佛山智能医疗系统

# 项目概述

## 项目背景

由中科嘉速公司承接佛山智能医疗系统的信息安全技术研究、及主流移动设备安全开发框架研发工作。

## 项目目标

主要目标为实现佛山智能医疗系统的安全认证功能。用户可采用各大网站已注册账号登录系统，在保证用户密码安全的前提下，免除用户重复注册的烦恼，从而扩大系统的潜在用户群。

## 主要内容

### 系统架构

1. **系统架构**

如图 1‑1所示，服务器端采用MySQL数据库保存用户和预约信息，通过Restlet发布WebService服务，供手机客户端调用。客户端根据OAuth协议在各开放平台上完成用户授权并获取access token（由授权服务器实现），然后利用access token调用开放平台提供的API，获取各种信息（由资源服务器实现）。

|  |  |
| --- | --- |
| 软硬件名称 | 说明 |
| 操作系统 | Windows 7|8 32或64位操作系统 |
| 数据库 | MySQL Server 5.6 |
| Web服务器 | Apache Tomcat 7.0 |
| IDE | Eclipse J2EE 4.2以上 |
| 客户端 | Android SDK18 |

表 1‑1系统软硬件平台

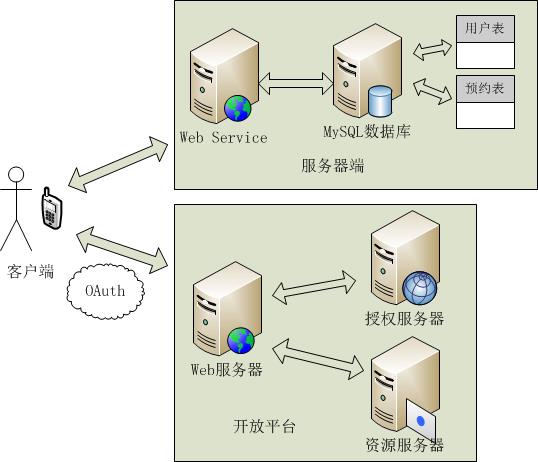


图 1‑1系统架构图

1. **OAuth授权流程**

OAuth授权流程如图 1‑2所示：

1. 用户登录客户端；
2. 客户端向开放平台发出数据请求；
3. 开放平台询问用户是否授权；
4. 用户在授权页面同意授权；
5. 开放平台返回客户端authorization token；
6. 客户端利用authorization token请求用户信息；
7. 开放平台返回用户信息。



图 1‑2OAuth授权流程图

### 系统流程

1. 用户登录界面，用户可采用以下开放平台帐号登录系统，以新浪微博为例，单击按钮。



图 1‑3登录界面

1. 用户跳转至新浪微博的授权页面，输入用户名密码，完成授权。

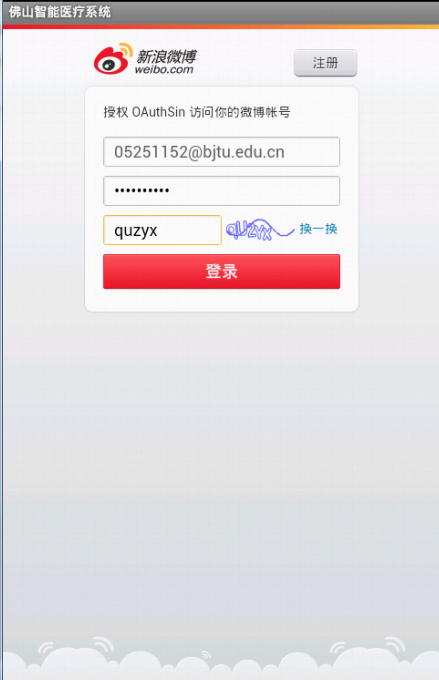


图 1‑4授权界面

1. 用户跳转回本系统，系统从新浪微博获取用户头像、姓名、性别等信息，用户还需填写电话、科室、就诊日期等信息，单击挂号。系统将用户信息保存到数据库中，当用户下次登录时可直接从数据库获取相应信息。



图 1‑5主界面

1. 用户跳转至预约成功界面，系统将挂号信息保存到数据库中。



图 1‑6预约界面

## 应用方式

该项目为原型系统，其意义在于：

1. 作为项目前期调研的一部分，进行实验性的开发，解决了如何利用OAuth实现手机客户端授权和信息获取的技术难点。
2. 开发者可借鉴项目中采用的主要技术，将其应用到后续的开发过程中，缩短开发周期。

## 文档内容及结构

第一章为项目概述，介绍项目概况。

第二章为项目配置，依照详细的配置步骤，用户可顺利安装运行demo，并在其基础上进行后续开发。

第三章为开发指南，详细介绍demo中采用的主要技术和快速入门方法，用户可借鉴这些技术，应用到具体的项目开发中。

第四章为Android开发，介绍如何在Eclipse中配置Android开发环境和Android开发过程中的常见问题。

第五章为OAuth客户端，详细介绍了OAuth客户端实现流程，以及版本对比、技术选择和常见问题等。

第六章为OAuth服务器，详细介绍了OAuth服务器实现流程。

第七章为spring security oauth，在官网推荐的几种实现方式中选择spring security oauth进行调研，详细介绍了sample如何运行，客户端和服务器端如何配置。

第八章为spring security，由于spring security oauth是构建在spring security上的，因此本章对spring security快速入门进行介绍。

# 项目配置

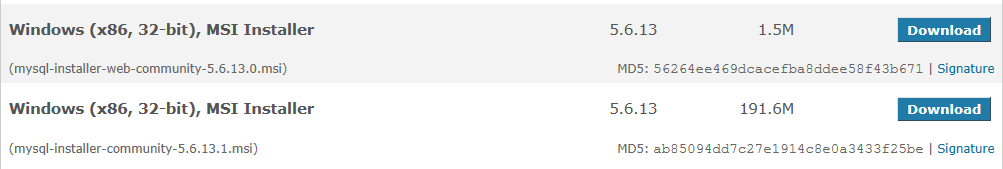
## software

software目录下包含所有安装包，32bit和64bit分别支持32位和64位Windows操作系统。

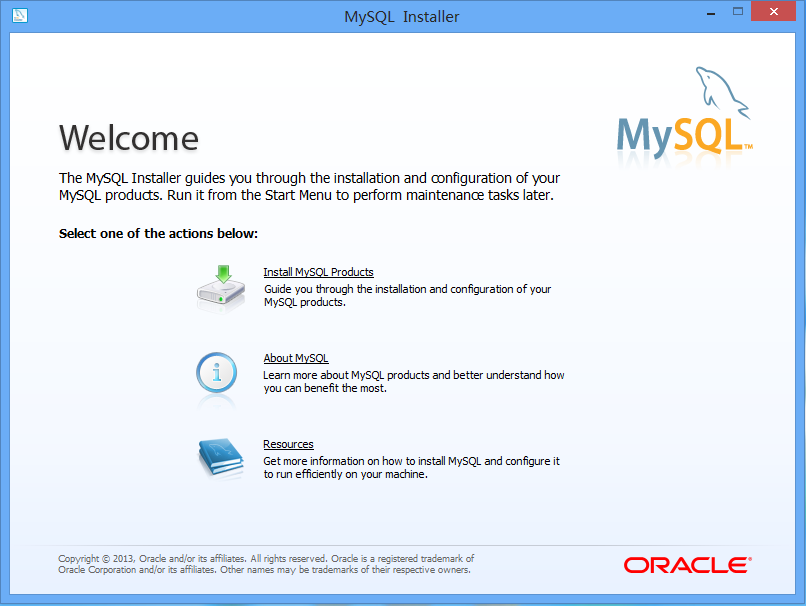
### 安装MySQL

<http://dev.mysql.com/downloads/windows/installer/>

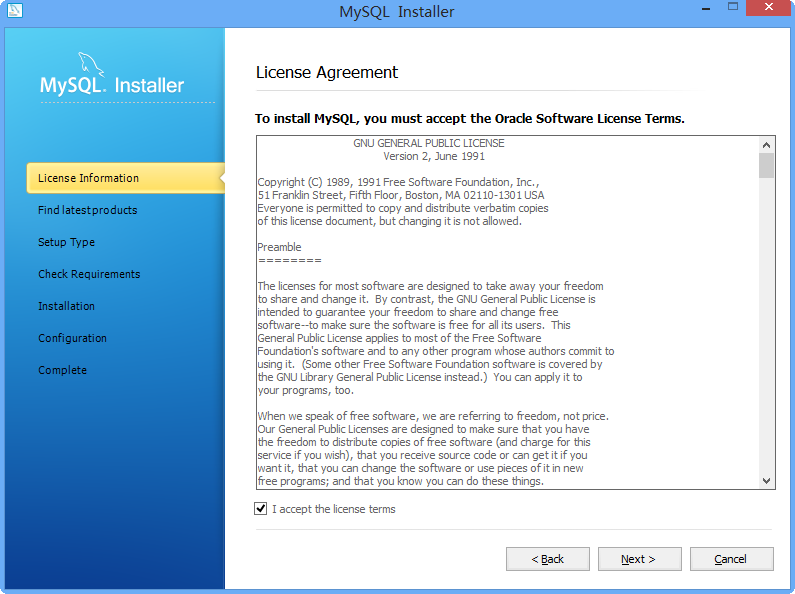
该安装包支持32位和64位操作系统



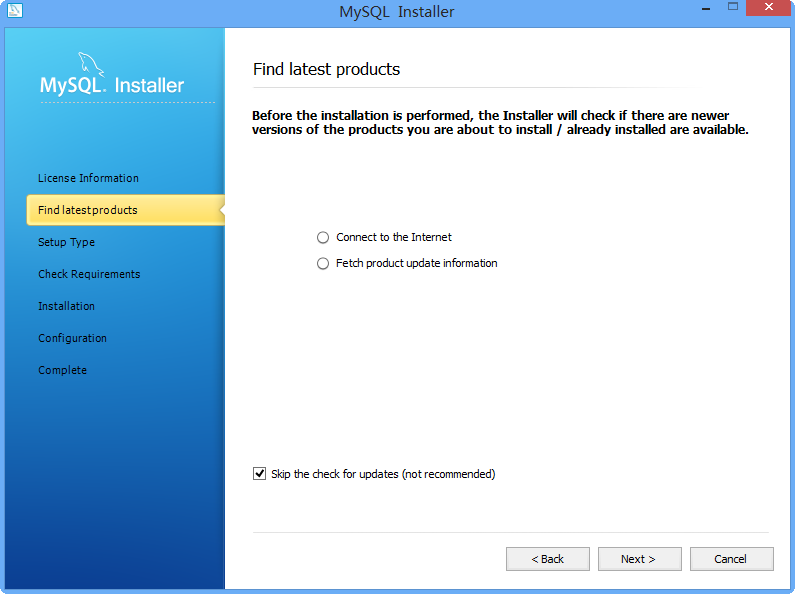
运行software/mysql-installer-community-5.6.13.1.msi



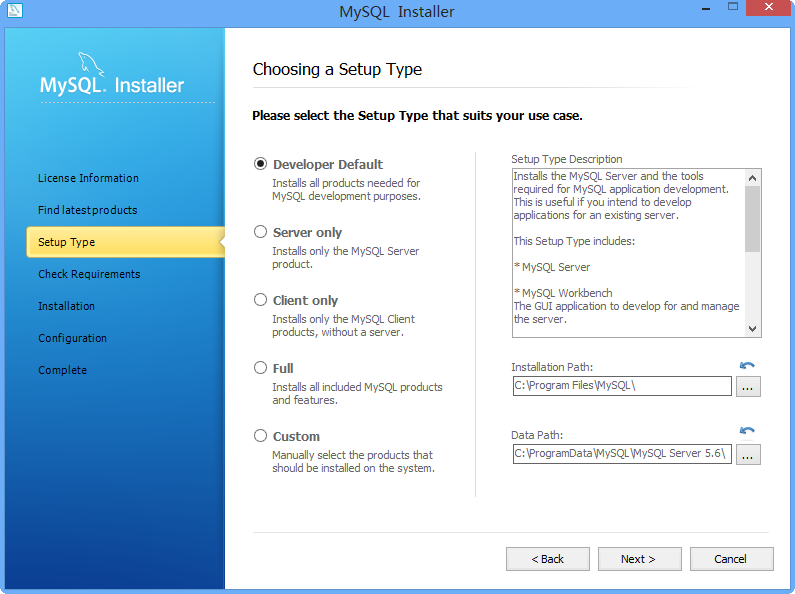
选Install MySQL Products



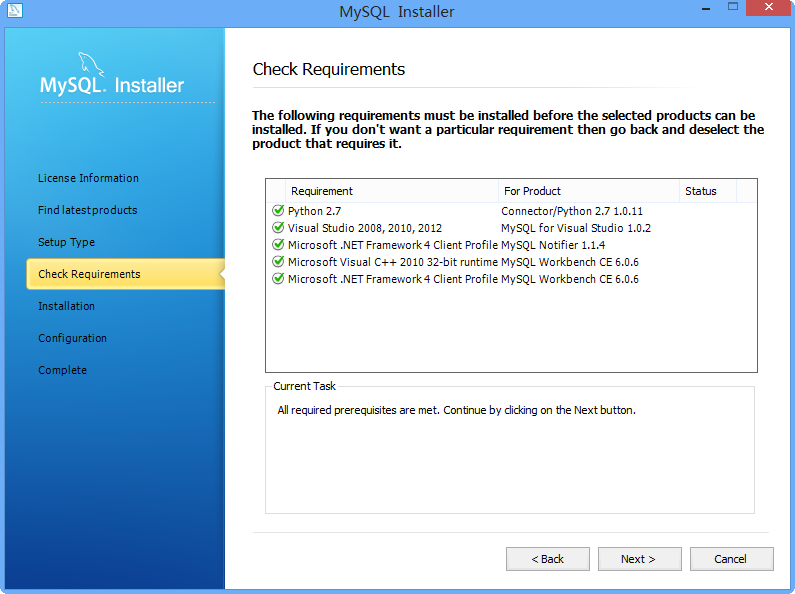
选I accept the licence terms，Next



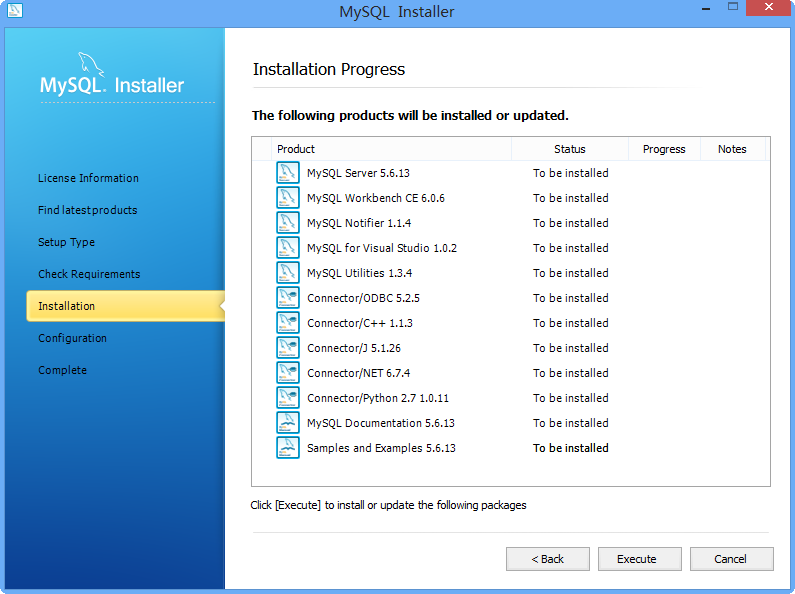
选Skip the check for updates（不检查更新），Next



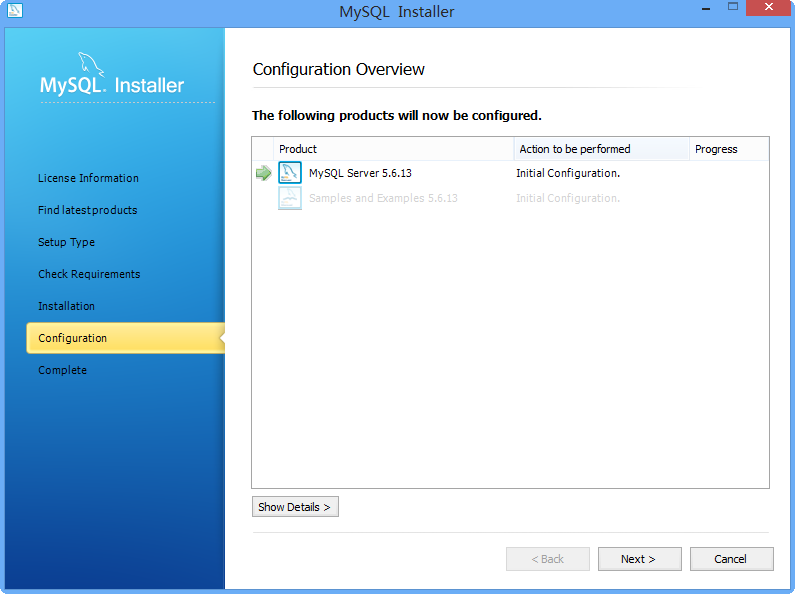
选Developer Default，设置安装路径和数据存储路径，推荐采用默认值



