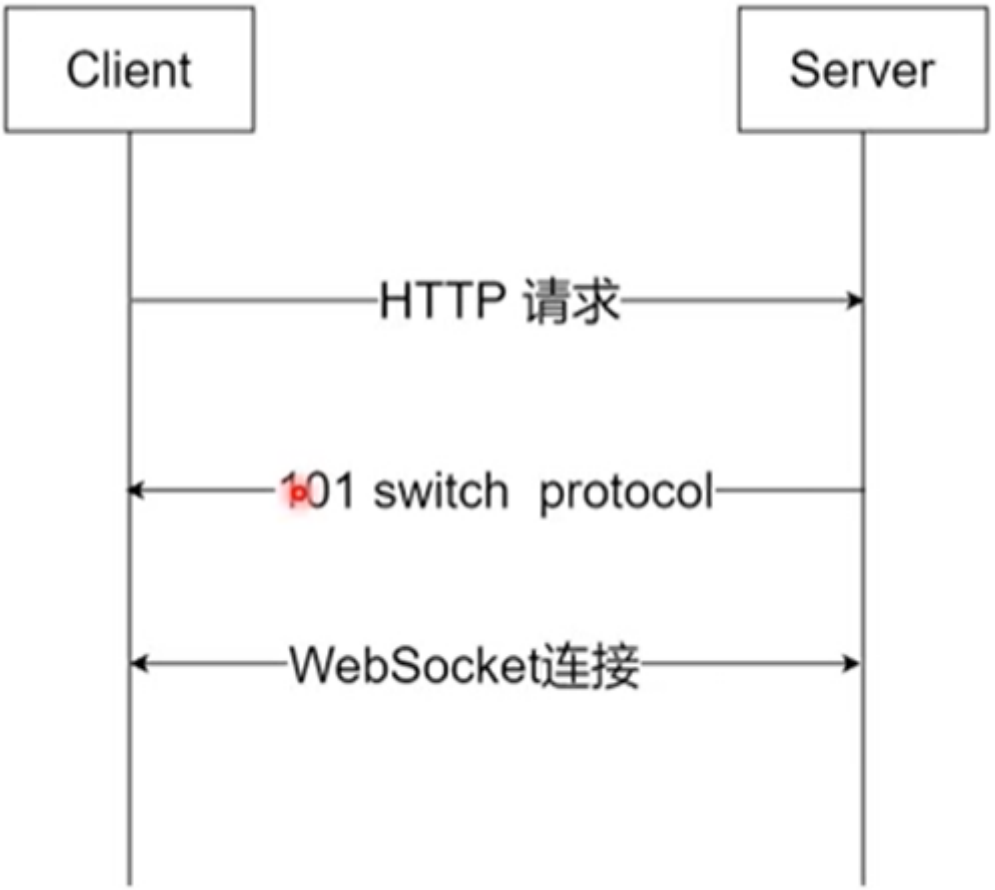
课程目标

WebSocket协议初探

websocket是一个基于TCP的通信协议，和http协议一样都是在应用层。但是websocket要复用HTTP的握手



client先以HTTP请求向Server发送一次请求，这个请求会带上一个头部，客服端希望以websok cet协议，如果服务端实现了webSocket协议的,会给的client端返回一个101 切换的协议的头，客户端收到协议之后，以websocket的方式以跟服务端建立连接。下面就

实现实战一个Socket连接的建立过程。观察浏览器/node端websocket连接建立的过程。观察对握手的处理和协议转换的过程。

客户端demo:

```
//简单的demo
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>webSocket聊天室</title>
</head>
<body>
<h1 style="text-align: center">web聊天</h1>
 <button onclick="chat()">开始聊天</button>
 <script>
  function chat(){
  let webSocket = new WebSocket('ws://dev.websocket:8090')
  //wss ->websocket Secure
  //https- tls/ssl - tcp/ip
  // wss - tls- tcp/ip
  webSocket.onopen = function () {
  console.log('connection open');
  }
  }
 </script>
 </body>
 </html>
```

