Http协议：

1. 超文本传输协议，明文传输，是浏览器与服务器之间传输信息的规范；
2. Http协议传输无状态；
3. 常用的http请求包含get，post，put，delete，header；
4. Http请求包含请求头和请求体；
5. 请求头常包含的字段有：
6. GET ：请求类型 路径：被请求资料的绝对路径 HTTP/1.1 http协议的版本
7. Accept：该请求可以处理的内容类型，如text/html,application/xhtml+xml,\*/\*
8. Referer: <https://www.baidu.com/>

解释：页面的引用，发送了请求A（https://www.baidu.com/）后，后面返回了多个请求（重定向），这多个请求的Referer就是请求A；

1. Referrer Policy:no-referrer-when-downgrade

解释：仅当发生协议降级（如 HTTPS 页面引入 HTTP 资源，从 HTTPS 页面跳到 HTTP 等）时不发送 Referrer 信息；在页面引入图片、JS 等资源，或者从一个页面跳到另一个页面，都会产生新的 HTTP 请求，浏览器一般都会给这些请求头加上表示来源的 Referrer 字段，Referrer 在分析用户来源时很有用，有着广泛的使用；

新的 Referrer Policy 规定了五种 Referrer 策略：No Referrer、No Referrer When Downgrade、Origin Only、Origin When Cross-origin、和 Unsafe URL

1. Accept-Language：请求可以接收的语言类型，如zh-CN
2. User-Agent：Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0) 解释：Mozilla/5.0(协议本身固定好的，不会变化)；compatible兼容Mozilla5.0，客户端浏览器的版号IE10.0；操作系统内核版本是windows nt 6.1，即windows7操作系统；64位系统；微软的渲染Trident6.0
3. Accept-Encoding：gzip,deflate

解释：请求可以接受的编码格式，即响应内容的压缩格式gzip,deflate是二种标准的用于http内容压缩的规范，一般响应内容较大时，会进行压缩后传输以减少带宽的使用，一般与响应的响应头的Content-Encoding:gzip对应（只有当响应的内容被压缩了，响应头里才会这一项）

1. X-requested-with:XMLHttpRequest：

解释：代表这个请求是通过Ajax方式发送的，Ajax是目前比较流行的客户端交互机制；

1. Host: [www.baidu.com](http://www.baidu.com)

解释：表式访问的主机，一般是IP或是域名

1. Connection:keep-alive

解释：长连接，就是建立一次连接后，就保持连接，不断开；因为如果不长连接，每次请求都要通过TCP协议的三次握手，非常消耗资源；长连接有时效性或是次数性，过期就要重新建立连接

1. Cookie:
2. Content-Length:46 请求体正文的大小
3. Cache-Control:no-cache 没有使用缓存
4. 响应头常包含的字段：
5. HTTP/1.1 200 OK
6. Server: Apache/2.2.9(Win32) 服务器的版本
7. Date:Wed 29 Apr 2018 19：43：20 GMT 接收请求的时间
8. 响应头Transfer-Encoding:chunked

解释：传输编码:分块，采用分块传输主要的考量是对于体积较大的文件（eg：图片、文件），服务器对响应内容按块压缩传输后，响应头就不会有Content-Length字段；服务器是否会压缩内容的条件有四个：

1> conf配置文件中compression="on"； 支持压缩

2> compressionMinSize="2048"，响应内容的大小达到压缩要求的最小值2048；

3> compressableMimeType="text/xml,application/octet-stream,application/json"，响应传输的内容类型属于这里配置的类型之一；

4> 请求头中包含Accept-Encoding字段，即请求可以接收的压缩方式

满足以上四个要求时，响应内容就会被分块压缩

conf/server.xml文件配置的属性：

compression="on"

compressionMinSize="2048"

compressableMimeType="text/xml,application/octet-stream,application/json"

1. 响应头Pragma:no-cache和Cache-Control:no-cache,no-store,max-age=0的区别：

解释：Pragma:no-cache --> 编译控制：无缓存

Cache-Control:no-cache --> 超高速缓存控制，响应是有被缓存的，只是每次发头响应数据时，缓存的响应数据都要向服务器评估缓存响应是否有效；

总结：Pragma: no-cache：跟Cache-Control: no-cache相同，Pragma: no-cache兼容http 1.0 ，Cache-Control: no-cache是http 1.1提供的。因此，Pragma: no-cache可以应用到http 1.0 和http 1.1,而Cache-Control: no-cache只能应用于http 1.1