Execute，安装所需插件



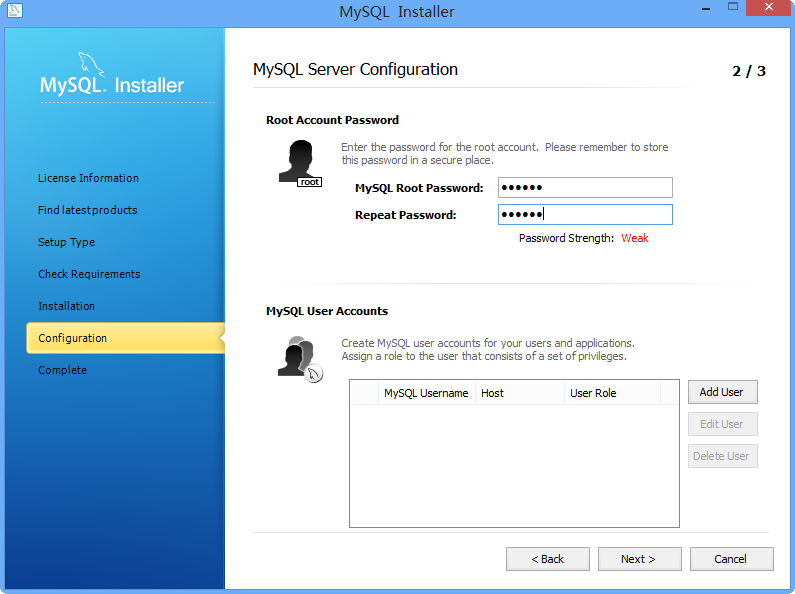
Execute，安装完后Next



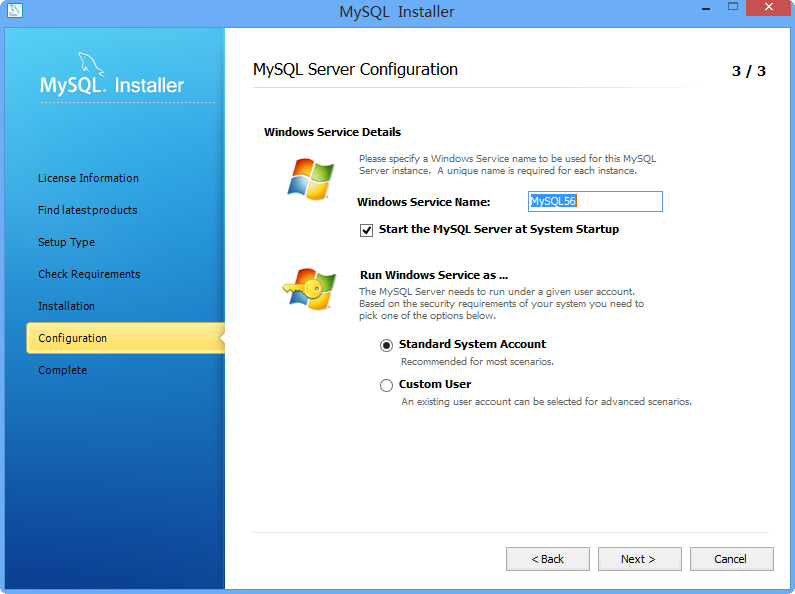
Next



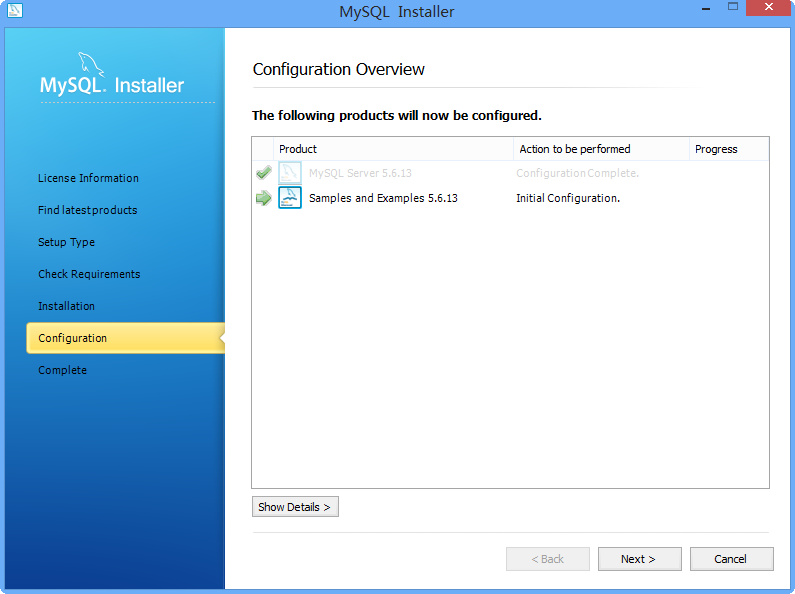
Next



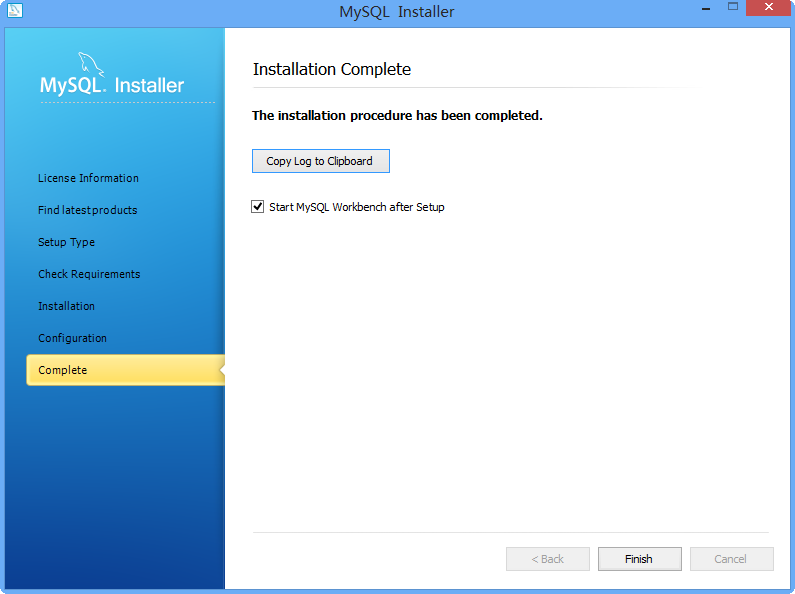
设置Root密码，该密码要记得，下面要用到。Next



Next

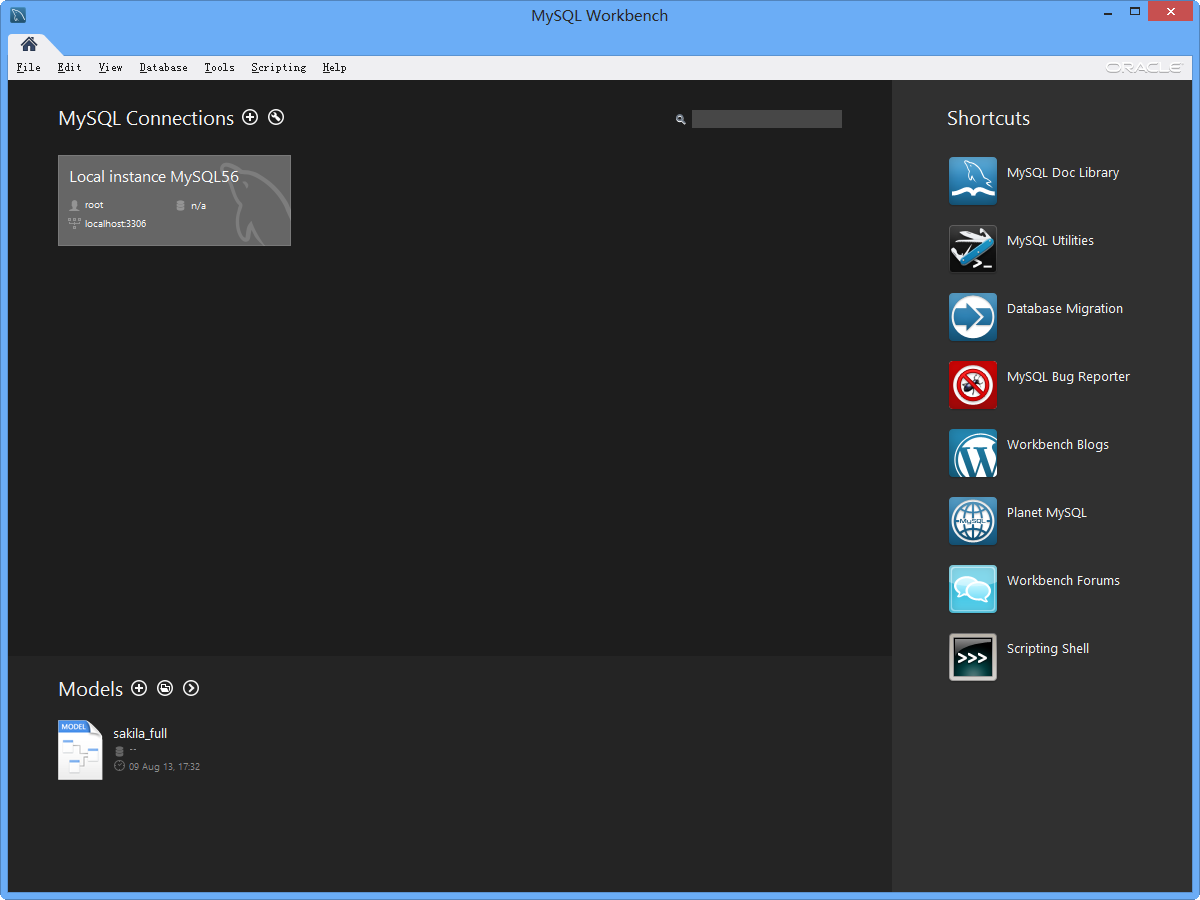


Next

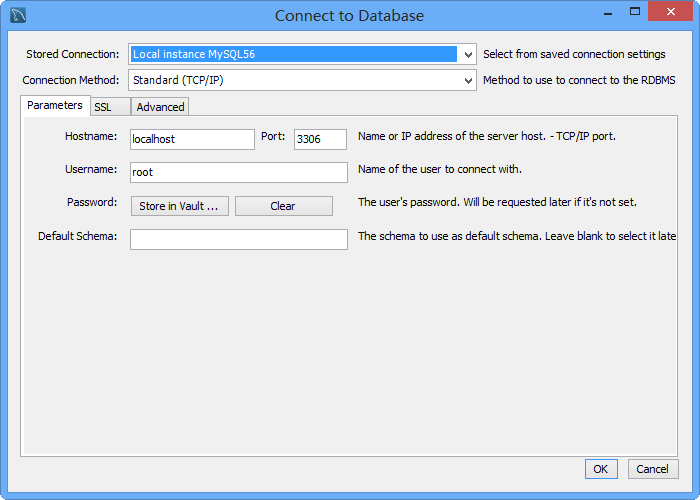


Finish

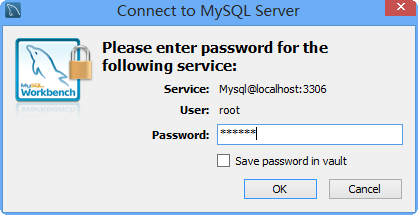
### 配置数据库



弹出主界面，选Database->Connect to Database…

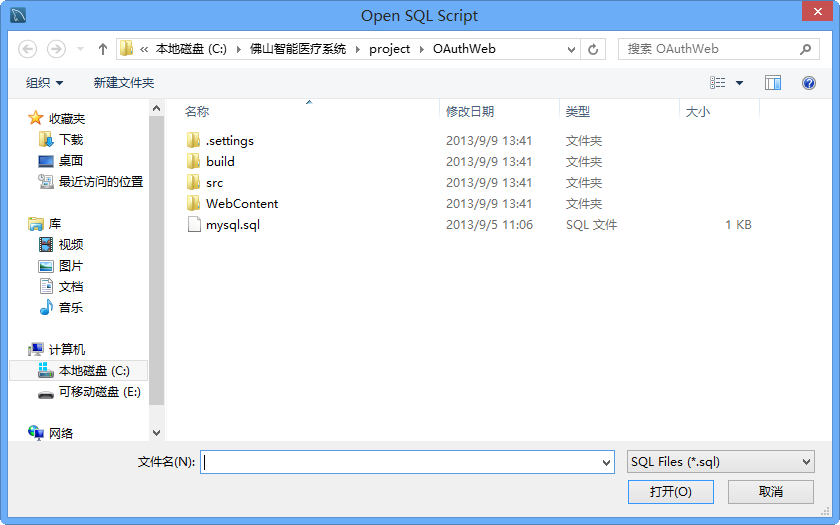


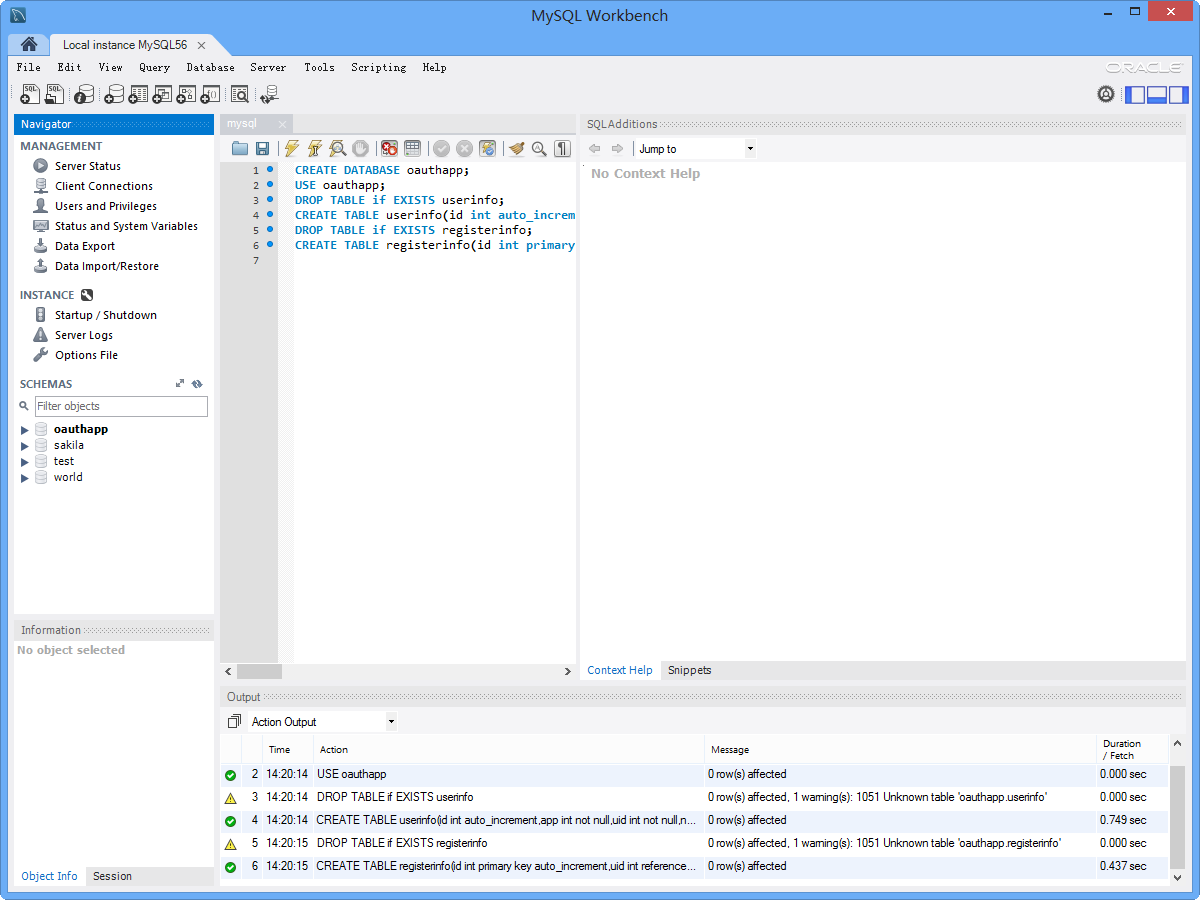
Stored Connection，选Local Instance MySQL56



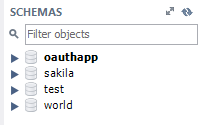
输入刚才设置的Root密码。OK。

单击“导入脚本”按钮，选择project/OAuthWeb/mysql.sql文件。





单击“执行”按钮，新建名为oauthapp的数据库，生成userinfo和registerinfo两张表。



单击“刷新”按钮，可看到生成的oauthapp数据库。

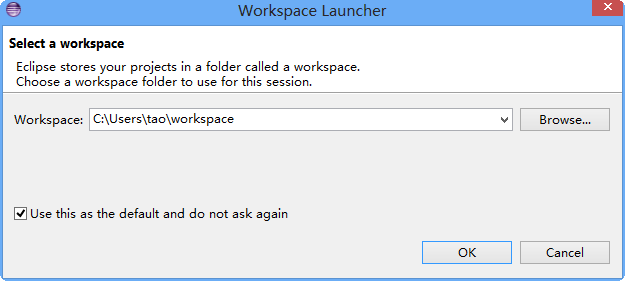
### 配置ADT

**注意：以下步骤需根据操作系统位数到software的64bit或32bit目录找相应安装包。**

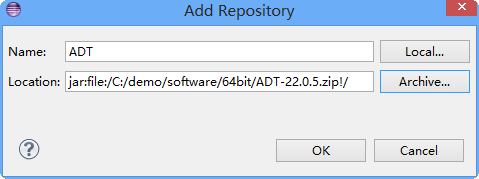
解压adt-bundle-windows-xxx.zip到任意目录（**注：目录路径中不要包含中文**）。

解压eclipse-jee-kepler-xxx.zip到任意目录。

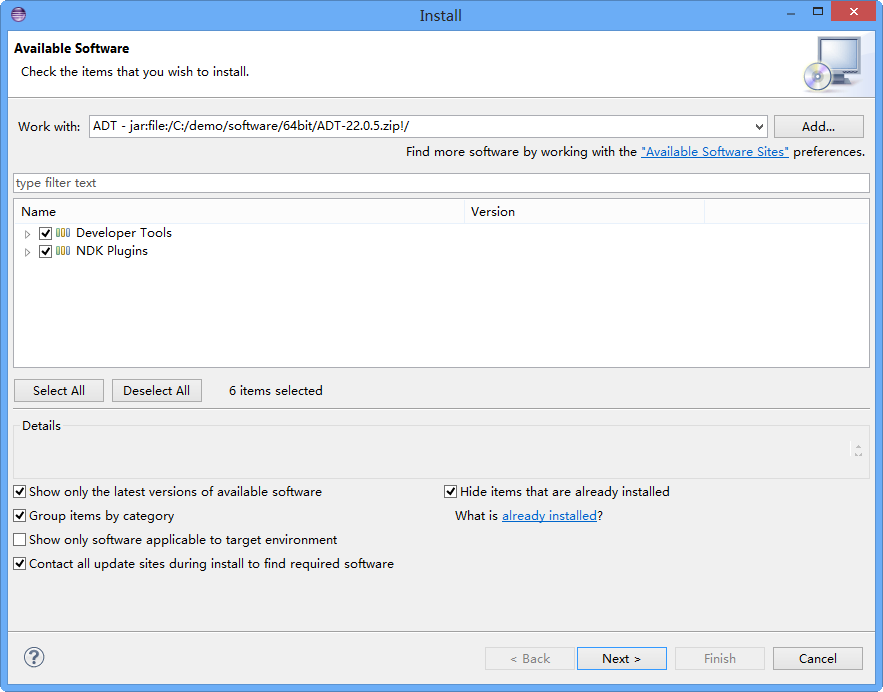
双击eclipse.exe，启动eclipse，设置工作目录，OK。



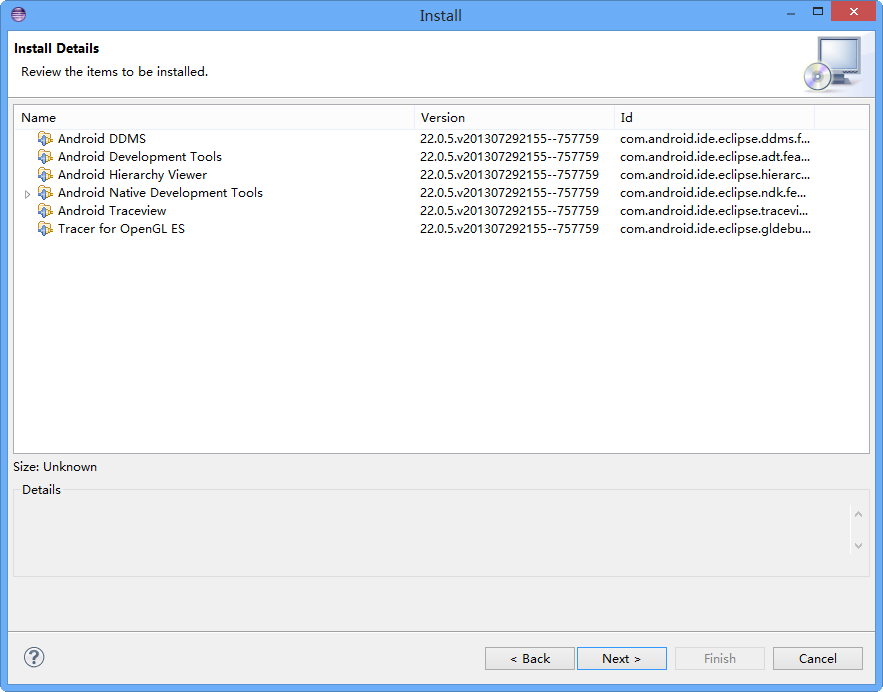
Help->Install New Software…->Add…->Archive…->选择ADT-22.0.5.zip->OK



