1.理解一件事情，最好尽可能的知道其来龙去脉，这样才能尽可能的接近事务的本质。

2.关于浏览器客户端和服务端的会话，有几个问题，为什么会有会话的产生，会话怎么实现，会话能解决什么问题，是不是一定要用原生的Session对象来建立和保持会话。如果自己实现有哪几种方式。

先说第一个问题：为什么会有会话的产生。

我们知道浏览器和服务器之间的通讯是短连接无状态的，每一次交互，对于服务器来说都是一次全新的请求，那么如果我们没有一个什么东西在客户端和服务器端做勾连维护，那么可以说他们直接没有什么关系。

但是我们是否需要维护浏览器和服务器直接的关系呢，好让服务器知道这次请求是a客户端浏览器发出的请求，而不是b客户端浏览器发出的请求，需要吗？当然需要，服务器需要稳定的知道每一个请求是来自哪一个客户端，有了这样一个前提，

服务器端和每一个客户端浏览器都维护了一份勾连的钥匙，让服务器稳定的知道请求来自哪个客户端浏览器，这就是浏览器和服务器间的会话。进一步当用户在某一个客户端浏览器做了登陆操作时，可以在这个会话中保留用户登录过的信息，这样，用户在这个客户端

与服务器端做通讯时，服务器端就知道了，当前请求来的这个客户端浏览器在与我维护的会话中，用户是处于登陆状态的，那么服务器端对这个客户端浏览器响应的数据就是用户登录后有权访问的所有功能数据（也就是能登陆进首页看到所有的功能和数据）。

另外一种情况是客户端和服务器端维护了一个会话，但是用户并没有在这个会话中进行登陆，那么这个会话会依然存在，这个会话可以证明客户端就是a客户端，而不是b客户端，这个会话凭证依然可以做为服务器端对客户端的确认凭证，可以做一些事情，比如验证码。

还有一种情况，就是客户端浏览器跟服务器之间没有任何会话，什么时候产生会话是由服务器端决定的，当一个客户端浏览器触发的一个请求，在服务器端处理过程中，如果服务器端代码逻辑使用到了会话，那么就会有会话信息产生，并把会话凭证回传给浏览器客户端。

这样他们就建立起了会话，在此之前，他们的通信都是处于无状态的。这样做也是服务器端从性能考虑，没必要建立会话时，就不建立会话，当请求触发了服务器代码逻辑涉及到会话内容时再建立会话，这样就减少了不必要会话的产生，节省了服务器宝贵的内存资源。

再说第二个问题：会话怎么实现。

会话是如何实现的，是如何保持会话状态的，是如何进行勾连的，这不得不提他们直接的通信凭证，jsessionId，还有cookie机制。

客户端访问服务器端的接口中，如果某一个接口在后端服务器是需要访问session对象的，那么服务器会生成一个session对象并将session对象的id值写入浏览器的cookie中即JSESSIONID=xxxxxxxxxxxx。而cookie的本域保存机制保证了JSESSIONID会正确的返回到本次请求的客户端浏览器，

而不是其他端客户浏览器，这样JSESSIONID是服务器端生成的，通过cookie正确的返回给客户端保存的，它就是客户端访问服务器端的会话凭证。

他为什么能成为会话凭证？

比如说图形验证码的功能，客户端请求验证码，服务器端生成验证码，同时生成sessioon对象，将验证码保存在session对象中，即验证码依附在了session对象上，然后服务器端将JSESSIONID和图形验证码同时返回到客户端浏览器，当客户端浏览器下一次请求服务器验证验证码时，

由于http通讯是无状态的，服务器端要知道这次请求的客户端浏览器是谁，要知道可能同时有n多的客户端浏览器向服务器生成了验证码，服务器也都分别产生了对应的session对象并给予了JSESSIONID，那么现象现在到底是哪一个客户端浏览器再申请验证呢？