1. Expires:Thu, 01 Jan 2018 10:20:50 GMT

解释：缓存的过期时间，一般是和Cache-Control字段配配套来控制缓存的；

1. 响应头Set-Cookie: 服务器生成cookie，响应给客户端，客户端会将cookie保存到本地cookie文件；
2. Content-Length:16

解释：响应内容的长度，用于浏览器判断当接收的内容到达16字节后，就说明响应接收完成了

1. Cache-control常见的取值有private、no-cache、max-age、must-revalidate等，默认为private。其作用根据不同的重新浏览方式分为以下几种情况：
2. 打开新窗口

如果指定cache-control的值为private、no-cache、must-revalidate，那么打开新窗口访问时都会重新访问服务器获取响应。而如果指定了max-age值，那么在此值内的时间里就不会重新访问服务器，而是在缓存中获取响应数据，例如：Cache-control: max-age=5，表示当访问此网页后的5秒内再次访问时不会去访问服务器获取响应，而是从缓存中获取响应数据；

1. 在地址栏回车

如果值为private或must-revalidate（和网上说的不一样），则只有第一次访问时会访问服务器，以后就不再访问。如果值为no-cache，那么每次都会访问服务器重新获取响应数据。如果值为max-age，则在过期之前不会重复访问服务器。

1. 按后退按扭

如果值为private、must-revalidate、max-age，则不会重新访问服务器，而如果为no-cache，则每次都重复访问服务器；

1. 按刷新按扭

无论为何值，都会重复访问服务器

1. Session的作用：解决http协议无状态的问题，服务器为接收到的每个请求分配一个sessionid（会话id）来标识每个请求，以达到保存http请求状态的目地；sessionid管理机制：将sessionid保存在内存里面进行管理，或是以表格的形式保存在数据库中；
2. session的常见实现方法是基于cookie，客户端需要支持Cookie，Web Server在返回Response的时候，在Response的Header部分，加入一个“set-cookie: jsessionid=XXXX”header属性，把jsessionid放在Cookie里传到客户端；客户端会把Cookie存放在本地文件里，下一次访问Web Server的时候，再把Cookie的信息放到HTTP Request的“Cookie”header属性里面，这样jsessionid就随着HTTP Request返回给Web Server；
3. session生命周期：session在用户第一次访问服务器时自动创建，注意只有访问JSP，Servlet等程序时才会创建session，只访问html，image等静态资源并不会创建session；服务器会把长时间没有活跃的session(超过了session的有效期时间)自动失效，且从内存中删除以回收内存；
4. http-关于application/x-www-form-urlencoded等字符编码的解释说明：

在Form元素的语法中，EncType表明提交数据的格式 用 Enctype 属性指定将数据回发到服务器时浏览器使用的编码类型。 下边是说明： application/x-www-form-urlencoded： 窗体数据被编码为名称/值对。这是标准的编码格式。 multipart/form-data： 窗体数据被编码为一条消息，页上的每个控件对应消息中的一个部分。 text/plain： 窗体数据以纯文本形式进行编码，其中不含任何控件或格式字符。

form的enctype属性为编码方式，常用有两种：application/x-www-form-urlencoded和multipart/form-data，默认为application/x-www-form-urlencoded。 当action为get时候，浏览器用x-www-form-urlencoded的编码方式把form数据转换成一个字串（name1=value1&name2=value2…），然后把这个字串append到url后面，用?分割，加载这个新的url。 当action为post时候，浏览器把form数据封装到http body中，然后发送到server。 如果没有type=file的控件，用默认的application/x-www-form-urlencoded就可以了。 但是如果有type=file的话，就要用到multipart/form-data了。浏览器会把整个表单以控件为单位分割，并为每个部分加上Content-Disposition(form-data或者file),Content-Type(默认为text/plain),name(控件name)等信息，并加上分割符(boundary)。

