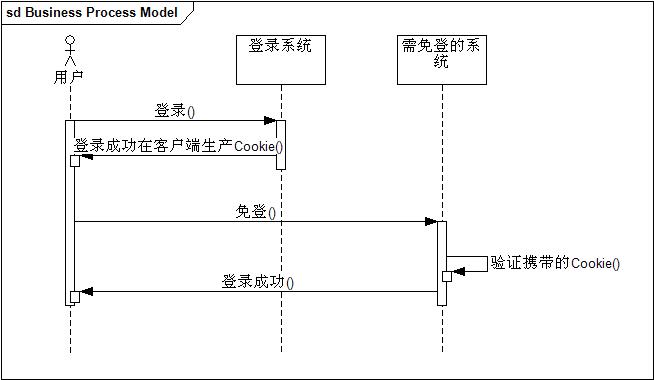
Ios端APP轮播广告图应用内外跳转

应用内外跳转分内部跳转和外部跳转，内部跳转就和普通的应用内部视图的变换控制器之间的跳转通信一样，关键的是应用的外部跳转，外部跳转和我们通常用的单点登录差不太多。就是在一个多系统共存的环境下，用户在一处登录后，就不用在其他系统中登录，也就是用户的一次登录能得到其他所有系统的信任。

APP外部跳转使用得非常频繁，例如像第三方分享的sdk里面有本地分享对象的APP客户端的情况下，用户点击分享会自动跳转信任的对象APP，当然实现外部跳转说到底就是要解决如何产生和存储那个信任，再就是其他系统如何验证这个信任的有效性，因此要点也就以下几个：

* 存储信任
* 验证信任

只要解决了以上的问题，达到了开头讲得效果就可以说是app外部跳转最简单实现方法就是用Cookie，实现流程如下所示：

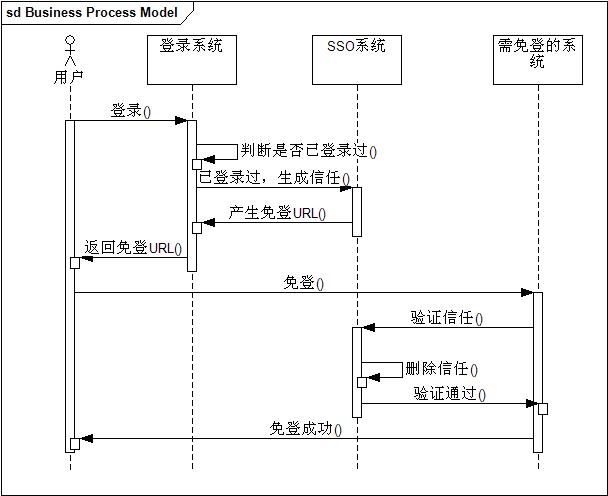


不然发现以上的方案是把信任存储在客户端的Cookie里，这种方法虽然实现方便但立马会让人质疑两个问题：

* Cookie不安全
* 不能跨域免登

对于第一个问题一般都是通过加密Cookie来处理，第二个问题是硬伤，其实这种方案的思路的就是要把这个信任关系存储在客户端，要实现这个也不一定只能用Cookie，用flash也能解决，flash的Shared Object API就提供了存储能力。

一般说来，大型系统会采取在服务端存储信任关系的做法，实现流程如下所示：



以上方案就是要把信任关系存储在单独的SSO系统（暂且这么称呼它）里，说起来只是简单地从客户端移到了服务端，但其中几个问题需要重点解决：

* 如何高效存储大量临时性的信任数据
* 如何防止信息传递过程被篡改
* 如何让SSO系统信任登录系统和免登系统

对于第一个问题，一般可以采用类似与memcached的分布式缓存的方案，既能提供可扩展数据量的机制，也能提供高效访问。对于第二个问题，一般采取数字签名的方法，要么通过数字证书签名，要么通过像md5的方式，这就需要SSO系统返回免登URL的时候对需验证的参数进行md5加密，并带上token一起返回，最后需免登的系统进行验证信任关系的时候，需把这个token传给SSO系统，SSO系统通过对token的验证就可以辨别信息是否被改过。对于最后一个问题，可以通过白名单来处理，说简单点只有在白名单上的系统才能请求生产信任关系，

