**Request & Response**

**Request格式**

<request-line> 比如：GET /api/index.json HTTP/1.1 <headers> 比如：Accept: \*/\*; User-Agent: Mozilla/4.0;…… <blank line> [<request-body>] 比如：id=1&timestamp=xxxxxx

**Response格式**

<status-line> 比如：HTTP/1.1 200 OK <headers> 比如：Content-Type: application/json;…… <blank line> [<response-body>] 比如：{"id":1,"username":"testuser"}

**Status Code**

http的状态码有将近60个，我这里主要记录一些常见的非正常情况下产生的状态码，在平常应用中或多或少会碰到，有助于我们去理解和发现问题。

206 - 断点下载时用到，客户端请求了一部分内容，服务器成功把这部分内容返回给它，这时候就是用这个状态。

301 - 永久跳转，原地址不存在了，url被指向到另一个地址。这个主要是搜索引擎相关，影响爬虫的检索行为。

302 - 临时跳转，服务器会返回一个新的url给客户端，客户端可以继续访问这个url来获取内容。

304 - 资源没有改变，客户端可以使用本地缓存的内容，常见于静态内容访问。

413 - 请求实体太大。常见的情况是上传大文件，但超出了服务器(比如nginx)限制。或者请求头或请求体超出了后端的server(比如tomcat)的设置（比如当前域名下cookie太多，超出了请求头限制）

416 - 跟断点续传有关，客户端请求的范围超出了服务器上文件大小。

500 - 服务器内部错误，不能返回正常的结果。比如最常见的应用抛出空指针异常未进行处理。

502 - 网关错误。常见的情况是反向代理后端的服务器(比如resin或tomcat)没有启动。

503 - 服务不可用。比如服务器负载太高或者服务器已经停止服务。

504 - 网关超时。比如请求时长超出了服务器的响应时间限制。

**Headers**

http headers分为请求头(Request Header)和响应头(Response Header)两类。下面是我们经常用到的一些header.

**1.缓存控制**

在互联网站的应用中，缓存几乎无处不在，在基于http的服务中，我们也可以对一些不常改变的内容在客户端进行缓存，这样在多次访问中可以复用缓存内容，加快访问速度，提升用户体验。http的协议里规定了一些用于缓存控制的http消息头：

**Cache-Control(HTTP/1.1)/Pragma(HTTP/1.0)**：指示客户端是否进行缓存以及缓存的时间是多长。默认值是private，也就是把内容缓存在用户私有空间。比如：Cache-Control:max- age=86400,must-revalidate，这是告诉客户端所请求的资源缓存一天（max-age单位是秒，相对时间），过期之后必须进行重新检验。

**Expires**：指定客户端（如果不强制刷新的话）在多长时间里可以不向服务器发请求，直接读本地缓存。

注意：

1. 优先级：Cache-Control > Expires；
2. 详细参数说明：<http://condor.depaul.edu/dmumaugh/readings/handouts/SE435/HTTP/node24.html>
3. 不同浏览器的不同行为（刷新，后退，地址栏回车等）在实现上可能有差异；

**Last-Modified/If-Modified-Since**：Last-Modified是服务器端返回给客户端的资源最后修改时间戳，这样，客户端在下次请求时（比如强制刷新）会带上If-Modified-Since参数来校验资源是否有更新，没有更新的话服务器就返回304状态码，客户端直接取本地缓存的资源。这个时候只有请求开销，没有网络传输开销。注意：时间戳必须是格林威治(GMT)时间，比如：Last-Modified:Sat, 19 Oct 2013 09:20:15 GMT

**ETag/If-None-Match**：ETag是根据文件属性通过一定算法生成的资源标识，也是用来确定客户端请求的资源是否有更新。如果服务器返回了一个ETag值给客户端，那么下次客户端请求时会带上If-None-Match参数来校验资源是否更新，没有更新的就返回304状态码。（效果基本等同于Last-Modified）

注意：

1. ETag需要计算，对于计算资源紧张的服务器来说是一种消耗，所以有些网站直接不使用ETag；
2. 如果服务器在负载均衡后面，同一个资源的请求可能分发到不同的后端机器上，由于ETag的计算依赖于文件属性，不同机器上内容相同的文件可能生成的ETag不同，这样就可能使本来内容没变的文件通过ETag校验失败。这里有两种解决方案：一是etag计算不依赖于本地机器，比如直接算文件内容的md5值；二是在负载均衡器上把相同的url请求分发到同一台后端机器。

在我们的实际业务场景下，http的缓存具有非常大的用途，下面列举一些：

1. 充分利用客户端的资源，比如一些客户端需要频繁访问的静态文件，像LOGO，广告图等，完全可以缓存在客户端本地。这样可以减少网络请求，加快客户端展示，还能减少服务器请求的压力。
2. 我们的一些静态内容，比如新闻，博客等，在被搜索引擎爬虫抓取的时候，通过控制缓存参数，就可以减少爬虫的抓取频率，减少不必要的资源浪费。
3. 如果我们的静态资源使用了CDN，那么设置了http缓存就可以在CDN节点上保存一份文件，减少CDN的回源次数，减少网络延时和源站服务器压力。