聪明的服务器通过JSESSIONID就可以轻松的找到浏览器客户端在之前请求过程中生成的session对象，JSESSIONID为空导致没找到或者session对象过去导致没找到或者客户端恶意伪造JSESSIONID导致没有找到，那就认为会话不存在，会从新产生新的session对象和JSESSIONID，重复刚才

的过程返回给客户端浏览器，当然本次从session对象中获取验证码信息是拿不到了，因为是新的session对象，所以本次验证码验证失败。

如果服务器通过JSESSIONID拿到了对应的session对象，则会从中取出验证码与用户输入的验证码进行比对，看是否正确。

整个过程串起来讲就很简单，一个客户端向我服务器端申请了一个验证码，我为这个客户端在服务器端的专门区域保存了验证码，当下次这个客户端浏览器请求验证码是否正确时，我服务器端正确的根据之前给浏览器的JSESSIONID找到了这个浏览器在我服务器开辟的专属会话区域，

在这个区域中我找到了属于这个浏览器客户端的所有相关信息，在里面找到了其中的一项验证码，进而进行验证。http的每次请求都是无状态的，但是JSESSIONID，cookie，和session保证了多次http会话是有状态的一样，这就是会话。

有人会问JSESSIONID泄露怎么办，当然没有绝对的安全，如果恶意第三方拿到了别人的JSESSIONID，通过技术手段放入自己的cookie中，就会进行正常的访问，但问题是恶意第三方通过什么手段能拿到别人的JSESSIONID呢？

1.我们想到了暴力破解，这很难，成功的概率很低，并且服务器端发现客户端大量的恶意访问会做次数验证，比如限定ip的访问次数，相信没有黑客会这么干。

2.在服务器响应给客户端浏览器JSESSIONID时做了请求拦截，这有可能，防御手段就是https，这样传输过程中即使被拦截，拿到的也都是加密数据，且无法解密，所以相当于没有拿大JSESSIONID。

3.硬件窃取，这一般就是关系比较亲密了，或者是偷盗到被害人的电脑设备，查询cookie中的值，进而将JSESSION放入自己的cookie中进行访问。如果是偷盗，那没办法了。如果是关系亲密，你自己没有防备，那也没办法了，如果是你自己主动给的，那还不如直接让他看你的首页功能权限呢。

所以说JSESSIONID在很大程度上是安全的，不需要怀疑。

再说第三个问题：会话能解决什么问题。

会话解决了服务器端对客户端的身份确认问题，一旦确认，就在会话期间，他们直接的交互看起来就像是长连接一样。

刚才说的解决验证码的问题也是一样。能在多端客户端浏览器的多次请求中，聪明的区分出每一个客户端浏览器的多次访问的信息区域存储，为每一个客户端浏览器保存了多次请求的专属会话数据存储区，保证了交互的连贯，每次交互的的输入如果想下次读取到是可以的，这个会话期间的数据是一致的。

解决用户登录也是一样，用户的登陆操作在要某一个浏览器客户端输入用户名密码完成，只要用户的用户名密码是对的，就会在浏览器端对应的服务器端的sesssion对象中存在用户信息，表明用户是登陆状态，本质上是说，服务器任务用户在这个JSESSIONID浏览器是处于登陆状态的，

用户在这个客户端浏览器可以访问他能访问的所有资源。

用户的同一个账号允许多端登陆和限制多端登陆也是一样的道理，不再多说了。

再说第四个问题：是不是一定要用原生的Session对象来建立和保持会话。

当然不是。

知道了整个会话的建立和保持的原理，我们可以自己实现会话，原生的Session对象只是Sevlet帮我们的默认实现，因为是一个太常用的功能，它就帮我们开发好了，我们一般直接用就好，只是集群环境下传统的session会话会比较吃力，我们更倾向于使用自己的会话机制。

再说第五个问题：如果自己实现有哪几种方式。

实现方式很多。

一.我们可以模拟原生session的做法，自己做一套实现，用法和传统用法一样，只是我们背后使用redis做session对象的存储，这样用法与原生session保持一致的基础上，还实现了服务端集群环境session的共享，推荐这种做法。

