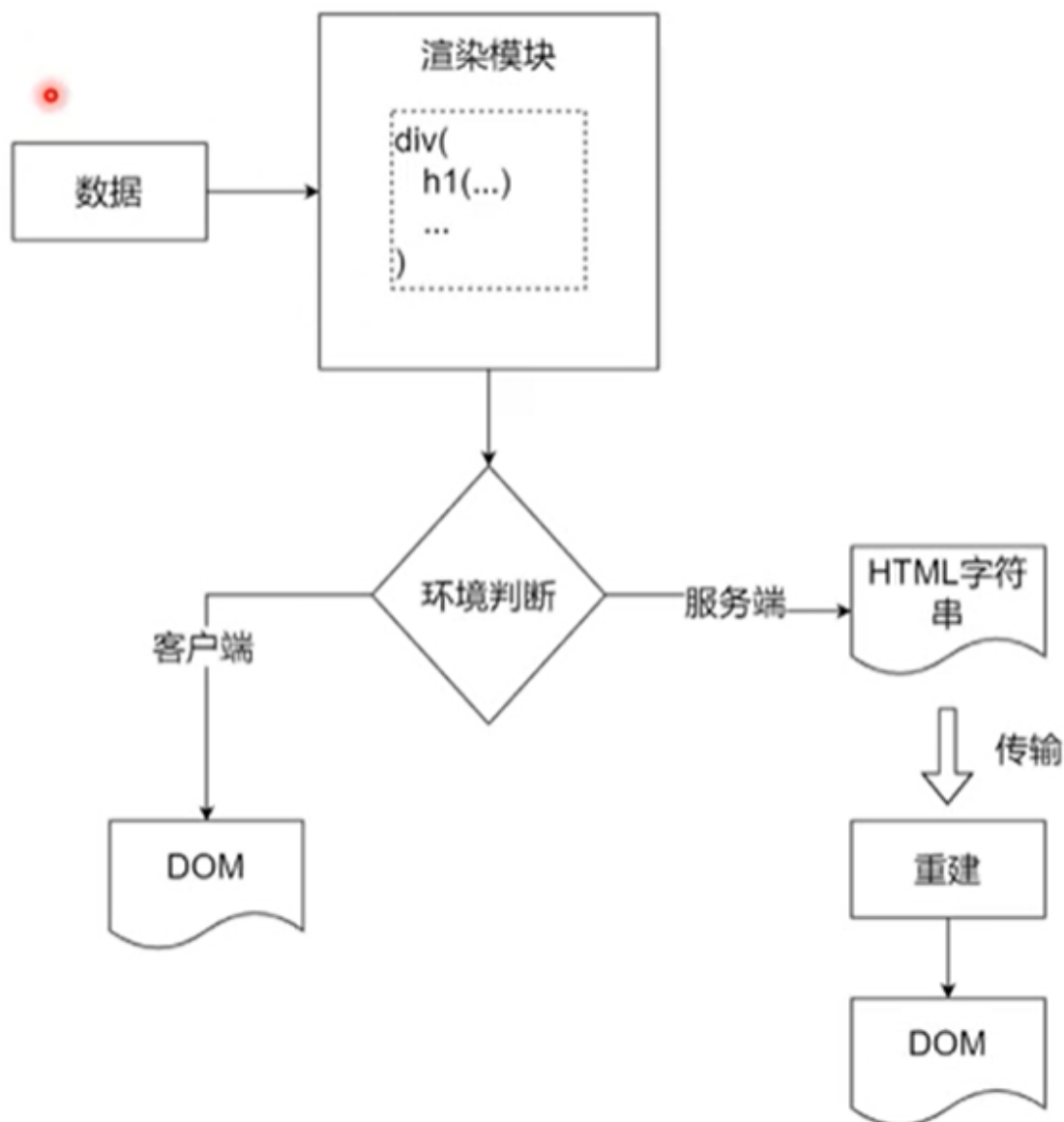
- **生态:**大部分逻辑都搬迁到前端控制，主要是一些跳转啊用户的逻辑。前端做的越来越多就产生了一个生态，就Baas(后台即服务)/FaaS(函数即服务)，这两个可称为一个完整的 Serverless 的实现。把服务端看成一个服务，并不是看作一个接口 API。比如我们接了一个地图的应用，我们直接跟百度 高德对接好了，无需经过我们的服务端了。如果中间需要一些行为，简单的处理一些服务，我们可以用node搭建一个Faas服务,由它来作函数的衔接(用这个既就可以减轻大量依赖以后端服务了)。服务端渲染(国内做的比较好的，就是知乎)
- **性能&容错:** 性能方面大量减少I/O，更好做缓存。容错做的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