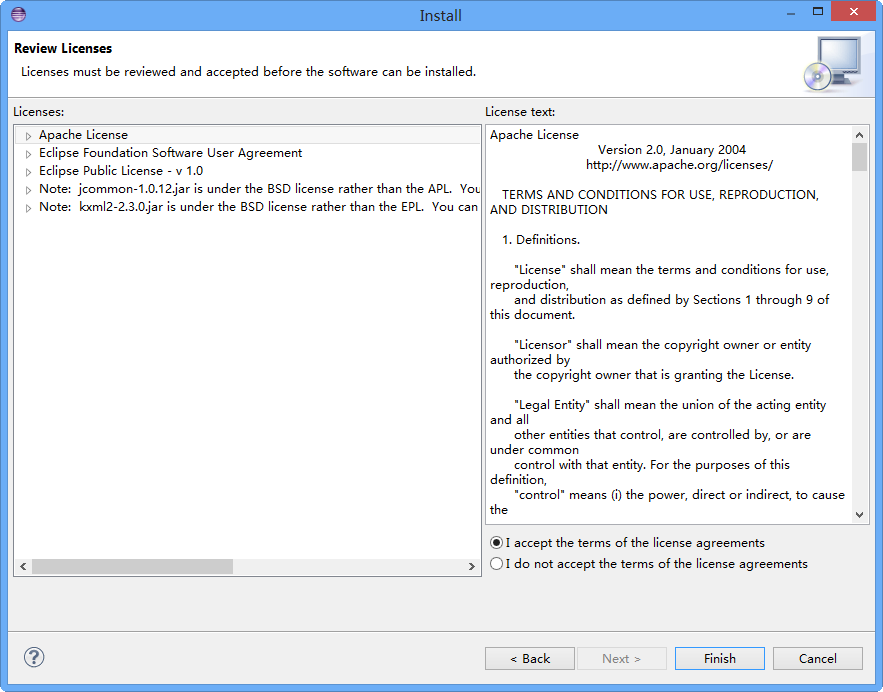
选择Developer Tools和NDK Plugins，Next



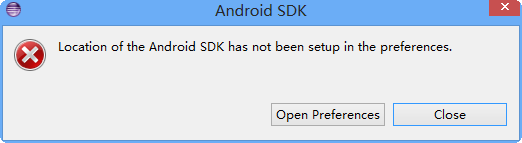
Next



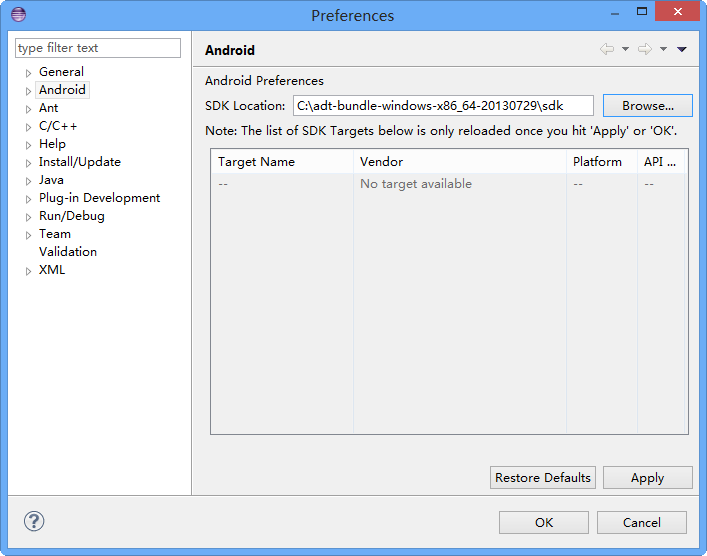
选择I accept…，Finish



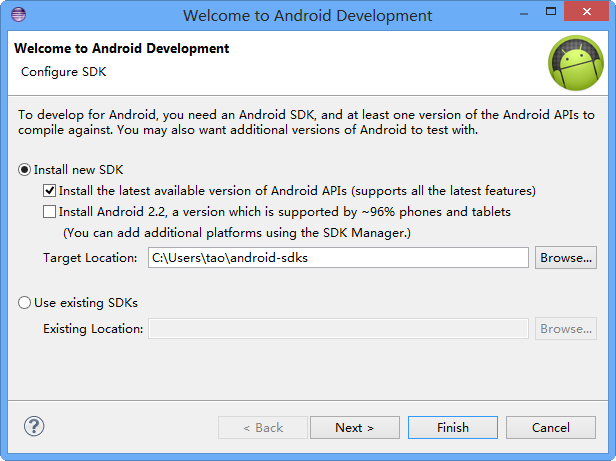
安装好ADT Plugin后，重启Eclipse，单击Open Preferences。



选择刚才adt-bundle-windows-xxx.zip的解压路径。Apply。OK。



选Use existing SDKs，选择adt-bundle-windows-xxx.zip的解压路径。

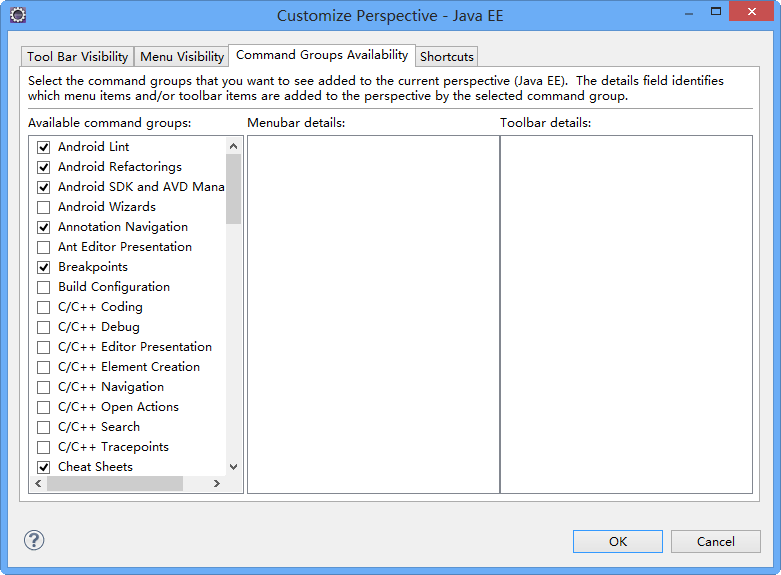


安装成功后，工具栏会出现，用于启动SDK Manager.exe。 用于配置模拟器。

**问题1：如果工具栏找不到**

Window->Customize Perspective->Command Groups Availability

选择Android Lint，Android Refactorings，Android SDK and AVD Manager



### 配置Tomcat

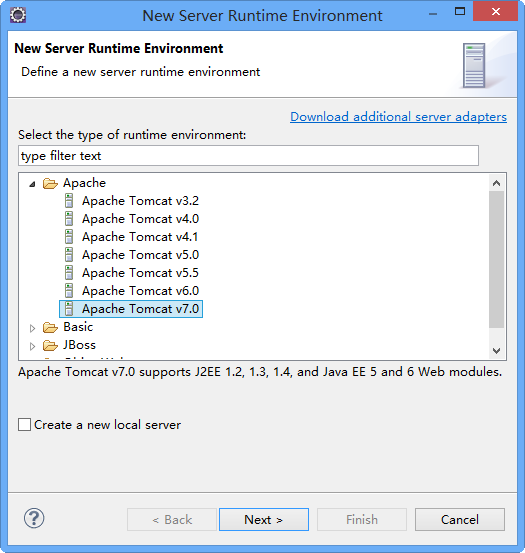
解压apache-tomcat-xxx-windows-xxx.zip到任意目录。

Window->Preferences->Server->Runtime Enviroment->Add

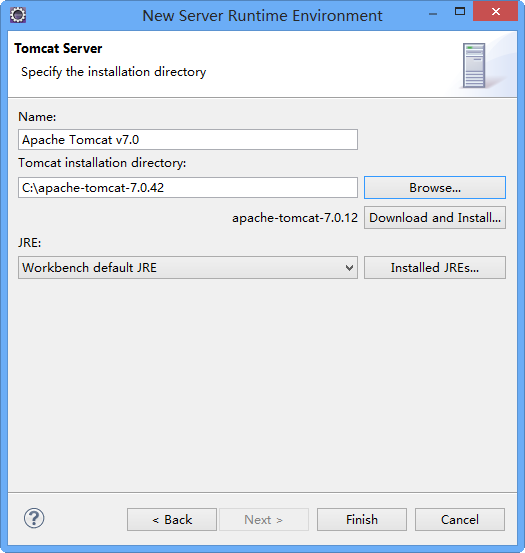
选择tomcat版本和安装路径



选择Apache Tomcat v7.0，Next



Browse…选择解压apache-tomcat-xxx-windows-xxx.zip的路径，Finish



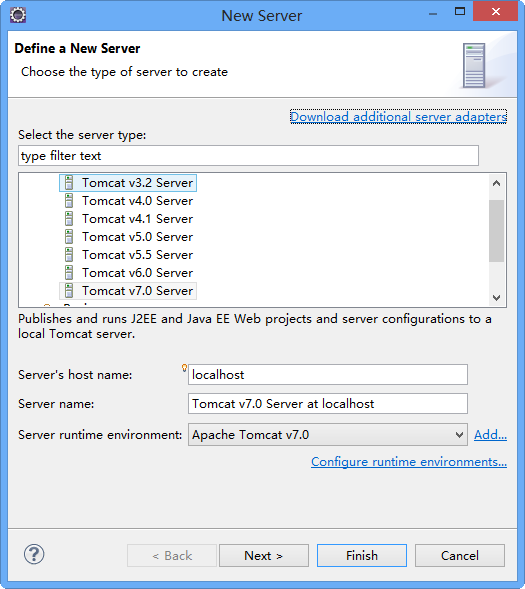
Window->Show View->Other打开Server视图



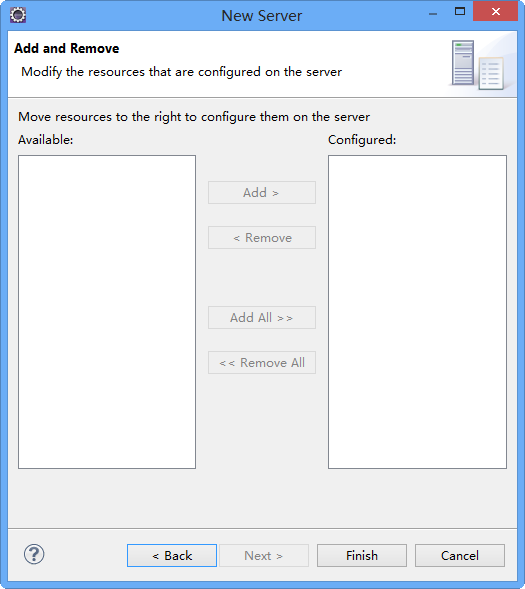
单击Click this link to create a new server

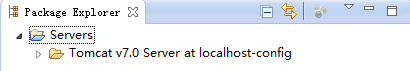


Next



Finish



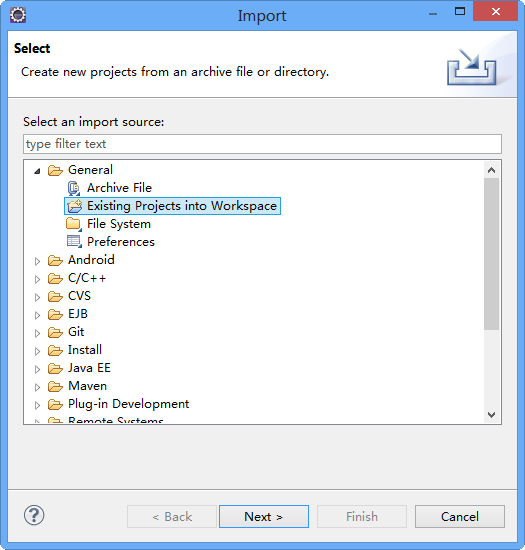


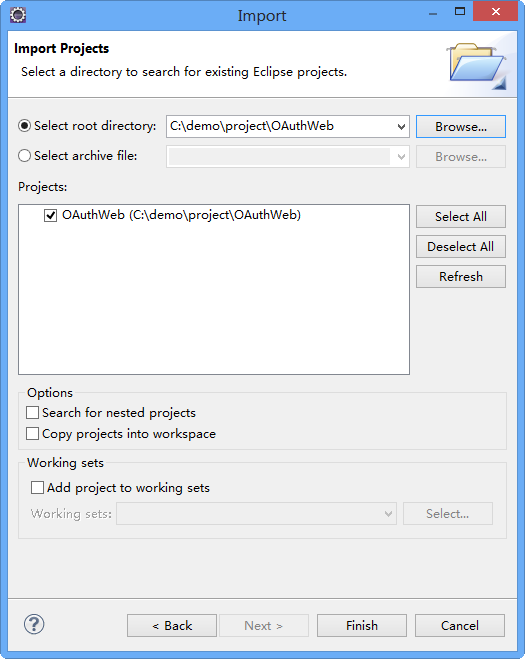
## project

project目录下为项目源码。

### 运行OAuthWeb

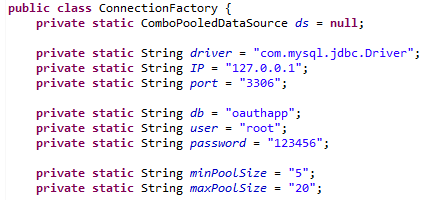
File->Import->Existing Projects into Workspace导入OAuthWeb



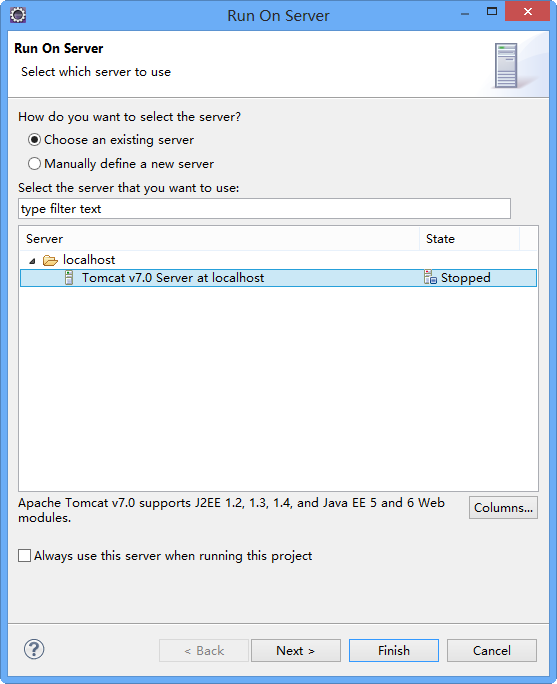


打开OAuthWeb中src/com.sinoparasoft.oauth.db/ConnectionFactory文件

根据数据库配置修改，IP，port，db，user，password的值。IP为安装数据库的IP（如是本机，则为127.0.0.1），port为MySQL的端口号（默认为3306），db为数据库名称（默认为oauthapp），user为数据库帐号（默认为root），password为安装MySQL时设置的密码。



项目右键Run As->Run On Server。

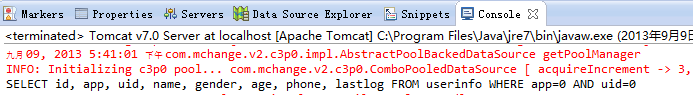


在浏览器中分别输入

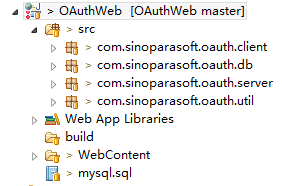
<http://localhost:8080/OAuthWeb/userinfo?app=0&uid=0>

<http://本机IP/OAuthWeb/userinfo?app=0&uid=0>

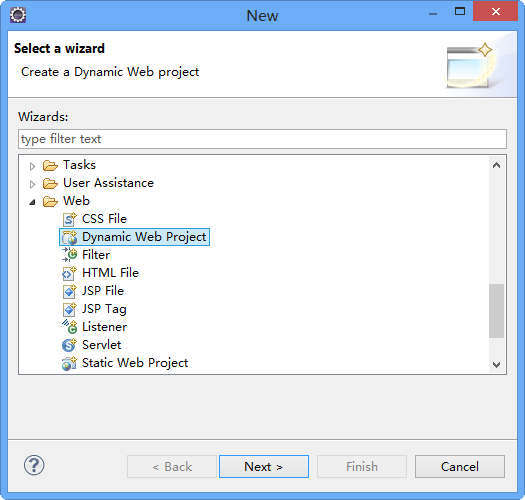
如果控制台出现SELECT id,uid,name,gender,age,phone,lastlog FROM userinfo WHERE app=0 AND uid=0，则配置成功。

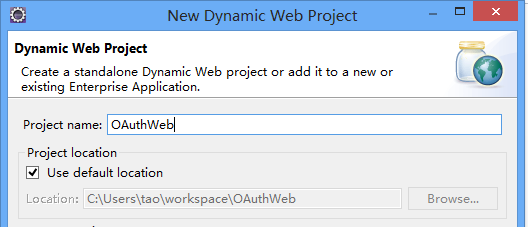


**问题1：如果导入项目后，项目上出现感叹号或叉，说明项目配置有问题。**



File->New->Other->Dynamic Web Project





将project/OAuthWeb中的WebContent，src，mysql.sql覆盖到Use default location的工程下，再右键refresh工程。

