面经预热-Day7（计网专题）

1、常见的请求方式？GET和POST请求的区别？

1. 作用不同
    - GET用于从服务端获取资源
    - POST一般用来向服务器端提交数据
2. 参数传递方式不同
    - GET请求的参数一般写在URL中，且只接受ASCII字符
    - POST请求的参数一般放在请求体中，对于数据类型也没有限制
3. 安全性不同
    - 因为参数传递方式的不同，所以两者安全性也不同，GET请求的参数直接暴露在URL中，所以更不安全，不能用来传递敏感信息
4. 参数长度限制不同
    - GET传送的数据量较小，不能大于2KB
    - POST传送的数据量较大，一般被默认为不受限制
    - HTTP协议没有BODY和URL字段的长度限制，对URL限制的大多是浏览器和服务器的原因
5. 编码方式不同
    - GET请求只能进行URL编码 `application/x-www-form-urlencoded`
    - POST请求支持多种编码方式 `application/x-www-form-urlencoded` 或 `multipart/form-data`。为二 进制数据使用多种编码。
6. 缓存机制不同
    - GET请求会被浏览器主动cache，而POST不会，除非手动设置

- GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留
- GET产生的URL地址可以被保存为书签，而POST不可以
- GET在浏览器回退时是无害的，而POST会再次提交请求

7. 时间消耗不同

    - GET产生一个TCP数据包
    - POST产生两个TCP数据包

    对于GET方式的请求，浏览器会把header和data一并发送出去，服务器响应200（返回数据）；

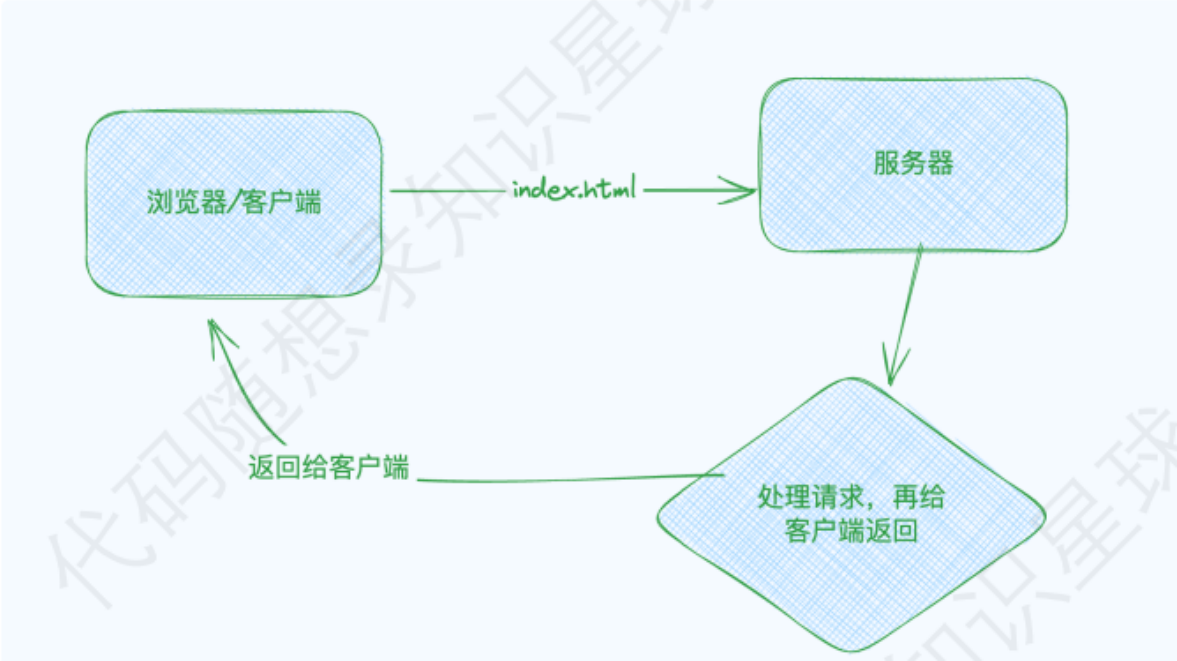
    而对于POST，浏览器先发送Header，服务器响应100continue，浏览器再发送data，服务器响应200ok（返回数据）

8. 幂等

    意思是多次执行相同的操作，结果都是【相同】的

    - GET方法就是安全且幂等的，因为它是【只读】操作，无论操作多少次，服务器上的数据都是安全的，且每次的结果都是相同的
    - POST因为是【新增/提交数据】的操作，会修改服务器上的资源，所以是不安全的，且多次提交数据就会创建多个资源，所以不是幂等的

