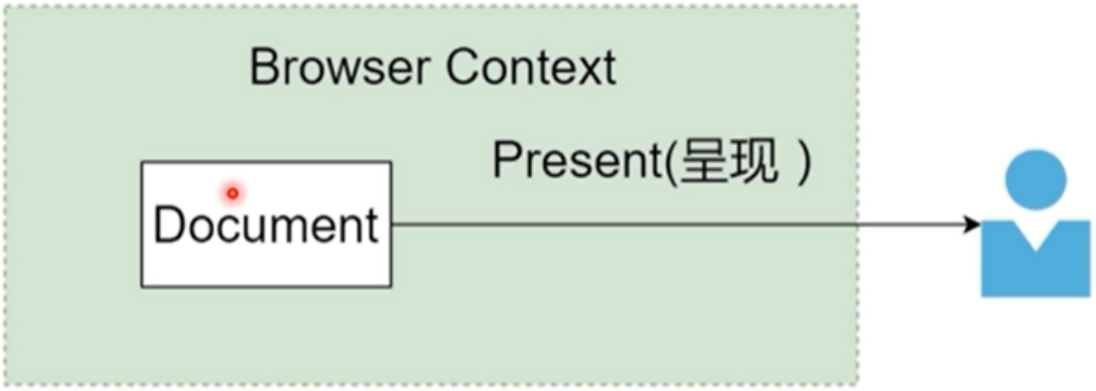
课程目标

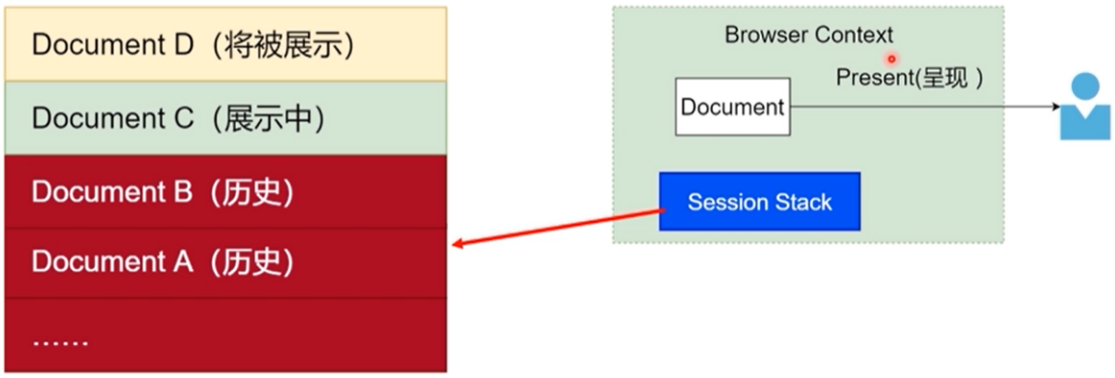
前端路由和History API

浏览器文境



我们说浏览器其实是这样的东西，它将Document，浏览器内部的一种对象(数据的集合)呈现给用户,这是浏览器核心的能力，document dom对象集合是这浏览器需要解析的东西。展现给用户的是网页。在docment的周边是知识，这些知识有哪些？现在用户浏览了哪些网页，现在的网址是什么，这些都是浏览器的背景知识。也就叫浏览器的上下文(Browser Context), 浏览器就是样的 浏览器的上下文包裹住Document，呈现给用户。

会话历史(Session History)



浏览器上下文Context 其实还有包裹着一个 Session Stack的，通过浏览器上下文进行数据共享。所以要这个会话历史可以堪称一个是会话栈，类似一个数组一样的。

