|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 인증 방식 | Authorization Code Grant  (Web Server Flow) | Implicit Grant  (User-agent, Mobile) | Password Credentials Grant  (2-legged) | Client Credentials Grant  (confidential client) |
| 추가설명 | 인증과정시 client\_secet 이용  response\_type=code | 인증과정에서 client\_secret 전달하지 않음, response\_type=token | 클라이언트에 ID/PWD를 저장해놓고 직접 액세스 토큰 요청  (보안상 권장하지 않음) | 클라이언트가 confidential 일 경우만 사용  (보안상 권장하지 않음) |
| 개발자 등록 단계 | 개발자는 자신의 아이디로 로그인한 뒤 client app을 등록한다.  1. 입력하는 값  - 클라이언트 앱명, 클라이언트 설명, 클라이언트 url, 클라이언트 type(Web server,Native app, User-Agent),  퍼미션(R,W,RW,M 등 직접 정의하면됨), redirect uri(인증 후 callbak할 url)  2. 생성하는 값  - 입력한 값과 함께 client\_id와 client\_secret을 생성하여 저장하고 개발자에게 다음 정보를 제공한다.  \* client\_id, client\_secret, 나머지 등록한 내용 | | | |
| Auth 단계 | - GET 방식 파라미터  \* 필수 : client\_id, redirect\_uri, response\_type  \* 옵션  \*\* scope : 사용권한, 앱 등록시에 규정한 권한 범위를 벗어나서는 안됨. 이것을 사용자는 승인화면에서 확인하고 앱을 승인하게 됨.  \*\* state : cross-site request forgery(CSRF) 공격을 방지하기 위한 애플리케이션의 고유값. 랜덤 유니크 스트링이어야 함.  -->이값은 code를 받을 때, access\_token을 받을 때 다시 돌려받아 확인하면 됨.(클라이언트가 확인하는 용도)  - 처리 내용  \* 확인해야 할 값  \*\* response\_type이 code인지 확인(token이면 우측으로)  \*\* client\_id 으로 client\_app 정보 확인  \*\* (옵션) scope 확인(유효한 범위 내에 있는지 확인)  \* 확인 후 인증 화면 보여줌  \* 인증 후 redirect\_uri로 code와 state(옵션) 값 전달(GET)  -- app 등록시 redirect\_uri와 동일한지 반드시 확인해야 함  \* 이미 로그인 상태라면 승인 버튼만 보여줌  \* token을 생성하여 DB에 저장  \* redirect\_uri?code=XXXXXX&state=XXXXX 로 리다이렉트!  \* code : verifier의미, client\_id에 대해 사용자가 승인했음을 Provier가 확인하기 위한 용도 | | - GET 방식 파라미터  \* client\_id, redirect\_uri, response\_type  \* response\_type은 token으로 설정  \* option : scope, state  - 사용자의 승인 처리를 하고 나면 다음과 같이 응답한다.  [redirect\_uri]#access\_token=XXXXX&refresh\_token=XXXXX  &expires\_in=3600&token\_type=Bearer  위의 코드를 WebView 에서 access\_token 획득하고 WebView는 사라짐. | |
| - 에러 처리 : 전달 포맷은 "resource 접근" 참조  \* invalid\_request, unauthorized\_client,  unsupported\_response\_type, invalid\_scope, server\_error,  temporarily\_unavailable, | |  | |
| Access Token 획득 | - endpoint : /oauth2/access\_token  - 필수 파라미터  \* code : 이전 단계에서 획득한 code  \* redirect\_uri : 미리 등록ㄷ한 redirect\_uri  \* grant\_type : authorization\_code 로 부여함.(이방식에서는)  - Basic Authorization Header : client\_id, client\_secret을 이용하여 Base64 인코딩한 값을 헤더에 추가함.  🡺 Authorization : Basic XXXXXXXXXXXX 형식으로 (표준)  - client\_id&client\_secret 연결하여 인코딩  - 서버측 처리과정  \* client\_id, client\_secret 확인  \* code 값으로 DB 조회하여 정보 검토  \* grant\_type이 authorization\_code인지 검토  \* redirect\_uri가 일치하는지 여부 확인  \* 유효기간 확인 : token 생성된 시간과 expries\_in 값 비교  \* state : authorization시에 보낸 state인가 아닌가 확인(option)  - 응답결과  \* access\_token  \* token\_type : bearer  \* issued\_at : timestamp  \* signature : 클라이언트의 secret으로 서명된 값(HMAC-SHA256 이용, 클라이언트가 자신이 받은 accesstoken을 검증할 떄 사용) --> optional  \* expires\_in : (optional)  \* refresh\_token : (optional) 유효기간이 종료되면 재요청할 때 사용할 token  \* scope : 권한  - form urlencoded 또는 json 포맷으로 응답함.  예)  { "access\_token" : "XXXX", "token\_type" : "Bearer",  "expires\_in" : 3600, "refresh\_token" : "XXXX" } | |  | |
| Reource 접근 | \* 헤더에 다음 추가하여 브라우저 캐싱 방지할 것  Pragma: no-cache  Cache-Control :no-cache    - API 요청시 access\_token을 Authorization 헤더에 추가하여 요청함.  \* Authorization : Bearer XXXXXXXXX(access\_token)  \* 만일 form urlencoded 형태로 이 값을 전달하려면 반드시 POST로만 처리해야 함.  - 에러 처리  \* 에러형식은 정해진 바 없으나 요즘은 json으로 응답하는 것이 추세  \* 최근 표준은 헤더로 처리하는 것임  \*\* facebook의 예 : 400 status 코드와 함께  {  error : {  type : "OAuthException",  message : "invalid access token"  }  }  \*\* 최근 표준  WWW-Authenticate : Bearer realm="aaa.com",  Error="invalid\_token",  Error\_description="access token expired"  \* invalid\_token 오류이면서 client가 refresh\_token을 보유하고 있다면 refresh token을 이용해 access token 갱신을 시도함  --> refresh token 참조 | | 왼쪽과 동일함. | |
| Refresh token | Endpoint : /oauth2/token  - refresh\_token을 이용해 새로운 access\_token 획득 시도함.  - 전달 파라미터  \* grant\_type : refresh\_token  \* client\_id, client\_secret, refresh\_token 값 추가  \* 이중 client\_id와 client\_secret은 GET으로 요청시 반드시 헤더에 집어넣어야 함.  --> Authorization : Basic XXXXXXX  - 서버측에서 할일  \* client\_id 정보 확인 grant\_type 확인  \* refresh token을 이용해 expire 된 access token인지 확인 후 token들을 재생성하여 DB 업데이트하고 다시 응답  \*\* refresh token, access token 모두를 재생성할 것을 권장  - 응답 파라미터 형식  \* access\_token의 응답형식과 동일함. | | 왼쪽과 동일  Client\_secret을 제외한 나머지 값들을 POST로 요청하여 Token을 갱신함.  응답형식은 access\_token 형식과 동일함. | |
|  |  | |  | |
|  | < password credential >  - 한번에 사용자ID, pwd, client\_id, client\_secret,redirect\_uri를 모두 이용해 한번에 access\_token 획득  - Client과 신뢰할 수 있어서 사용자가 안심하고 사용자 ID, PWD를 저장할 수 있을 때 사용함.  GET /oauth2/token  Authorizatiion : Basic XXXXXXXXX  grant\_type=password&username=XXX&password=XXXXX  만일 client\_id와 client\_secret을 form urlencoded로 전달하려면 반드시 POST를 사용할 것 | | < client\_credential >  - confidential client(예: App 관리자용 클라이언트)는 client\_id, client\_secret만으로 인증하고 access\_token을 발급함.(user 정보 없음)  POST /oauth2/token  grant\_type=client\_credentials&client\_id=XXXX&client\_secret=XXXX | |