**2.断点请求**

**Accept-Ranges**：服务端支持断点下载时会返回这个响应头给客户端，当客户端知道这个以后就可以发送断点请求了。

**Content-Length**：响应信息的长度，告诉客户端当前请求返回了多少数据。这里要注意一下，用head方法提交请求时不会返回具体数据，但是这个Content-Length会返回完整数据的大小。

**Range/Content-Range**：客户端请求时提交名为Range的header，告诉服务器自己要请求哪部分的数据。比如：Range: bytes=0-1023表示请求第0到1023个字节.然后服务器返回这1024个字节的内容给客户端，响应头中会带上Content-Range。即：Content-Range: bytes 0-1023/4096，这个4096就是文件总大小。客户端下次请求可以从第1024个字节处开始，Range: bytes=1024-xxxx

**3.编码**

**Accept-Encoding/Content-Encoding**：前者是客户端支持接收的消息编码类型。默认是identity，可选值有gzip,compress等。后者是服务器端响应信息的内容编码类型，常用的就是压缩。压缩的好处显而易见，可以大大减少网络传输的开销，相对于服务器端压缩产生的cpu消耗，网络传输的减少显然更实在。常见形式：Content-Encoding: gzip,deflate,compress.通常我们对html,js,css,xml,json之类的响应结果可以进行压缩传输。

**Transfer-Encoding**：response header.响应消息的传输编码类型，规定了网络传输的形式。一般都是下面这种形式：Transfer-Encoding: chunked。当服务器产生动态内容，不知道响应信息的具体长度时，可以通过这个指定分块进行传输，处理多少数据就返回多少数据，这样不用等到数据都准备好了一次性返回。结合上面的内容编码，比如gzip，可以分块压缩并进行传输。另外，请注意，在使用这种编码传输时，我们是看不到Content- Length的，因为内容还没有完全生成。

**4.其他**

**X-Forward-For**：request header. 用来标识用户的真实ip，特别是通过代理(正向或反向)访问服务器或是服务器在负载均衡设备后面的情况。格式：X-forward-For: client,proxy1,proxy2,…最左边的是最接近客户端的ip。

**User-Agent**：request header.服务器用来识别客户端基本信息的请求头。一般这个在识别搜索爬虫的时候有用，某些场景下也可以用这个来做一些客户端的统计。

**Referer**：request header.客户端访问服务器时，这个Referer来指定请求来源，比如是从哪个网站链接过来的，我们在一些统计中会经常用到这个。另外，还有一个重要的用途就是在需要资源防盗链的场景中来过滤非法的请求来源（但是，这个referer是客户端可以伪造的）。

**Location**：response header.在301/302状态码的响应头中，都会带上这个Location头，来指示客户端用新的地址去访问需要的资源。

**Connection**：request/response header.在http/1.1中，客户端和服务端默认都是保持连接的，也就是Connection: keep-alive.如果任何一方不想保持连接，都可以把这个值设置为close.默认情况下，客户端和服务端会保持一个长连接，这样客户端就可以用这个连接发送多次http请求，减少频繁创建连接带来的消耗。对于这个参数，在服务端可能要做更多的设置，比如连接keep-alive的时间，服务器内核的一些网络参数设置(针对tcp)。

**Session和Cookie**

http请求是无状态的请求，但是在我们的互联网应用中，经常需要标识用户状态信息来完成一些交互性的操作，比如用户认证要记录用户登录状态，购物车应用要记住用户选择的商品，广告投放应用要记录用户的历史浏览行为等等。这里就会用到session和cookie了。

**session**:是指http请求-响应的过程中客户端与服务器端的交互状态，这些信息被保存在服务器端，比如内存，数据库等。每个session都有一个唯一标识，由服务器生成，这个标识也要在客户端进行保存，这样客户端在下次请求时可以带上这个标识，方便服务器判断客户端的状态。

客户端对session的支持：

1. 通过cookie保存session id，在请求时发送给服务器。
2. 通过url的参数携带session id与服务器通信。
3. 通过表单的隐藏字段携带session id与服务器通信。

session共享的问题：

在分布式应用中，我们的http server一般都架在反向代理或是负载均衡设备后面，这就会面临一个session共享的问题。也就是同一个用户的多个请求可能被分发到多个不同的机器，如果我们把session保存在机器本地内存中的话，就无法在多个机器间共享用户的session。这个问题，一般来说，我们可以有两种方式来解决：

1. 把session存放到分布式的内存(eg:memcached)或是集中式存储中(eg:database)。
2. 在反向代理或负载均衡设备上把相同用户的请求分发到同一台机器（这里要处理好机器宕机后请求重新分配的问题）。

**cookie**:在客户端保持状态化信息，每个cookie内容都属于特定的域(domain)和路径(path)，出于安全考虑，不同域或路径下的cookie不能共享。

会话cookie：没有指定过期时间，保存在内存，浏览器关闭后就失效。

持久cookie：指定了过期时间，保存在浏览器本地。

详细内容可以参考：<http://en.wikipedia.org/wiki/HTTP_cookie>

需要注意的是cookie会存在一些安全方面的问题。

在这里我只是总结了自己在工作中遇到的与http协议相关的一些内容的理解，http协议还有很多需要挖掘的东西，也需要不断去探索，对http协议的理解将会给我们的开发应用带来很大的便利。

最后，推荐两个很NB的http调试工具：fiddler(windows)和charles(mac)有http代理功能，对于不是基于浏览器的http应用(比如mobile app)，可以用这两个工具来监控http请求。