这种做法不再做太多描述，因为原生session会话我们太熟悉了，图形验证码，用户登录信息，手机或邮箱验证码等等功能都可以依附在session对象上，服务器与客户端浏览器只保留一个jsesion的会话凭证即可。

当然手机号或邮箱的发送次数限定是完全的另外一回事，要单独做，与会话通讯完全无关。

二.了解了原理，针对不同的功能，我们也可以做多套标记浏览器客户端和服务器端关系的jsessionId，这样没问题，原理也没问题，但是不推荐这样做，因为在一个系统中本来一个jsessionId可以搞定的事情，又根据不同的功能拆分出了多条线进行实现，每一条线分别保持自己的会话，不推荐这样。

因为这样随着功能线的拓展，可能每来一条线，就要保持自己这条线的服务器和浏览器之间的通信，增加了扩展成本，服务器端也被这么多条线的会话维护搞的可读性变差。

差归差，但仁者见仁智者见智，有人喜欢这么搞，那我们也可以说说它是怎么玩转的。

比如登陆功能图形验证码，会单做一个captcha\_login\_jsessionId的uuid在服务器端搞一个对象存储到redis中，同时将captcha\_login\_jsessionId作为uuid的键返回到浏览器客户端作为浏览器客户端“图形验证码”的专属会话凭证。

比如导出功能图形验证码，会单做一个captcha\_export\_jsessionId的uuid在服务器端搞一个对象存储到redis中，同时将captcha\_export\_jsessionId作为uuid的键返回到浏览器客户端作为浏览器客户端“图形验证码”的专属会话凭证。

比如登陆，会单做一个jsessionId的uuid在服务器端搞另一个对象存储到redis中，同时将jsessionId作为uuid的键返回到浏览器客户端作为浏览器客户端“用户登录状态”的专属会话凭证。

比如手机号或邮箱验证码登陆功能，会单做一个code\_phone\_ login\_jsessionId的uuid在服务器端再另搞一个对象存储到redis中，同时将code\_phone\_ login\_jsessionId作为uuid的键返回到浏览器客户端作为浏览器客户端“图形验证码”的专属会话凭证。

比如手机号或邮箱验证码修改密码功能，会单做一个code\_phone\_ modify\_pwd\_jsessionId的uuid在服务器端再另搞一个对象存储到redis中，同时将code\_phone\_ modify\_pwd\_jsessionId作为uuid的键返回到浏览器客户端作为浏览器客户端“图形验证码”的专属会话凭证。

比如手机号或邮箱验证码查看某重要功能二次确认验证码以验证用户身份，会单做一个code\_phone\_ xxx\_jsessionId的uuid在服务器端再另搞一个对象存储到redis中，同时将code\_phone\_ xxx\_jsessionId作为uuid的键返回到浏览器客户端作为浏览器客户端“图形验证码”的专属会话凭证。

三. 有没有一种做法可以同时做到上面两点呢。

有。那就是在redis中使用一个uuid来存储一个hash的对象，图形验证码，用户登陆信息，手机或邮箱验证码分别在hash中存入自己的key值。这样对于前后端交互依然是只用一个jsessionId即那个uuid。

好处是轻量级的实现了传统session会话的功能。

坏处是使用方法上不能像传统session用法一样，不过这也不算坏处，本身这种做法已经很轻量级，很简单了，没必要搞的跟传统session使用方式一模一样，就像第一种实现方式，用起来跟传统session一样，代码实现起来就很复杂了。

所以第三种方式也非常推荐。

针对于上面的三种做法，图形验证码可以按第一种做法依附到session上，也可以按第二种做法自己来单做一个captcha\_jsessionId，也可以按照第三种做法，也是依附到session上。当然，我们建议还是依附到session对象上就好，同一个客户端浏览器没必要跟服务器端建立两个会话对象来处理对应的事情，把所有跟会话相关的数据都封装到一个会话对象就好，客户端也就保持一个jsessionId就好了。