1. PHP超文本预处理器，是一种html内嵌式的语言，是一种在服务器端执行的嵌入html文档的脚本语言。
2. 浏览器的缓存是在硬盘中的，一般常说的缓存是在内存中的；
3. Web服务器一般指网站服务器，是指驻留于因特网上某种类型计算机的程序，可以向浏览器等Web客户端提供文档， 也可以放置网站文件，让全世界浏览；可以放置数据文件，让全世界下载。目前最主流的三个Web服务器是Apache Nginx IIS；
4. Apache软件基金会（也就是Apache Software Foundation，简称为ASF）是专门为运作一个开源软件项目的Apache 的团体提供支持的非盈利性组织，这个开源软件的项目就是 Apache 项目；
5. Tomcat是由Apache软件基金会下属的Jakarta项目开发的一个Servlet容器，按照Sun Microsystems提供的技术规范，实现了对Servlet和JavaServer Page（JSP）的支持，并提供了作为Web服务器的一些特有功能，如Tomcat管理和控制平台、安全域管理和Tomcat阀等；
6. Apache是web服务器，Tomcat是应用（java）服务器，它只是一个servlet容器，是Apache的扩展。 Apache和Tomcat都可以做为独立的web服务器来运行，但是Apache不能解释java程序（jsp,servlet）；
7. 响应token：LtpaToken（Lightweight Third Party Authentication Token），LTPA(Lightweight Third Party Authentication)技术是IBM的标准。当某用户访问某WebSphere URL时,系统会提示他输入用户名和口令进行登录。这时用户可以输入他的惟一标识符,通过验证后,Web服务器将把该用户的Web 浏览器中显示的Web 站点内容发送回来。在场景后台,WebSphere入口网站服务器将会建立包含已鉴别使用者认证的单点登录Cookie(默认值是LTPA记号),并且会一直发送该cookie, 而浏览器通常的默认设置是允许接收cookie的,因此用户的浏览器将保存这个cookie。 LTPA cookie是临时的,只在浏览器内存中存留,用户如果关闭浏览器,cookie就会被永久删除。LTPA cookie的特点如下：

(1)LTPA cookie是一种典型的浏览器cookie。LTPA cookie特有的名称是LtpaToken。当配置 SSO时,在配置实用工具中,LTPA cookie有一个被编码值,隐藏起cookie中包含的重要信息并且通过Internet传输。

(2)LTPA cookie 不能跨域，通常SSO环境必须部署到单一DNS域中,即每台服务器都在同一DNS域中。

(3)在用户已经登录并且该用户的浏览器接收到 LTPA cookie以后,在HTTP通信中不再需要进行特定的配置,浏览器运行的标准方法就是浏览器将自动发送该cookie。浏览器不断地向任何正确的DNS域中的URL目标发送HTTP请求，通过这种途径不断地向外发送LPTA cookie。当SSO服务接收到HTTP请求并且发现请求中包含了LTPA cookie时,服务器将验证cookie，随即可知道该cookie属于哪一位已经登录的用户，服务器就可以允许这个用户对这台服务器进行适当的访问。浏览器的任务就是确定在什么时候应该随同HTTP通信一起发出LTPA cookie。当用户浏览到一个不在同一DNS域中的URL时, 因为该cookie不适用于这个新的DNS域,浏览器则不会发送 LTPA cookie,新的DNS目标的接收服务器就不知道用户是谁,这时会提示用户输入他的用户名和口令。

(4)LTPA cookie是安全的,因为服务器在创建它时，使用一组加密密钥进行了安全加密。加密密钥用于对cookie进行编码,编码后的cookie传送到用户浏览器,而浏览器只对有加密密钥的cookie进行解码和验证cookie的完整性,并随时检测cookie是否被篡改过。在SSO环境中的所有服务器必须共享同一个加密密钥。当SSO服务器接收到HTTP请求并发现其中包含LTPA cookie时,就使用它共享的加密密钥副本验证cookie,这时有效的cookie信息就使服务器能够识别出登录的用户。

SSO服务器使用的安全加密确保了没有任何伪造cookie的机会。没有加密密钥,其他非法 的cookie不会通过验证,伪造的cookie将被忽略。因此，SSO服务器不会被入侵。

在WebSphere Portal环境中,LTPA加密密钥通常在配置SSO时由WebSphere 创建。管理员可以将密钥导出到文件中,然后转移该文件到其他的SSO服务器（例如Domino）,在那里导入密钥。系统的管理维护人员应该非常小心地处理密钥文件,把所有的副本保护好。

这样,LTPA技术就实现了WebSphere Portal门户网站的安全性；