### 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|hydrate) renderAs三种 一种是直接dom 一种是html字符串 一种是hydrate，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

```
1  const app = require('express')()
2  const render = require('./render')
3  const path = require('path')
4  //html数据模板
5  const html = render({
6    name: 'div',
7    props: {
8      onclick: (e) => {
9        e.preventDefault()
10       window.alert('这是外层div')
11      }
12    },
13    child: [
14      {
15        name: 'ul',
16        child: [
17          {
18            name: 'li',
19            props: {
20              onclick: function () {
21                console.log('我是ul的li---1')
22              },
23              style: {
24                background: 'red'
25              }
26            },
27            child: 'apple'
28          },
29          {
30            name: 'li',
31            props: {
32              onclick: function () {
33                console.log('我是ul的li---2')
34              },
35              style: {
```

```
36   background: 'blue'
37   }
38 },
39   child: 'apple1'
40 },
41 {
42   name: 'li',
43   props: {
44     onclick: function () {
45       console.log('我是ul的li----3')
46     },
47     style: {
48       background: 'yellow'
49     }
50   },
51   child: 'apple2'
52 }
53 ]
54 }
55 ]
56 }, 'html')
57
58 console.log(html);
59
60 //请求静态js page-ssr.js
61 app.get('/page-ssr.js', (req, res) => {
62   res.sendFile(path.resolve(__dirname, 'page-ssr.js'))
63 })
64
65 // 请求静态js page-ssr.js
66 app.get('/page.js', (req, res) => {
67   res.sendFile(path.resolve(__dirname, 'page.js'))
68 })
69
70
71 //请求html page
72 app.get('/page', (req, res) => {
73   const htmlStr = `<div id="root"></div>
74   <script src="page.js"></script>
75   `
```

```

76
77   res.send(htmlStr)
78 })
79
80 // 请求html page-ssr
81 app.get('/page-ssr',(req,res)=>{
82   const htmlStr = `

{
90   console.log('进入9001服务');
91 })
92


```

## render渲染函数

```

1
2 //node节点是一个数据格式是一个递归
3 function render(node,renderAs = 'dom',path = [] ) {
4   const {name,props,child} = node
5   const Strategyer = {
6     dom: function () {
7       //外层创建一个的element
8       let element = document.createElement(name)
9       // 是否有props属性click事件
10      if(props&&props.onclick){
11        element.addEventListener('click',props.onclick)
12      }
13      if(typeof child === 'string'){
14        element.innerHTML = child
15      }else if(Array.isArray(child)){
16        //遍历children,把相关的child的elemnt追加进去
17        child.forEach((item,i)=>{
18          element.appendChild(render(item,renderAs,path.concat(i))) //需要递归
19        })
20      }
21      return element

```

```

22  },
23  html: function () {
24    let childStr = ''
25    if(typeof child === 'string'){
26      childStr = child
27    }else if( Array.isArray(child)){
28      childStr = child.map((item,index)=>{
29        return render(item,renderAs,path.concat(index))
30      }).join("")
31    }
32    return `<${name} id='node-${path.join('-')} '>${childStr}</${name}>`
33  },
34  rehydrate: function () {
35    //第外层
36    if(props&&props.onclick){
37      document.getElementById(`node-${path.join('-')}`).addEventListener('cli
ck',props.onclick)
38    }
39    if(Array.isArray(child)){
40      //递归一下
41      child.forEach((item,index)=>{
42        render(item,renderAs,path.concat(index))
43      })
44    }
45
46  },
47  }
48  if( renderAs in Strateyer){
49    return Strateyer[renderAs]()
50  }else {
51    throw "not support renderAs Type"
52  }
53
54  }
55
56  //node环境的导出 module方式
57  module.exports = render

