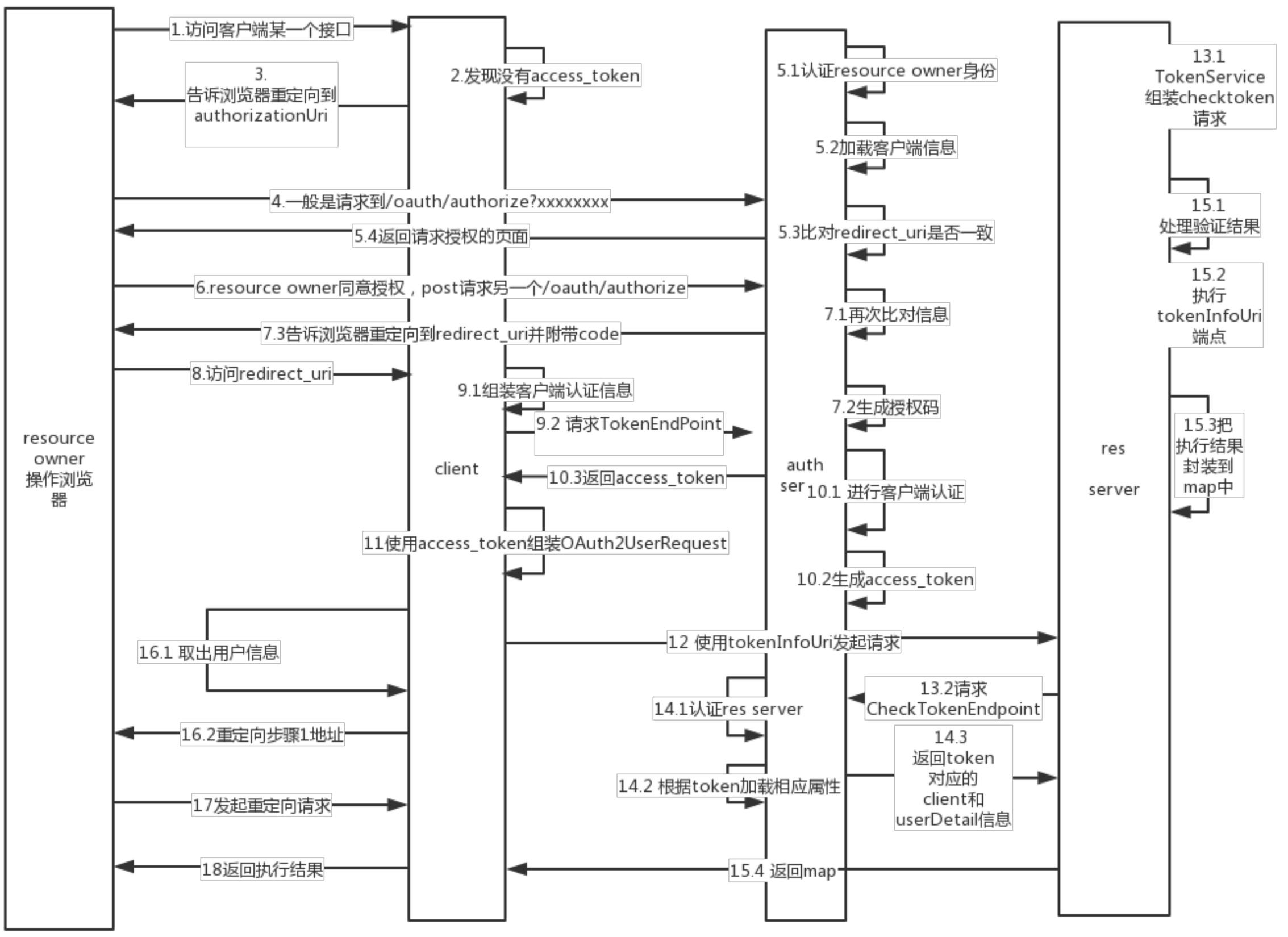

步骤1.用户(resource owner)操作浏览器访问客户端（client）中某一个接口，接口会发现请求中没有access_token

步骤2.因此client会给浏览器提供一个重定向地址，这个地址就是某个认证服务器（auth server)的认证地址。地址来自玉client中配置的spring.security.oauth2.client.provider.authorizationUri属性。并且，authorizationUri还会附带一个参数:spring.security.oauth2.client.registration.redirectUri。用于auth server在用户授权成功后重定向回客户端。

步骤3.为了防止恶意攻击client的redirect_uri接口。我们需要验证访问redirect_uri地址的请求的合法性。因此这里生成了一个state属性。也一并附带到redirect_uri后面发送给auth server.auth server会在步骤7中原样返回state以便我们验证。然后告诉浏览器进行重定向。地址就是步骤2所说的authorizationUri

步骤4.浏览器访问authorizationUri（默认就是/oauth/authorization)。这个地址的目的地就是auth server的AuthorizationEndPoint.