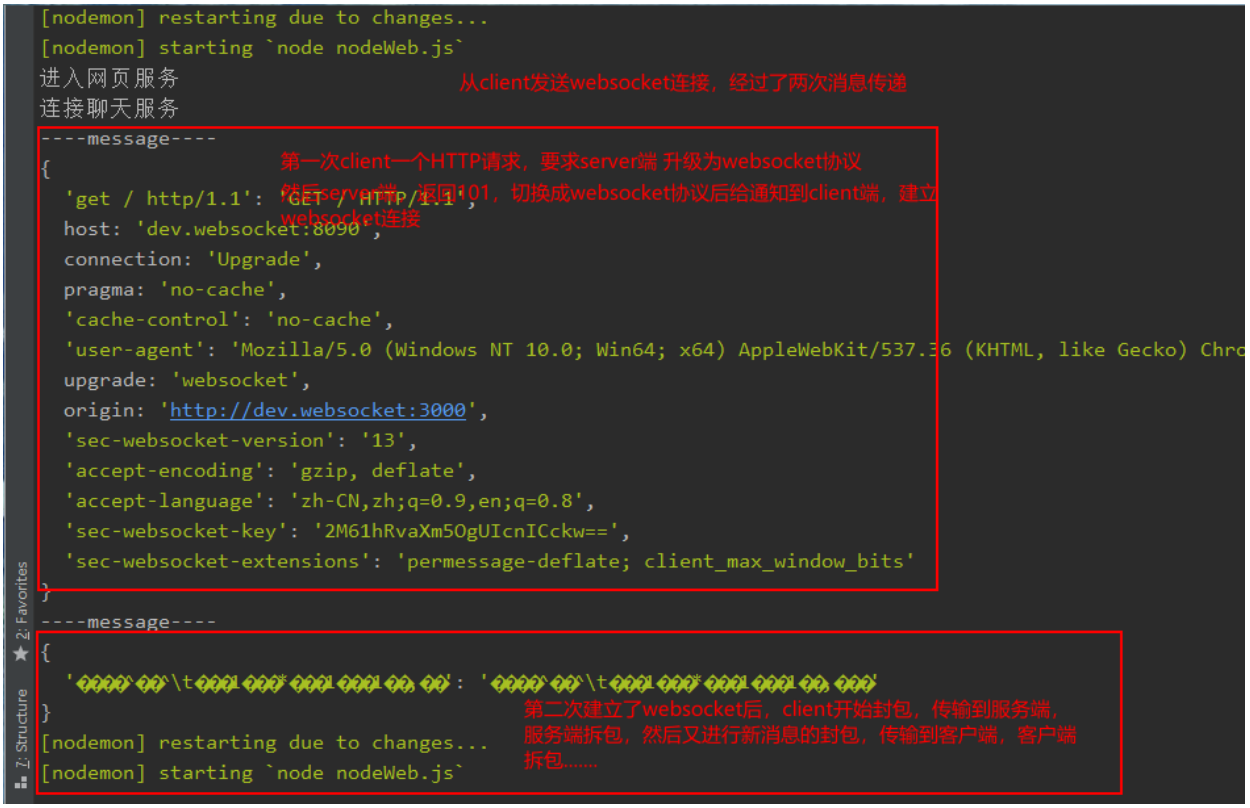
node原生实现服务端demo:

```
//网页服务
const express = require('express')
const path = require('path')
const app = express()

app.get('/',(req,res)=>{
 res.sendFile(path.resolve(__dirname,'index.html'))
})
app.listen(3000,()=>{
  console.log('进入网页服务');
```

```
})


// websocket server
const net = require('net') //node自带的一个包
//net 这个服务是tcp/IP的一个服务和express的http服务还是有一定差别的
const server = net.createServer()
//解析头部的包
const parseHeader = require('parse-headers')

//crypto 加密包
const crypto = require('crypto')
server.on('connection',socket => {
 //监听socket的数据来源
 socket.on('data',(buffer)=>{
 //数据都是二进制的buffer
 const str = buffer.toString()
 //查看传输了几次消息 第一次是握手 后面是websokcet 封包后的传输
 console.log("----message----");
 //解析头部
 const header = parseHeader(str)
 console.log(header)

 //crypto 加密
 const sha1 = crypto.createHash('sha1')
 sha1.update(header['sec-websocket-key'] + '258EAFA5-E914-47DA-95CA-C5AB0DC85B11')
 const acceptKey = sha1.digest('base64')
 let response =`HTTP/1.1 101 Switch protocols
Upgrade:websocket
Connection:Upgrade
Sec-Websocket-Accept:${acceptKey}


 `
 socket.write(response)
 })
})

server.listen(8090,()=>{
 console.log('连接聊天服务')
```

```
51 })
```