至于短信验证码或者邮箱验证码，要看你想要实现什么要的效果了，如果你想限制用户使用哪个浏览器客户端发送的验证码，就必须在这个浏览器输入验证码进行验证的话，那么他的过程跟图形验证码没有什么区别，只不过是验证码的显示由浏览器客户端推迟到了手机短信或邮箱中显示而已。

剩下的过程和图形验证码一样，一般我们会这样做，使用这种限制。

如果你不限制用户浏览器客户端，比如用户使用A浏览器点击发送手机验证码，然后接收到验证码后，在B浏览器客户端进入同样的页面输入验证码，然后依然可以通过，即只要验证码是合法的，哪个浏览器客户端都可以进行验证通过。这样也说的通，因为发送验证码的本质就是要确认是本人操作，

因为能收到验证码的一定是本人，至于他想通过什么浏览器客户端进行验证码的输入和提交认证是他的事情。

比如上面A浏览器点击发送验证码，B浏览器直接在对应模块输入验证码并提交认证，服务器端会单独验证用户这个手机号的服务器保存验证码是否在有效期内，在有效期内是否与用户传入的验证码匹配上，匹配上说明验证通过，用户身份合法，然后可以建立session会话，回传JSESSIONID到B

浏览器，这样用户最后是完成了在B浏览器客户端的登陆，而不是A浏览器。

这种场景，他的验证逻辑是独立的，是独立于session会话之外的，验证码验证是单独的逻辑，验证通过了，用户可以与最后提交验证码的浏览器终端建立会话信息，就是这么个过程。

如果你考虑是在用户登录到系统内，有的模块需要发送验证码再次验证用户身份，而这时如果用户会话已经过期，或者用户退出登录怎么办，没关系，各自走各自的，用户输入了验证码，验证码验证通过，但发现已经会话过期了，自然会让用户重新登陆系统，重新发送验证码，从新输入验证码就好了。

但是我们一般不会这样做，我们通常的做法还是会限制用户使用哪个浏览器终端发送的手机或邮箱验证码，那么就只能在这个浏览器客户端进行验证码的输入和提交认证。这样最大程度的避免不必要的麻烦，能满足客户需要，能满足安全性，越简单越清晰越好。

当然对于第一个场景，这个过程中，用户在提交验证码认证之前，可以频繁的换浏览器客户端进行验证码发送，而我们的验证码通常是有次数限制的， 比如一个小时之内最多发5次，一天最多发10次，这个限制应该是与浏览器客户端无关的，只是针对于某一个手机号或邮箱做的限制，所以这个限制应该在服务器端单独处理，与会话无关，至于手机号邮箱和其操作的功能模块有关。

不论上面哪种场景，手机和邮箱验证码的发送次数都是单独处理的。

广义会话本质：

服务器端为每一个访问服务器端会话的客户端生成并授予了唯一秘钥即jsessionId，当客户端下一次访问时，客户端凭借该唯一秘钥jsessionId，来让服务器唯一找到与该客户端对应的数据信息即会话信息即session对象，并保持在会话有效期内的交互一致连贯性。

这种连贯性在session会话失效之前一直保持，潜台词是用户在会话期间一直有对session对象的操作包括读取写入等连续操作对广义会话进行了续期。会话失效一般只有两种情况即会话自身时间到期（即用户长时间未进行操作）或者用户主动退出

（这种情况还看你如何处理，用户退出可以选择只删除session中的用户信息，然后静静的等待广义session会话的过期，这种情况广义会话依然存在，之前的就可以保持客户端和服务器端的通信。）。

服务器端永远不会直接去验证某一个jsessionId是否有效合法。它的处理逻辑很简单：

如果jsessionId是空的或者是服务器端不存在的jsessionId值，那么必然是非法的，我就直接new一个全新的session对象，那么你会发现从该session对象中取任何会话信息都是空的，因为jsessionId就是错的，

