**OAuth开发文档**

# OAuth介绍

OAUTH协议为用户资源的授权提供了一个安全的、开放而又简易的标准。与以往的授权方式不同之处是OAUTH的授权不会使第三方触及到用户的帐号信 息（如用户名与密码），即第三方无需使用用户的用户名与密码就可以申请获得该用户资源的授权，因此OAUTH是安全的。同时，任何第三方都可以使用 OAUTH认证服务，任何服务提供商都可以实现自身的OAUTH认证服务，因而OAUTH是开放的。业界提供了OAUTH的多种实现如 PHP，JavaScript，Java，Ruby等各种语言开发包，大大节约了程序员的时间，因而OAUTH是简易的。目前互联网很多服务如Open API，很多大头公司如Google，Yahoo，Microsoft等都提供了OAUTH认证服务，这些都足以说明OAUTH标准逐渐成为开放资源授权 的标准。

在官方网站的首页，可以看到下面这段简介：

**An open protocol to allow secure API authorization in a simple and standard method from desktop and web applications.**

    大概意思是说OAUTH是一种开放的协议，为桌面程序或者基于BS的web应用提供了一种简单的，标准的方式去访问需要用户授权的API服务。OAUTH 类似于Flickr Auth、Google's AuthSub、Yahoo's BBAuth、 Facebook Auth等。OAUTH认证授权具有以下特点：

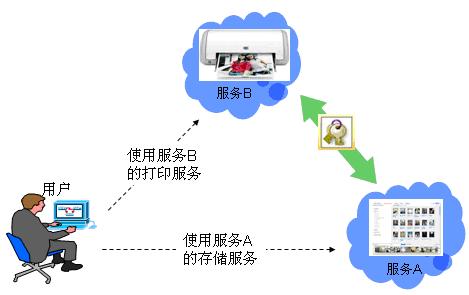
1. 简单：不管是OAUTH服务提供者还是应用开发者，都很容易于理解与使用；

2. 安全：没有涉及到用户密钥等信息，更安全更灵活；

3. 开放：任何服务提供商都可以实现OAUTH，任何软件开发商都可以使用OAUTH；

# OAUTH产生的背景

典型案例：如果一个用户拥有两项服务：一项服务是图片在线存储服务A， 另一个是图片在线打印服务B。如下图所示。由于服务A与服务B是由两家不同的服务提供商提供的，所以用户在这两家服务提供商的网站上各自注册了两个用户， 假设这两个用户名各不相同，密码也各不相同。当用户要使用服务B打印存储在服务A上的图片时，用户该如何处理？法一：用户可能先将待打印的图片从服务A上 下载下来并上传到服务B上打印，这种方式安全但处理比较繁琐，效率低下；法二：用户将在服务A上注册的用户名与密码提供给服务B，服务B使用用户的帐号再 去服务A处下载待打印的图片，这种方式效率是提高了，但是安全性大大降低了，服务B可以使用用户的用户名与密码去服务A上查看甚至篡改用户的资源。



    很多公司和个人都尝试解决这类问题，包括Google、Yahoo、Microsoft，这也促使OAUTH项目组的产生。OAuth是由Blaine Cook、Chris Messina、Larry Halff 及David Recordon共同发起的，目的在于为API访问授权提供一个开放的标准。OAuth规范的1.0版于2007年12月4日发布。通过官方网址：[http://oauth.net](http://oauth.net/)可以阅读更多的相关信息。