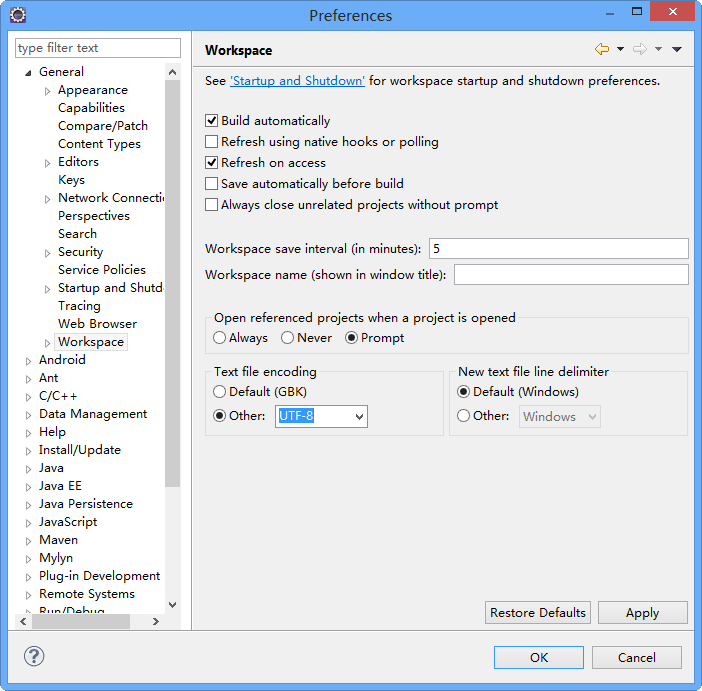
**问题2：localhost可以访问，127.0.0.1和本机IP无法访问**

<http://127.0.0.1:8080>访问tomcat，如果不是Tomcat主页，则表示其他程序占用8080端口。卸载该程序。

### 运行OAuthApp

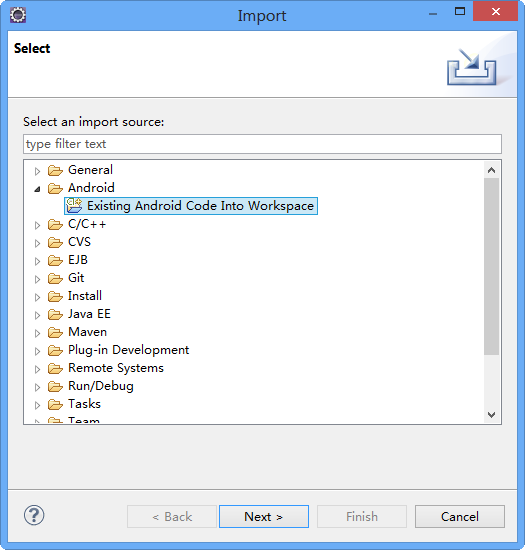
1. **导入工程前先设置编码方式**

Window->Preferences->General->Workspace->Text file encoding->Other->UTF-8，Apply，OK

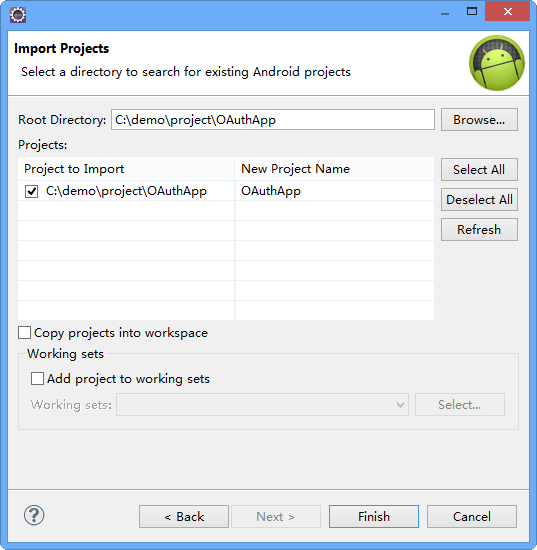


1. **导入OAuthApp**

File->Import->Existing Android Code Into Workspace。

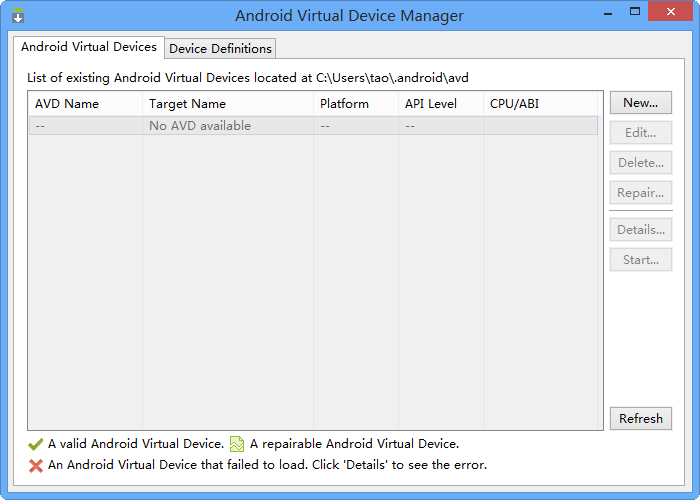


选择project/OAuthApp，Finish

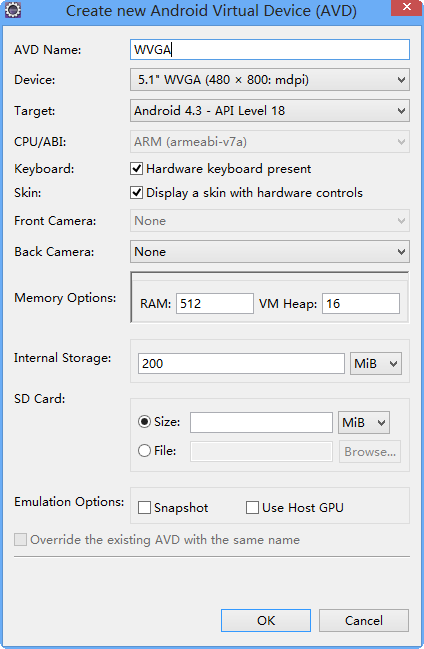


1. **配置模拟器**

单击工具栏的Android Virtual Device Manager，New…。



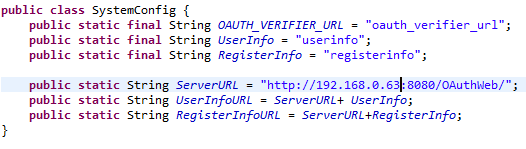
AVD Name：给模拟器取个名字，Device：选择硬件型号，Target：支持API等级。



1. **运行OAuthApp**

**注：运行OAuthApp前必须保证OAuthWeb正确运行。**

打开OAuthApp的src/com.sinoparasoft.oauth.util/SystemConfig文件，修改ServerURL为运行OAuthWeb的服务器IP地址（不能是localhost或127.0.0.1）。



项目上右键Run As->Android Application。



1. **增加测试用户**

**问题1：如果新浪提示“applications over the unaudited use restrictions”，或人人提示“无效的测试用户”**

由于应用未提交审核，尚处于测试阶段，必须由app key的申请者到开放平台，将用户增加到测试用户中。

打开开始中的，输入root密码，回车。







# 开发指南

OAuthWeb是服务器端的动态Web项目，**采用Restlet发布WebSerivce供客户端调用，**向MySQL读写用户和预约信息。

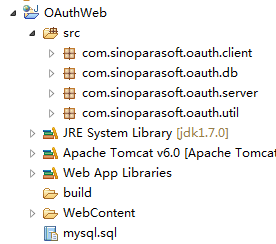
OAuthApp是客户端的Android项目，**采用Scribe实现OAuth授权**，**将Restlet提供的ClientResource封装成HttpClient访问服务器**，实现用户和预约信息的读取和保存。

OAuthExample：是Java普通项目，方便跟踪测试Scribe对各开发平台的授权过程。实验成功后可直接加入OAuthApp的com.sinoparasoft.oauth.scribe.ConfigUtil，让OAuthApp支持其他开放平台。

## 服务器开发

参考WebServer项目可快速入门，如何用Restlet发布WebService服务。

### OAuthWeb源码介绍



* db包含操作数据库的类
* client为客户端测试程序
* server为服务器端配置程序
* util为通用方法

具体请参考源码中的注释。

**Http请求**

（Get，Post…）

**Restlet发布WebService**

（UserResource->UserInfo）

（RegisterResource->RegisterInfo）

**MySQL**

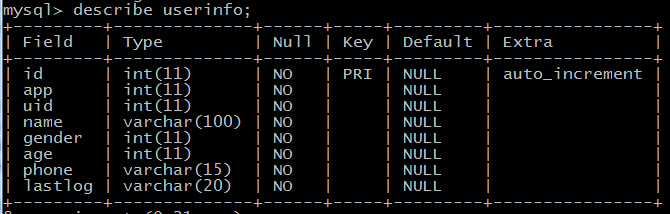
userinfo（用户信息）

registerinfo（预约信息）

### 数据库设计

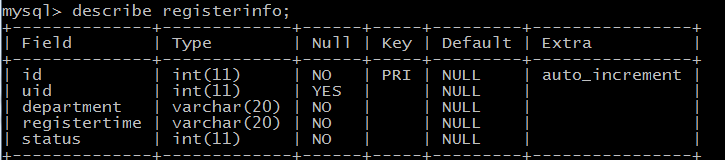
project/OAuthWeb/mysql.sql为建立数据库的脚本。该数据库设计比较简陋，应根据项目需求增加字段，且注意字段类型，例如uid应该为varchar。

1. **用户信息表**



|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 含义 |
| id | int | 用户ID，自增序列号，建议id的类型改为BIGINT |
| app | int | 开放平台ID，1：开心网，2：人人，3：新浪… |
| uid | int | 开放平台保存的用户ID，由于QQ的用户巨大，int会溢出，建议uid的类型改为varchar或bigint |
| name | varchar | 用户名 |
| gender | int | 性别，1为男，0为女 |
| age | int | 年龄 |
| phone | varchar | 电话 |
| lastlog | varchar | 上次登录时间 |

1. **预约信息表**

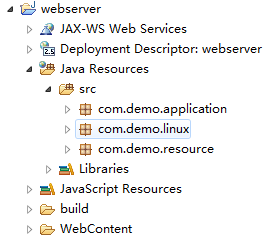


|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 含义 |
| id | int | 预约ID，自增序列号，建议id的类型改为BIGINT |
| uid | int | 用户ID，与userinfo表的id类型一致 |
| department | varchar | 科室 |
| registertime | varchar | 为简化采用varchar，建议考虑timestamp |
| status | int | 预约状态，0：预约，1：取消，2：结束… |

### Restlet发布WebService

#### 第一个WebService项目

File->Import->Existing Projects into Workspace->Browse->选择webserver路径->Finish



这是一个最简单的webserver项目，在WebContent/WEB-INF/lib/下包含org.restlet.jar（核心包）和org.restlet.ext.servlet.jar（提供Servlet服务）。

打开web.xml

org.restlet.application配置成相应的Application类。

<context-param>

<param-name>org.restlet.application</param-name>

<param-value>com.demo.application.WebApplication</param-value>

</context-param>

servlet配置成org.restlet.ext.servlet.jar提供的org.restlet.ext.servlet.ServerServlet类，并对所有url过滤

<servlet>

<servlet-name>restlet</servlet-name>

<servlet-class>org.restlet.ext.servlet.ServerServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>restlet</servlet-name>

<url-pattern>/\*</url-pattern>

</servlet-mapping>

WebApplication继承自Application，用于配置Restlet服务，对/demo路径下的请求，都由DemoResource响应。

public class WebApplication extends Application {

@Override

public Restlet createInboundRoot() {

Router router = new Router(getContext());

router.attach("/demo", DemoResource.class);

return router;

}

}

DemoResource继承自ServerResource类，以Get和Post标签分别相应Http的Get和Post请求，Get返回Hello，Post在Linux平台执行what命令。

public class DemoResource extends ServerResource {

@Get

public String sayHello() {

return "Hello! Your Web Service Works Good.\n";

}

@Post

public String sayWhat(String what) {

return Command.execute(what);

}

}

#### 入门教程

1. **客户端发送get请求，输出到console**

new ClientResource("http://restlet.org").get().write(System.out);

1. **服务器端启动在8182端口，对Get请求，返回hello,world。**[http://localhost:8182](http://localhost:8182/)

public class Part03 extends ServerResource {

      public static void main(String[] args) throws Exception {

        // Create the HTTP server and listen on port 8182

        new Server(Protocol.HTTP, 8182, Part03.class).start();

    }

    @Get

    public String toString() {

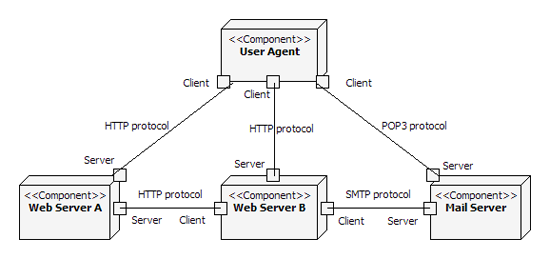
        return "hello, world";

    }

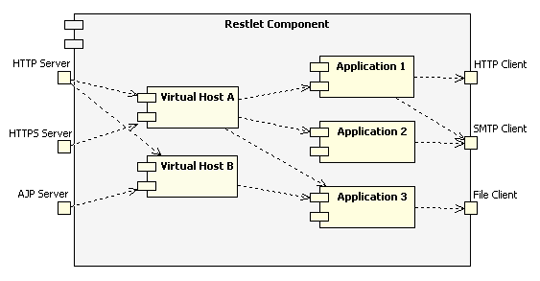
 }

1. **Restlet架构**

以Web Service B为例，即是User Agent的服务器端，又是Mail Server的客户端。直接采用不同协议，如HTTP，POP3，SMTP。



Component管理Virtual Hosts和Application。Virtual Hosts支持灵活配置，如同个IP共享多个域名，同个域名在多个IP间均匀分配负载。Application管理Resource和Representation。Application在多个Restlet实现和不同Virtual Hosts间可移植。



public class Part05 extends ServerResource {

public static void main(String[] args) throws Exception {

// Create a new Restlet component and add a HTTP server connector to it

Component component = new Component();

component.getServers().add(Protocol.HTTP, 8182);

// Then attach it to the local host

component.getDefaultHost().attach("/trace", Part05.class);

// Now, let's start the component!

// Note that the HTTP server connector is also automatically started.

component.start();

}

@Get

public String toString() {

// Print the requested URI path

return "Resource URI : " + getReference() + '\n' + "Root URI : "

+ getRootRef() + '\n' + "Routed part : "

+ getReference().getBaseRef() + '\n' + "Remaining part: "

+ getReference().getRemainingPart();

}

}

<http://localhost:8182/trace/abc/def?param=123>访问得到

Resource URI : http://localhost:8182/trace/abc/def?param=123

Root URI : http://localhost:8182/trace

Routed part : http://localhost:8182/trace

Remaining part: /abc/def?param=123

1. **提供静态文件**

public static final String ROOT\_URI = "file:///c:/restlet/docs/api/";

[...]

// Create a component

Component component = new Component();

component.getServers().add(Protocol.HTTP, 8182);

component.getClients().add(Protocol.FILE);

// Create an application

Application application = new Application() {

    @Override

    public Restlet createInboundRoot() {

            return new Directory(getContext(), ROOT\_URI);

    }

};

// Attach the application to the component and start it

component.getDefaultHost().attach(application);

component.start();

1. **URI重写和重定向**

Redirector允许重写URI，需要三个参数，第一个是parent context，第二个定义目标URI，第三个定义重定向类型，这里采用客户端重定向。从初始请求中提取kwd参数，复制到keywords，然后由Redirector配置成目标URI。

// Create a root router

Router router = new Router(getContext());

// Create a Redirector to Google search service

String target = "http://www.google.com/search?q=site:mysite.org+{keywords}";

Redirector redirector = new Redirector(getContext(), target,

        Redirector.MODE\_CLIENT\_TEMPORARY);

// While routing requests to the redirector, extract the "kwd" query

// parameter. For instance :

// http://localhost:8182/search?kwd=myKeyword1+myKeyword2

// will be routed to

// http://www.google.com/search?q=site:mysite.org+myKeyword1%20myKeyword2

Extractor extractor = new Extractor(getContext(), redirector);

extractor.extractQuery("keywords", "kwd", true);

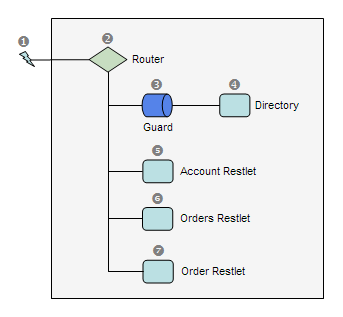
// Attach the extractor to the router

router.attach("/search", extractor);

1. **Routers和分层URI**

Redirector，Router，Finder，Filter可用于管理URI。

1. /docs/ to display static files
2. /users/{user} to display a user account
3. /users/{user}/orders to display the orders of a particular user
4. /users/{user}/orders/{order} to display a specific order



// Create a root router

Router router = new Router(getContext());

// Attach a guard to secure access to the directory

Guard guard = new Guard(getContext(), ChallengeScheme.HTTP\_BASIC,

        "Restlet tutorial");

guard.getSecrets().put("scott", "tiger".toCharArray());

router.attach("/docs/", guard);

// Create a directory able to expose a hierarchy of files

Directory directory = new Directory(getContext(), ROOT\_URI);

guard.setNext(directory);

// Create the account handler

Restlet account = new Restlet() {

    @Override

    public void handle(Request request, Response response) {

        // Print the requested URI path

        String message = "Account of user \""

                + request.getAttributes().get("user") + "\"";

        response.setEntity(message, MediaType.TEXT\_PLAIN);

    }

};

// Create the orders handler

Restlet orders = new Restlet(getContext()) {

    @Override

    public void handle(Request request, Response response) {

        // Print the user name of the requested orders

        String message = "Orders of user \""

                + request.getAttributes().get("user") + "\"";

        response.setEntity(message, MediaType.TEXT\_PLAIN);

    }

};

// Create the order handler

Restlet order = new Restlet(getContext()) {

    @Override

    public void handle(Request request, Response response) {

        // Print the user name of the requested orders

        String message = "Order \""

                + request.getAttributes().get("order")

                + "\" for user \""

                + request.getAttributes().get("user") + "\"";

        response.setEntity(message, MediaType.TEXT\_PLAIN);

    }

};

// Attach the handlers to the root router

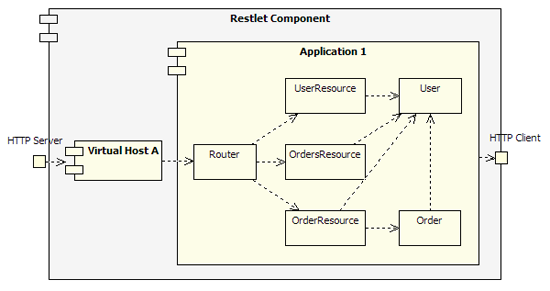
router.attach("/users/{user}", account);

router.attach("/users/{user}/orders", orders);

router.attach("/users/{user}/orders/{order}", order);

1. **查找对应的Resource**

Application通过attch()方法，将uri和ServerResource class绑定，生成Finder，当请求来到时，根据请求类型（如POST或Get等），解析出对应的ServerResource，



// Create a router

Router router = new Router(getContext());

 // Attach the resources to the router

router.attach("/users/{user}", UserResource.class);

router.attach("/users/{user}/orders", OrdersResource.class);

router.attach("/users/{user}/orders/{order}", OrderResource.class);

UserResource 需继承自ServerResource ，重写init()方法从/users/{user}提取出user属性，通过注释@Get让toString()方法支持Get请求。

public class UserResource extends ServerResource {

    String userName;

     Object user;

     @Override

    public void init() {

        this.userName = (String) getRequestAttributes().get("user");

        this.user = null; // Could be a lookup to a domain object.