所以必然不存在对应的session会话对象与之对应，new出一个新的，所有的从新session中获取会话数据验证逻辑必然不能通过，然后巧妙的将这个新session对象的id值写入客户端浏览器，在之后的请求中，该浏览器就有了正确的凭证。

如果jsessionId对应的有服务器session对象，那么就说明这个jsessionId是合法的并且还处于广义会话期内，这样从就这个session对象中取对应的会话数据进行验证，必然就能验证通过。就是这样简单的道理。

对于用户的请求，不存在高并发的的情况，可以说不存在并发的情况，都是单线程操作，因为一个客户端就对应自己的一份服务器会话数据，不存在服务器会话数据可以多客户端同时操作的情况。

这就好像登陆用户可以访问自己的数据库数据一样，对于大部分功能模块，用户访问的服务器数据都是自己账号独有的数据，这种情况下，用户的每个操作都是属于串行操作，不存在并发访问相同服务器数据的情况。

而对于某些特殊功能模块，如果登陆用户可以访问并修改公共数据库数据，例如下单操作，库存就是公共数据，这才是高并发的情况，才会考虑并发问题解决，比如分布式锁的使用。

这里的session信息就属于服务器的数据信息，只是redis存储而非数据库存储，对于代码来讲，只是存储方式不同而已。session数据信息属于用户私有信息，不存在并发操作。

所谓用户登录了，本质上是指用户在当前的客户端与服务器的广义session会话中，通过输入用户名密码的方式证明了自己是合法用户然后在对应jsessionId的session对象中保留了用户信息，这样我们就说用户在当前的客户端浏览器与服务器端的广义会话中是处于登陆状态的。

简而言之，就说浏览器和服务器广义session会话（JSESSIONID,session对象对应起来的一种机制）中裹挟了用户登录的信息，那用户在这个广义session会话中就说登陆状态，反之，用户就是非登录状态。根本前提是广义会话要存在，即对应的jsessionId有与之的对应的session对象与之对应。

至于如何区分【未登录】和【会话已过期】，实在是比较不容易清晰区分的，比如用户在【未登录】和【会话已过期】的情况下，直接在浏览器请求首页地址的情况。

因为会话session过期后，session对象会从服务器端直接删除掉，用户登录的时候携带JsessionId来了，没有找到与之对应的sesssion对象，

找不到对应的session对象有很多种情况，比如：

1.比如如果session.invalidate()方法的策略是删除旧会话的同时生成新会话并回写新会话的jsessionId到客户端的情况，回写完成后，过了半个小时，服务端session删除，客户端浏览器再次请求服务器端时，使用的是新的jsessionId了，但是也找不到对应的session对象。

2.比如如果验证码生成了，对应的jsessionId就生成了，但是用户没有进一步的操作，然后过了半个小时服务器端的session对象失效了，而jsessionId在客户端cookie依然存在，此后，客户端再访问服务器时，这个jsessionId是找不到与之对应的session对象的。

3.比如如果session.invalidate()方法的策略是删除旧会话。当客户端浏览器再次请求时，jsessionId是找不到与之对应的session对象的。

4.比如用户长时间没有操作，服务器session对象过期了。当客户端浏览器再次请求时，jsessionId是找不到与之对应的session对象的。

这只是简单列举的几种情况，可能还有其他没列出的情况，你不好说是从来都没有登陆过，还是刚才登陆过，只是由于时间到了，我把session对象删除了。

所以我们最好就统一给一个【未登录或会话超时，请重新登陆。】的提示就完了，或者【会话失效，请重新登陆。】比较模糊的提示。

当然话说回来，如果硬要明确区分这两种情况，也必然是有办法的，做还是可以做的，自己要再做一些外围的操作。只是会大大增加会话的复杂度，有些得不偿失，没有必要。【未登录或会话超时，请重新登陆。】已经足够用了。