OAuth

최근의 인터넷 서비스는 그 자체가 SaaS(Software as a Service)의 형태이다다. 서비스 중에서 사용자가 일부 필요한 것만 사용할 수 있게 한다는 것이다.

Facebook 이나 트위터가 세상에 널리 퍼지게 된 이유 중에 하나가 외부 서비스에서도 Facebook 이나 트위터의 일부 기능을 사용할 수 있게 한 것이다.

외부 서비스와 연동되는 Facebook 이나 트위터의 기능을 이용하기 위해 사용자가 반드시 Facebook 이나 트위터에 로그인해야 하는 것이 아니라, 별도의 인증 절차를 거치면 다른 서비스에서 Facebook 과 트위터의 기능을 이용할 수 있게 되는 것이다.

이런 방식은 Facebook 이나 트위터 같은 서비스 제공자뿐만 아니라 사용자와 여러 인터넷 서비스 업체 모두에 이익이 되는 생태계를 구축하는데 기여하게 될 것이며, 이 방식에서 사용하는 인증 절차가 OAuth 이다.

OAuth 의 탄생과 사용

OAuth 는 인증을 위한 오픈 스탠더드 프로토콜로, 사용자가 Facebook 이나 트위터 같은 인터넷 서비스의 기능을 다른 애플리케이션(데스크톱, 웹, 모바일 등)에서도 사용할 수 있게 한 것이다.

OAuth 의 탄생 이전에도 다른 애플리케이션에 사용자의 아이디와 암호가 노출되지 않도록 하면서 API 접근 위임(API Access Delegation)이 가능한 여러 인증 방법이 있었다. Google 과 Yahoo!, AOL, Amazon 등에서는 각각의 인증 방식을 제작하여 사용했다.
OAuth 1.0 이 나온 때는 2007 년이며, 이후 보안 문제를 해결한 수정 버전인 OAuth 1.0 revision A 가 2008 년에 나왔다.

OAuth 의 시작은 2006 년에 트위터의 개발자와 소셜 북마크 서비스인 Gnolia 의 개발자가 만나 인증 방식을 논의한 때부터였다. 두 회사의 개발자들은 그때까지 API 접근 위임에 대한 표준안이 없다는 것을 알았다. 그래서 2007 년 4 월 인터넷에 OAuth 논의체를 만든 뒤 OAuth 드래프트 제안서를 만들어 공유했다. 그 뒤에 이 활동을 지지하는 사람이 생기게 되었고, 2008 년 IETF 모임(73 회, 미네소타에서 개최)에서 OAuth 가 IETF 표준안으로 관리되야 하는 가에 대한 논의가 있었다. 그리고 2010 년에 OAuth 1.0 프로토콜 표준안이 RFC5849 로 발표되었다.

2007 년에 나온 OAuth 1.0 은 비공식 논의체에 의해 최초로 만들어진 것이고, 2010 년 IETF OAuth 워킹그룹에 의해 이 프로토콜이 IETF 표준 프로토콜로 발표된 것이다.
현재 나와 있는 OAuth 2.0 은 드래프트 단계에 있는 것으로, IETF OAuth 워킹그룹이 주축이 되어 만든 것이다. OAuth 2.0 은 OAuth 1.0 과 호환되지 않지만, 인증 절차가 간략하다는 장점이 있다.

그래서 아직 최종안이 나오지 않았음에도 여러 인터넷 서비스에서 OAuth 2.0 을 사용하고 있다. 다음 표는 대표적인 인터넷 서비스 기업에서 사용하는 OAuth 의 버전을 정리한 것이다.

표 1 인터넷 서비스 기업이 사용하는 OAuth 버전

인터넷 서비스 기업	OAuth 버전
Facebook	2.0
Foursquare	2.0
Google	2.0
Microsoft (Hotmail, Messenger, Xbox)	2.0
LinkedIn	2.0
Daum(티스토리	2.0
NHN (오픈 API)	1.0a
Daum(요즘, 오픈 API)	1.0a
MySpace	1.0a
Netflix	1.0a
트위터	1.0a
Vimeo	1.0a
Yahoo!	1.0a

OAuth 와 로그인

OAuth 와 로그인은 반드시 분리해서 이해해야 한다. 일상 생활을 예로 들어 OAuth 와 로그인의 차이를 설명해 보겠다.

사원증을 이용해 출입할 수 있는 회사를 생각해 보자. 그런데 외부 손님이 그 회사에 방문할 일이 있다. 회사 사원이 건물에 출입하는 것이 로그인이라면 OAuth 는 방문증을 수령한 후 회사에 출입하는 것에 비유할 수 있다.

다음과 같은 절차를 생각해 보자.

- 나방문씨(외부 손님)가 안내 데스크에서 업무적인 목적으로 김목적씨(회사 사원)를 만나러 왔다고 말한다.
- 안내 데스크에서는 김목적씨에게 나방문씨가 방문했다고 연락한다.
- 김목적씨가 안내 데스크로 찾아와 나방문씨의 신원을 확인해 준다.
- 김목적씨는 업무 목적과 인적 사항을 안내 데스크에서 기록한다.
- 안내 데스크에서 나방문 씨에게 방문증을 발급해 준다.
- 김목적씨와 나방문씨는 정해진 장소로 이동해 업무를 진행한다.

위 과정은 방문증 발급과 사용에 빗대어 OAuth 발급 과정과 권한을 이해할 수 있도록 한 것이다. 방문증이란 사전에 정해진 곳만 다닐 수 있도록 하는 것이니, '방문증'을 가진 사람이 출입할 수 있는 곳과 '사원증'을 가진 사람이 출입할 수 있는 곳은 다르다. 역시 직접 서비스에 로그인한 사용자와 OAuth 를 이용해 권한을 인증받은 사용자는 할 수 있는 일이 다르다.

OAuth 에서 'Auth'는 'Authentication'(인증)뿐만 아니라 'Authorization'(허가) 또한 포함하고 있는 것이다. 그렇기 때문에 OAuth 인증을 진행할 때 해당 서비스 제공자는 '제 3 자가 어떤 정보나 서비스에 사용자의 권한으로 접근하려 하는데 허용하겠느냐'라는 안내 메시지를 보여 주는 것이다. 다음의 화면 두 개는 각각 네이버와 미투데이에 접근 권한 요청이 있음을 알려 주는 화면이다. 미투데이의 경우에는 OAuth 인증 방식을 사용하지 않지만 인증 절차에 대한 예시 정도로 생각해 주기 바란다.