node原生实现服务端相关日志



聊天室:node端

使用socket.io实现的node服务端demo

```
1  //网页服务
2  const express = require('express')
3  const path = require('path')
4  const app = express()
5
6  app.get('/',(req,res)=>{
7   res.sendFile(path.resolve(__dirname,'index.html'))
8  })
9  app.listen(3000,()=>{
10   console.log('进入网页服务');
11  })
12
13  const soceketIO = require('socket.io')(8090)
14
15  function broadcast(type,message,sender){
16   for(let socket of users.keys()){
```

```
  socket.send({type,message,sender}) //每个socket对应每个用户的 发送相关给到客户端
  }
}
const users = new Map()
soceketIO.on('connect',(socket)=>{
  // 创建socket的成本很低的,维护的成本不高，进来一个用户创建一个io
  // 高成本的是计算 一个聊天室的能支撑多少个用户的，得看后面数据得存储 计算得效率
  socket.on('message',data=>{
  //这里用得switch 只是用作简单得demo 真正的登录还得需要用户token进行校验
  switch (data.type) {
  case 'LOGIN':
  users.set(socket,{name:data.name}) //map中的每一个key都是一个socket
  broadcast('LOGIN',`${data.name}加入了聊天`)
  break
  case 'CHAT':
  const user = users.get(socket) //聊天的时候的时候 先获取的当前socket
  // socket关联的map的值 是之前存的{name:data.name}
  broadcast('CHAT',data.message,user.name)
  break
  }
  })
})

```

观察node端的实现过程

思考：下载全部聊天记录应该如何实现?

1. 在demo实例情况下 用数组缓存起来即可
2. 是以什么样的形式给到client端 是以http协议 还是socket直接发呢？ 其实都可以但是一个socket的复用的是一个TCP链接，如果聊天记录数据很多的时候,用socket就会照成阻塞。这是可以使用另外一个连接，这个用socket的还是http这个就无关紧要了。

聊天室:web端(demo)

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

```
 <meta charset="UTF-8">
 <title>webSocket聊天室</title>
</head>
<body>
<h1 style="text-align: center">web聊天</h1>
<input type="text" id="chatValue" onkeydown="calAge(event)"; >
 <button onclick="send()">发送</button>
 <div id="textContent">

 </div>
 <script src="/socket.io.js"></script>
 <script>
  var socket = io.connect('ws://dev.websocket:8090',{transports: ['websoc
ket']});
  const name = "user" +new Date().getTime()
  socket.send({
  type:'LOGIN',
  name
  })
  socket.on('message',(data)=>{
  console.log(data)
  const {message,sender} = data
  let senderName = sender
  if(!sender){
  senderName = '系统'

  }else if(sender === name){
  senderName = "我"
  }
  Message(senderName,message)
  })


  function Message(sender,message) {
  let textDiv = document.createElement('div')
  textDiv.innerHTML =`<span>${sender}:<span><span>${message}</span>`
  document.getElementById('textContent').append(textDiv)

  }
  function calAge(e) {
  var evt = window.event || e;
```

```
44  if (evt.keyCode == 13) {
45  //回车后要干的业务代码
46  send()
47  }
48  }
49  function send() {
50
51  let chatInpt = document.getElementById("chatValue")
52  socket.send({
53  type:'CHAT',
54  message:chatInpt.value
55  })
56  chatInpt.value =""
57  }
58  </script>
59  </body>
60  </html>
```

课程小结

websockt握手和协议转换的过程很自然[我们可以看到用原生的node和用socket.io] 比较socket网络插座 为客户端/服务端提供通信机制