    }

     @Get

    public String toString() {

        return "Account of user \"" + this.userName + "\"";

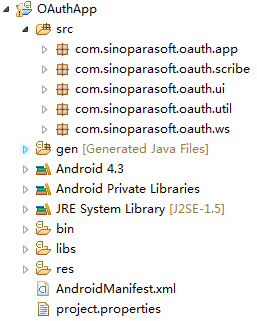
    }

}

## 客户端开发

参考OAuthExample的com包下的内容可快速入门，如何用Scribe实现OAuth授权。

### OAuthApp源码介绍



* ui包含Android界面，LoginActivity为启动界面，AuthActivity为授权界面。
* util包含通用方法
* ws包含与WebSerivce相关的类
* scribe包含利用Scribe实现OAuth授权的类，项目对scribe进行重新包装，增加ConfigUtil单例类，用于配置各平台的参数
* app包含从开放平台获取信息的实现类，主要参考AppInterface，AppFactory

具体请参考源码中的注释。

**LoginActivity**

ConfigUtil初始化

**AuthActivity**

页面跳转，用户授权

**MainActivity**

OAuth获取用户信息，调用WebService注册用户

**RegisterActivity**

调用WebService注册预约

### 运行OAuthExample

File->Import->Existing Projects into Workspace导入OAuthExample。



com是自己的测试程序，其他都是Scribe源码。

以Kaixin20Example为例，将apiKey和apiSecret改为注册获得的值后，右键该文件Run As->Java Application。

将Now go and authorize Scribe here:下面的url，输入浏览器，完成授权后，获取浏览器跳转URL，为回调地址加code格式

<http://localhost:8080/?code=6295f089a55f4cf9377f84cfd5458993>

将code的值粘贴到And paste the authorization code here下面，获取access token，最后获取PROTECTED\_RESOURCE\_URL返回的信息。

### Scribe实现OAuth授权

1. **配置scribe**

从<https://github.com/fernandezpablo85/scribe-java>下载Jar包

对Eclipse普通项目，将scribe-x.x.x.jar加入classpath。

对maven项目，在pom.xml中添加如下内容：

<dependency>

<groupId>org.scribe</groupId>

<artifactId>scribe</artifactId>

<version>1.3.2</version> // please use always the latest version

</dependency>

1. **调用ServiceBuilder生成OAuthService，提供DefaultApi20实现类，API\_KEY和API\_SECRETE**

OAuthService service = new ServiceBuilder().provider(TwitterApi.class).apiKey("your\_api\_key")

.apiSecret("your\_api\_secret").build();

如果要指定回调页面，则在build()之前调用callback("http://your\_callback\_url")。

**第4和第5步，oauth1和oauth2流程不同：**

**oauth1需首先获取request token，再利用request token换取access token，oauth2不需利用request token换取access token，而是直接调用方法，将需要request token的地方设为null即可**

1. **获取授权URL**

**oauth1流程：**

获得request token地址

Token requestToken = service.getRequestToken();

用Request token换取授权URL

String authUrl = service.getAuthorizationUrl(requestToken);

**oauth2流程：**

直接获得授权URL

String authUrl = service.getAuthorizationUrl(null);

1. **获取access token**

**oauth1流程：**

用户登录authUrl，完成授权后，返回code，将code作为参数赋给Verifier，通过requestToken和Verifier换取accessToken

Verifier v = new Verifier("verifier you got from the user");

Token accessToken = service.getAccessToken(requestToken, v);

**oauth2流程：**

通过verifer换取access token，不需要request token。

Verifier verifier = new Verifier(code);

Token accessToken = service.getAccessToken(null, verifier);

1. **利用access token获取信息**

生成OAuthRequest，需提供HTTP请求方式（GET或POST，有些网站支持GET方式，有些支持POST方式，请参看具体网站开放平台的API）和请求URL（参看具体开放平台提供的URL）

OAuthRequest request = new OAuthRequest(Verb.GET, "http://api.twitter.com/1/account/verify\_credentials.xml");

利用前面获取的access token完成注册

service.signRequest(accessToken, request); // the access token from step 4

发送HTTP请求

Response response = request.send();

打印返回的HTTP请求体，各开放平台一般以xml或json方式返回信息，需根据具体网站返回方式进行解析

System.out.println(response.getBody());

### 支持新的开放平台

scribe主要支持国外的开放平台，国内支持开心网，新浪微博，人人等。

具体支持哪些网站请参看下面的链接。

<https://github.com/fernandezpablo85/scribe-java/tree/master/src/test/java/org/scribe/examples>

如果要支持新的开放平台，需根据OAuth版本继承自DefaultApi10a或DefaultApi20。

1. **oauth1实现**

DefaultApi10a是oauth1的默认父类，具体需实现三个接口：

* getRequestTokenEndpoint返回获取request token的URL
* getAccessTokenEndpoint返回获取access token的URL
* getAuthorizationUrl返回授权URL

public abstract class DefaultApi10a implements Api{

……

public abstract String getRequestTokenEndpoint();

public abstract String getAccessTokenEndpoint();

public abstract String getAuthorizationUrl(Token requestToken);

}

以sohu微博为例：

public class SohuWeiboApi extends DefaultApi10a

{

private static final String REQUEST\_TOKEN\_URL = "http://api.t.sohu.com/oauth/request\_token";

private static final String ACCESS\_TOKEN\_URL = "http://api.t.sohu.com/oauth/access\_token";

private static final String AUTHORIZE\_URL = "http://api.t.sohu.com/oauth/authorize?oauth\_token=%s";

@Override

public String getRequestTokenEndpoint()

{

return REQUEST\_TOKEN\_URL;

}

@Override

public String getAccessTokenEndpoint()

{

return ACCESS\_TOKEN\_URL;

}

@Override

public String getAuthorizationUrl(Token requestToken)

{

return String.format(AUTHORIZE\_URL, requestToken.getToken());

}

}

1. **oauth2实现**

DefaultApi20是oauth2的默认父类，需实现2个接口：

* getAccessTokenEndpoint返回获取access token的URL
* getAuthorizationUrl返回授权URL

public abstract class DefaultApi20 implements Api

{ ……

public abstract String getAccessTokenEndpoint();

public abstract String getAuthorizationUrl(OAuthConfig config);

}

以开心网为例：

public class KaixinApi20 extends DefaultApi20

{

private static final String AUTHORIZE\_URL = "http://api.kaixin001.com/oauth2/authorize?client\_id=%s&redirect\_uri=%s&response\_type=code";

private static final String SCOPED\_AUTHORIZE\_URL = AUTHORIZE\_URL + "&scope=%s";

@Override

public AccessTokenExtractor getAccessTokenExtractor()

{

return new JsonTokenExtractor();

}

@Override

public String getAccessTokenEndpoint()

{

return "https://api.kaixin001.com/oauth2/access\_token?grant\_type=authorization\_code";

}

@Override

public String getAuthorizationUrl(OAuthConfig config)

{

// Append scope if present

if (config.hasScope())

{

return String.format(SCOPED\_AUTHORIZE\_URL, config.getApiKey(), OAuthEncoder.encode(config.getCallback()), OAuthEncoder.encode(config.getScope()));

}

else

{

return String.format(AUTHORIZE\_URL, config.getApiKey(), OAuthEncoder.encode(config.getCallback()));

}

}

}

**国内目前主流网站都已采用oauth2，虽然仍提供oauth1，但会逐渐淘汰oauth1。因此如果网站支持oauth2，则首选oauth2。**

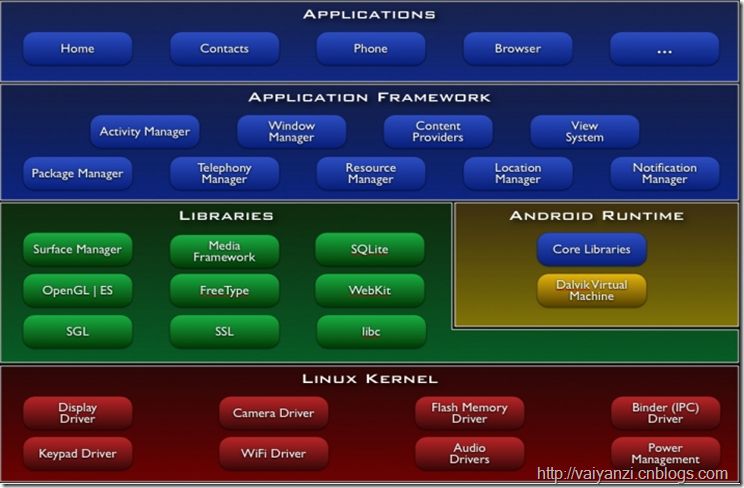
## 注意事项

由于只是Demo，因此还有如下未完善的地方：

1. 数据库设计未完善。
2. 当服务器挂掉时，OAuthApp没有提供超时连接功能。
3. WebService未提供安全认证功能。
4. 客户端与服务器端的通信如何保证，丢包。

可参考：<http://blog.csdn.net/is_zhoufeng/article/category/1540901>

# Android开发



Android系统分为5部分：应用程序层、应用程序框架、函数库、Android运行时、Linux内核。

Android运行时分为Android核心库集和Dalvik虚拟机。Dalvik虚拟机运行专有的.dex文件，基于寄存器实现。

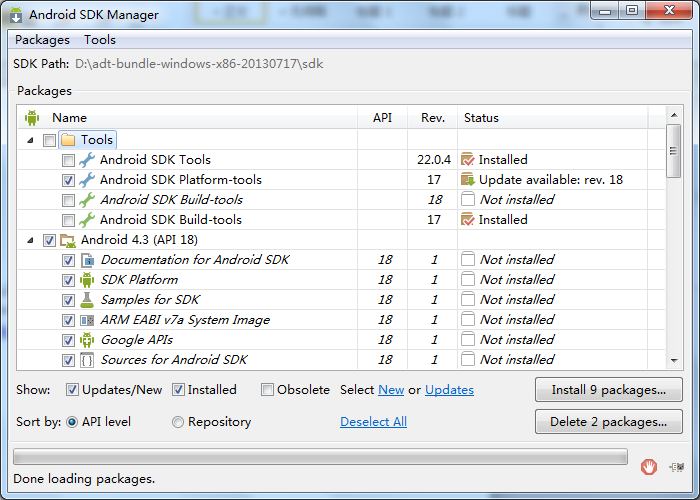
## 搭建Android开发环境

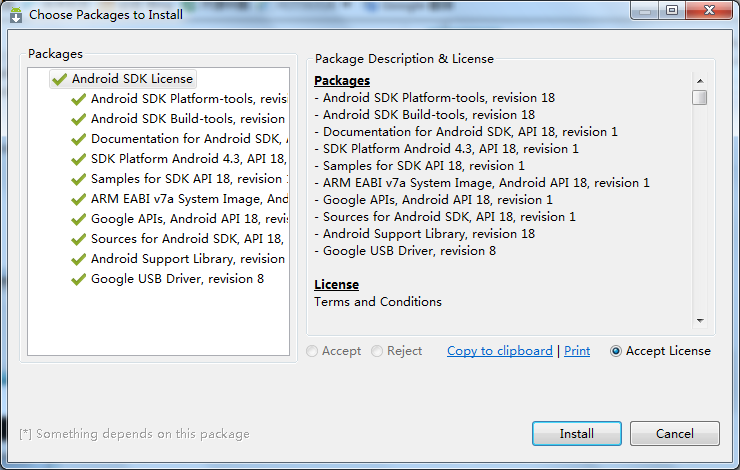
首先要下载并安装Android SDK，然后在eclipse中添加ADT插件，并配置安装好的SDK路径。

### 安装Android SDK

<http://developer.android.com/sdk/index.html>

下载并解压adt-bundle-windows-x86-20130717.zip，双击SDK Manager.exe，选择需要安装的API。Accept License，Install。





### eclipse安装ADT

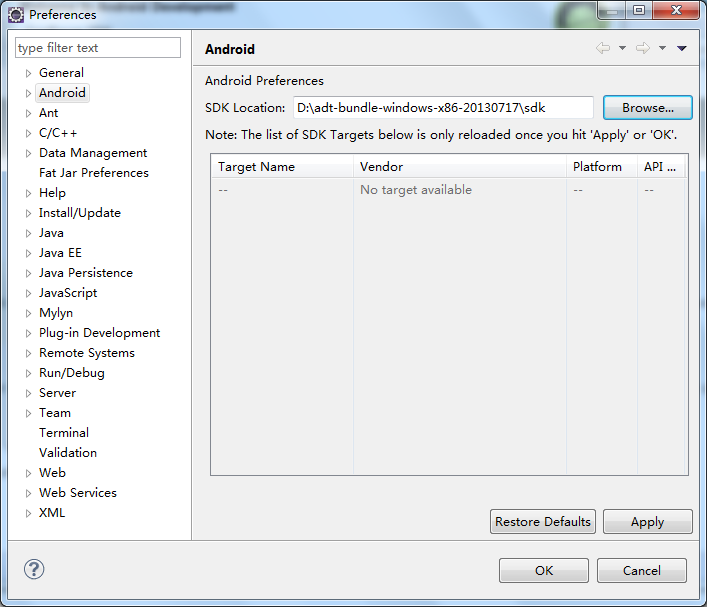
安装文档：<http://developer.android.com/sdk/installing/installing-adt.html>

Help > Install New Software->Add->添加库ADT Plugin，

<https://dl-ssl.google.com/android/eclipse/>（如果出错，则将https改为http），OK。然后选择需安装的插件，install。重启eclipse。

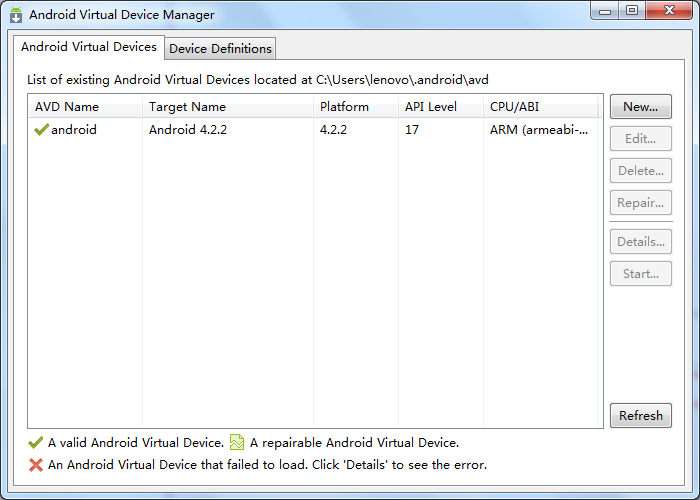
安装成功后，工具栏会出现，用于启动SDK Manager.exe。

设置Android SDK的安装路径。

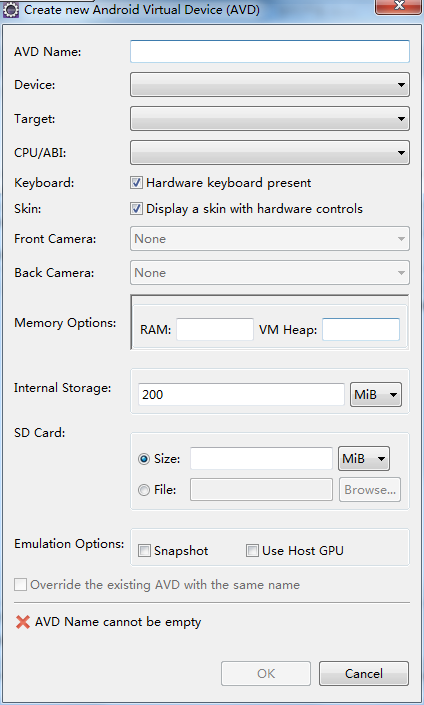


### 新建AVD

单击，新建AVD(Android Virtual Device)虚拟手机。



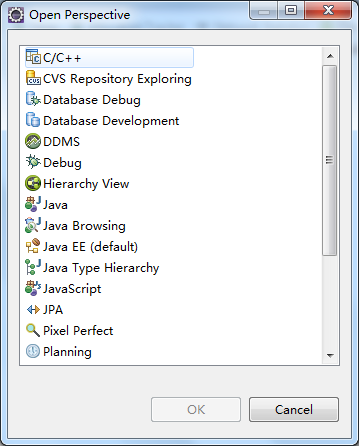
单击New…，设置AVD Name，选择设备类型，设置Memory Options内存大小，OK。