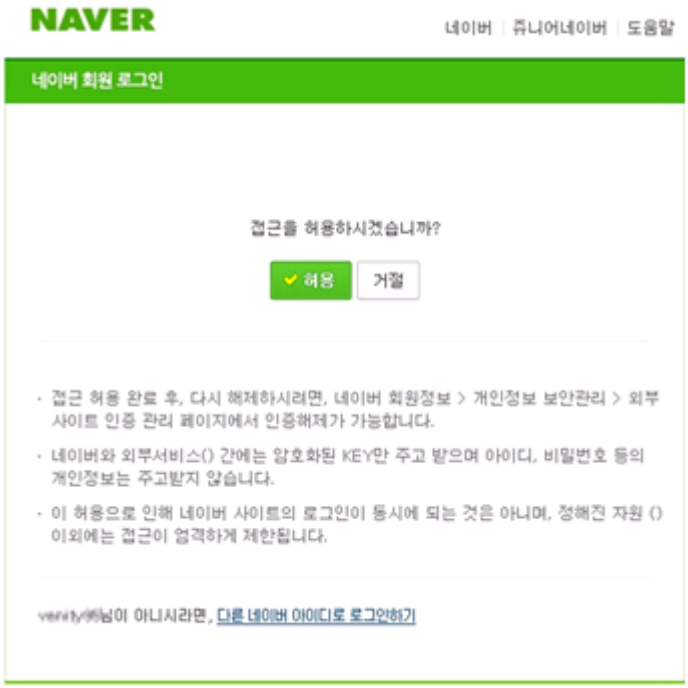


그림 1 네이버 OAuth 접근 권한 요청 화면

그림 2 미투데이 접근 권한 요청 화면

OpenID 와 OAuth

OpenID 도 인증을 위한 표준 프로토콜이고 HTTP 를 사용한다는 점에서는 OAuth 와 같다. 그러나 OpenID 와 OAuth 의 목적은 다르다.

OpenID 의 주요 목적은 인증(Authentication)이지만, OAuth 의 주요 목적은 허가(Authorization)이다. 즉, OpenID 를 사용한다는 것은 본질적으로 로그인하는 행동과 같다. OpenID 는 OpenID Provider 에서 사용자의 인증 과정을 처리한다. Open ID 를 사용하는 여러 서비스(Relying Party)는 OpenID Provider 에게 인증을 위임하는 것이다.
물론 OAuth 에서도 인증 과정이 있다.

가령 Facebook 의 OAuth 를 이용한다면 Facebook 의 사용자인지 인증하는 절차를 Facebook(Service Provider) 처리한다. 하지만 OAuth 의 근본 목적은 해당 사용자의 담벼락(wall)에 글을 쓸 수 있는 API 를 호출할 수 있는 권한이나, 친구 목록을 가져오는 API 를 호출할 수 있는 권한이 있는 사용자인지 확인하는 것이다.

OAuth 를 사용자 인증을 위한 방법으로 쓸 수 있지만, OpenID 와 OAuth 의 근본 목적은 다르다는 것을 알아야 한다.

OAuth Dance, OAuth 1.0 인증 과정

OAuth 를 이용하여 사용자를 인증을 하는 과정을 OAuth Dance 라고 한다. 두 명이 춤을 추듯 정확하게 정보를 주고받는 과정을 재미있게 명명한 것이다.
OAuth 를 이해하려면 몇 가지 용어를 먼저 알아 두어야 한다. 다음 표에 OAuth 의 대표 용어를 정리해 보았다.

표 2 OAuth 1.0 대표 용어

용어	설명
User	Service Provider 에 계정을 가지고 있으면서, Consumer 를 이용하려는 사용자
Service Provider	OAuth 를 사용하는 Open API 를 제공하는 서비스
Consumer	OAuth 인증을 사용해 Service Provider 의 기능을 사용하려는 애플리케이션이나 웹 서비스
Request Token	Consumer 가 Service Provider 에게 접근 권한을 인증받기 위해 사용하는 값. 인증이 완료된 후에는 Access Token 으로 교환한다.
Access Token	인증 후 Consumer 가 Service Provider 의 자원에 접근하기 위한 키를 포함한 값

다음은 OAuth 인증 과정을 그림으로 표현한 것이다.

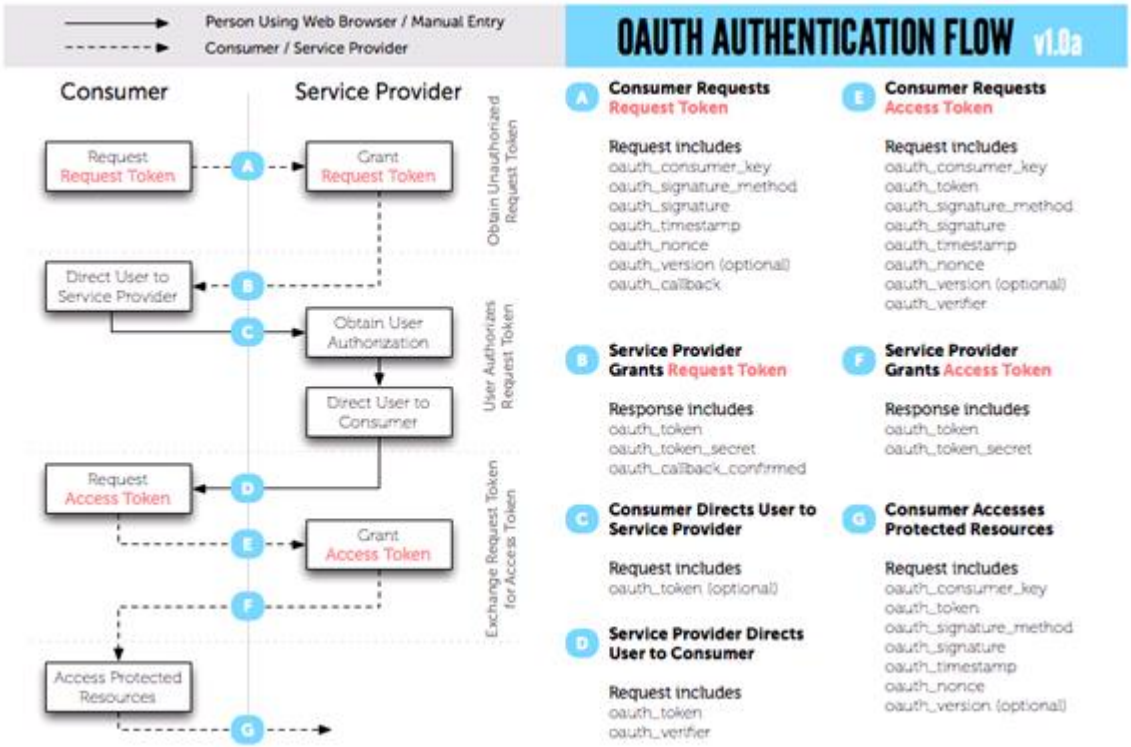


그림 3 OAuth 1.0a 인증 과정(원본 출처: http://oauth.net/core/diagram.png)

OAuth 인증 과정을 앞에서 설명한 회사 방문 과정과 연결하면 다음 표와 같다.

표 3 회사 방문 과정과 OAuth 인증 과정

회사 방문 과정	OAuth 인증 과정
1. 나방문씨가 안내 데스크에서 업무적인 목적으로	Request Token 의 요청과 발급

	김목적씨를 만나러 왔다고 말한다.	
2.	안내 데스크에서는 김목적씨에게 나방문씨가 방문했다고 연락한다.	사용자 인증 페이지 호출
3.	김목적씨가 안내 데스크로 찾아와 나방문씨의 신원을 확인해 준다.	사용자 로그인 완료
4.	김목적씨는 업무 목적과 인적 사항을 안내 데스크에서 기록한다.	사용자의 권한 요청 및 수락
5.	안내 데스크에서 나방문 씨에게 방문증을 발급해 준다.	Access Token 발급
6.	김목적씨와 나방문씨는 정해진 장소로 이동해 업무를 진행한다.	Access Token 을 이용해 서비스 정보 요청

위의 표에 따르면, Access Token 은 방문증이라고 이해할 수 있다. 이 방문증으로 사전에 허락된 공간에 출입할 수 있다. 마찬가지로 Access Token 을 가지고 있는 Consumer 는 사전에 호출이 허락된 Service Provider 의 오픈 API 를 호출할 수 있는 것이다.