Document D是用户输入网址，还没有按回车时候的一个文档网址地址。

Document C是用户输入网址，还有按了回车时候的一个文档网址地址。

Document B是历史

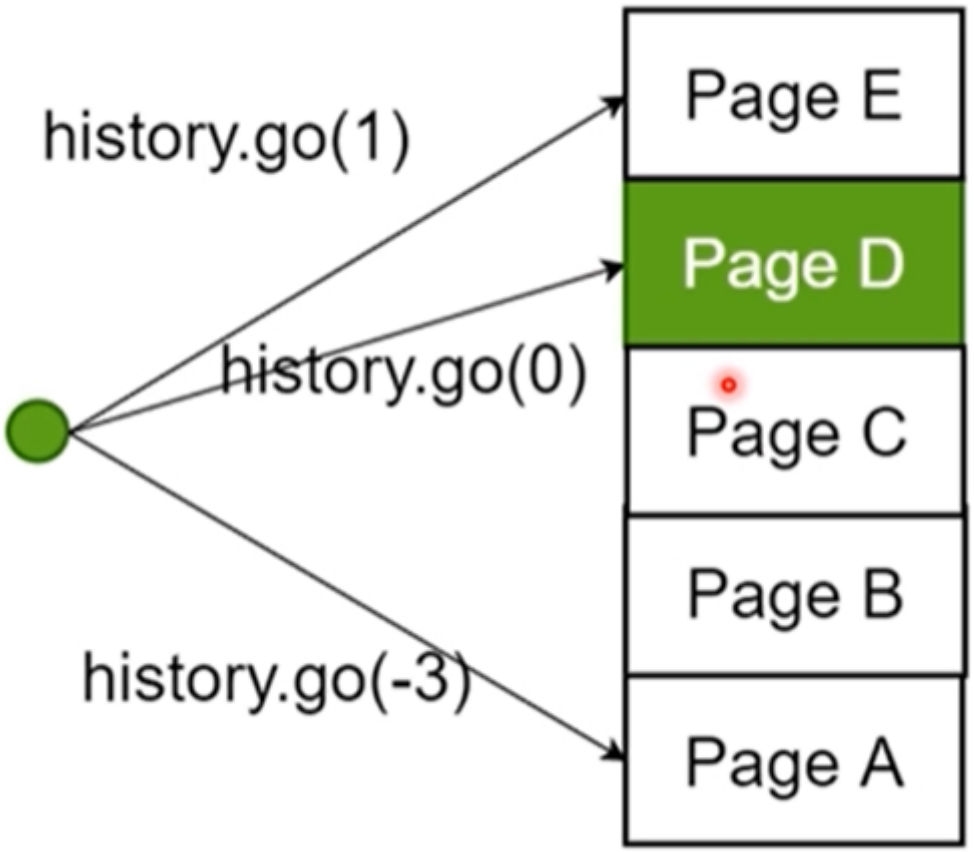
Document A是也是历史

.....

