**http-something：**

一：基本概述

1. http是一种客户端-服务端进行数据交换、通信的协议。
   1. 客户端发送request请求到服务端，服务端处理该请求，并返回一个response。
   2. 客户端我总结为两种：**浏览器发送http请求、sdk构造http请求发送给服务器。**
   3. 客户端、服务器之间的还有路由器等组件，这些组件由网络层、传输层管理，而http是应用层协议，所以描述http时不需要考虑下一层。
2. 客户端，服务端工作流程
   1. 往往都有客户端端发起请求获取服务端的html文档，然后浏览器解析html文档，然后再次发送对文档中资源（css、对象存储）的一些请求，最后浏览器将所有的资源整合在一起，得到一个网页。 **网页时超文本文档，这意味着有一部分展示的内容时链接，用户可以点击链接，浏览器将再次发送http请求，然后解析响应的文档。**
3. 代理：客户端和服务端之间还有许多计算机设备参与http的传递
   1. 传递的信息会经历传输层、网络层、物理层，但是这些通常对于http协议即应用层时透明的。
   2. 还有一些应用层的代理。它们可以时透明的，单纯帮助转发http请求到服务器；也可以是不透明的，会对修改http请求信息。
      1. Nginx、网关、认证服务等
4. http是**可扩展的**
   1. http/1.0、http/2.0,这样的被称为标头，客户端和服务端之需要校验http版本，就可以加入新功能。
5. http是**无状态的**
   1. 同一个tcp连接中，**两个执行成功的请求是没有关系的**，这也就带来了一个问题，用户没办法连贯的交互（例如购物车信息无法保存）。
   2. **但是http可以借助cookie，使得http看上去是有状态的**。Cookie使得每个请求都可以共享上下文信息，是的http是**有状态的会话**。
6. http的连接
   1. http依靠tcp的可靠性，所以http不使用udp。
   2. 客户端与服务端建立在发送请求和响应前，必须建立tcp连接。http/1.0默认每个http请求都需要建立一次tcp连接，客户端需要发送多次请求时，这种方式效率必然很低。
   3. **所以http/1.1引入了流水线和持久化连接：可以通过Connection标头来控制tcp连接**。http/2.0更进一步，在一个连接中复合多个消息，让效率更高。
   4. 服务端客户端通信的步骤：
      1. 建立tcp连接、客户端发送请求、服务端响应
      2. http/1.1以后，只需要建立一次tcp连接即可，重复后面两个步骤即可。

二：http信息

1. http请求Request
   1. 起始行：方法、path（请求目标）、http版本
   2. 标头（Header）：**不区分大小写。**整个标头（kv键值对）由一行组成，可以很长。
      1. 分类：
         1. **请求标头**：出现在http请求中，包含客户端和请求的详细信息，用于帮助服务器更好的处理请求如：User-Agent(客户端软件的名称、版本)、Accept（客户端能够处理的内容类型）、Authorization（服务端进行身份校验凭证）、Connection、Accept-Encoding（客户端能够处理的文件压缩算法）
         2. **响应标头：**是http响应中的额外信息，帮助客户端理解如何处理服务端返回的数据如：Age（告诉客户端响应在缓存中存储的时间）、Location（指示客户端重定位到新的url，在响应码为3xx时使用）、Server（服务器软件的版本）、Content-Endoding（服务端响应采用的压缩算法）
         3. **表示标头：**可以出现在请求中也可以出现在响应中，用于表示消息主体信息body，描述和解释主体信息，如：Content-Type(资源MIME类型)、Content-Encoding（主体编码类型）、Content-Length（body长度）
         4. **有效负载标头：和表示标头一样？？？？？**
   3. Body主体：**不是所有的请求都有主体信息，**像GET、DELETE等没有主体信息。而POST请求会提交信息到服务器，会有主体信息。
      1. 主体分为两种
         1. 单一资源
         2. 多资源
2. http响应response
   1. 状态行：协议版本、状态吗、状态文本
   2. 标头
   3. Body主体：不是所有的响应都有主体，如响应码为201、204的响应不会有主体
      1. 主体分为三类
         1. 单资源主体，已知单文件长度。
         2. 单资源主体，未知长度单文件，
         3. 多资源主体

三http协议中的数据压缩

1. 端到端的压缩技术
   1. 消息主体在服务端压缩完成后，在传输过程中不变（不管中间遇到什么样的节点，都不会改变压缩数据），抵达客户端后，客户端根据响应的压缩算法进行解压缩。
   2. 服务端与客户端采用哪一种压缩算法，是通过主动协商选择的。客户端发送的request中会设置**Accept-Encoding**，表示自己能够处理的压缩算法。服务端选取一种，进行消息主体压缩后，会在返回的response中设置**Content-Encoding**告诉客户端采用的压缩算法。
   3. 常见的压缩算法：gzip、br