Request Token
OAuth 에서 Consumer 가 Request Token 발급을 요청하고 Service Provider 가 Request Token 을 발급하는 과정은 "저 나방문입니다. 김목적씨를 만날 수 있을까요?"라고 말하는 절차에 비유할 수 있다.

Request Token 을 요청하는 Request 전문을 살펴보자. 다음은 네이버의 OAuth API 로 Request Token 을 요청하는 예이다.

```
GET
/naver.oauth?mode=req_req_token&oauth_callback=http://example.com/OAuthRequestToken.do&oauth_consumer_key=WEhGuJZWUasHg&oauth_nonce=zSs4RFI7lakpADpSsv&oauth_signature=wz9+ZO5OLUnTors7HlyaKat1Mo0=&oauth_signature_method=HMAC-SHA1&oauth_timestamp=1330442419&oauth_version=1.0 HTTP/1.1


Accept-Encoding: gzip, deflate
```

```
Connection: Keep-Alive

Host: nid.naver.com
```

보기 쉽도록 위의 내용을 아래와 같이 매개변수를 기준으로 정리했다.

```
GET http://nid.naver.com/naver.oauth?mode=req_req_token&

oauth_callback=http://example.com/OAuthRequestToken.do&

oauth_consumer_key=WEhGuJZWUasHg&

oauth_nonce=zSs4RFI7lakpADpSsv&

oauth_signature=wz9+ZO5OLUnTors7HlyaKat1Mo0=&

oauth_signature_method=HMAC-SHA1&

oauth_timestamp=1330442419&

oauth_version=1.0 HTTP/1.1
```

oauth_sinature 는 다음과 같이 네 단계를 거쳐 만든다.

1. 요청 매개변수를 모두 모은다.
oauth_signature 를 제외하고 'oauth_'로 시작하는 OAuth 관련 매개변수를 모은다. POST body 에서 매개변수를 사용하고 있다면 이 매개변수도 모아야 한다.

2. 매개변수를 정규화(Normalize)한다.
모든 매개변수를 사전순으로 정렬하고 각각의 키(key)와 값(value)에 URL 인코딩(rfc3986)을 적용한다. URL 인코딩을 실시한 결과를 = 형태로 나열하고 각 쌍 사이에는 &을 넣는다. 이렇게 나온 결과 전체에 또 URL 인코딩을 적용한다.

3. Signature Base String 을 만든다.
HTTP method 명(GET 또는 POST), Consumer 가 호출한 HTTP URL 주소(매개변수 제외), 정규화한 매개변수를 '&'를 사용해 결합한다. 즉 '[GET|POST] + & + [URL 문자열로 매개변수는 제외] + & + [정규화한 매개변수]' 형태가 된다. 이 예제에서는 'http://nid.naver.com/naver.oauth' 을 URL 로 사용하고, 이 URL 에 URL 인코딩을 적용한 값을 사용했다.

4. 키 생성
3 번 과정까지 거쳐 생성한 문자열을 암호화한다. 암호화할 때 Consumer Secret Key 를 사용한다. Consumer Secret Key 는 Consumer 가 Service Provider 에 사용 등록을 할 때 발급받은 값이다. HMAC-SHA1 등의 암호화 방법을 이용하여 최종적인 oauth_signature 를 생성한다.
사용자 인증 페이지의 호출
OAuth 에서 사용자 인증 페이지를 호출하는 단계는 '안내데스크에서 김목적씨에게 방문한 손님이 있으니 안내 데스크로와서 확인을 요청하는 것'에 비유할 수 있다. Request Token 을 요청하면, Service Provider 는 Consumer 에 Request Token 으로 사용할 oauth_token 과 oauth_token_secret 을 전달한다. Access Token 을 요청할 때는 Request Token 의 요청에 대한 응답 값으로 받은 oauth_token_secret 을 사용한다. Consumer 가 웹 애플리케이션이라면 HTTP 세션이나 쿠키 또는 DBMS 등에 oauth_token_secret 를 저장해 놓아야 한다.
oauth_token 을 이용해 Service Provider 가 정해 놓은 사용자 인증 페이지를 User 에게 보여 주도록 한다. 네이버의 경우 OAuth 용 사용자 인증 페이지의 주소는 다음과 같다.

```
https://nid.naver.com/naver.oauth?mode=auth_req_token
```

여기에 Request Token 을 요청해서 반환받은 oauth_token 을 매개 변수로 전달하면 된다. 예를 들면 다음과 같은 URL 이 만들어 지게 되는 것이다. 이 URL 은 사용자 인증 화면을 가리킨다.

```
https://nid.naver.com/naver.oauth?mode=auth_req_token&oauth_token=wpsCb0Mcpf9dDDC2
```

로그인 화면을 호출하는 단계까지가 '안내 데스크에서 김목적씨에게 전화하는 단계'라 보면 되겠다. 이제 김목적씨가 안내 데스크로 와서 나방문씨를 확인해야 한다. 정말 나방문씨가 맞는지 아닌지 확인하는 과정이 필요할 것이다. 이 과정이 OAuth 에서는 Service Provider 에서 User 를 인증하는 과정이라고 볼 수 있다.

인증이 완료되면 앞에서 말한 바와 같이 어떤 권한을 요청하는 단계에 이르게 된다. "업무 약속이 있어 오셨으니 나방문씨가 출입할 수 있게 해 주세요"와 같은 것 말이다.
인증을 마치면 Consumer 가 oauth_callback 에 지정한 URL 로 리다이렉트한다. 이때 Service Provider 는 새로운 oauth_token 과 oauth_verifier 를 Consumer 에 전달한다. 이 값들은 Access Token 을 요청할 때 사용한다. Access Token 요청하기
OAuth 에서의 AccessToken 은 나방문 씨에게 지급할 방문증과 같다.

Access Token 을 요청하는 방법은 Request Token 을 요청하는 방법과 거의 같다. 다만, 사용하는 매개변수의 종류가 약간 다르고 oauth_signature 를 생성할 때 사용하는 키가 다르다. Access Token 을 요청할 때에는 매개변수 oauth_callback 는 없고, oauth_token 와 oauth_verifer 가 있다.

Request Token 발급을 요청할 때에는 Consumer Secret Key 를 사용해 oauth_token_secret 를 생성했다. Access Token 발급을 요청할 때에는 Consumer Secret Key 에 oauth_token_secret 을 결합한 값(Consumer Secret Key + & + oauth_token_secret)을 사용해 oauth_token_secret 를 생성한다. 암호화 키를 변경하여 보안을 더 강화하는 것이다.

