目录

1.网络分层

- 计算机网络
  - `ISO`定义了 `7`层的网络分层
  - 大家通用的， `5`层的网络分层

  - 通用的`5`层:
    - 自下向上顺序
    1. 物理层： 网线、WiFi、光纤
    2. 数据链路层：数据的转换
    3. 网络层
    4. 传输层
    5. 应用层： qq、微信

- 网络传输协议：网络数据在互联网进行传输的时候, 需要遵从传输规则, 叫做网络传输协议。
  - 网络层 : IP
  - 传输层 : `Port` => `TCP/UDP`

    - 假设 `A` 给 `B` 在微信上发一条消息
      `A`的传输步骤:
      1. 先通过 *应用层* 编辑未发送的消息
      2. 通过 *传输层* 和 *网络层* 给这串数据添加 `ip:Port`; 知道这串数据 从哪来,要去哪去
      3. 数据链路层 将数据转换为 `01`二进制,再进一步转换为 电信号或光信号
      4. 通过网线或wifi发送出去
- 网络分层的概念 是为了更好的理解网络传输, 人为的概念; 实际传输时不存在

2.TCP协议（传输层）

- `TCP`: 面向连接协议。 （双向链接）
  - 传输数据:
    1. 建立连接 **三次握手**
    2. **传输确认** 数据传输
    3. 断开连接 **四次挥手**

    - **三次握手** 建立连接:
      1. 客户端 向 服务端发起连接请求
         - 客户端 发送 `SYN`包 -> 服务器
      2. 服务端 回复 准备好连接
         - 服务端 回发 `SYN+ACK`包 -> 客户端
      3. 服务端 回复 我也准备好了
         - 客户端 发送`ACK` 包 -> 服务器
      - 服务器收到 `ACK`包就 进行数据传输
      - 为什么不两次握手?
        如果两次握手, 在网络拥堵情况下, 第一个`SYN`包滞留, 客户端会再次发送`SYN2`包, 服务器根据`SYN2` 返回 `SYN2+ACK`包; 如果此时 第一个`SYN`包滞留结束,发送到服务器上, 服务器就会以为建立了两次链接

    - 数据传输 **传输确认**:
      - 一包数据可能会被拆成多包发送, 需要**处理 丢包问题**
      - 数据包到达的顺序不同, 需要**处理 乱序问题**

      - `TCP`协议为每一个链接建立了发送缓冲区,从建立连接的第一个序列号为`0`,后面每一个字节的序列号就会增加`1`
      - 发送数据时, 从发送缓冲区取出一部分数据,组成发送报文,在`TCP`协议头中会附带 序列号和长度
        - `发送报文` = `序列号` | `长度` | `数据内容`
      - 回复确认: `ACK` = `序列号` + `长度` = `下一包数据起始序列号`
      - 根据序列号和长度重组数据， 丢失包后,会要求服务端重发此包

    - **四次挥手** 断开连接:
        - **客户端和服务端都可以 发送关闭连接请求**
        1. 客户端发送 `Fin` 包 -> 服务端， **客户端**进入 **终止等待1状态**
        2. 服务端 发送 `ACF` 包 -> 客户端 , **服务端**进入**关闭等待状态**,**客户端**进入**终止等待2状态**
        3. 服务端 发送`FIN`包 -> 客户端 ,**服务端**进入 **最后确认状态**
        4. 客户端收到后, 回复`ACF`包 -> 服务端 ,**客户端**进入**超时等待状态**,经过超时时间后,关闭连接;
            - 服务端收到`ACK`包后,立即关闭连接
            - 服务端没有收到`ACK`包, 将重复`3步骤`, 客户端收到`AFC`包,重置超时等待时间
- TCP协议
  优点：1. 安全，2. 准确度非常高
  缺点：1. 传输效率低，2. 耗资源

3.UDP协议 (传输层)

- `UDP`:无连接协议 (适用于 及时性要求高, 准确度要求不高的应用)
    - 传输数据: 直接传,不管是否收到数据。
    - 缺点：1. 不安全，2. 准确度非常低，3. 经常丢包
    - 优点：1. 及时性非常高，2. 消耗资源低
        - 例: 视频聊天

4.本地存储技术 Cookie

- `LocalStorage`在笔记9
    1. 永久存储（除非主动删除）
    2. 最大存储`5M` ， 客户端的一个微型数据库
    3. 只能去存储字符串`string` (放入什么类型的数据都会被自动转换为字符串)
       /下面了解，不重要的点/
    4. `localStorage` 在浏览器隐私模式下是不可读取的
    5. `localStorage` 本质上是对字符串的读取,如果存储内容多的话，会消耗内存空间，导致页面变卡。
    6. `localStorage` 不能被爬虫抓取到
- `Cookie` 叫做 会话跟踪技术
    1. 可以设置过期时间
    2. 最大可以存储`4kb`, 每一个域名最多可以存储`50`条`cookie` (不同浏览器有些许区别)
    - 只能存储字符串，大小有限,**一般只存储重要的信息**
        - 一般存储: 是否登录、点赞, 购物车信息, 视频播放进度等
- `cookie`的语法
    - 格式: `name=value;[expires=date];[path=path];[domain=somewhere.com];[secure]`
        - `[]`中括号的内容,是可选项
    - 火狐支持本地加载的文件缓存`cookie`。