步骤5.1请求到达auth server的时候,如果用户(resource owner)没有认证，那么就会弹出登录框，要求用户认证身份

步骤5.2然后校验客户端(client）的信息(但不是认证client),auth server事先从AuthorizationServerConfigurerAdapter配置中或者通过clientservice,加载好auth server认可的client(一般使用的是inMemoryDetailClientService，如果使用jdbcDetailClientService可能表设计需要满足这个service。或者自定义clientservice)

步骤5.3 比对配置的redirect_uri和请求中附带的redirect_uri是否一致,一致才能让用户授权。

步骤5.4 返回给浏览器一个用户授权页面。

步骤6.这里假定用户在浏览器中同意给client授权访问他在res server上的个人信息。然后用户点击提交。浏览器发送post请求(另一个/oauth/authorize)到AuthorizationEndPoint.

步骤7.1 再次比对userdetail ，clientDetail，AuthorizationRequest是否一致或者有改变

步骤7.2 比对完成后，会修改对应authorizationRequest请求的相关属性，并生成授权码authorization_code,然后把授权码，认证请求(AuthorizationRequest)，认证用户(authentication)三方一 一关联。方便以后的场景进行快速验证等操作。

步骤7.3 给redirect_uri附带上authorization_code，以及client发送给我们的state。并通知浏览器重定向到redirect_uri.

步骤8. 浏览器进行重定向操作，地址是redirect_uri。此时回到了client中.

步骤9.1 当redirect_uri到达client之后，与步骤2不一样的是，因为配置的原因，client会识别到redirect_uri和普通的请求不一样。因此会使用OAuth2AuthorizationCodeAuthenticationProvider(spring security provider的知识还记得吗)来处理。然后，这个provider就开始组装OAuth2AuthorizationCodeGrantRequest请求对象。然后根据ClientRegistration.ClientAuthenticationMethod(basic或者post）来组装请求参数。最后附带上授权码。访问ClientRegistration.tokenUri.

步骤9.2 client发起到auth server的请求。一般client中配置的tokenUri就是auth server的TokenEndPoint地址：/oauth/token。

步骤10.1 auth server根据请求头中是否有authorization参数。来判定为client basic 认证或者是 client post认证。从而采用不同的客户端认证方法(一般我们都用原生的basic 或者post方式，也有其他的认证方式但是没有研究过)。basic认证使用的是BasicAuthenticationFilter。post认证使用的是ClientCredentialsTokenEndpointFilter

步骤10.2 客户端认证通过后，才能正式进入TokenEndPoint.这个端点会先封装一个tokenRequest对象。并最终生成access_token。而且access_token ，tokenRequest，已认证客户端。三个是一 一关联的。方便以后的场景中进行快速验证等操作。

步骤10.3 经过10.2之后，auth server会返回access_token给client。至此client就得到了access_token

步骤11 经过上一步之后，access_token会流转到Oauth2LoginAuthenticationProvider。这个provider会使用access_token和配置中的userInfoUri来组装UserRequest.准备给res server发起请求。

步骤12 经过步骤11之后，client向res server发起了一个请求用户信息的请求（UserInfoUri)。这个请求的期望返回结果是一个map。当这个请求返回了一个map之后.我们使用client配置文件中的provider.userNameAttribute作为map的key获取我们需要的用户信息。

步骤13.1 res server根据请求头中的header是否有Authorization=Bear + token。来判定是否为UserRequest.然后使用RemoteTokenService来处理UserRequest。RemoteTokenSerivce的主要工作是:取出access_token。与步骤10.1一样，auth server也需要对 res server进行认证。所以在res server中,我们取出配置文件的clientId和clientSecret（当然这里的clientid和clientSecret和client的不一样）。组装basic请求头(就像步骤10.1 需要对client进行认证一样)（或者也可以采用其他认证方式）。组装了请求头之后，在把access_token做成form表单。

步骤13.2 经过步骤13.1 res server向auth server发送验证access_token的请求,也就是配置文件中的security.oauth2.resource.tokenInfoUri，目的地就是CheckTokenEndPoint。

步骤14.1 和步骤10.1一样，进行res server认证。

步骤14.2 认证通过之后，正式进入CheckTokenEndPoint。然后根据access_token获取到10.2中一 一对应的三种属性。并组装相应结果

步骤14.3 向res server返回client和UserDetail信息。

步骤15.1 res server处理返回结果，判断access_token验证是否成功

步骤15.2 验证成功之后，才能执行UserInfoUri(一般是我们在res server中自定义的一个controller).

步骤15.3 请注意，UserInfoUri执行之后，controller需要返回一个map。因此请注意返回值类型。

步骤15.4 返回给client一个map结果

步骤16.1 client使用配置文件中配置的provider.usernameAttribute属性作为key.来取出15.4中返回的map.至此client便获取到了 res server的提供的关于 resource owner的个人信息

步骤16.2 告诉浏览器重定向到步骤1中请求的接口。

步骤17 浏览器发起重定向请求到client。

步骤18 client执行接口请求。完成业务处理，并返回请求结果给浏览器。至此完整的流程结束