Access Token 사용하기
드디어 방문증이 발급됐다. 이제 출입문을 통과하는 일만 남았다. 방문증을 가지고 출입문을 통과한다는 것은 User 의 권한으로 Service Provider 의 기능을 사용하는 것과 비슷하다. 다시 말해, 권한이 필요한 오픈 API 를 호출할 수 있게 되는 것이다.
가령 네이버 카페에서 게시판 목록을 가져온다고 한다면 호출해야 하는 URL 은 다음과 같다.

```
http://openapi.naver.com/cafe/getMenuList.xml
```

특정 User 의 권한을 가지고 카페 게시판 목록 반환 URL 을 요청해야 해당 User 가 가입한 카페의 게시판 목록을 반환받을 수 있을 것이다. 이 URL 을 호출할 때는 OAuth 매개변수를 함께 전달해야 한다.

다음은 Access Token 을 사용해 오픈 API 를 요청하는 예이다. HTTP 헤더에 Authorization 필드를 두었고, Authorization 필드의 값 부분에 OAuth 매개변수 적는다. Access Token 을 사용할 때는 GET 이나 POST 가 아닌 HEAD 방식을 사용한다.

```
POST /cafe/getMenuList.xml HTTP/1.1

Authorization: OAuth oauth_consumer_key="dpf43f3p2l4k3l03",oauth_token="nSDFh734d00sl2jdk"

,oauth_signature_method="HMACSHA1",oauth_timestamp="1379123202",oauth_nonce="chapoH",
oauth_signature="MdpQcU8iPSUjWoN%2FUDMsK2sui9I%3D"

Accept-Encoding: gzip, deflate

Connection: Keep-Alive

Host: http://openapi.naver.com
```

보기 쉽도록 Authorization 필드를 아래와 같이 매개변수를 기준으로 정리했다.

```
Authorization: OAuth oauth_consumer_key="dpf43f3p2l4k3l03",

oauth_token="nSDFh734d00sl2jdk",
```

```
oauth_signature_method="HMACSHA1",

oauth_timestamp="1379123202",

oauth_nonce="csrrkjsd0OUhja",

oauth_signature="MdpQcU8iPGGhytrSoN%2FUDMsK2sui9I%3D"
```

OAuth 2.0

OAuth 1.0 은 웹 애플리케이션이 아닌 애플리케이션에서는 사용하기 곤란하다는 단점이 있다. 또한 절차가 복잡하여 OAuth 구현 라이브러리를 제작하기 어렵고, 이런저런 복잡한 절차 때문에 Service Provider 에게도 연산 부담이 발생한다.

OAuth 2.0 은 이러한 단점을 개선한 것이다. OAuth 1.0 과 호환성이 없고, 아직 최종안이 발표된 것은 아니지만 여러 인터넷 서비스 기업에서 OAuth 2.0 을 사용하고 있다.
OAuth 2.0 의 특징은 다음과 같다.

- 웹 애플리케이션이 아닌 애플리케이션 지원 강화
- 암호화가 필요 없음 HTTPS 를 사용하고 HMAC 을 사용하지 않는다.
- Siganature 단순화 정렬과 URL 인코딩이 필요 없다.
- Access Token 갱신 OAuth 1.0 에서 Access Token 을 받으면 Access Token 을 계속 사용할 수 있었다. 트위터의 경우에는 Access Token 을 만료시키지 않는다. OAuth 2.0 에서는 보안 강화를 위해 Access Token 의 Life-time 을 지정할 수 있도록 했다.

이외에도 OAuth 2.0 에서 사용하는 용어 체계는 OAuth 1.0 과 완전히 다르다. 같은 목적의 다른 프로토콜이라고 이해는 것이 좋다. 하지만 아직 최종안이 나오지 않았기 때문에, 현재로서는 OAuth 2.0 의 특징만 파악하는 것으로도 충분할 듯 하다.

OAuth 로 인한 인터넷 서비스의 변화

OAuth 는 요즘의 인터넷 생태계의 주요 요소로 자리매김하고 있다.

신생 업체들은 고유의 인증 방식을 사용하기 보다는 Facebook 이나 트위터의 인증을 이용하고 있다. 개발 비용과 운영 비용을 줄이는 효과도 있지만 Facebook 이나 트위터를 통해 서비스를 홍보하는 효과를 만들 수도 있다. 그리고 Service Provider 입장에서는 핵심 기능을 더욱 공고히 하는 효과를 얻고 있다.
표준 인증 방법 하나가 인터넷 업계에 변화를 주고 있는 것이다.

== 트위터의 데이터 수집==========

Twitter 데이터 수집

우리는 2. Twitter API 사용하기에서에서 트위터 App 을 설정하였다. 이제 해당 앱 ID 와 비밀키를 가지고 파이썬을 이용하여 데이터를 가지고 오는 방법에 대하여 설명하고자 한다. 페이스북의 경우에는 앱 아이디와 토큰을 요청 URL 에 파라미터로 전송하여 간단하게 데이터를 가지고 올 수 있는 기능을 제공한다. 그러나 트위터의 경우에는 OAuth1.0a 의 기본에 충실하게 억세스 토큰을 가지고 와야 실제 원하는 데이터를 조회 할 수 있다.

1 OAuth 란?

사용자는 특정 서비스를 사용하기 위하여 아이디와 비밀번호라는 고전적인 방법을 이용하였다. 이를 위하여 서비스 제공자는 사용자의 아이와 비밀번호를 관리하여야 하는데 서버의 공격등으로 인해 아이디와 비밀번호가 노출되는 사고가 빈번히 발생하고 이를 위하여 데이터베이스에 비밀번호를 암호화 하는 등 다양한 형태의 보안 기술이 발생하였다.

OAuth 의 시작은 2006 년에 트위터와 소셜 북마크 서비스인 Gnolia 사의 개발자들은 접속한 사용자들이 서비스를 사용하기 위하여 사용자를 인증(Authentication)하고 특정한 서비스만 사용하기 위한 권한(Authorization)을 주기 위한 적당한 인증 알고리즘이 없다고 판단하여 개발에 착수, 2007 년 10 월에 OAuth1.0 을 발표하였다. 그 이후 세션 고정 공격(Session Fixation Attack: 세션 하이재킹(Hijacking) 기법중의 하나로 유요한 유저 세션을 탈취하여 인증을 회피하는 공격 방법)에 취약함이 발견되어 이를 개선하고 "The OAuth 1.0 Protocol"이라는 이름으로 2010 년에 표준안이 RFC5849 로 IETF 에 의해 체택 되었다. 기존의 1.0 에서 보안된 개념으로 실제 1.0 이라고 제정되었지만 많은 사람들은 OAuth 1.0a 라고 부른다. 현재 OAuth 는 처음 설계를 한 트위터를 필두로, 구글, 페이스북, 마이크로 소프트 등의 외국 거대 서비스 회사 및 국내의 네이버, 다음 카카오 등에서 사용하고 있다.

