基于Android手机DeviceID的短信验证机制

2011-9-16

# 背景

基于Android手机开发的应用Inspector打算采取收费模式。希望采用的注册机制如下：

1. 每个账号可购买若干个装机数。比如用户Liming，购买5个装机数，则Liming可以在5部Android手机上安装Inspector应用，超过该数目则不能获得注册。
2. 辨识手机是根据该手机的DeviceID。事实上，每一部手机出厂时的DeviceID都是唯一的，不会重复，意味着对同一部手机，可以多次安装Inspector。即使在用户刷机之后，依然可以再次安装Inspector，因为其DeviceID与ROM无关，是固化在手机硬件里的。

# 方案

注册由两部分组成：

1. 用户名+注册码 （基本验证，为确保是合法用户）
2. 手机短信验证 （短信验证，为确保该手机可被注册）

该验证是为了确保inspector仅能被安装在不多于账号有效装机数的手机上。

其中1采用AES加密验证，即用户名及注册码是否能满足AES算法匹配。

其中2采用CS结构，即用户手机作为Client，注册时向服务机（Service）发送一条短信，Service接收后在数据库中验证该机器是否可以获得有效注册，并发回注册码短信，Client手机收到短信进行校验，校验成功即注册成功。

下文仅详细讨论2，即手机短信验证的具体设计。

# 设计

我们把Server端称为AuthServer，AuthServer将以Android手机上的service形式存在，即手机即是我们的验证服务器。

## 数据库

AuthServer在Android的SQLite数据库中创建并维护若干张table，定义如下：

**Table1：tbl\_inspector\_auth\_user**

**该table记录用户的有效记录数**

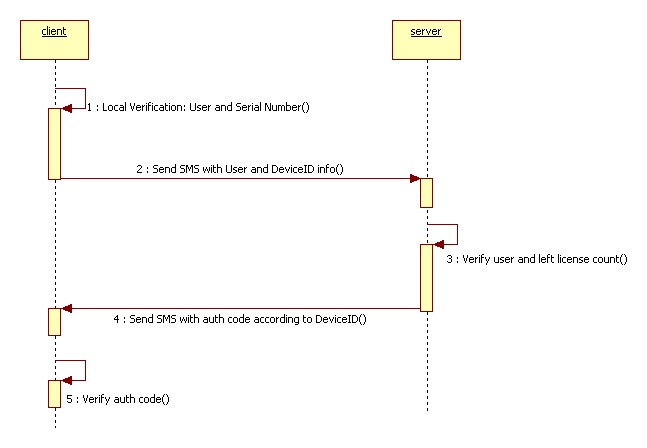
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 类型 | 能否为空 | 说明 |  |
| user | varchar(100) | N | 用户名 |  |
| auth\_all\_count | uint | N | 有效装机数 |  |
| auth\_left\_count | uint | N | 剩余装机数 |  |
| buydate | datetime | N | 购买日期 |  |
| reserve\_string\_1 | varchar(100) | Y | 预留 |  |
| reserve\_string\_2 | varchar(100) | Y | 预留 |  |
| reserve\_int\_1 | int | Y | 预留 |  |
| reserve\_int\_2 | int | Y | 预留 |  |

**Table2：tbl\_inspector\_auth\_record**

**该table记录每一次的注册信息**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 类型 | 能否为空 | 说明 |  |
| user | varchar(100) | N | 用户名 |  |
| device\_id | varchar(100) | N | 手机ID |  |
| phone\_number | varchar(20) | Y | 最新手机号码（如果同一部手机多次注册，则phone\_number只记录最后一次注册的手机号码，也存在无法取得手机号码的情况，所以该字段可能为空） |  |
| auth\_datetime | datetime | N | 最新注册日期时间（如果同一部手机多次注册，则auth\_datetime只记录最后一次的日期时间） |  |
| auth\_count | uint | N | 记录该手机的注册次数（通常情况下只会注册一次，auth\_count = 1） |  |
| reserve\_string\_1 | varchar(100) | Y | 预留 |  |
| reserve\_string\_2 | varchar(100) | Y | 预留 |  |
| reserve\_int\_1 | int | Y | 预留 |  |
| reserve\_int\_2 | int | Y | 预留 |  |

## 验证流程



上图解释了短信验证流程，client通过短信发送User和DeviceID给server，如果该User还有可用装机数（auth\_left\_count > 0），则使用AES算法得到DeviceID对应的加密后结果，取出结果的前8位通过短信发送回手机，该手机比较该8位字串与本地AES计算结果，匹配则验证成功。

# 程序实现

采用通常的Android应用开发方式，Android SDK + Java + Eclipse的开发套件。

## 要点1：注册BoardcastReceiver拦截SMS

<!--接受短消息需要注册的receiver，其中的android:name是继承了BroadCastReceiver的类的位置 android:priority="100" 指定了接收器的级别，这里设置的值比系统的值要高，这意味着程序的接受 短消息要早于系统接受短消息，这也就给我们提供了拦截短消息，不让系统发出短消息提示的机会。 -->

<receiver android:name=".SmsBroadCast">

<intent-filter android:priority="100">

<action android:name="android.provider.Telephony.SMS\_RECEIVED" />

</intent-filter>

</receiver>

## 要点2：SQLite数据库的性能和扩展性

在中短期内，用户数小于1w和注册数少于5w的情形下，数据库压力不大。如果到10w级别上必须考虑扩展性。（TBD）