# OAUTH相关术语

在弄清楚OAUTH流程之前，我们先了解下OAUTH的一些术语的定义：

* **OAUTH相关的三个URL：**
  + Request Token URL: 获取未授权的Request Token服务地址；
  + User Authorization URL: 获取用户授权的Request Token服务地址；
  + Access Token URL: 用授权的Request Token换取Access Token的服务地址；
* **OAUTH相关的参数定义：**
  + oauth\_consumer\_key: 使用者的ID，OAUTH服务的直接使用者是开发者开发出来的应用。所以该参数值的获取一般是要去OAUTH服务提供商处注册一个应用，再获取该应用的oauth\_consumer\_key。如Yahoo该值的注册地址为：<https://developer.yahoo.com/dashboard/>
  + oauth\_consumer\_secret：oauth\_consumer\_key对应的密钥。
  + oauth\_token  
    oAuth进行到最后一步得到的一个“黄金招牌”。有了这个“黄金招牌”，**求资源的网站B**就可以大摇大摆去 **拥有资源的网站A** 抓取任意有权限可以被抓取的资源了。  
    oauth\_token\_secret  
    上面那个东西的私钥
  + oauth\_signature\_method: 请求串的签名方法，应用每次向OAUTH三个服务地址发送请求时，必须对请求进行签名。签名的方法有：HMAC-SHA1、RSA-SHA1与PLAINTEXT等三种。
  + oauth\_signature: 用上面的签名方法对请求的签名。
  + oauth\_timestamp: 发起请求的时间戳，其值是距1970 00:00:00 GMT的秒数，必须是大于0的整数。本次请求的时间戳必须大于或者等于上次的时间戳。
  + oauth\_nonce: 随机生成的字符串，用于防止请求的重放，防止外界的非法攻击。
  + oauth\_version: OAUTH的版本号，可选，其值必须为1.0。

**OAUTH HTTP响应代码：**

* HTTP 400 Bad Request 请求错误
  + Unsupported parameter 参数错误
  + Unsupported signature method 签名方法错误
  + Missing required parameter 参数丢失
  + Duplicated OAuth Protocol Parameter 参数重复
* HTTP 401 Unauthorized 未授权
  + Invalid Consumer Key 非法key
  + Invalid / expired Token 失效或者非法的token
  + Invalid signature 签名非法
  + Invalid / used nonce 非法的nonce

# OAUTH认证授权流程

在弄清楚了OAUTH的术语后，我们可以对OAUTH认证授权的流程进行初步认识。其实，简单的来说，OAUTH认证授权就三个步骤，三句话可以概括：

## 获取未授权的request token

### 请求参数

oauth\_consumer\_key:

消费方键值。

oauth\_signature\_method:

消费方签署本请求所用的签名方法。

oauth\_signature:

签名，定义于[签署请求 (签署请求)](http://www.rollingcode.org/blog/f/oauth-core-1.0-final-cn.html#signing_process)。

oauth\_timestamp:

定义于[Nonce and Timestamp (单次值与时间戳)](http://www.rollingcode.org/blog/f/oauth-core-1.0-final-cn.html#nonce)。

oauth\_nonce:

定义于[Nonce and Timestamp (单次值与时间戳)](http://www.rollingcode.org/blog/f/oauth-core-1.0-final-cn.html#nonce)。

oauth\_version:

可选。如果存在，其值必须为1.0。如果参数不存在，服务提供方必须假定协议版本为1.0。 服务提供方对1.0以外取值的响应尚未定义。

额外参数:

由服务提供方定义的任意额外参数

### 服务方返回结果

响应包含如下参数：

oauth\_token:

请求令牌

oauth\_token\_secret:

令牌密钥

附加参数:

由服务提供方定义的任意参数。

## 获取用户授权的Request Token

### 请求参数

oauth\_token:

可选。在前述步骤中获得的请求令牌。服务提供方可以声明此参数为必须，也可以允许不包含在授权URL中并提示用户手工输入。

oauth\_callback:

可选。消费方可以指定一个URL，当 [获取用户授权 (获取用户授权)](http://www.rollingcode.org/blog/f/oauth-core-1.0-final-cn.html#auth_step2)成功后，服务提供方将重定向用户到这个URL。

附加参数:

由服务提供方定义的任意参数。

### 服务提供方将用户引导回消费方

如果消费方在oauth\_callback中提供了回调URL(在[消费方引导用户至服务提供方 (消费方引导用户至服务提供方)](http://www.rollingcode.org/blog/f/oauth-core-1.0-final-cn.html#user_auth_redirected)中描述)，则服务提供方构造一个HTTP GET请求URL，重定向用户浏览器到该URL，并包含如下参数：

oauth\_token:

被用户授权或否决的请求令牌

回调URL可以包含消费方提供的查询参数，服务提供方必须保持已有查询不变并追加oauth\_token参数。

## 用授权的Request Token换取Access Token

### 消费方请求访问令牌参数

oauth\_consumer\_key:

消费方键值。

oauth\_token:

之前获取的请求令牌。

oauth\_signature\_method:

消费方使用的签署方法。

oauth\_signature:

[签署请求 (签署请求)](http://www.rollingcode.org/blog/f/oauth-core-1.0-final-cn.html#signing_process)中定义的签名。

oauth\_timestamp:

在[单次值与时间戳 (单次值与时间戳)](http://www.rollingcode.org/blog/f/oauth-core-1.0-final-cn.html#nonce)中定义。

oauth\_nonce:

在[单次值与时间戳 (单次值与时间戳)](http://www.rollingcode.org/blog/f/oauth-core-1.0-final-cn.html#nonce)中定义。

oauth\_version:

可选。如果存在，其值必须为1.0。如果参数不存在，服务提供方必须假定协议版本为1.0。 服务提供方对1.0以外取值的响应尚未定义。

### 返回参数

oauth\_token:

访问令牌。

oauth\_token\_secret:

令牌密钥。

## 访问受保护资源

### 请求参数

oauth\_consumer\_key:

消费方键值。

oauth\_token:

访问令牌。

oauth\_signature\_method:

消费方使用的签署方法。

oauth\_signature:

[签署请求 (签署请求)](http://www.rollingcode.org/blog/f/oauth-core-1.0-final-cn.html#signing_process)中定义的签名。

oauth\_timestamp:

定义于[单次值与时间戳 (单次值与时间戳)](http://www.rollingcode.org/blog/f/oauth-core-1.0-final-cn.html#nonce).

oauth\_nonce:

定义于[单次值与时间戳 (单次值与时间戳)](http://www.rollingcode.org/blog/f/oauth-core-1.0-final-cn.html#nonce).

oauth\_version:

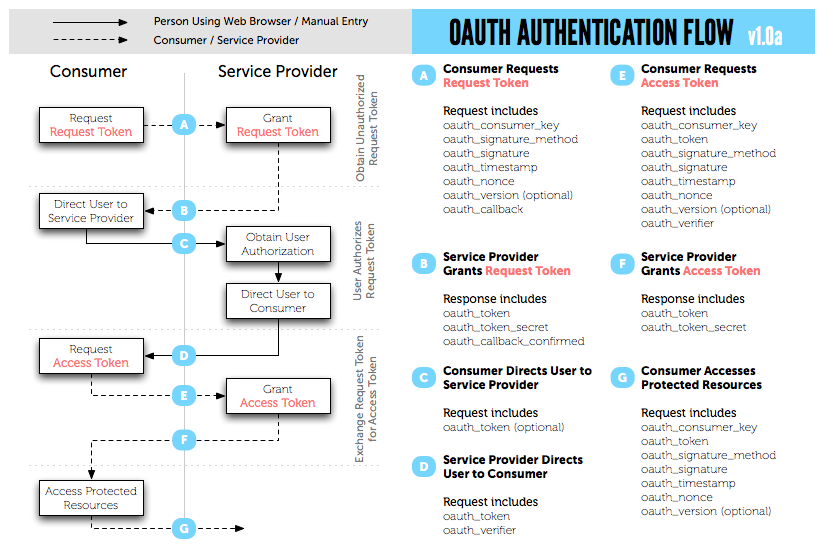
可选。如果存在，其值必须为1.0。如果参数不存在，服务提供方必须假定协议版本为1.0。 服务提供方对1.0以外取值的响应尚未定义。

附加参数:

