http特点

- web应用是基于HTTP协议的，而HTTP协议恰恰是一种无状态协议

cookie

cookie是为了辩别用户身份，进行会话跟踪而 *存储在客户端* 上的数据

- 通过 响应头 向客户端设置cookie

```
Set-Cookie: name=zfpx
```

- *读取* 客户端过来的cookie

```
Cookie: key1=value1; key2=value2
```

设置

```
res.cookie(name,value,[,options]);
```

参数	chrome对应属性	类型	说明	示例
domain	Domain	String	域名，默认是当前域名	{domain:'a.zfpx.cn'}
path	Path	String	域名，默认是/	{path:'/visit'}
expires	Expires	Date	过期时间，如果没以有指定或为0表示当前会话有效	{expires:new Date(Date.now()+20*1000)}
maxAge	Max-Age	Number	有效时间(单位是毫秒)	{maxAge:20*1000}
httpOnly	HTTP	Boolean	不能通过浏览器javascript访问	{httpOnly:true}
secure	Secure	String	只通过https协议访问	

- 获取cookie

使用cookie-parser中间件

```
app.use(require('cookie-parser')());
```

- 设置cookie

```
response.cookie(key,value)
```

- 获取cookie

```
request.cookies
```

- 清除cookie

```
response.clearCookie('username')
```

cookie使用注意事项

- 可能被客户端`篡改`，使用前验证合法性
- 不要存储`敏感`数据，比如用户密码，账户余额
- 使用`httpOnly`保证安全
- 尽量减少cookie的`体积`
- 设置正确的`domain`和`path`，减少数据传输

session

- 会话跟踪，数据存放在 `服务器端`
- 需要借助 `cookie` 存储一个 `会话ID`，服务器可以根据会话ID来查询出详细的session数据

session步骤

- 在服务器端生成全局唯一标识符(session_id)
- 在服务器内存里开辟此session_id对应的数据存储空间
- 将session_id作为全局唯一标示符通过cookie发送给客户端
- 以后客户端再次访问服务器时会把session_id通过请求头中的cookie发送给服务器
- 服务器再通过session_id把此标识符在服务器端的数据取出

express中的session

```
app.use(require('cookie-parser')('zfpx'));
app.use(require('express-session')({secret:'zfpx',resave:true,saveUninitialized:true}
req.session.username = 'zfpx'
console.log(req.session.username);
```

cookie VS session

- 应用场景
- 安全性
- 性能
- 时效性
- 存储