```

## 客户端渲染html

```

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

```

```

2  const {name,props,child} = node
3  const Strategyer = {
4    dom: function () {
5      //外层创建一个的element
6      let element = document.createElement(name)
7      // 是否有props属性click事件
8      if(props&&props.onclick){
9        element.addEventListener('click',props.onclick)
10     }
11     if(typeof child === 'string'){
12       element.innerHTML = child
13     }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(item,renderAs,path.concat(index))
28       }).join("")
29     }
30     return `<${name} id='node-${path.join('-')} '>${childStr}</${name}>`
31   },
32   rehydrate: function () {
33     //第外层
34     if(props&&props.onclick){
35       document.getElementById(`node-${path.join('-')}`).addEventListener('click',props.onclick)
36     }
37     if(Array.isArray(child)){
38       //递归一下
39       child.forEach((item,index)=>{
40         render(item,renderAs,path.concat(index))

```



```
41  })
42  }
43
44  },
45  }
46  if( renderAs in Strateyer){
47  return Strateyer[renderAs]()
48  }else {
49  throw "not support renderAs Type"
50  }
51
52  }
53  const html = render({
54  name:'div',
55  props:{
56  onclick:(e)=>{
57  e.preventDefault()
58  window.alert('这是外层div')
59  }
60  },
61  child:[
62  {
63  name:'ul',
64  child:[
65  {
66  name:'li',
67  props:{
68  onclick:function () {
69  console.log('我是ul的li---1')
70  },
71  style:{
72  background:'red'
73  }
74  },
75  child:'apple'
76  },
77  {
78  name:'li',
79  props:{
80  onclick:function () {
```

```

81 console.log('我是ul的li---2')
82 },
83 style:{
84 background:'blue'
85 }
86 },
87 child:'apple1'
88 },
89 {
90 name:'li',
91 props:{
92 onclick:function () {
93 console.log('我是ul的li----3')
94 },
95 style:{
96 background:'yellow'
97 }
98 },
99 child:'apple2'
100 }
101 ]
102 }
103 ]
104 }, 'dom')
105 document.getElementById('root').appendChild(html)

```

## 服务端渲染html

```

1 function render(node,renderAs ='dom',path = [] ) {
2   const {name,props,child} = node
3   const Strateyer = {
4     dom: function () {
5       //外层创建一个的element
6       let element = document.createElement(name)
7       // 是否有props属性click事件
8       if(props&&props.onclick){
9         element.addEventListener('click',props.onclick)
10      }
11      if(typeof child === 'string'){
12        element.innerHTML = child
13      }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 `
```

```
53 }
54
55
56 render({
57   name: 'div',
58   props: {
59     onclick: (e) => {
60       e.preventDefault()
61       window.alert('这是外层div')
62     }
63   },
64   child: [
65     {
66       name: 'ul',
67       child: [
68         {
69           name: 'li',
70           props: {
71             onclick: function () {
72               console.log('我是ul的li---1')
73             },
74             style: {
75               background: 'red'
76             }
77           },
78           child: 'apple'
79         },
80         {
81           name: 'li',
82           props: {
83             onclick: function () {
84               console.log('我是ul的li---2')
85             },
86             style: {
87               background: 'blue'
88             }
89           },
90           child: 'apple1'
91         },
92         {
```