    - 谷歌需要服务器加载的文件才能缓存 `cookie` 。

- 设置 `cookie`
    - `document.cookie = 'name=value'`
    - `cookie` 设置 同名 `cookie` 会进行覆盖,可以设置多个 `cookie`

- 读取 `cookie`
    - `alert(document.cookie)`
    - `cookie` **存储中文会乱码 -> 先将中文编码, 读取时再解码**

```
document.cookie = 'user=' + encodeURIComponent('王保乐')
alert(decodeURIComponent( document.cookie ))
```

        - `encodeURIComponent` 和 `decodeURIComponent` 可以编码和解码 `URI` 特殊字符（如 `#` ， `/` ， `￥` 等），而 `decodeURI` 则不能。

5.Cookie的可选项

- `expires` : 过期时间
    - 必须填写, 日期对象
        - 当会话结束后 系统会自动清除过期的 `cookie`
    - 可以 通过 日期函数方法 : `setDate()` 等 设置过期时间

```
var d = new Date()
var day = d.getDate();
d.setDate(day + 3)
```

        - 默认过期时间 是 会话结束 (关闭浏览器)
    - 手动删除 `cookie`
        - 设置重名 `cookie` ,覆盖之前的 `cookie` ,过期时间设置到昨天或更早，在关闭浏览器时会被浏览器清除

```
document.cookie = 'user=;expires=' + new Date(0);
```

- `path` : 限制访问路径（可访问 `cookie` 的路径）
    - 如果不去设置, 默认加载 `html` 页面同级的路径的 `cookie`
        - 如果我们设置的 `cookie` 路径,和加载当前文件的路径不一致, `cookie` 访问会失败
    - 例:
        - `5_2path.html` 在服务器 `17` 文件夹下

```
document.cookie = 'name=xxx;path=' + '/17/demo';
```

        - 此时, `cookie` 设置成功, 但是无法访问, 因为源文件和 `cookie` 文件的路径不一致
        - 在 `demo` 文件夹中, `cookie` 是设置成功的, `demo` 中其他源文件可以访问此条 `cookie`

    - 同级目录下的 `html` 文件 `cookie` 是共享的

- `domain`: 限制访问域名
    - 如果不去设置, 默认是加载 `html` 页面的域名/ `IP`
        - 如果加载当前文件的域名 和 设置 `cookie` 的域名不一致的话, 设置 `cookie` 会失败

```
// 域名错误 无法加载cookie
document.cookie = 'u=111;domain=localhosx'
```

- `secure`: 安全(https: 证书认证协议)
    - 如果不设置, 设置 `cookie` ,可以通过 `HTTP` 协议加载文件设置 `cookie` ， 也可以通过 `HTTPS` 协议加载文件设置 `cookie`
        - 设置这个字段以后, 只能通过 `https` 协议 加载 `cookie`

6.封装 cookie

- `setCookie()` ： 设置 `cookie`
- `getCookie()` ： 获取 `cookie`
- `removerCookie()` ： 删除 `cookie`

```
// 快速设置 过期时间
function afterOfDate(n){
  var d = new Date();
  var day = d.getDate();
  d.setDate(day + n);
  return d;
}
```

- `setCookie()`

```
function setCookie(name, value,{expires=7,path,domain,secure}={}){
  var str = encodeURIComponent(name) + '=' + encodeURIComponent(value);
  str += ';expires=' + afterOfDate(expires);
  if(path){
    str += ';path=' + path;
  }
  if(domain){
    str += ';domain=' + domain;
  }
  if(secure){
    str += ';secure';
  }
  document.cookie = str;
  }
```

    - 调用： `setCookie('user','xxxx',{})`

- `getCookie()`
  - `cookie`获取的是当前所有的`cookie`, 想要的效果是 传入对应的键, 获取对应的值

```
function getCookie(name){
  // 先对cookie进行解码
  var cookie = decodeURIComponent(document.cookie)
  // 查找name 是否存在cookie, 在cookie的其实位置
  var start = cookie.indexOf(name);
  if(start == -1){
    return null;
  }else{
    var end = cookie.indexOf(';',start);
    // 如果end为-1 说明这是最后一条cookie
    if(end == -1){
      end = cookie.length;
    }

    var str = cookie.substring(start,end);
    var str = str.split('=')[1]
    return str
  }
}
alert(getCookie('超级英雄'))
alert(getCookie('user'))
```