History API

提供操作控制浏览器会话历史，维护会话栈(Session stack)的能力

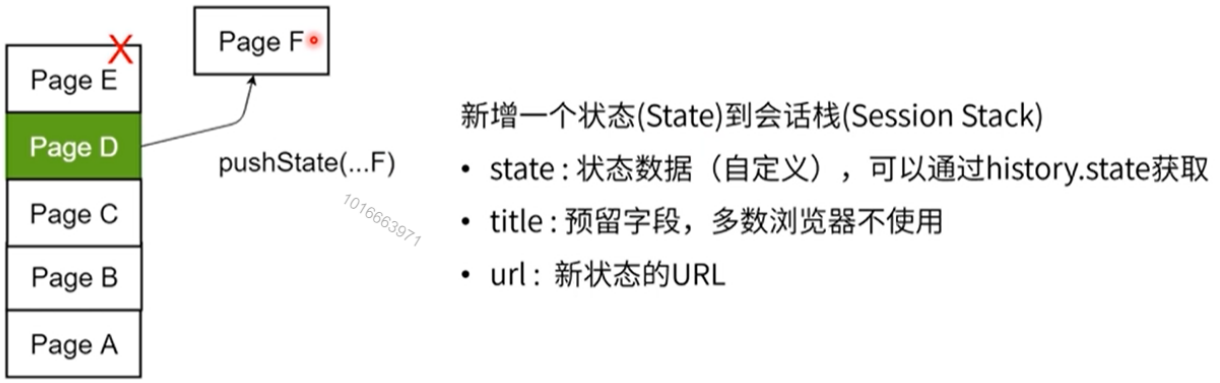
history.go() 切换会话栈,不会改变会话栈。



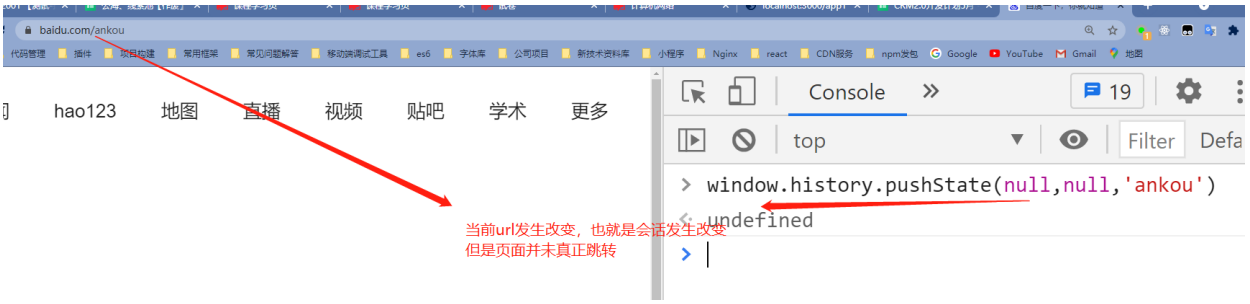
go(1) 当前的页面的下一个页面 也就是pageE
go(0) 当前的页面
go(-1) 当前的页面的上一个页面 也就是PageC
go(-2) 当前的页面的下一个页面 也就是pageB
go(-3) 当前的页面的下一个页面 也就是pageA

history.back()和history.forward() 和go具有同样功能的API 只是back是回退，forward是前进

history.pushState(state,title,url) 新增一个的状态(state) 到会话栈(Session Stack)
state:状态数据(可以自定义一些数据title:预留字段 多数浏览器不适用 url:新状态url



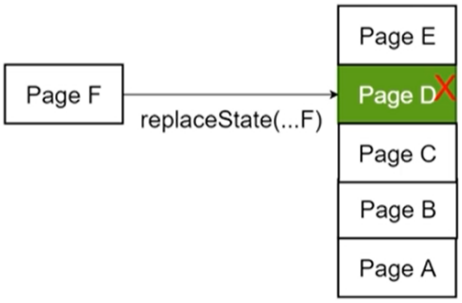
pushState一个状态F进去，它的会把原来的E擦掉，栈是一个线性结构，不能分叉的。当前是是pageD，预加的就是变成的了pageF，因为线性结构只能保留一个。pushState不会真的发生页面的跳转，为了响应单页面应用渲染的一个趋势，是替换一个内容。但是当前的url(当前的会话)会改掉 。如图所示。



history.pushState({name:'ankou'},null，url) 第一个参数，可以记录当前会话的数据，无需拼接在url上。如下图所示

浏览器的pushState是有限制的

history.replaceState(state,title,url) 是替换当前会话栈，例如的当前的会话是Page D 使用replaceState(..F),房前的会话栈中pageF就替换了pageD



替换会话栈（Session Stack）中当前的状态

- state：状态数据（自定义），可以通过history.state获取
- title：预留字段，多数浏览器不使用
- url：新状态的URL

注意：通过pushState进去的状态， 调用go() back() forward() 都不会真正跳转页面的，如果不是通过pushState进去的，使用就会页面重新跳转刷新。