选中该设备，单击Start，启动AVD。

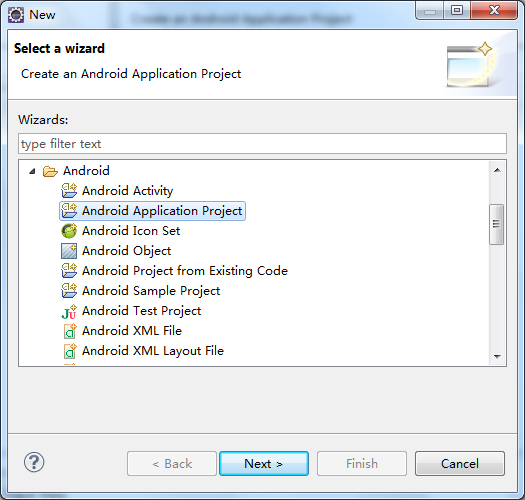
打开调试页面

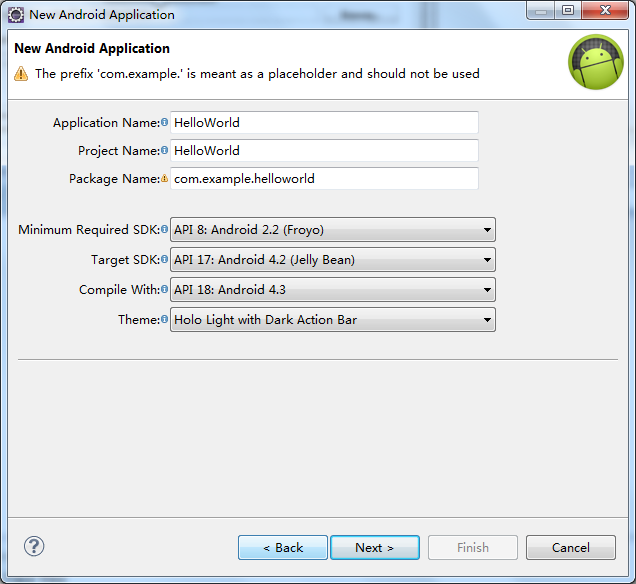
Window->Open Perspective->Other->DDMS

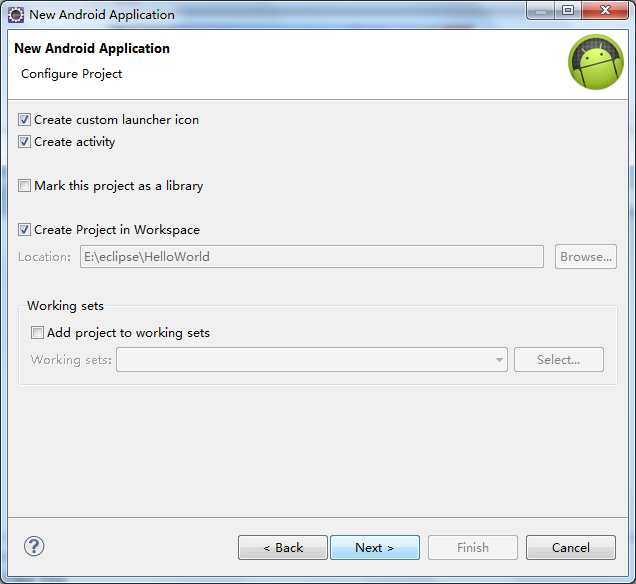


### 新建项目

File->New->Others，选Android Application Project。







运行Run->Android Application

## 问题

**问题1：AVD启动报错：android unfortunately UI has stopped**

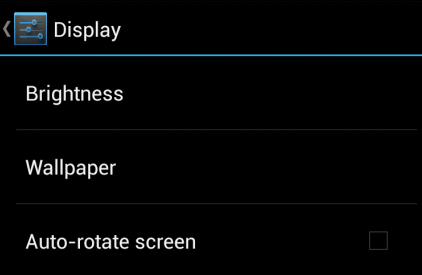
C:\Users\lenovo\.android\avd\Nexus.avd

C:\Users\lenovo\.android\avd\nexus.avd\hardware-qemu.ini

hw.mainKeys=**yes**

**问题2：AVD启动过慢**

Setting->Display->Auto-rotate screen去掉勾

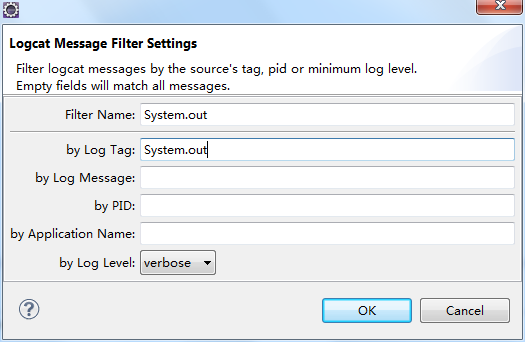


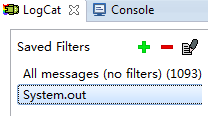
**问题3：Activity not started, its current task has been brought to the front**

重启模拟器

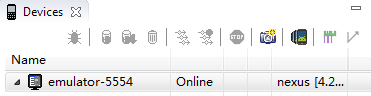
**问题4：Android调试，System.out.println不输出**

用LogCat查看AVD输出，打开DDMS，单击加号，在Filter Name和Log Tag输入System.out，也可以设置Application Name为OAuthApp



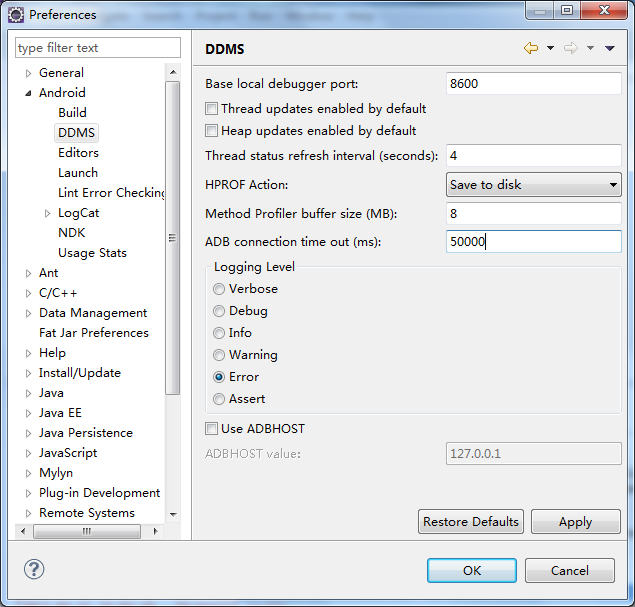


单击Devices的emulator-5554，显示该AVD的输出。



**问题6：Failed to install .apk on device 'emulator-5554': timeout**

Window->Preferences->DDMS的5000改为50000



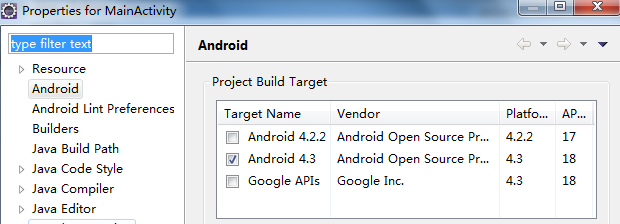
**问题7：调节AVD屏幕大小**

打开Android Virtual Device Manager对话框。选中AVD，单击start，修改Screen Size。



**问题8：导入android项目后没有引入Android Library**

项目右键->Properties->单击Android->选中Android 4.3



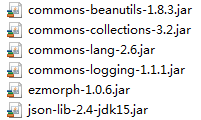
**问题9：java.lang.NoClassDefFoundError: com.mchange.v2.c3p0.ComboPooledDataSource**

Android不支持直接读写数据库，必须通过调用web service服务

**问题10：Unable to execute dex: Multiple dex files define**

Android Dependencies有重复引入的.jar包，解决的方法是在libs删除重复的jar即可

**问题11：使用json，报ClassNotFound错误**



**问题12：网络连接错误**

在Activity中直接用HTTP连接网络，再加上这句

<uses-sdk android:minSdkVersion=*"12"* android:targetSdkVersion=*"18"*/>会导致网络无法HTTP连接。

解决方法：

[Adding the uses-sdk fields in Androids manifest file causes crash](http://stackoverflow.com/questions/12712630/adding-the-uses-sdk-fields-in-androids-manifest-file-causes-crash)

The error is NetworkOnMainThreadException Starting with ICS, you cannot do any network access on main thread. You should probably use AsyncTask where you do network access

主线程中使用网络连接会导致NetworkOnMainThreadException，因此将网络连接放到AsyncTask中实现。

# OAuth客户端

## 获取app id和app secret

以新浪为例：

<http://open.weibo.com/wiki/%E7%A7%BB%E5%8A%A8%E5%BA%94%E7%94%A8>

<http://open.weibo.com> 新浪微博帐号登录

进入管理中心，单击创建应用，填写应用信息。



单击管理应用，查看基本信息中的App Key和App Secret，编辑高级信息中的授权回调页。





## 流程

### OAuth1流程（已过时）

<http://open.weibo.com/wiki/OAuth>

1. 注册应用后获得App Key和App Secret。客户端用App Key和App Secret向新浪微博发起请求，新浪微博经过验证后返回Reuqest Token和Request Secret。
2. 客户端用Reuqest Token和Request Secret作为参数引导用户跳转至新浪微博的授权页面，用户输入帐号密码，微博根据回调地址返回oauth\_verifier。
3. 客户端用Reuqest Token、Request Secret、oauth\_verifier向微博发请求，微博返回user\_id、Access Token和Access Secret给客户端。客户端用其代替用户的新浪帐号和密码，调用微博开放接口。

### OAuth2流程

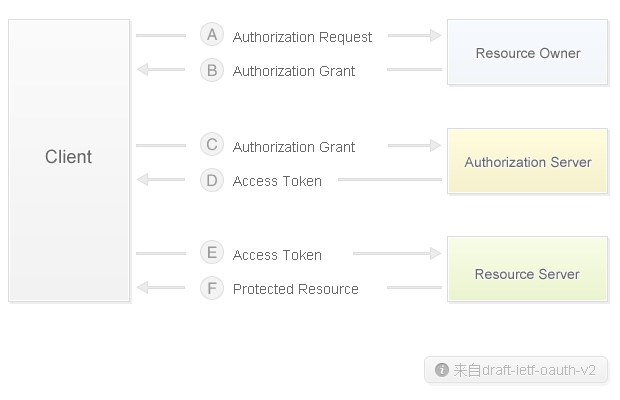


图1: 抽象的协议流程

(A)客户端从资源拥有者那里请求授权。授权请求能够直接发送给资源拥有者，或者间接地通过授权服务器这样的中介，而后者更为可取。

(B) 客户端收到一个访问许可，它代表由资源服务器提供的授权。

(C) 客户端使用它自己的私有证书到授权服务器上验证，并出示访问许可，来请求一个访问令牌。

(D) 授权服务器验证客户端私有证书和访问许可的有效性，如果验证通过则分发一个访问令牌。

(E) 客户端通过出示访问令牌向资源服务器请求受保护资源。

(F) 资源服务器验证访问令牌的有效性，如果验证通过则响应这个资源请求。

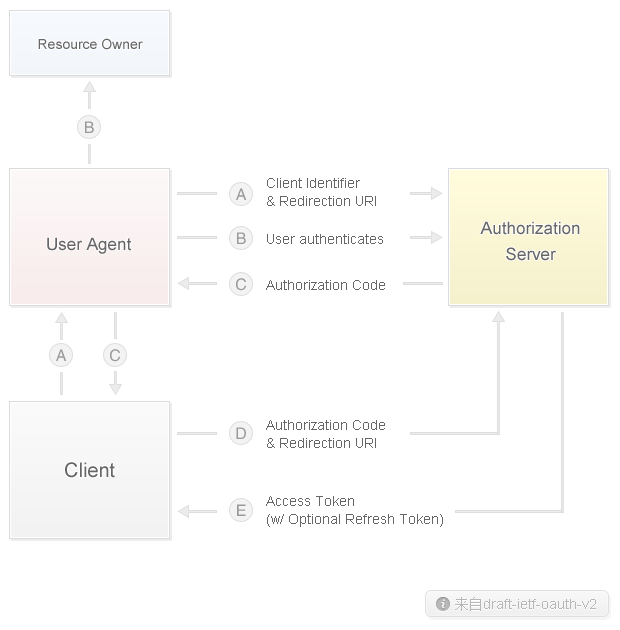


图2: 获取授权码

1. 引导用户到授权页面。

https://api.weibo.com/oauth2/authorize?client\_id=YOUR\_CLIENT\_ID&response\_type=code&redirect\_uri=YOUR\_REGISTERED\_REDIRECT\_URI

1. 用户输入帐号密码，同意授权，页面跳转至 YOUR\_REGISTERED\_REDIRECT\_URI/?code=CODE
2. 用CODE换取Access Token。

https://api.weibo.com/oauth2/access\_token?client\_id=YOUR\_CLIENT\_ID&client\_secret=YOUR\_CLIENT\_SECRET&grant\_type=authorization\_code&redirect\_uri=YOUR\_REGISTERED\_REDIRECT\_URI&code=CODE

返回值

{ "access\_token":"SlAV32hkKG", "remind\_in ":3600, "expires\_in":3600 }

1. 使用获得的OAuth2.0 Access Token调用API

### OAuth1.0 vs OAuth2.0

OAuth2相对OAuth1.0流程进行简化，去掉了RequestToken的过程。

* 流程不同。OAuth1.0的Request\_Token换Authorization\_Code，Authorization\_Code再换Access\_Token的授权模式；OAuth2.0的 Authorization\_Code直接换Access\_Token模式；
* 返回值不同。1.0里包含3个字段: UserID, AccessToken, AccessTokenSecret。2.0里也包含3个字段: AccessToken，ExpiresIn(AccessToken的过期时间，按秒计，很短，默认可能是1个小时)，RefreshToken (AccessToken过期时，用来获取新的AccessToken)；
* 格式不同。所有API只返回Json格式，没有XML格式。

## 技术选择

<http://blog.bizmate.com/?p=140>这篇文章很好

* 使用各开放平台提供的sdk。各开放平台都提供了对应的sdk，大部分sdk都支持java语言，使用这些sdk能够很快地实现功能，但是使用了这些具体的网站sdk之后，容易导致程序与这些具体的接口耦合，不利于网站代码复用和扩展。
* 网上很多使用Signpost，只支持OAuth1，目前主流网站都已停止OAuth1，而采用OAuth2，作者也在google groups上说明由于OAuth2相对OAuth1协议改动较大，暂未实现。<https://groups.google.com/forum/#!topic/signpost-users/h79celW908w>
* scribe-java是一个java语言的oauth库。优点是代码结构良好且使用方便，支持的第三方登录平台包括：Google、Facebook、Yahoo、LinkedIn、 Twitter、Foursquare、Evernote、Vimeo、Yammer、Windows Live。缺点是国内平台支持较少，目前仅支持开心、新浪、搜狐、人人。需实现如腾讯等的认证和各种信息的获取。

对比总结：

|  |  |  |
| --- | --- | --- |
| 技术 | 优点 | 缺点 |
| SDK | 各开放平台都提供了Android SDK和Demo，能快速实现认证和信息获取功能 | 系统必须集成各平台的SDK，根据不同要求调用不同API |
| Scribe-java | 代码结构良好且使用方便，能统一各平台的认证过程 | 国内平台支持较少，目前仅支持开心、新浪、搜狐、人人。需实现如腾讯等平台的认证和信息获取过程。 |

## 问题

**问题1：oauthcommunicationexception communication with the service provider failed**

Mainfest.xml中的设置版本号导致出错，去掉uses-sdk

<uses-sdk android:minSdkVersion="12" android:targetSdkVersion="18"/>

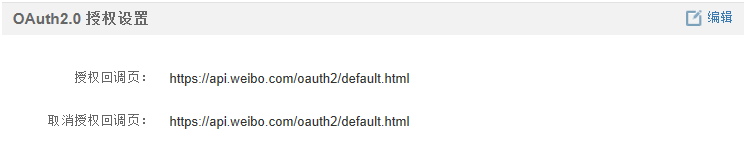
<http://blog.segmac.com/post/android-manifest-uses-sdk-element>

**问题2：consumer\_key\_refused**

新浪已禁止调用V1接口，在调用V1接口时会返回”40109 Oauth Error: consumer\_key\_refused!” 。采用V2接口。

**问题3：新浪微博 oauth2.0 redirect\_uri\_mismatch**

程序中的回调地址必须与“应用信息”-“高级信息”中的授权回调页一致。



**问题4：授权页面过大，显示不全**

在授权URL的后面添加&display=mobile

**问题5：applications over the unaudited use restrictions**

测试帐号中增加测试帐号，就是自己的新浪帐号，半小时后生效

## 密码

### 开心

App Key：2366683097160e568ef45d61168bfd7f

App Secret：c89b3baf9cb507297fdeaf5138af04d5

CallBackURL：<http://localhost:8080>（貌似随便设什么都可以）

### 新浪

App Key：381189709

App Secret：76f5da26e9b19f94c3575c5c53ba26f4

CallBackURL： https://api.weibo.com/oauth2/default.html

### 人人

API KEY：bc2b06565eb145b09597255afeff5f01

Secret Key：432168db355840298f464b218534872d

CallBackURL：<http://graph.renren.com/oauth/login_success.html>

### 搜狐

Consumer Key：pMK6mqIKYsE57AbGbK7w

Consumer secret：FuuSBhi6v5nWjlMIfnYT4dfaU3fsQeeqIGFSb7Yl

### 腾讯

腾讯微博

无线应用

App Key：801394503

App Secret：ed7febe2ac81c286e82098502c37daf6

CallBackURL：[www.sina.com](http://www.sina.com)（不小心设错了）

网站应用

App Key:801395342

App Secret:bbe6ac26a0814923dd331d85f296d728

CallBackURL：www.qq.com

腾讯社区

APP ID：100496179

APP KEY：28a82879e5490ac05f3398f7a70a4255

# OAuth服务器

Authorization Server是授权服务器

Resource Server是API服务器。

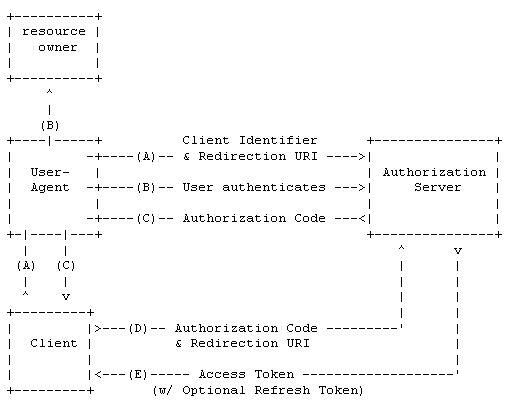
## 4种获取Access Token的方式

|  |  |  |
| --- | --- | --- |
|  | 服务端流程 | 客户端流程 |
| 适用场景 | 有Web Server | 无Web Server |
| 是否生成Authorization Code | 需要进行两步：  1. 获取Authorization Code；  2. 通过Authorization Code获取Access Token | 只需要一步：构造授权地址，即可获取Access\_Token  不生成Authorization Code，直接返回Access Token |
| Access Token返回方式 | 以Json文本返回  String code = uri.getQueryParameter("code"); | 在URI Fragment中返回，  1. 可通过js方法：window.location.hash来获取URL中#后的参数值。 2. 建议用js设置cookie存储token。 |