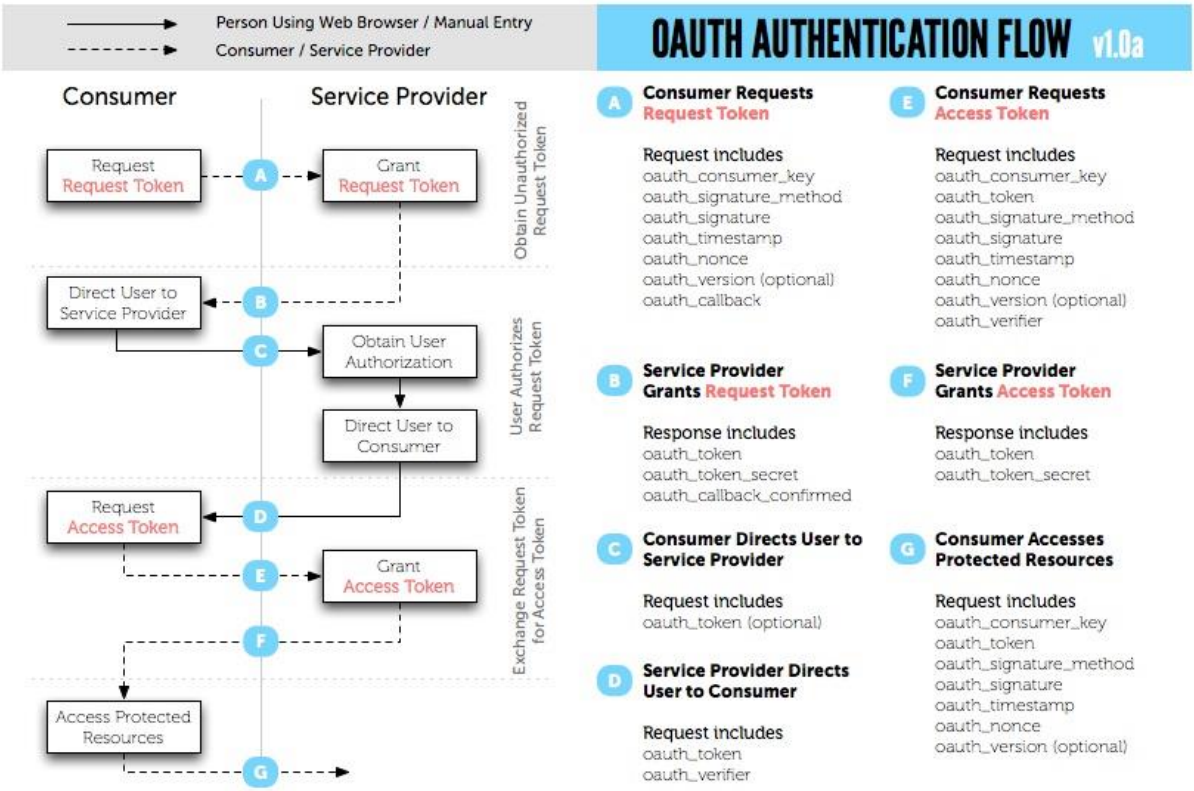
OAuth 의 장점은 사용자의 아이디와 비밀번호를 이용하여 인증을 하는 것이 아니고, 특정 서비스를 사용하고자 할 때 임시 사용 티켓을 발급하여, 서비스가 만료되거나 특정 기간이

종료하면 티켓을 사용할 수 없는 구조로 되어있다.

1.1 OAuth 1.0a 인증 과정

OAuth 인증을 위하여 사용하는 주체를 표현하는 용어는 다음과 같다

소비자 Consumer	OAuth 를 사용해 서비스 제공자의 기능을 사용하려는 프로그램 : 우리가 제작하는 프로그램의 입장
서비스 제공자 Service Provider	OAuth 를 사용하는 Open API 제공자 : 트위터, 페이스 북 등
요청 토큰 Request Token	OA 소비자(Consumer)가 서비스제공자(Service Provider)에게 접근 권한을 받기 위해 사용하는 값, 인증이 완료되면 접근 토큰(Access Token)으로 변경된다
접근 토큰 Access Token	인증 후 소비자(Consumer)가 서비스제공자(Service Provider)의 자원에 접근하기 위한 키를 포함한 값
사용자 User	서비스제공자(Service Provider)에 계정을 가지고 있으면서, 소비자(Consumer)를 이용하려는 사용자



[그림 1] OAuth 1.0a 인증 과정 (http://oauth.net/core/diagram.png)

1) Request Token 요청 및 발급

- Consumer Key 와 요청 시간(timestamp), 요청자료를 암호화 한 방식(HMAC-SHA1 등), 버전(1.0) 정보, 악의적인 정보 요청을 방지하기 위한 임의 문자열과 서명값을 만든후 해당 서버로 요청한다.
- 서명값(oauth_signature) 앞에서 설명한 각 변수값과 HTTP 요청 방식(POST 또는 GET)을 지정한 암호화 방식으로 암호화하여 만들어 낸다.
- 서비스 제공자에게 Request Token 을 요청하면, 서비스 제공자는 'oauth_token'과 'oauth_token_secret'값을 회신한다.

2) 사용자 인증 페이지 호출 및 수락

- 수신한 'oauth_token'을 가지고 서비스 제공자가 지정한 인증페이지로 인증 수락을 요청한다. 인증 페이지는 각 서비스제공자가 제공한다.

3) Access Token 요청 및 발급

- 'consumer_key'와 'oauth_token' 및 기타 변수들을 이용하여 'oauth_signature' 를 생성한 후 요청하면 서비스 제공자는 'oauth_token'과 'oauth_token_secret' 및 부가

정보를 회신한다.

4) 해당 API 접근

- 수신한 'oauth_token'을 이용하여 서비스제공자의 API 를 사용한다.

2 트윗(Tweet) 가지고 오기

파이썬에서 트위터를 사용하기 위해서는 다양한 모듈이 공개되고, 업그레이도 되고 있다. 그중에서 가장 보편적으로 사용하는 것은 "Tweepy" 모듈이다. 그러나 본 책에서는 최대한 모듈의 종속성을 배제하고 기본적인 접근 방법을 인지하기 위하여 최소한의 모듈만을 사용하는 것을 원칙으로 한다.

앞 절에서 언급하였듯이 트위터는 OAuth 1.0a 표준을 사용하여 응용 프로그램의 인증을 하고, 서비스 API 를 사용할 수 있는 키값을 제공하여 준다. 이를 위하여 OAuth 를 이해하는 것이 이전 절의 목적이었다면, 이번절에서는 OAuth 를 이용하여 인증하는 방법에 대하여 알아 볼 것이다. OAuth 는 암호화 알고리즘 및 시그네처(signature)의 구성을 위하여 많은 코딩이 필요하므로 현재 OAuth 인증을 위하여 개발되고 업그레이드 되고 있는 파이썬 "OAuth2" 모듈을 import 하여 사용하고, 각 인증 과정은 3 절에서 살펴 보도록 하겠다.

특정 트위터의 트윗을 가져오기 위하여 다음의 코드를 작성한다.