服务提供方指定的附加参数。

### 具体流程

    当应用拿到Access Token后，就可以有权访问用户授权的资源了。大家肯能看出来了，这三个步骤不就是对应OAUTH的三个URL服务地址嘛。一点没错，上面的三个步骤 中，每个步骤分别请求一个URL，并且收到相关信息，并且拿到上步的相关信息去请求接下来的URL直到拿到Access Token。具体的步骤如下图所示：



具体每步执行信息如下：

A. 使用者（第三方软件）向OAUTH服务提供商请求未授权的Request Token。向Request Token URL发起请求，请求需要带上的参数见上图。

B. OAUTH服务提供商同意使用者的请求，并向其颁发未经用户授权的oauth\_token与对应的oauth\_token\_secret，并返回给使用者。

C. 使用者向OAUTH服务提供商请求用户授权的Request Token。向User Authorization URL发起请求，请求带上上步拿到的未授权的token与其密钥。

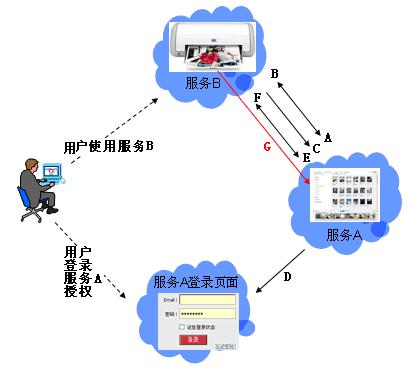
D. OAUTH服务提供商将引导用户授权。该过程可能会提示用户，你想将哪些受保护的资源授权给该应用。此步可能会返回授权的Request Token也可能不返回。如Yahoo OAUTH就不会返回任何信息给使用者。

E. Request Token 授权后，使用者将向Access Token URL发起请求，将上步授权的Request Token换取成Access Token。请求的参数见上图，这个比第一步A多了一个参数就是Request Token。

F. OAUTH服务提供商同意使用者的请求，并向其颁发Access Token与对应的密钥，并返回给使用者。

G. 使用者以后就可以使用上步返回的Access Token访问用户授权的资源。

    从上面的步骤可以看出，用户始终没有将其用户名与密码等信息提供给使用者（第三方软件），从而更安全。用OAUTH实现背景一节中的典型案例：当服务 B（打印服务）要访问用户的服务A（图片服务）时，通过OAUTH机制，服务B向服务A请求未经用户授权的Request Token后，服务A将引导用户在服务A的网站上登录，并询问用户是否将图片服务授权给服务B。用户同意后，服务B就可以访问用户在服务A上的图片服务。 整个过程服务B没有触及到用户在服务A的帐号信息。如下图所示，图中的字母对应OAUTH流程中的字母：



# 关于签名说明

 在第一步获取Request Token时，需要使用Consumer Key和API Key Secret进行签名 的Consumer Key Secret。

 在第二步换取Access Token时，需要使用Consumer Key，API Key Secret、Request Token和Request Token Secret进行签名。

 而在第三步访问受限资源时，需要使用Consumer Key，API Key Secret、Access Token和Access Token Secret进行签名

举例说明：

sha.getSignature("Request Toke&API Key Secret");

# 关于服务端如何生成Request Token以及Access Token等说明

## 获取未授权的Request Token和密匙tokenSecret

System.nanoTime()是返回Long类型的毫微秒的系统计时器的当前值。

requesttToken=DigestUtils.md5Hex(consumer\_key+ System.nanoTime())；

tokenSecret=consumer\_key + System.nanoTime() + requesttToken;

## 获取用户授权的Request Token

获取用户授权的request\_token可以不返回也可以使用上面的requesttToken和tokenSecret

## 用授权的Request Token换取Access Token

accessToken= DigestUtils.md5Hex(consumer\_key+ requestToken+ System.nanoTime());

accessSecret=DigestUtils.md5Hex(consumer\_key+requestToken+System.nanoTime()+accessTokn);