2、什么是强缓存和协商缓存



缓存可以解决什么问题：

- 减少不必要的网络传输，节约带宽
- 更快的加载页面
- 减少服务器负载，避免服务过载的情况出现

强缓存

强缓存：浏览器判断请求的目标资源是否有效命中强缓存，如果命中，则可以直接从内存中读取目标资源，无需与服务器做任何通讯。

- `Expires强缓存`：设置一个强缓存时间，此时间范围内，从内存中读取缓存并返回，因为`Expires`判断强缓存过期的机制是获取本地时间戳，与之前拿到的**资源文件中的Expires字段的时间做比较。来判断是否需要对服务器发起请求。这里有一个巨大的漏洞：“如果我本地时间不准咋办？”所以**目前已经被废弃了。
- `Cache-Control强缓存`：`http1.1`中增加该字段，只要在资源的响应头上写上需要缓存多久就好了，单位是秒。`Cache-Control:max-age=N`，有**max-age、s-maxage、no-cache、no-store、private、public**这六个属性。
  - max-age决定客户端资源被缓存多久。
  - s-maxage决定代理服务器缓存的时长。
  - no-cache表示是强制进行协商缓存。
  - no-store是表示禁止任何缓存策略。
  - public表示资源既可以被浏览器缓存也可以被代理服务器缓存。
  - private表示资源只能被浏览器缓存，默认为private

协商缓存（了解）

1. 基于 `last-modified` 的协商缓存

- 首先需要在服务器端读出文件修改时间，
- 将读出来的修改时间赋给响应头的last-modified字段。
- 最后设置Cache-control:no-cache
- 当客户端读取到last-modified的时候，会在下次的请求标头中携带一个字段:If-Modified-Since，而这个请求头中的If-Modified-Since就是服务器第一次修改时候给他的时间
- **之后每次对该资源的请求，都会带上If-Modified-Since这个字段，而服务端就需要拿到这个时间并再次读取该资源的修改时间，让他们两个做一个比对来决定是读取缓存还是返回新的资源。**

缺点：

- **因为是更具文件修改时间来判断的，所以，在文件内容本身不修改的情况下，依然有可能更新文件修改时间（比如修改文件名再改回来），这样，就有可能文件内容明明没有修改，但是缓存依然失效了。**
- **当文件在极短时间内完成修改的时候（比如几百毫秒）。因为文件修改时间记录的最小单位是秒，所以，如果文件在几百毫秒内完成修改的话，文件修改时间不会改变，这样，即使文件内容修改了，依然不会 返回新的文件。**

2. **基于 `ETag` 的协商缓存：**将原先协商缓存的比较**时间戳**的形式修改成了比较**文件指纹（根据文件内容计算出的唯一哈希值）**。

- 第一次请求某资源的时候，服务端读取文件并计算出文件指纹，将文件指纹放在响应头的Etag字段中跟资源一起返回给客户端。
- 第二次请求某资源的时候，客户端自动从缓存中读取出上一次服务端返回的ETag也就是文件指纹。并赋给请求头的if-None-Match字段，让上一次的文件指纹跟随请求一起回到服务端。
- 服务端拿到请求头中的if-None-Match字段值（也就是上一次的文件指纹），并再次读取目标资源并生成文件指纹，两个指纹做对比。如果两个文件指纹完全吻合，说明文件没有被改变，则直接返回304状态码和一个空的响应体并return。如果两个文件指纹不吻合，则说明文件被更改，那么将新的文件指纹重新存储到响应头的ETag中并返回给客户端

缺点：

- **ETag需要计算文件指纹这样意味着，服务端需要更多的计算开销。。如果文件尺寸大，数量多，并且计算频繁，那么ETag的计算就会影响服务器的性能。显然，ETag在这样的场景下就不是很适合。**

3、HTTP1.0和HTTP1.1的区别

1. 长连接
    - `HTTP1.1` 支持长连接，每一个TCP连接上可以传送多个HTTP请求和响应，默认开启`Connection：Keep-Alive`
    - `HTTP1.0`默认为短连接，每次请求都需要建立一个TCP连接

2. 缓存
    - `HTTP1.0`主要使用 `If-Modified-Since/Expires`来做为缓存判断的标准
    - `HTTP1.1`则引入了更多的缓存控制策略例如 `Entity tag / If-None-Match`等更多可供选择的缓存头来控制缓存策略
3. 管道化
    - 基于`HTTP1.1`的长连接，使得请求管线化成为可能，管线化使得请求能够并行传输，但是响应必须按照请求发出的顺序依次返回，性能在一定程度上得到了改善
4. 增加Host字段
    - 使得一个服务器能够用来创建多个Web站点
5. 状态码
    - 新增了24个错误状态响应码
6. 带宽优化
    - `HTTP1.0`中，存在一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了，并且不支持断点续传功能
    - `HTTP1.1`则在请求头引入了`range`头域，它允许只请求资源的某个部分，即返回码是 `206（Partial Content）`

4、HTTP2.0和HTTP1.1的区别

1. 二进制分帧

   在应用层 （HTTP/2.0） 和传输层 （TCP or UDP） 之间增加一个二进制分帧层，从而突破 HT TP1.1 的性能限制，改进传输性能，实现低延迟和高吞吐量。

2. 多路复用（MultiPlexing）

   允许同时通过单一的 HTTP/2 连接发起多重的请求-响应消息，这个强大的功能则是基于“二 进制分帧”的特性。

3. 首部压缩

HTTP1.1 不支持 header 数据的压缩， HTTP/2.0 使用 HPACK 算法对 header 的数据进行压缩，这样数据体积小了，在网络上传输就会更快。高效的压缩算法可以很大的压缩 h eader ，减少发送包的数量从而降低延迟。

4. 服务端推送（Server Push）

   在 HTTP/2 中，服务器可以对客户端的一个请求发送多个响应，即服务器可以额外的向客户端 推送资源，而无需客户端明确的请求。