2. Bearer 대신에 Mac Token 을 이용하는 경우 아래 URL 참조

--> OAuth 1.0의 서명 생성과정과 유사함. (BaseString 생성하여 MacKey로 서명함)

Mac Key Identifier는 access Token임

Base String 생성 방법에서 각 필드는 \n으로 구분함.

<http://tools.ietf.org/html/draft-hammer-oauth-v2-mac-token-05>

3. JWT(JSON Web Token) 이용시는 다음 참조

<http://tools.ietf.org/html/draft-ietf-oauth-json-web-token-10>

3. salesforce의 refresh token flow

<https://help.salesforce.com/help/doc/en/remoteaccess_oauth_refresh_token_flow.htm>

3. salesforce의 web server flow

<http://help.salesforce.com/HTViewHelpDoc?id=remoteaccess_oauth_web_server_flow.htm&language=en_US>

3. salesforce의 user agent flow

<http://help.salesforce.com/apex/HTViewHelpDoc?id=remoteaccess_oauth_user_agent_flow.htm&language=en_US>

4. HTTPS는 query string도 암호화함.

4. URL Fragment는 암호화 되는가? 답 된다.

<http://blog.httpwatch.com/2011/03/01/6-things-you-should-know-about-fragment-urls/>

location : Location:http://localhost:8000/tistory2/callback.html#access\_token=XXXXXX 시에

브라우저는 이 정보를 받아 http://localhost:8000/tistory2/callback.html 로만 재요청함. 이동되는 문서 내용이 수신되면 #을 부여하여 문서위치로 이동시킴.

따라서 302 로 Move될 때는 Http Header에 안전하게 암호화된 상태이므로 이것으로 전송 끝.