- `removerCookie()` -> 将时间设置为过去时间

```
function removeCookie(name){
  // 先判断这个 name 存不存在于cookie, 也可以不用判断, 直接删除
  if(!getCookie(name)){
    return null
  }else{
    setCookie(name,'',{expires:-10})
  }
}
```

- `$cookie`函数: 设置/获取/删除 三合一
  - 判断传入参数的个数, 决定是什么操作
    - `setCookie(name,value)` 或 `setCookie(name,value,{})`
    - `getCookie(name)`
    - `removeCookie(name,null)` -> 因为和`get`一致,所以要将`value`设置成`null`

```
function $cookie(name){
  switch(arguments.length){
    case 1:
      return getCookie(arguments[0])
    case 2:
      if(arguments[1] == null){
        removeCookie(arguments[0],null,{expires:-10})
      }else{
        setCookie(arguments[0],arguments[1])
      }
      break;
    case 3:
      setCookie(arguments[0],arguments[1],arguments[3])
    default:
      return 'error';
  }
}
```

7.认识闭包

1. **满足以下特点的叫做闭包**
    1. 函数嵌套函数
    2. 内部函数使用外部函数的形参和变量
        - 被引用的形参和变量就不会被【垃圾回收机制】回收
        - 闭包中 `return` 只是为了指针返回, 可以使用闭包中的变量和对象

```
function aaa(res){
  var num = 10;
  alert(res + "," + num);
  function bbb(){
    num += 2;
    alert('bbb' + num + "," + res)
  }
  return bbb;
}
var bbb = aaa(20); // 20,10
bbb(); // bbb12,20
```

2. 闭包的好处:
    1. **变量可以常驻内存**
    2. **避免全局变量污染** 避免声明全局变量(多人同时开发时)
    3. **可以声明私有成员**

3. 立即执行函数 `(function (){})(传参)` `( function(){}() ))`
    - 立即执行函数： 不需要调用，立马能够自己执行的函数
    1. 写法 可以传递参数进来

```
// 第一种
( function (a, b){    多使用
    console.log(a + b);
})(1, 2);

// 立即执行函数也可以写函数名(sum)
( function sum(a, b){
    console.log(a + b);
})(1, 2);
```

```
// 第二种
  ( function (b, a){
      console.log(b + a);
  }(2, 3) );

  ( function sum(b, a){
      console.log(b + a);
  }(2, 3) );
```

- `()`包裹`function`是告诉浏览器，这不是一个函数声明。

2. 立即执行函数最大的**作用**就是 **独立创建了一个作用域。**
   - 立即执行函数外部不可以调用，**加名字没意义**

```
// 立即执行函数写 闭包
var ccc = (function(){
  var a = 2;
  return function(){
    a++;
    alert(a)
  }
})()
ccc() // 3
ccc() // 4
```

4. 私有变量(`private`)

```
var moduleA = (function(){
  var count = 10;      // 私有变量
  function aaa(){      // 私有方法
    count += 2;  alert(count)
  }
  function bbb(){
    count *= 10; alert(count)
  }
  return{
    funcA: aaa, funcB: bbb
  }
})()
moduleA.funcA() // 12
moduleA.funcB() // 120
```

- 只能通过`moduleA`访问, `count`和`aaa`都是`moduleA`的私有变量和方法

8.闭包的作用和注意事项(IE) 内存泄漏

- 使用闭包 获取当前点击按钮的下标

```
var aBtns = document.getElementsByTagName('button');

for(var i = 0; i<aBtns.length;i++){
  aBtns[i].onclick = (function(index){
    return function(){
      alert(index)
    }
  })(i)
}
```

  - 使用闭包的注意事项

```
// 我们天天都在使用闭包:
window.onload = function(){
  var oDiv = document.getElementById('div1');
  oDiv.onclick = function(){
    alert(oDiv.id)
  }
}
```

    - `oDiv`满足 闭包条件; (1)函数嵌套函数, (2)内部函数使用外部函数的变量或参数
    - 这种无法释放的内存被称为: 内存泄漏 (常在IE浏览器,其他浏览器的内存管理机制很好)
    - 内存泄漏解决方法:
      1. 在外部获取 内部函数要使用的属性, 减少闭包内存

```
window.onload = function(){
  var oDiv = document.getElementById('div1');
  var id = oDiv.id;
  oDiv.onclick = function(){
    alert(id)
  }
  oDiv = null;
}
```

         - `oDiv=null` 后，点击idv还会执行点击函数?
      2. 页面解构(`onunload`) IE支持
         - 解构时，将 点击函数和元素节点 指向`null`

```
var oDiv = document.getElementById('div1');
oDiv.onclick = function(){ alert(oDiv.id) }
window.onunload = function(){
  oDiv.onclick = null;
  oDiv = null;
}
```