实战服务端路由

- **观察node.js实现服务端路由**
- **观察Cluster启动多个实例进行负载均衡**

1.服务端路由

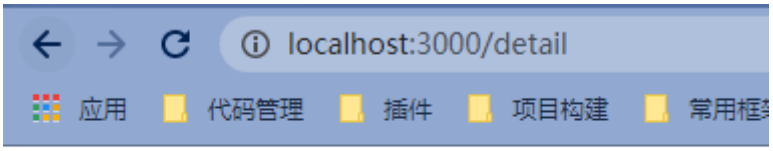
```
const express = require('express')
const app = express()
const fs = require('fs')
const path = require('path')
//解析当前pages的相对路径为绝对路径 __dirname 找到当前的路径
const pageDir = path.resolve(__dirname,"pages")
//读取文件夹中的文件
const htmls = fs.readdirSync(pageDir)


 //高阶函数显示
 function displayHtmlFile(name,ext){
  return (req,res)=>{
  res.sendFile(path.resolve(pageDir,`${name}.${ext}`))
  }
 }

 htmls.forEach(file=>{
  const [name,ext] = file.split('.')
  app.get(`/${name}`,displayHtmlFile(name,ext))
 })

```

```
23
24 app.listen(3000,()=>{
25  console.log('欢迎进入路由服务');
26 })
```



这是详情页面

这是list页面

2. 观察Cluster启动多个实例进行负载均衡

```
const cluster = require('cluster');
const numCPUs = require('os').cpus().length;
const express = require('express')

if (cluster.isMaster) { var process: NodeJS.Process
  console.log(`Master ${process.pid} is running`);

  // Fork workers.
  for (let i = 0; i < numCPUs; i++) {
    cluster.fork();
  }

  cluster.on('exit', (worker, code, signal) => {
    console.log(`worker ${worker.process.pid} died`);
  });
```

实战前端路由

- 在node端实现wildcard路由/product/*
- 前端解析/product/:id和product/list形成单页面

server端代码

```
const express = require('express')
const app = express()
const path = require('path')
const htmlFile = path.resolve(__dirname,"pages/spa.html")


//url正则匹配 一下
//以product开头的url和铺面跟随数字，都跳转到同一个页面
app.get(/\/product(s|\/d+)/,(req,res)=>{
  res.sendFile(htmlFile)
 })

 app.listen(8090,()=>{
  console.log('spa 服务');
 })

```

client端代码

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<meta charset="UTF-8">
<title>单页面</title>
<style>
a{
color: blue;
cursor: pointer;
 }
 </style>
</head>
<body>
<h2>当页面应用</h2>
<ul>
 <li><a onclick="route('/products')">列表</a></li>
 <li><a onclick="route('/product/123')">详情</a></li>
</ul>
<div id="content"></div>

</body>
<script>
 //进入页面匹配，初始化
 //程序切换路由配置
 //浏览器前进 后退切换

 //先配置一份路由表
 function List() {
 const html = `
 <ul>
 <li> apple</li>
 <li> 华为</li>
 <li> 三星</li>
 </ul>
 `
 document.getElementById('content').innerHTML = html
 }
 function Detail() {
 document.getElementById('content').innerHTML = 'Detail'
 }
 const pages = [
 {
 match:/\/products/,
```

```
  route:List,
  },
  {
  match:/\/product\/\d+/,
  route:Detail,
  },
  ]
  //匹配路由
  function matchRoute(pages,href) {
  //匹配路由
  const page = pages.find(item=>item.match.test(href))
  //当页面替换内同
  page.route()
  }
  //进入页面初始化
  matchRoute(pages,window.location.href)
  //程序切换路由
  function route(url) {
  //切换url，不刷新跳转页面
  window.history.pushState(null,null,url)
  // 再做相关内容切换
  matchRoute(pages,url)
  }
  //浏览器前进后退 通过onpopstate监听状态变化
  window.onpopstate =function () {
  matchRoute(pages,window.location.href)
  }
</script>
</html>
```