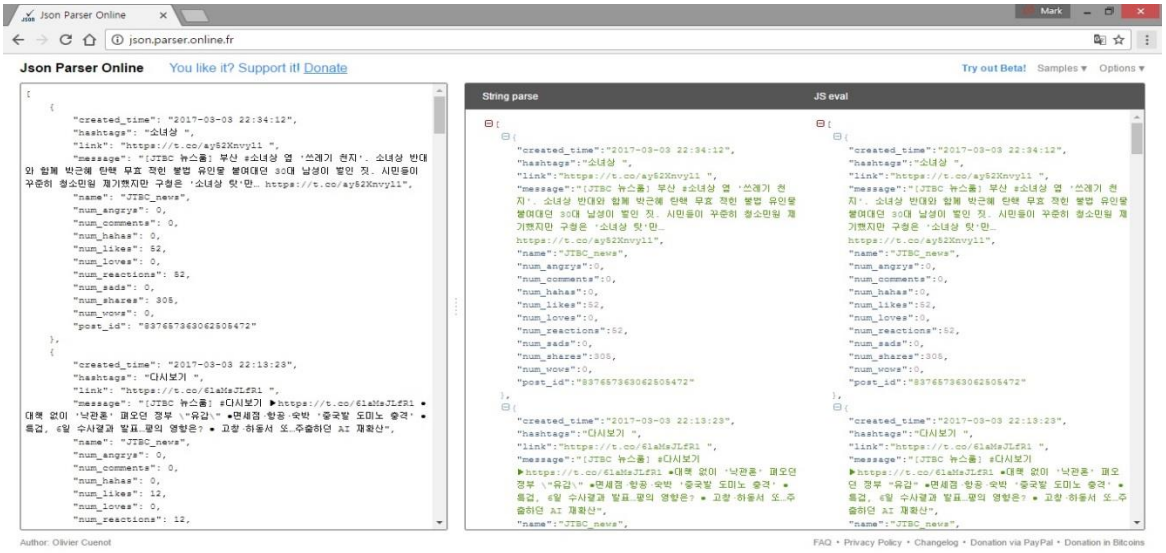
```
import oauth2
import json
import datetime
import time
from config import *

#[CODE 1]

def oauth2_request(consumer_key, consumer_secret, access_token, access_secret):
    try:
        consumer = oauth2.Consumer(key=consumer_key, secret=consumer_secret)
        token = oauth2.Token(key=access_token, secret=access_secret)
        client = oauth2.Client(consumer, token)
        return client
    except Exception as e:
        print(e)
        return None
```

```
#[CODE 2]

def get_user_timeline(client, screen_name, count=50, include_rts='False'):
    base = "https://api.twitter.com/1.1"
    node = "/statuses/user_timeline.json"
    fields = "?screen_name=%s&count=%s&include_rts=%s" % (screen_name, count, include_rts)
    #fields = "?screen_name=%s" % (screen_name)

    url = base + node + fields

    response, data = client.request(url)

    try:
        if response['status'] == '200':
            return json.loads(data.decode('utf-8'))
    except Exception as e:
        print(e)
        return None

#[CODE 3]

def getTwitterTwit(tweet, jsonResult):

    tweet_id = tweet['id_str']
    tweet_message = '' if 'text' not in tweet.keys() else tweet['text']

    screen_name = '' if 'user' not in tweet.keys() else tweet['user']['screen_name']

    tweet_link = ''
    if tweet['entities']['urls']: #list

        for i, val in enumerate(tweet['entities']['urls']):
            tweet_link = tweet_link + tweet['entities']['urls'][i]['url'] + ' '
    else:
        tweet_link = ''
```

```
        hashtags = ''
        if tweet['entities']['hashtags']: #list

            for i, val in enumerate(tweet['entities']['hashtags']):
                hashtags = hashtags + tweet['entities']['hashtags'][i]['text'] + ' '
        else:
            hashtags = ''

        if 'created_at' in tweet.keys():
            # Twitter used UTC Format. EST = UTC + 9(Korean Time) Format ex: Fri Feb 10 03:57:27
+0000 2017

            tweet_published = datetime.datetime.strptime(tweet['created_at'],'%a %b %d %H:%M:%S
+0000 %Y')
            tweet_published = tweet_published + datetime.timedelta(hours=+9)
            tweet_published = tweet_published.strftime('%Y-%m-%d %H:%M:%S')
        else:
            tweet_published = ''

        num_favorite_count = 0 if 'favorite_count' not in tweet.keys() else tweet['favorite_count']
        num_comments = 0
        num_shares = 0 if 'retweet_count' not in tweet.keys() else tweet['retweet_count']
        num_likes = num_favorite_count
        num_loves = num_wows = num_hahas = num_sads = num_angrys = 0

        jsonResult.append({'post_id':tweet_id, 'message':tweet_message,
                           'name':screen_name, 'link':tweet_link,
                           'created_time':tweet_published, 'num_reactions':num_favorite_count,
                           'num_comments':num_comments, 'num_shares':num_shares,
                           'num_likes':num_likes, 'num_loves':num_loves,
                           'num_wows':num_wows, 'num_hahas':num_hahas,
                           'num_sads':num_sads, 'num_angrys':num_angrys, 'hashtags': hashtags})

def main():
    screen_name = "jtbc_news"
```

```
    num_posts = 50

    jsonResult = []

    client = oauth2_request(CONSUMER_KEY, CONSUMER_SECRET, ACCESS_TOKEN,
ACCESS_SECRET)
    tweets = get_user_timeline(client, screen_name)

    for tweet in tweets:
        getTwitterTwit(tweet, jsonResult)

    with open('%s_twitter.json' % (screen_name), 'w', encoding='utf8') as outfile:
        str_ = json.dumps(jsonResult,
                          indent=4, sort_keys=True,
                          ensure_ascii=False)
        outfile.write(str_)

    print ('%s_twitter.json SAVED' % (screen_name))

if __name__ == '__main__':
    main()
```

코드를 수행하면 jtbc_news_twitter.json 파일이 .py 가 있는 폴더에 저장되어 있을 것이다. 해당 파일을 JSON 뷰어를 이용하여 읽어 보면 [그림 2]와 같이 트윗(tweet)이 JSON 형태로 저장된 것을 확인할 수 있다.



[그림 2] JTBC_NEWS 트위터를 크롤링한 JSON 모습

[그림 2]의 처음 JSON 데이터는 [그림 3]의 내용을 나타내고 있다.

[그림 3] 저장한 JTBC 뉴스 트윗

실제 이전 절에서 페이스북 API 를 이용하여 JSON 형태의 데이터를 수신하고, 취급하는 방법에 대하여 충분히 이해하였으리라 생각하여 중복되는 내용은 5. Facebook 데이터 수집을 확인하기 바란다.

먼저 빈번하게 사용하는 환경변수의 값인 Cusomer_key 와 Customer_Secret, Access_token 과 Access_token 은 congi.py 라는 파일로 따로 작성한 후 import 하여 사용하였다.

```
CONSUMER_KEY = "[COUSUMER_KEY]"
CONSUMER_SECRET = "[COSUMER_SECRET]"
ACCESS_TOKEN = "[ACCESS_TOKEN]"
ACCESS_SECRET = "[ACCESS_SECRET]"
```

해당 .py 파일의 키값은 트위터에서 제공받은 값으로 변경하기 바란다.(Part I.의 트위터 개발자 계정 부분을 참조하기 바란다)

```
#[CODE 1] oauth2 모듈을 이용한 키 인증

def oauth2_request(consumer_key, consumer_secret, access_token, access_secret):
    try:
        consumer = oauth2.Consumer(key=consumer_key, secret=consumer_secret)
        token = oauth2.Token(key=access_token, secret=access_secret)
        client = oauth2.Client(consumer, token)
        return client
    except Exception as e:
        print(e)
        return None
```

oauth2 모듈은 OAuth 의 복잡한 과정을 간단하게 해결해주는 모듈을 제공한다. 우리는 부여받은 consumer_key 와 해당 secret, 그리고 acces_token 과 secret 을 oath2 에 전달해 줌으로써 기본적인 인증을 위한 준비를 마친다.