**四.http缓存(缓存的是服务端的响应)**

1. http缓存技术作用
   1. 使得客户端（浏览器）响应更快
   2. 服务端不需要再次处理请求减轻服务器压力。
2. http缓存分类：**私有缓存**、**共享缓存**
   1. **私有缓存：**浏览器缓存
      1. 私有存储的缓存数据不与其他用户共享，因此可以缓存一些个性数据、私有数据。
      2. 如果响应中具有个性消息，想要将响应缓存在私有存储中。可以设置一个header叫：Cache-Control: private
   2. **共享缓存：**CDN、反响代理等
3. 基于age的缓存策略：
   1. 缓存中的响应有两种状态：fresh和stale。Fresh表示响应还可用，stale表示响应过期了。
   2. 如何知道缓存返回给客户端的响应时fresh还是stale呢？
      1. 基于age，age是自响应在缓存中生成后经历的时间。
      2. 第一，服务端返回response中设置了一个header叫：Cache-Control: max-age=604800，**max-age表示响应在缓存中能够待的最大不过期的时间。**第二**，**客户端发送请求后，缓存返回该响应并且计算了缓存已经待的时间Age。**若缓存在浏览器**，那么会直接判断Age是否小雨max-age，小于则是fresh。**若在共享缓存中**，则客户端发送请求后，缓存返回响应时，会设置Age这个Header，收到响应的客户端会判断是否时fresh。
4. Expire和max-age
   1. Expire是Header，在http/1.0版本中使用。Expire使用明确是的时间，而不是通过计算经过的时间来指定缓存。
   2. 由于时间格式难以解析，故http/1.1中推出了max-age方式。如果Expire和max-age均出现的话，首先考虑max-age。
   3. http/1.1无需特供Expire
5. **Vary响应**
   1. 缓存中区分不同响应是根据URL，即URL作为Key。但是往往客户端发送的请求的URL是相同的，但是想要获得不同Accept、Accept-Encoding、Accept-Language等标头的响应。
      1. 例如：缓存中已经有一个响应，该响应有Accept-Language: en，但是客户端发送相同URL请求，想要获得的响应时Accept-Language: ja。
      2. 这种情况可以在Header中添加Vary: Accept-Language，根据语言单独缓存。
      3. 这会导致缓存根据URL和Vary共同作为key
6. 验证响应：过时的响应不会立即删除，而是询问服务器将原响应更新为新的响应。
   1. 标头**If-Modified-Since**：响应过期后，客户端显示发送请求给服务器，带有标头If-Modified-Since: 原响应生成时间，服务器会判断响应是否有修改过。如果没有修改过，则服务器返回303 Not Modifed，响应非常小。客户端会延长缓存中的响应过期时间。
   2. Etag/If-None-Match：服务器第一次返回响应中会携带Etag标头，对应的值的是body的hash值，响应存储到客户端。当客户端发现缓存的响应过期了，则会显示发送请求，并携带Not-None-Match标头，对应的值时过期响应中Etag的值，以询问服务器请求的资源是否修改过。如果hash值和Not-None-Match相同，则返回响应303 Not Modified。
   3. 强制重新验证:不会重复使用缓存中的响应，即使没有过期。
      1. Cache-Control: no-cache:响应会携带Cache-Control: no-cache和Etag、Last-Modified三个标头。 每次客户端都会去询问服务器是否过期，如果过期则返回200 ok，如果没过期则返回303 Not Modified
      2. Cache-Control: max-age: 0, must-revalidate和Cache-Control: no-cache作用一样。在HTTP/1.1以后使用no-cache
7. 不使用缓存：
   1. no-cache并不是阻止存储缓存，而是在没有验证情况下，不使用缓存。若要阻止存储缓存应该用Cache-Control: no-store
   2. 但是不建议使用no-store。第一有兼容问题，第二这样会丢弃http缓存优势。而是使用no-cache，每次询问一下服务端。如果考虑用户个性化数据安全问题，就是用Cache-Control: no-cache, private。
8. 删除存储中的响应
   1. 如果Cache-Control: max-age设置的过大,导致客户端的请求每次都直接使用缓存中的响应，服务器长时间无法收到客户端的请求，使得服务端失去对响应的控制。
   2. 我们应该考虑在服务端返回的响应中添加Cache-Control: no-cache，是的每次服务端都能收到请求，并能够够控制响应。
9. 请求折叠：针对于http共享缓存
   1. 多个客户端发送对相同服务器资源的请求，经过共享缓存服务器时，共享缓存服务器只会发送一个请求到服务器，然后服务器返回响应到共享缓存服务器。
   2. 如果针对特定用户个性化，服务器返回的响应中可以使用Cache-Control: private

**五.http重定向**

1.原理：**服务器返回响应是具有3xx的状态吗，以及设置了Location标头，浏览器收到响应后，会立即加载Location的URL然后发送新的请求。这个过程对用户是不可知的。**

**六.http条件请求**

1.概述：响应结果会因为特定首部标头不同而不同。

2.使用场景：

**http缓存更新**中涉及到的一些标头：Cache-Control等

**增量下载/断点续传**

**七.http协商机制**

**八.http连接管理**

1.http传输协议主要是依靠tcp建立端到端的连接。早起的http/1.0版本的http协议只有**短连接**方式，即每次http请求都需要tcp建立连接，这很浪费时间并且消耗服务器资源。

2.http/1.1提供了**长连接、http流水线**两种模型

**a.**长连接：http/1.1默认使用长连接模型，除非主动设置Connection: close。但是长连接也会消耗服务器资源、有dos攻击的风险。

**b.**http流水线：默认浏览器采用长连接模型，即请求都是顺序的发出，只有接收到响应后才能发送下一个请求，如果上一个请求因为网络原因阻塞了，会有等待的感觉。http流水线模型会在同一个连接上连续的发送多个请求，并且响应可以打包到同一个tcp包中返回，这样效率很高。**但是不是所有的http请求都支持流水线模型，只有满足幂等的请求方法才允许比如GET、PUT、DELETE、HEAD。**

**3.**