### 服务端流程Authorization Code Flow

此流程适用于在Web服务端调用REST API的的应用。例如：网站，站内应用。

采用Authorization Code获取Access Token的授权验证流程又被称为Web Server Flow，适用于所有有Server端的应用，如Web/Wap站点、有Server端的手机/桌面客户端应用等。其流程示意图如下，其流程由获取Authorization Code和通过Authorization Code获取Access Token这2步组成。

[](http://wiki.open.kaixin001.com/lib/exe/detail.php?id=%E4%BD%BF%E7%94%A8Authorization_Code%E8%8E%B7%E5%8F%96Access_Token&media=authorization.png)

#### 获取Authorization Code

**1. 请求数据包格式**

其获取方式是通过重定向用户浏览器（或手机/桌面应用中的浏览器组件）到 “<http://api.kaixin001.com/oauth2/authorize>”地址上，并带上以下参数：

|  |  |  |
| --- | --- | --- |
| 参数名 | 必选 | 介绍 |
| client\_id | true | 申请组件时获得的API Key |
| response\_type | true | 此值固定为“code” |
| redirect\_uri | true | 授权后要回调的URI，即接收Authorization Code的URI。 对于无Web Server的应用，其值可以是“oob”，此时用户同意授权后， 授权服务会将Authorization Code直接显示在响应页面的页面中及页面 title中。 |
| scope | false | 以空格分隔的权限列表，若不传递此参数，代表请求默认的basic权限。 如需调用扩展权限，必需传递此参数，详细请参考[权限列表](http://wiki.open.kaixin001.com/index.php?id=%E6%8E%A5%E5%8F%A3%E8%AE%BF%E9%97%AE%E6%9D%83%E9%99%90%E8%AF%B4%E6%98%8E#权限列表) |
| state | false | 用于保持请求和回调的状态，防止第三方应用受到CSRF攻击。授权服务器在回调时（重定向用户浏览器到 “redirect\_uri”时），会原样回传该参数（如果state是0或false，接口会认为没有传递state参数）。 请务必遵守流程传入state并进行合法性检查。 |
| display | false | 登录和授权页面的展现样式，默认为“page”。手机访问时，此参数无效 |
| oauth\_client | false | 是否为手机访问。手机访问：oauth\_client=1；不是手机，无需此参数 |
| Forcelogin（开心）  x\_renew（人人） | false | 是否强制登录。授权过程中需要验证用户的身份，支持Cookie验证。已经使用OAuth2.0登录的浏览器，不会再次出现登陆页面。强制登录：forcelogin=1；不强制登录，无需此参数 |

例如：“client\_id”为“Va5yQRHlA4Fq4eR3LT0vuXV4”的应用要请求某个用户的默认权限，并在授权后需跳转到“<http://www.example.com/oauth_redirect>”，同时希望在弹出窗口中展现用户登录、授权界面，则应用需要重定向用户的浏览器到如下URL：

http://api.kaixin001.com/oauth2/authorize?

response\_type=code&

client\_id=Va5yQRHlA4Fq4eR3LT0vuXV4&

redirect\_uri=http%3A%2F%2Fwww.example.com%2Foauth\_redirect&

scope=basic&

display=popup

**2. 响应数据包格式**

此时授权服务会根据应用传递参数的不同，为用户展现不同的授权页面。如果用户在此页面同意授权，授权服务则将重定向用户浏览器到应用所指定的“redirect\_uri”，并附带上表示授权服务所分配的Authorization Code的code参数，以及state参数（如果请求authorization code时带了这个参数）。

例如：继续上面的例子，假设授权服务在用户同意授权后生成的Authorization Code为“ANXxSNjwQDugOnqeikRMu2bKaXCdlLxn”，则授权服务将会返回如下响应包以重定向用户浏览器到“<http://www.example.com/oauth_redirect>”地址上：

HTTP/1.1 200 OK

Location: http://www.example.com/oauth\_redirect?code=ANXxSNjwQDugOnqeikRMu2bKaXCdlLxn

注意：

① “code”参数可以在“redirect\_uri”对应的应用后端程序中获取。

② 每一个Authorization Code的有效期为10分钟，并且只能使用一次，再次使用将无效。

#### 通过Authorization Code获取Access Token

**1. 请求数据包格式**

通过上面第一步获得Authorization Code后，便可以用其换取一个Access Token。获取方式是，应用在其服务端程序中发送请求（推荐使用POST）到 开心网OAuth2.0授权服务的“<https://api.kaixin001.com/oauth2/access_token>”地址上，并带上以下5个必须参数：

|  |  |  |
| --- | --- | --- |
| 参数名 | 必选 | 介绍 |
| grant\_type | true | 此值固定为“authorization\_code” |
| code | true | 通过上面第一步所获得的Authorization Code |
| client\_id | true | 应用的API Key |
| client\_secret | true | 应用的Secret Key |
| redirect\_uri | true | 该值必须与获取Authorization Code时传递的“redirect\_uri”保持一致 |

例如：

https://api.kaixin001.com/oauth2/access\_token?

grant\_type=authorization\_code&

code=ANXxSNjwQDugOnqeikRMu2bKaXCdlLxn&

client\_id=Va5yQRHlA4Fq4eR3LT0vuXV4&

client\_secret=0rDSjzQ20XUj5itV7WRtznPQSzr5pVw2&

redirect\_uri=http%3A%2F%2Fwww.example.com%2Foauth\_redirect

2. 响应数据包格式

若参数无误，服务器将返回一段JSON文本，包含以下参数：

|  |  |  |
| --- | --- | --- |
| 参数名 | 介绍 | 类型 |
| access\_token | 要获取的Access Token | string |
| expires\_in | Access Token的有效期，以秒为单位 | string |
| refresh\_token | 用于刷新Access Token 的 Refresh Token | string |
| scope | Access Token最终的访问范围，即用户实际授予的权限列表（用户在授权页面时，有可能会取消掉某些请求的权限） | string |

例如：

HTTP/1.1 200 OK

Content-Type: application/json

Cache-Control: no-store

{

"access\_token": "12932845\_de81cfe747ad7e06f2a904e73646f04c",

"expires\_in": 2592000,

"refresh\_token": "12932845\_f139bd04b513c4abb880e48f21cee542",

"scope": "basic",

}

若请求错误，服务器将返回一段JSON文本，包含以下参数：

例如：

HTTP/1.1 400 Bad Request

Content-Type: application/json

Cache-Control: no-store

{

"error\_code":"400",

"request":"\/oauth2\/access\_token",

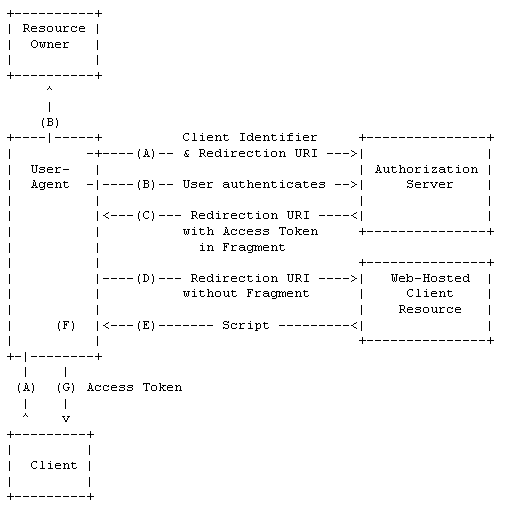
"error":"40018:Error: 提供的Access Grant是无效的、过期的或已撤销的，例如，Authorization Code无效、Refresh Token无效、redirect\_uri与获取Authorization Code时提供的不一致等"

}

### 客户端流程Implicit Grant Flow

此流程适用于在客户端调用REST API的应用。例如：后端无Web Server支持的手机客户端和桌面客户端，运行于浏览器内部的JavaScript、ActionScript应用（浏览器插件、纯Flash应用）。

采用Implicit Grant方式获取Access Token的授权验证流程又被称为User-Agent Flow，适用于所有无Server端配合的应用（由于应用往往位于一个User Agent里，如浏览器里面，因此这类应用在某些平台下又被称为Client-Side Application），如手机/桌面客户端程序、浏览器插件等，以及基于JavaScript等脚本客户端脚本语言实现的应用，他们的一个共同特点是，应用无法妥善保管其应用密钥（App Secret Key），如果采取Authorization Code模式，则会存在泄漏其应用密钥的可能性。其流程示意图如下：

[](http://wiki.open.kaixin001.com/lib/exe/detail.php?id=%E4%BD%BF%E7%94%A8Implicit_Grant%E8%8E%B7%E5%8F%96Access_Token&media=implicit.png)

#### 获取Access Token

**1. 请求数据包格式**

为了获取Access Token，应用需要将用户浏览器（或手机/桌面应用中的浏览器组件）到开心网开放平台OAuth2.0授权服务的“<http://api.kaixin001.com/oauth2/authorize>”地址上，并带上以下参数：

|  |  |  |
| --- | --- | --- |
| 参数名 | 必选 | 介绍 |
| client\_id | true | 申请组件时获得的API Key |
| response\_type | true | 此值固定为“token” |
| redirect\_uri | true | 授权后要回调的URI，即接受code的URI。对于无Web Server的应用， 其值可以是“oob”。 |
| scope | false | 以空格分隔的权限列表，若不传递此参数，代表请求默认的basic权限。 如需调用扩展权限，必需传递此参数，详细请参考[权限列表](http://wiki.open.kaixin001.com/index.php?id=%E6%8E%A5%E5%8F%A3%E8%AE%BF%E9%97%AE%E6%9D%83%E9%99%90%E8%AF%B4%E6%98%8E#权限列表) |
| state | false | 用于保持请求和回调的状态，授权服务器在回调时（重定向用户 浏览器到“redirect\_uri”时），会在Query Parameter中原样回传该参数 |
| display | false | 登录和授权页面的展现样式，默认为“page”。手机访问时，此参数无效 |
| client | false | 是否为手机访问。手机访问：client=1；不是手机，无需次参数 |

例如：“client\_id”为“Va5yQRHlA4Fq4eR3LT0vuXV4”的应用要请求某个用户的默认权限，并在授权后需跳转到“<http://www.example.com/oauth_redirect>”，同时希望在弹出窗口中展现用户登录授权界面，则应用需要重定向用户浏览器到如下URL：

http://api.kaixin001.com/oauth2/authorize?

response\_type=token&

client\_id=Va5yQRHlA4Fq4eR3LT0vuXV4&

redirect\_uri=http%3A%2F%2Fwww.example.com%2Foauth\_redirect&

scope=basic&

display=popup&

state=XXX

**2. 响应数据包格式**

若用户登录并接受授权，授权服务将重定向用户浏览器到“redirect\_uri”，并在Fragment中追加如下参数：

|  |  |
| --- | --- |
| 参数名 | 介绍 |
| access\_token | 要获取的Access Token |
| expires\_in | Access Token的有效期，以秒为单位 |
| refresh\_token | 用于刷新Access Token 的 Refresh Token |
| scope | Access Token最终的访问范围，即用户实际授予的权限列表（用户在授权页面时，有可能会取消掉某些请求的权限），关于权限的具体信息参考“权限列表”一节 |
| state | 如果请求获取Access Token时带有state参数，则将该参数原样返回 |
| encodeuid | 加密过的开心用户ID，如不需要使用可以忽略 |

例如：

HTTP/1.1 200 OK

Location: http://www.example.com/oauth\_redirect#

access\_token=12932845\_de81cfe747ad7e06f2a904e73646f04c&

expires\_in=2592000&

refresh\_token=12932845\_386fd97c84df3ee0dea051ec684888e8&

scope=basic&

state=xxx&

encodeuid=jiVuOCVy1a

与服务端流程不同的是，客户端流程不会返回code参数，而是在URI Fragment中返回Access Token（#access\_token=...），只有客户端代码(例如：例如运行与浏览器中的JavaScript、有浏览器控件支持的客户端代码)才可以获取Access Token。

### Refresh\_Token

令牌刷新方式，适用于所有有Server端配合的应用。

无论通过OAuth2.0哪种方式获取Access Token，都会拿到一个有效期为1个月的Access Token和有效期为2个月的Refresh Token。对于这些应用，只要用户在两个月内登录，应用就可以使用Refresh Token刷新以获得新的Access Token（新的Refresh Token也会同时下发），因此用户只要在两个月内登录过你的应用，就不需要重新登录。

**1. 请求数据包格式**

使用Refresh Token刷新以获得新的Access Token，需要应用在其服务端发送请求（推荐用POST方法）到开心网OAuth2.0授权服务的“<https://api.kaixin001.com/oauth2/access_token>”地址上，并带上以下参数：

|  |  |  |
| --- | --- | --- |
| 参数名 | 必选 | 介绍 |
| grant\_type | true | 固定为“refresh\_token” |
| refresh\_token | true | 用于刷新Access Token用的Refresh Token |
| client\_id | true | 应用的API Key |
| client\_secret | true | 应用的Secret Key |
| scope | flase | 以空格分隔的权限列表，若不传递此参数，代表请求的数据访问操作权限 与上次获取Access Token时一致。通过Refresh Token刷新Access Token 时所要求的scope权限范围必须小于等于上次获取Access Token时授予的 权限范围，详细请参考[权限列表](http://wiki.open.kaixin001.com/index.php?id=%E6%8E%A5%E5%8F%A3%E8%AE%BF%E9%97%AE%E6%9D%83%E9%99%90%E8%AF%B4%E6%98%8E#权限列表) |

例如：

https://api.kaixin001.com/oauth2/access\_token?

grant\_type=refresh\_token&

refresh\_token=12932845\_f139bd04b513c4abb880e48f21cee542&

client\_id=Va5yQRHlA4Fq4eR3LT0vuXV4&

client\_secret=0rDSjzQ20XUj5itV7WRtznPQSzr5pVw2&

scope=basic

**2. 响应数据包格式**

HTTP/1.1 200 OK

Content-Type: application/json

Cache-Control: no-store

{

"access\_token": "12932845\_343bec2737979ca0ace85a80c243b1ec",

"expires\_in": 2592000,

"refresh\_token": "12932845\_de81cfe747ad7e06f2a904e73646f04c",

"scope": "basic",

}

若请求错误，服务器将返回一段JSON文本，包含以下参数：

HTTP/1.1 400 Bad Request

Content-Type: application/json

Cache-Control: no-store

{

"error\_code":"400",

"request":"\/oauth2\/access\_token",

"error":"40021:Error: 请求的scope参数是无效的、未知的、格式不正确的"

}

### 用户名密码流程Resource Owner Password Credentials Flow

采用用户名、密码获取Access Token，适用于任何类型应用。

使用用户名密码端流程的权限需要提前申请。

用户名密码流程需要使用HTTP POST请求OAuth 2.0的Access Token Endpoint，并需要带上一系列需要的参数：

|  |  |
| --- | --- |
| grant\_type | 使用[Resource Owner Password Credentials](http://wiki.dev.renren.com/wiki/Resource_Owner_Password_Credentials)作为Access Grant时，此值为“password”。 |
| username | 用户名 |
| password | 用户密码 |
| client\_id | 在开发者中心注册应用时获得的API Key; |
| client\_secret | 在开发者中心注册应用时获得的Secret Key |
| scope | 非必须参数。以空格分隔的权限列表，若不传递此参数，代表请求用户的默认权限。全部权限请参考'[权限列表](http://wiki.dev.renren.com/wiki/%E6%9D%83%E9%99%90%E5%88%97%E8%A1%A8)'。 |

请求如下：

https://graph.renren.com/oauth/token?

grant\_type=password&

username=USERNAME&

password=PASSWORD&

client\_id=YOUR\_API\_KEY&

client\_secret=YOUR\_SECRET\_KEY&

scope=...

如果用户验证通过（用户名密码正确），并且应用验证通过，OAuth 2.0会返回Access Token相关的信息：

{

"access\_token": "10000|5.a6b7dbd428f731035f771b8d15063f61.86400.1292922000-222209506",

"expires\_in": 87063,

"refresh\_token": "10000|0.385d55f8615fdfd9edb7c4b5ebdc3e39-222209506",

"scope": "read\_user\_album read\_user\_feed"

}

## 接口访问权限scope

每一个Access Token代表“一个用户”授予“一个应用”的“一系列权限”。这“一系列权限”就是在获取Access Token过程中传递的“scope”参数

在调用API时，API服务会检验请求中的Access Token是否包含本API需要的权限。只有包含所需要的权限，才能正常调用该API，否则就无法正常调用。

接口权限包括两类：

第一类是基本权限：也是默认访问权限，权限名为basic，第三方应用在请求获取Access Token时，所传递的“scope”参数中可以不包含basic权限，一旦用户同意授权，则basic权限会自动授予。

第二类是扩展权限：如果第三方应用需要调用的API必需用户授予扩展权限，那么在获取Access Token时，您就需要在scope参数中设定这些权限。用户会根据您设置的权限，来决定是否授予这些权限，只有得到用户授权，您才能调用相应的API。

例如：您的应用需要调用[/records/add](http://wiki.open.kaixin001.com/index.php?id=%E5%8F%91%E8%A1%A8%E8%AE%B0%E5%BD%95)接口发一条记录，您就需要设定scope=create\_records，用户授权后，您才能调用该接口。

http://api.kaixin001.com/oauth2/authorize?response\_type=code

&client\_id=Va5yQRHlA4Fq4eR3LT0vuXV4

&redirect\_uri=http%3A%2F%2Fwww.example.com%2Foauth\_redirect

&scope=basic create\_records

&display=popup

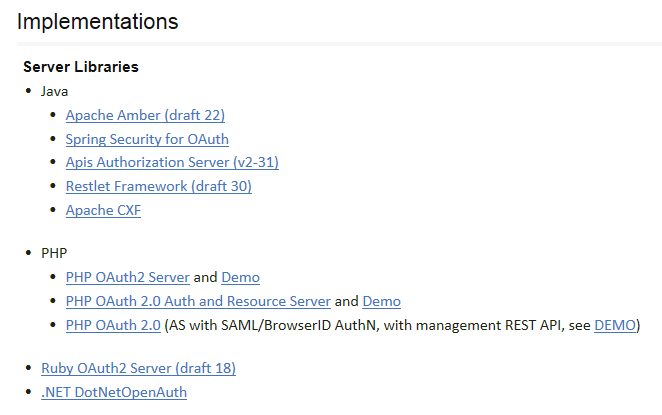
注意：basic可有可无，是默认权限；多个scope权限之间用一个半角空格分隔。

## 跨站请求伪造 (CSRF)

CSRF通过伪装来自受信任用户的请求来利用受信任的网站。为了防止CSRF攻击，在请求人人OAuth 2.0 Authorize Endpoint的时候，需要使用state传递一个用户的唯一标识，这个标识在调转到rediret\_uri时会被原样返回。人人网强烈推荐使用这种方式来保持用户的状态，防止CSRF攻击。