기본값을 지정한 후 consumer 와 token 오브젝트(object)를 Client 클래스로 전달하면 복잡한 댄싱과정을 마치고 access_token 을 포함하는 OAuthClient 를 반환한다.

```
#[CODE 2] 사용자의 timeline 트윗을 수신

def get_user_timeline(client, screen_name, count=50, include_rts='False'):
    base = "https://api.twitter.com/1.1"
    node = "/statuses/user_timeline.json"
    fields = "?screen_name=%s&count=%s&include_rts=%s" % (screen_name, count, include_rts)
    url = base + node + fields

    response, data = client.request(url)

    try:
        if response['status'] == '200':
            return json.loads(data.decode('utf-8'))
    except Exception as e:
        print(e)
```

```
        return None
```

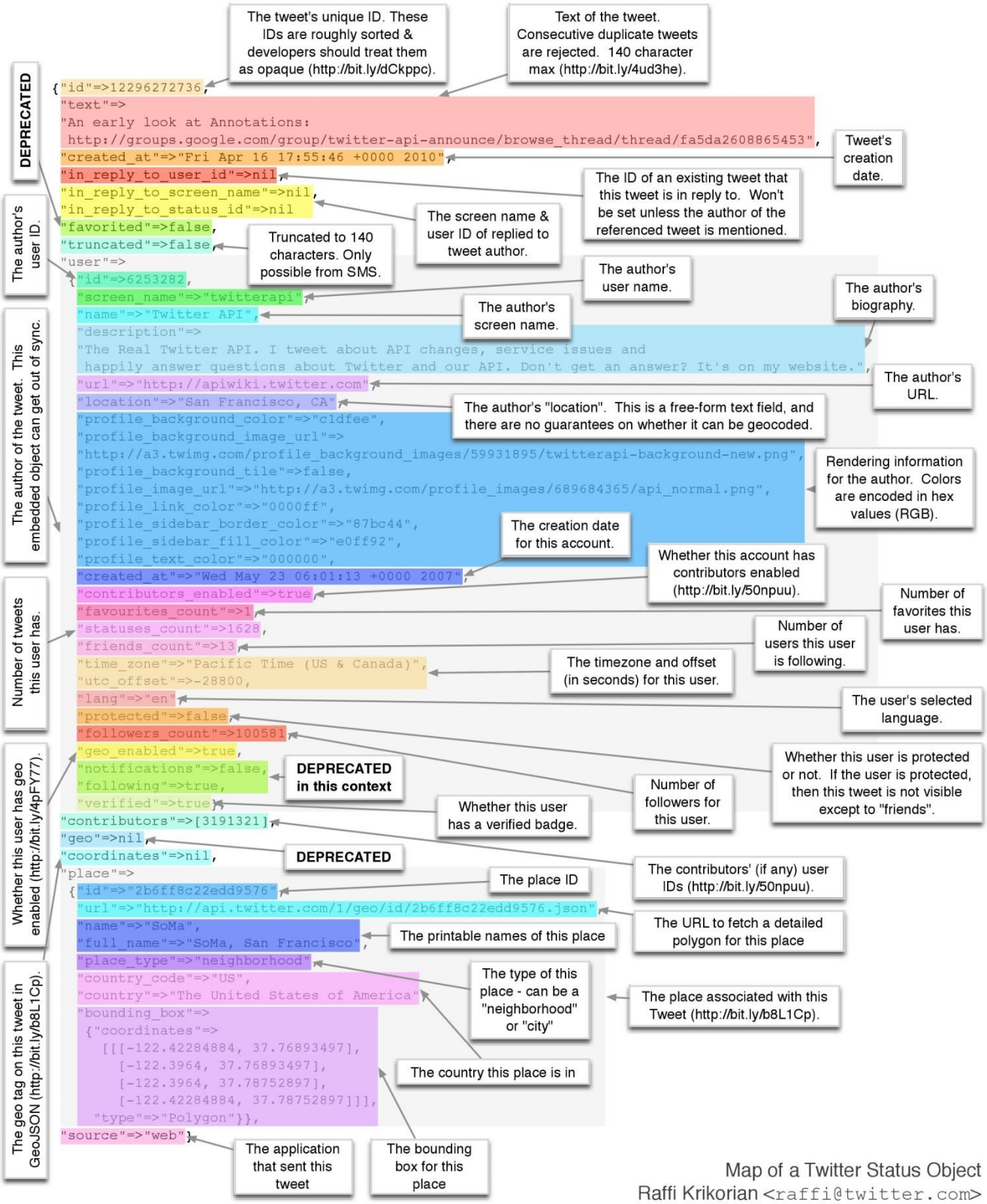
[CODE 2]의 경우에는 페이스북의 타임라인을 얻어 오는것과 동일한 형식을 가진다. 여기서 screen_name 은 트위터에서 사용하는 공식 이름(영문)이다. 페이스북은 Numeric ID 형식의 page id 를 가지고 타임라인을 요청하여야 하기 때문에 변환 과정을 거쳐야 하나, 트위터는 편하게 screen_name 을 가지고 요청이 가능하다.

수신받은 데이터는 페이스북과 마찬가지로 'utf-8'로 인코딩 되어있는 바이너리 JSON 데이터 형식이여서 'utf-8'로 디코딩 한 후 반환한다.

```
#[CODE 3] 사용자의 timeline 트윗을 수신

def getTwitterTwit(tweet, jsonResult):

    tweet_id = tweet['id_str']
    tweet_message = '' if 'text' not in tweet.keys() else tweet['text']

    screen_name = '' if 'user' not in tweet.keys() else tweet['user']['screen_name']

    tweet_link = ''
    if tweet['entities']['urls']: #list

        for i, val in enumerate(tweet['entities']['urls']):
            tweet_link = tweet_link + tweet['entities']['urls'][i]['url'] + ' '
    else:
        tweet_link = ''

    hashtags = ''
    if tweet['entities']['hashtags']: #list

        for i, val in enumerate(tweet['entities']['hashtags']):
            hashtags = hashtags + tweet['entities']['hashtags'][i]['text'] + ' '
    else:
        hashtags = ''

    if 'created_at' in tweet.keys():
```

```
        # Twitter used UTC Format. EST = UTC + 9(Korean Time) Format ex: Fri Feb 10 03:57:27 +0000 2017

        tweet_published = datetime.datetime.strptime(tweet['created_at'],'%a %b %d %H:%M:%S +0000 %Y')
        tweet_published = tweet_published + datetime.timedelta(hours=+9)
        tweet_published = tweet_published.strftime('%Y-%m-%d %H:%M:%S')
    else:
        tweet_published = ''

    num_favorite_count = 0 if 'favorite_count' not in tweet.keys() else tweet['favorite_count']
    num_comments = 0
    num_shares = 0 if 'retweet_count' not in tweet.keys() else tweet['retweet_count']
    num_likes = num_favorite_count
    num_loves = num_wows = num_hahas = num_sads = num_angrys = 0

    jsonResult.append({'post_id':tweet_id, 'message':tweet_message,
                       'name':screen_name, 'link':tweet_link,
                       'created_time':tweet_published, 'num_reactions':num_favorite_count,
                       'num_comments':num_comments, 'num_shares':num_shares,
                       'num_likes':num_likes, 'num_loves':num_loves,
                       'num_wows':num_wows, 'num_hahas':num_hahas,
                       'num_sads':num_sads, 'num_angrys':num_angrys, 'hashtags': hashtags})
```

[CODE 3]은 수신한 JSON 형식에서 우리가 원하는 데이터의 값을 키값과 그에 해당하는 데이터값을 조회한다. 페이스북은 reaction 이라는 레코드셋을 가지고 "좋아요, 사랑해요" 등의 리액션 데이터를 가지고 있지만 트위터는 단순하게 '좋아요'의 데이터만 가지고 있다. [그림 4]는 트위터의 JSON 구성 형태를 알아보기 편하도록 작성된 맵이다.