以上只是提供了些简单的实现技术，但需要强调的是这只是技术实现而已，仅仅是为了解决上面谈到的一些问题，然而我们如果说在自己的APP相应地跳转就不需要那么繁琐的验证，因为我们本身就是绝对信任的。主要实现如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1，添加URL Types项** ttp://static.oschina.net/uploads/space/2014/1229/152737_iGIH_1440723.png  **2，配置URL Scheme**  a,展开URL types，再展开Item1，将Item1下的URL identifier修改为URL Scheme  b,展开URL Scheme，将Item1的内容修改为myapp http://ugc.qpic.cn/adapt/0/bc5de610-8e91-07fb-59ca-d3ca8ec21774/800?pt=0&ek=1&kp=1&sce=0-12-12  **3，其他应用的跳转**  **作为调用者的我，需要通过：**  [?](http://my.oschina.net/u/1440723/blog/361939" \t "_blank)   |  |  | | --- | --- | | 1  2  3 | NSString \*paramStr = [NSString stringWithFormat:@"myAppTest://username=%@&age=%@&address=%@", @"test123", @"100", @"上海市"];      NSURL \*url = [NSURL URLWithString:[paramStr stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]];      [[UIApplication sharedApplication] openURL:url]; |   **4，参数的接收**  那么作为一个Provider怎么去接收Customer传递过来的参数呢？  首先，在找到项目中的AppDelegate.m文件，然后找到openURL方法(如果没有就去实现它)。OK，到这里你已经完成了90%了，接着继续  [?](http://my.oschina.net/u/1440723/blog/361939" \t "_blank)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | - (BOOL)application:(UIApplication \*)application openURL:(NSURL \*)url sourceApplication:(NSString \*)sourceApplication annotation:(id)annotation  {      NSString \*urlStr = [url absoluteString];      if ([urlStr hasPrefix:@"myAppTest://"]) {          NSLog(@"TestAppDemo1 request params: %@", urlStr);          urlStr = [urlStr stringByReplacingOccurrencesOfString:@"myAppTest://" withString:@""];          NSArray \*paramArray = [urlStr componentsSeparatedByString:@"&"];          NSLog(@"paramArray: %@", paramArray);          NSMutableDictionary \*paramsDic = [[NSMutableDictionary alloc] initWithCapacity:0];          for (int i = 0; i < paramArray.count; i++) {              NSString \*str = paramArray[i];              NSArray \*keyArray = [str componentsSeparatedByString:@"="];              NSString \*key = keyArray[0];              NSString \*value = keyArray[1];              [paramsDic setObject:value forKey:key];              NSLog(@"key:%@ ==== value:%@", key, value);          }        }      return NO;  } |   **5，设备中是否安装了对象App判断**   BOOL hasInstagram = [[UIApplication sharedApplication] canOpenURL:url];     if (hasInstagram) {         [[UIApplication sharedApplication] openURL:url];     } 但是有一个问题，由于iOS9苹果官方加 入对用户隐私以及禁止扫描系统信息的控制，目前通过canOpenURL的方法来判断用户是否安装特定app，则会出现-canOpenURL: failed for URL: " myAppTest://app/\*\*\*\*\*\*\*/" - error: "This app is not allowed to query for scheme myAppTest "的错误，同时也返回NO,修改起来很简单，只需要在plist中加入myAppTest的白名单即可    <key>LSApplicationQueriesSchemes</key>          <array>                  <string>myAppTest</string>              </array> 到目前我们的互相信任的APP之间的跳转基本结束我们已经可以随心所欲的想怎么跳就怎么跳了 嘿嘿 开心吧！   当然我们以上所说只是实现了跳转传值功能，具体的怎么跳转，怎么加密解密还是很重要的，仍需完善的是我们从哪里跳到哪里具体到某一个界面仍然要根据需求来定！ |