# spring security oauth

<http://oauth.net/2/>



以上是官网推荐的服务器实现方式，本章对Spring Security for OAuth进行详细调研，但最终须根据开发者采用的技术，选择合适的实现方式，例如demo采用Restlet提供WebService，则可考虑采用Restlet Framework。以上实现都处于开发测试阶段，没有详细文档。

## 运行Sample

源码在demo/project/spring-security-oauth。

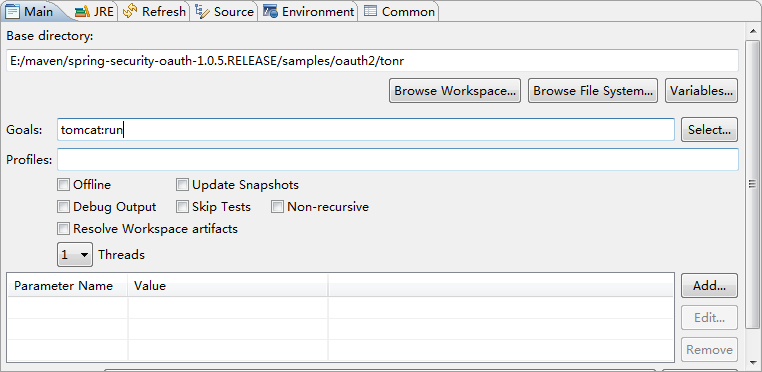
Sparklr:Provider，Tonr:Consumer

要看懂sample，首先须学习maven，spring mvc和spring security。

check out源码，用m2eclipse maven导入项目。由于只采用oauth2.0版本，因此删除oauth1.0的实现，即sparklr，tonr，spring-security-oauth三个工程和pom.xml中对应的module。

发布Sparklr2，在sparklr2的pom.xml上右键Run As->mvn install

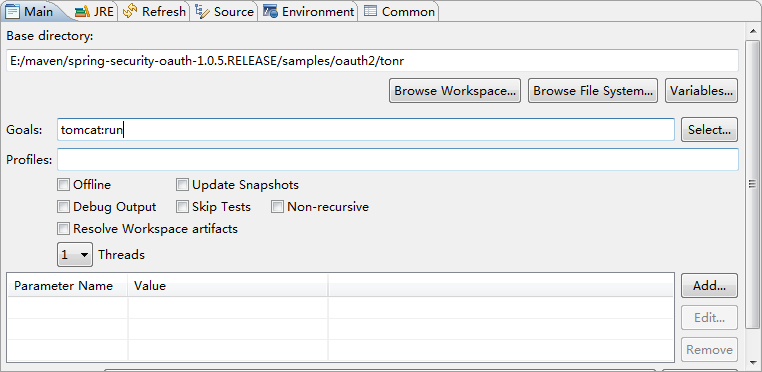
Run As->mvn build，Goals输入tomcat:run，发布成web服务



浏览器访问：http://localhost:8080/sparklr2，log in后看到图片，logout。关闭连接，此时sparklr2已在tomcat容器中，tonr2启动时会同时启动sparklr2。

发布Tonr，在tonr2的pom.xml上右键Run As->mvn install

Run As->mvn build，Goals输入tomcat:run，发布成web服务



浏览器访问：http://localhost:8080/tonr2，login后看到图片，logout。

Login后单击view Marissa's Sparklr photos，跳转到sparklr授权页面，登录授权后跳转回tonr2页面，就可以看到Marissa 在sparklr的照片。

## 客户端配置

1. **配置命名空间和schema**

<beans xmlns:oauth="http://www.springframework.org/schema/security/oauth2"

xsi:schemaLocation=[http://www.springframework.org/schema/security/oauth2 http://www.springframework.org/schema/security/spring-security-oauth2-1.0.xsd](http://www.springframework.org/schema/security/oauth2%20http://www.springframework.org/schema/security/spring-security-oauth2-1.0.xsd)>

1. **配置客户端Filter**

<http access-denied-page="/login.jsp">

<intercept-url pattern="/photos" access="ROLE\_USER,SCOPE\_READ" />

...

<custom-filter ref="**oauth2ClientFilter**" after="EXCEPTION\_TRANSLATION\_FILTER"/>

</http>

<oauth:client id="**oauth2ClientFilter**"/>

client用于配置OAuth2.0机制，包含redirect-strategy-ref属性，redirect-strategy bean的引用。默认值为DefaultRedirectStrategy。

1. **配置受保护资源**

resource用于定义受保护资源，每个resource对应一个OAuth2ProtectedResourceDetails实例，包含以下属性：

id：resource的id，用于客户端查找资源。

type：资源类型，指明如何获取access token，可选值有”authorization\_code", "implicit" and "client\_credentials”。默认生成BaseOAuth2ProtectedResourceDetails。

client-id：OAuth客户端id，从OAuth Provider处获取。

client-secret：资源密码，默认情况下，没有密码提供给资源。

access-token-uri：换取access token的URI。

user-authorization-uri：用户将被重定向到该URI，进行授权。

scope：以逗号分隔，指明资源访问的范围，默认为空。

client-authentication-schema：授权access token的scheme。可选值为http\_basic和form。默认为http\_basic。

例如：

<oauth:resource id="sparklr" type="authorization\_code" client-id="tonr" client-secret="secret"

access-token-uri="${accessTokenUri}" user-authorization-uri="${userAuthorizationUri}" scope="read,write" />

1. **访问受保护资源**

配置好资源后，可通过RestTemplate访问资源，需要为RestTemplate提供一个OAuth2ProtectedResourceDetails实例。

<oauth:resource id="sparklr" ... />

<bean id="sparklrService" class="org....SparklrServiceImpl">

...

<property name="sparklrRestTemplate">

<oauth:rest-template resource="sparklr" />

</property>

</bean>

## 服务器配置

1. **配置授权服务器**

authorization-server用于配置OAuth2.0授权服务器。

client-details-service-ref：指向定义client details service的 bean。

token-services-ref：指向定义token services的bean。

user-approval-handler-ref：指向定义user approval handler的bean。

user-approval-page：处理用户同意表单的URL。

approval-parameter-name：表单参数的名称，用于表明用户同意客户端授权请求。

authorization-endpoint-url：请求授权的URL，默认为/oauth/authorize，受Spring Security保护，只有认证用户可访问。

token-endpoint-url：请求access token的URL，默认为/oauth/token。

authorization-code：作为authorization-server的子节点，支持以下属性，disabled：指明授权码是否有效。authorization-code-services-ref：指向定义授权码services的bean，默认为InMemoryAuthorizationCodeServices。

例如：

<oauth:authorization-server client-details-service-ref="**clientDetails**" token-services-ref="**tokenServices**" user-approval-handler-ref="**userApprovalHandler**">

<oauth:authorization-code />

<oauth:implicit />

<oauth:refresh-token />

<oauth:client-credentials />

<oauth:password />

</oauth:authorization-server>

以上配置了五种授权方式：

* Authentication Code，又称Web Server Flow，适用于所有有Server端配合的应用和无Server端应用
* Implicit Grant，又称User-Agent Flow，适用于所有无Server端配合的应用（桌面客户端需要内嵌浏览器）
* Client Credentials，即采用应用公钥、密钥获取Access Token，适用于任何类型应用。
* Refresh Token，即令牌刷新方式，适用于所有有Server端配合的应用
* password，适用于无法使用浏览器发起服务端和客户端验流程的应用。

1. **配置client details service**

client-details-service用于定义客户端细节的内存实现，id属性和client子节点。

client-id：必须，客户端id。

secret：客户端密码，如果采用implicit方式则不需要。

scope：客户端限制范围，如果没有定义则无限制。

authorized-grant-types：授权流，默认值为authorization\_code,refresh\_token。

authorities：授权。

例如：

<oauth:client-details-service id="clientDetails">

<oauth:client client-id="tonr" resource-ids="sparklr"

authorized-grant-types="authorization\_code,implicit" authorities="ROLE\_CLIENT"

scope="read,write" secret="secret" />

</oauth:client-details-service>

1. **配置Token Service**

AuthorizationServerTokenServices定义了管理OAuth2.0 token的操作。

可采用DefaultTokenServices作为AuthorizationServerTokenServices的实现，它包括生成随机数等功能，除了token持久化用TokenStore。InMemoryTokenStore是TokenStore的内存实现。

也可采用JDBC实现保存tokens和认证，需要在数据库中增加下表，并提供DataSource。

CREATE TABLE oauth\_refresh\_token ( TOKEN\_ID VARCHAR(64) NOT NULL PRIMARY KEY, TOKEN BLOB NOT NULL, authentication BLOB NOT NULL);

例如：

<bean id="tokenStore" class="org.springframework.security.oauth2.provider.token.InMemoryTokenStore" />

<bean id="tokenServices" class="org.springframework.security.oauth2.provider.token.DefaultTokenServices">

<property name="tokenStore" ref="tokenStore" />

<property name="supportRefreshToken" value="true" />

<property name="clientDetailsService" ref="clientDetails" />

</bean>

1. **配置用户授权处理方法**

<bean id="userApprovalHandler" class="org.springframework.security.oauth.examples.sparklr.oauth.SparklrUserApprovalHandler">

<property name="autoApproveClients">

<set>

<value>my-less-trusted-autoapprove-client</value>

</set>

</property>

<property name="tokenServices" ref="tokenServices" />

<property name="requestFactory" ref="requestFactory" />

</bean>

Provider分为认证Service和资源Service，可以部署为同一应用，也可以多个资源Service共享一个认证Service。token请求由Spring MVC controller处理，获取资源由Spring Security request filter处理。

[AuthorizationEndpoint](http://static.springsource.org/spring-security/oauth/apidocs/org/springframework/security/oauth2/provider/endpoint/AuthorizationEndpoint.html) ：处理认证请求，返回code，Default URL： /oauth/authorize.

[TokenEndpoint](http://static.springsource.org/spring-security/oauth/apidocs/org/springframework/security/oauth2/provider/endpoint/TokenEndpoint.html) ：利用code换取access token，Default URL: /oauth/token

[OAuth2AuthenticationProcessingFilter](http://static.springsource.org/spring-security/oauth/apidocs/org/springframework/security/oauth2/provider/authentication/OAuth2AuthenticationProcessingFilter.html)：根据access token载入请求认证

# spring security

## 参考教材

下载[Spring Security distribution](http://www.springsource.com/download/community?project=Spring%20Security&version=3.1.2.RELEASE)，解压后spring-security-samples-tutorial-3.1.x.war，改名为tutorial.war，放到tomcat的webapps目录中，启动tomcat

<http://localhost:8080/tutorial/>

用户/密码

rod/koala：supervisor, user, teller

dianne/emu：user,teller

scott/wombat：user

源码地址，参考samples

git://github.com/SpringSource/spring-security.git

非常好的入门教程<http://www.mkyong.com/tutorials/spring-security-tutorials/>

<http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/>

## 第一个Spring security项目

源码在demo/project/springsecurity。

新建maven项目

第一步将DelegatingFilterProxy配置到web.xml中；

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

需要注意url-pattern最好对所有url都过滤，filter-name必须设为springSecurityFilterChain。

第二步配置security文件

最简单的配置如下：

<http auto-config='true'>

<intercept-url pattern="/\*\*" access="ROLE\_USER" />

</http>

指明所有路径必须以ROLE\_USER身份才能访问。

<authentication-manager>

<authentication-provider>

<user-service>

<user name="jimi" password="jimispassword" authorities="ROLE\_USER,ROLE\_ADMIN" />

<user name="bob" password="bobspassword" authorities="ROLE\_USER" />

</user-service>

</authentication-provider>

</authentication-manager>

配置用户名，密码及身份

### pom.xml

增加依赖

<properties>

<spring.version>3.1.2.RELEASE</spring.version>

</properties>

  <dependencies>

  <!—以下是 Spring 3 的依赖-->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring Security -->

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-core</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-web</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-config</artifactId>

<version>${spring.version}</version>

</dependency>

</dependencies>

### web.xml

引入spring mvc和spring security

<?xml version="1.0" encoding="UTF-8"?>

<!-- 采用2.4的命名空间，就不用在每个包含EL的jsp中设置<%@ page isELIgnored="false"%> -->

<web-app id="WebApp\_ID" version="2.4"

xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee

http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

<display-name>OAuth</display-name>

<!--

设置ContextLoaderListener的配置文件

spring-servlet.xml：MVC的配置文件，其文件名必须为DispatcherServlet的<servlet-name>-servlet格式

applicationContext-security.xml：spring security的配置文件

-->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>

/WEB-INF/spring-servlet.xml,

/WEB-INF/applicationContext-security.xml

</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!--

引入spring security，对所有url都用DelegatingFilterProxy过滤，其配置文件为applicationContext-security.xml

-->

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 引入spring mvc，对/所有路径都用DispatcherServlet进行分发 ，其配置文件为spring-servlet.xml-->

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

### applicationContext-security.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans:beans xmlns="http://www.springframework.org/schema/security"

xmlns:beans="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security.xsd">

<!--

use-expressions="true"后，就不可用简单的ROLE\_USER设置access属性，而要采用<intercept-url pattern="/\*" access="hasRole('ROLE\_USER')"/>

intercept-url：过滤是按照顺序执行的,所以最前面的是最想先过滤的

form-login：设置登录界面， 如果没指定登陆界面,spring security将用自身的登陆界面

login-page:指定登录界面;

default-target-url: 登录成功后跳转路径 ;

authentication-failure-url:认证失败跳转路径

logout：设置退出界面

logout-success-url: 退出成功跳转路径

在jsp中添加<a href="<c:url value="j\_spring\_security\_logout" />" > Logout</a>

<http-basic/>：采用基本认证，即弹出对话框，让用户输入用户名密码的方式，安全性差

<access-denied-handler error-page="/403"/>：设置拒绝进入时的页面

max-sessions="1": 一个用户第二次登陆时,第一次失效

exception-if-maximum-exceeded="true": 防止第二次登陆

access-denied-page: 该用户无权限时转到指定页面(转向) login-page="/login" default-target-url:默认跳转页面

-->

<http auto-config="true" use-expressions="true">

<intercept-url pattern="/welcome\*" access="hasRole('supervisor')" />

<intercept-url pattern="/secure/extreme/\*\*" access="hasRole('supervisor')"/>

<intercept-url pattern="/secure/\*\*" access="isAuthenticated()" />

<intercept-url pattern="/\*\*" access="permitAll" />

<form-login login-page="/login" authentication-failure-url="/loginfailed" />

<access-denied-handler ref="accessDeniedHandler"/>

<logout logout-success-url="/" />

</http>

<!-- 数据库控制用户名密码 -->

<!-- <authentication-manager>

<authentication-provider>

<jdbc-user-service data-source-ref="dataSource"

users-by-username-query="

select username,password, enabled

from users where username=?"

authorities-by-username-query="

select u.username, ur.authority from users u, user\_roles ur

where u.user\_id = ur.user\_id and u.username =? "

/>

</authentication-provider>

</authentication-manager> -->

<!-- 采用BCryptPasswordEncoder加密 -->

<beans:bean id="encoder" class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"/>

<!--

内存方式控制用户名密码

Usernames/Passwords are

rod/koala

dianne/emu

scott/wombat

peter/opal

-->

<authentication-manager>

<authentication-provider>

<password-encoder ref="encoder"/>

<user-service>

<user name="rod" password="$2a$10$75pBjapg4Nl8Pzd.3JRnUe7PDJmk9qBGwNEJDAlA3V.dEJxcDKn5O" authorities="supervisor, user, teller" />

<user name="dianne" password="$2a$04$bCMEyxrdF/7sgfUiUJ6Ose2vh9DAMaVBldS1Bw2fhi1jgutZrr9zm" authorities="user,teller" />

<user name="scott" password="$2a$06$eChwvzAu3TSexnC3ynw4LOSw1qiEbtNItNeYv5uI40w1i3paoSfLu" authorities="user" />

<user name="peter" password="$2a$04$8.H8bCMROLF4CIgd7IpeQ.tcBXLP5w8iplO0n.kCIkISwrIgX28Ii" authorities="user" />

</user-service>

</authentication-provider>

</authentication-manager>

</beans:beans>

## 问题

**问题1：No bean named 'springSecurityFilterChain' is defined**

<http auto-config=true> 属性，其实是下面这些配置的缩写：

<http>

<form-login />

<http-basic />

<logout />

</http>

如果没有配置form-login或logout，则必须设置auto-config=true

解决方法在applicationContext-security.xml中配置 <http auto-config="true"/>

**问题2：Configuration problem: Unable to locate Spring NamespaceHandler for XML schema namespace [http://www.springframework.org/schema/security]**

确保有以下依赖

spring-security-acl-3.1.0.M2.jar

spring-security-config-3.1.0.M2.jar

spring-security-core-3.1.0.M2.jar

spring-security-taglibs-3.1.0.M2.jar

spring-security-web-3.1.0.M2.jar

**问题3：Configuration problem: No AuthenticationEntryPoint could be established. Please make sure you have a login mechanism configured through the namespace (such as form-login) or specify a custom AuthenticationEntryPoint with the 'entry-point-ref' attribute**

<http>需用<form-login>或< entry-point-ref>指定登录界面，如果没有，则采用spring security默认的登录界面

**问题4：No mapping found for HTTP request with URI in DispatcherServlet with name**

<servlet-mapping>的<url-pattern>/</url-pattern>把所有的请求都交给spring去处理了，而所有available的请求url都是在Constroller里使用类似@RequestMapping(value = "/login/{user}", method = RequestMethod.GET)这样的注解配置的，这样的话对js/css/jpg/gif等静态资源的访问就会得不到。

spring-servlet.xml中增加mvc命名空间

xmlns:mvc=<http://www.springframework.org/schema/mvc>

http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.1.xsd

<mvc:default-servlet-handler/>

**问题5：jsp $ not working**

方法1：<%@ page isELIgnored="false"%>即表达式不被当字符串处理

但每个包含表达式的jsp文件前面都必须加这句话

方法2：

web.xml

设置web-app的命名空间为2.3，isELIgnored默认设为true

<!DOCTYPE web-app PUBLIC

"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app\_2\_3.dtd" >

设置web-app的命名空间为2.4，已经不用DTD，而只有schema声明

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"

version="2.4">

确定web.xml里的xsd版本之后一定要在JSP的声明(<%@page %>)部分加一行，如下：   
<%@ page isELIgnored="false" %> ，这样设为false才能解析EL表达式。

注意：其中servlets 2.4，这个版本的isELIgnored默认设置为false。所以使用web.xml里用web-app\_2\_4.xsd声明的时候在JSP页面不用特意声明。

**问题6：Maven项目导入Eclipse IDE，解决遇到“Unsupported IClasspathEntry kind=4 ”问题**

Eclipse导入maven项目后，项目右键->Maven->Disable Maven Nature

cmd中进入项目目录，运行mvn eclipse:clean

Eclipse中项目右键Configure->Convert to Maven Project