The tweet's unique ID. These IDs are roughly sorted & developers should treat them as opaque (http://bit.ly/dCkppc).
Text of the tweet. Consecutive duplicate tweets are rejected. 140 character max (http://bit.ly/4ud3he).
DEPRECATED
{"id"=>12296272736,
"text"=>
"An early look at Annotations:
http://groups.google.com/group/twitter-api-announce/browse_thread/thread/fa5da2608865453",
"created_at"=>"Fri Apr 16 17:55:46 +0000 2010",
Tweet's creation date.
"in_reply_to_user_id"=>nil,
The ID of an existing tweet that this tweet is in reply to. Won't be set unless the author of the referenced tweet is mentioned.
The author's user ID.
"in_reply_to_screen_name"=>nil,
"in_reply_to_status_id"=>nil
The screen name & user ID of replied to tweet author.
"favorited"=>false,
"truncated"=>false,
Truncated to 140 characters. Only possible from SMS.
"user"=>
{"id"=>6253282,
The author's user name.
"screen_name"=>"twitterapi",
The author's biography.
"name"=>"Twitter API",
The author's screen name.
The author of the tweet. This embedded object can get out of sync.
"description"=>
"The Real Twitter API. I tweet about API changes, service issues and
happily answer questions about Twitter and our API. Don't get an answer? It's on my website.",
"url"=>"http://apiwiki.twitter.com"
The author's URL.
"location"=>"San Francisco, CA"
The author's "location". This is a free-form text field, and there are no guarantees on whether it can be geocoded.
"profile_background_color"=>"c1dfee",
"profile_background_image_url"=>
"http://a3.twimg.com/profile_background_images/59931895/twitterapi-background-new.png",
"profile_background_tile"=>false,
"profile_image_url"=>"http://a3.twimg.com/profile_images/689684365/api_normal.png",
"profile_link_color"=>"0000ff",
"profile_sidebar_border_color"=>"87bc44",
"profile_sidebar_fill_color"=>"e0ff92",
"profile_text_color"=>"000000",
Rendering information for the author. Colors are encoded in hex values (RGB).
The creation date for this account.
"created_at"=>"Wed May 23 06:01:13 +0000 2007",
Whether this account has contributors enabled (http://bit.ly/50npuu).
"contributors_enabled"=>true,
Number of favorites this user has.
"favourites_count"=>1,
Number of tweets this user has.
"statuses_count"=>1628,
Number of users this user is following.
"friends_count"=>13,
"time_zone"=>"Pacific Time (US & Canada)",
The timezone and offset (in seconds) for this user.
"utc_offset"=>-28800,
"lang"=>"en"
The user's selected language.
"protected"=>false,
"followers_count"=>100581
Whether this user is protected or not. If the user is protected, then this tweet is not visible except to "friends".
"geo_enabled"=>true,
"notifications"=>false,
DEPRECATED in this context
"following"=>true,
Number of followers for this user.
"verified"=>true},
Whether this user has a verified badge.
"contributors"=>[3191321],
Whether this user has geo enabled (http://bit.ly/4pFY77).
"geo"=>nil,
"coordinates"=>nil,
DEPRECATED
The contributors' (if any) user IDs (http://bit.ly/50npuu).
"place"=>
{"id"=>"2b6ff8c22edd9576"
The place ID
"url"=>"http://api.twitter.com/1/geo/id/2b6ff8c22edd9576.json"
The URL to fetch a detailed polygon for this place
"name"=>"SoMa",
"full_name"=>"SoMa, San Francisco",
The printable names of this place
"place_type"=>"neighborhood"
The type of this place - can be a "neighborhood" or "city"
The place associated with this Tweet (http://bit.ly/b8L1Cp).
The geo tag on this tweet in GeoJSON (http://bit.ly/b8L1Cp).
"country_code"=>"US",
"country"=>"The United States of America"
The country this place is in
"bounding_box"=>
{"coordinates"=>
[[[-122.42284884, 37.76893497],
[-122.3964, 37.76893497],
[-122.3964, 37.78752897],
[-122.42284884, 37.78752897]]],
"type"=>"Polygon"}},
"source"=>"web"}
The application that sent this tweet
The bounding box for this place
Map of a Twitter Status Object
Raffi Krikorian <raffi@twitter.com>
18 April 2010

[그림 4] 트위터 Status 객체 구성 요소

트위터에서는 API 를 활용하기 위한 다양한 페이지 URL(endpoint)를 제공한다(코드에서 사용한 node 부분).

용도	**Endpoint**
내 타임라인	/statuses/home_timeline.json allowed_param:'since_id', 'max_id', 'count'
특정 사용자 타임라인	/statuses/user_timeline.json allowed_param:'id', 'user_id', 'screen_name', 'since_id'
맨션 타임라인	/statuses/mentions_timeline.json allowed_param:'since_id', 'max_id', 'count'
나를 리트윗한 정보	/statuses/retweets_of_me.json allowed_param:'since_id', 'max_id', 'count'
사용자 정보	/users/show.json allowed_param:'id', 'user_id', 'screen_name'
사용자 검색	/users/search.json allowed_param:'q', 'count', 'page'
팔로워	/followers/list.json allowed_param: 'id', 'user_id', 'screen_name', 'cursor', 'count', 'skip_status', 'include_user_entities'
검색	/search/tweets.json allowed_param: 'q', 'lang', 'locale', 'since_id', 'geocode', 'max_id', 'since', 'until', 'result_type', 'count', 'include_entities', 'from', 'to', 'source'

트위터는 큰 결과 세트를 검색하기 위하여 페이징 기법으로 커서링(cursoring)이라는 기법을 사용한다. 먼저 처음에는 cursor 의 값을 '-1'로 지정하여 결과를 검색하면 'previous_cursor', 'next_cursor', 'previous_cursor_str' 및 'next_cursor_str'을 반환한다. [그림 4]에서 JSON 구조를 잘

살펴 보았다면 트위터는 숫자문자열의 경우에는 항상 해당 '키'값과 '키_str'을 제공하는 것을 확인할 수 있다.

다음 페이지를 얻기 위하여 우리는 'cursor'에 'next_cursor'의 값을 전달하여 페이지를 요청하면 된다. 만약 'next_cursor'의 값이 '0'으로 반환되면 더 이상의 페이지는 없는 것을 의미한다(페이스북에서는 다음 페이지를 요청하는 URL 을 보내주므로 단순하게 해당 URL 을 호출하는 것으로 데이터를 획득할 수 있었다). 또한 페이스북의 since 와 until 의 개념은 트위터에서는 Search API 에서 사용되며, 'statuses'에서는 지원하지 않는다. 자세한 부분은 Search API 를 참조하기 바란다.