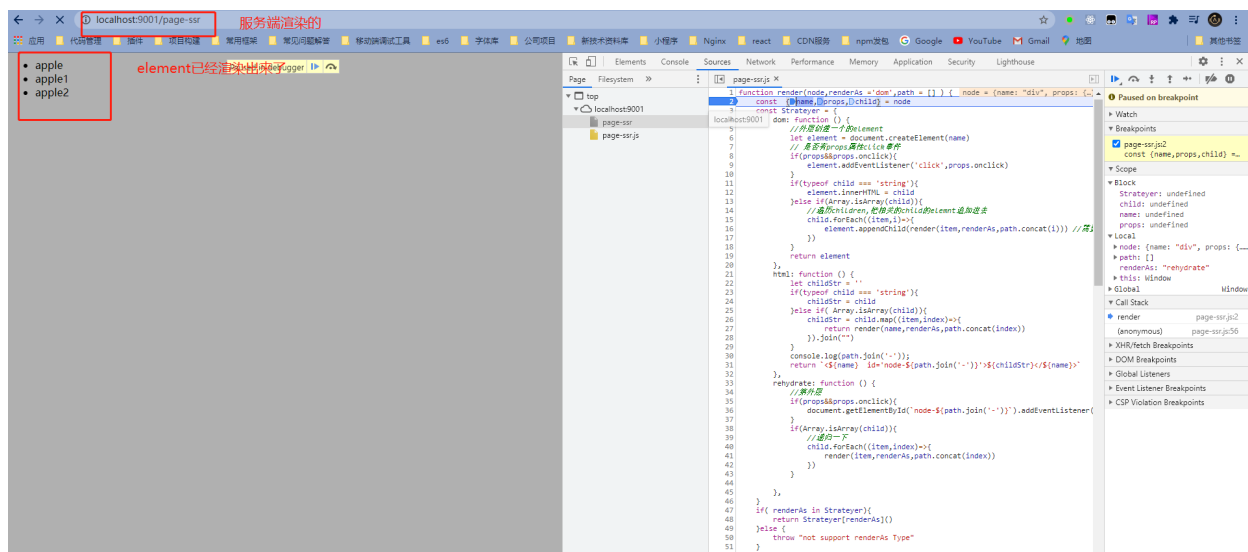
```

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

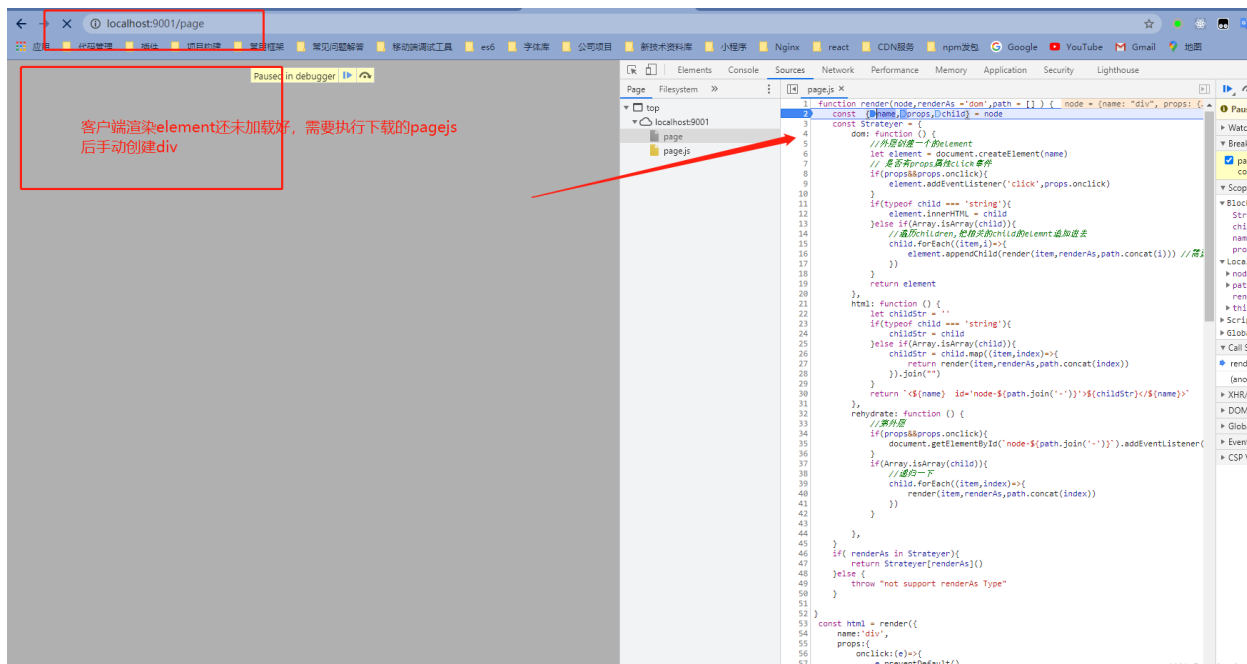
```

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