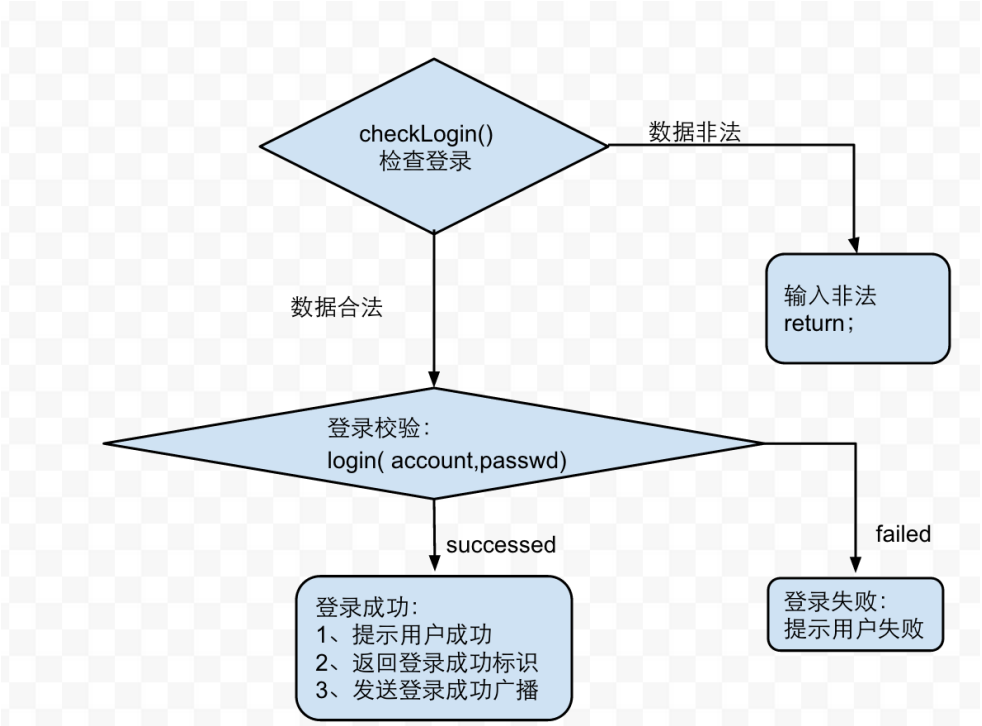
目前为止发现的关于用户信息的存储包括：

1、用户名、密码的存储；

2、登录成功后，从服务器获取的user信息的存储；

接上一篇，先看下登录请求的流程图：



1、用户名和密码的存储：

```
// 保存用户名和密码
mAppContext.saveAccountInfo(CyptoUtils.encode(Contanst.ACCOUNT_EMAIL, email),
CyptoUtils.encode(Contanst.ACCOUNT_PWD, passwd));
```

接着走到了：AppContext.java中的saveAccoutInfo()方法：

```
    /**
     * 保存用户的email和pwd
     * @param email
     * @param pwd
     */
    public void saveAccountInfo(String email, String pwd) {
        setProperty(ACCOUNT_EMAIL, email);
        setProperty(ACCOUNT_PWD, pwd);
    }
```

接下来是，setProperty方法调用了AppConfig.getAppConfig（）用来进行用户信息的设置，这里使用了单例模式，保证整个程序中只有一份AppConfig配置：

```
public void setProperty(String key, String value) {
		AppConfig.getAppConfig(this).set(key, value);
}
```

接下来是，set方法，获取Properties来进行数据的存储：

```
public void set(String key, String value) {
		Properties props = get();
		props.setProperty(key, value);
		setProps(props);
}
```

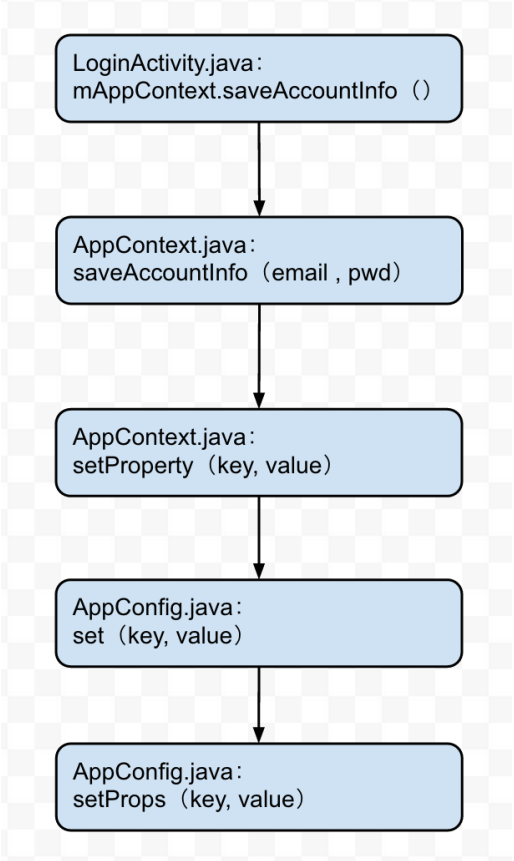
接下来是 ，setProps：

```
private void setProps(Properties p) {
		FileOutputStream fos = null;
		try {
			// 把config建在files目录下
			// fos = activity.openFileOutput(APP_CONFIG, Context.MODE_PRIVATE);

			// 把config建在(自定义)app_config的目录下
			File dirConf = mContext.getDir(APP_CONFIG, Context.MODE_PRIVATE);
			File conf = new File(dirConf, APP_CONFIG);
			fos = new FileOutputStream(conf);

			p.store(fos, null);
			fos.flush();
		} catch (Exception e) {
			e.printStackTrace();
		} finally {
			try {
				fos.close();
			} catch (Exception e) {
			}
		}
}
```

用户名和密码存储的流程图如下：



2、登录成功后，从服务器获取的user信息的存储：

向服务器请求数据是从login(email, passwd)方法发起的，整个过程中使用异步任务来完成的。

1、//异步登录

```
new AsyncTask<Void, Void, Message>() {
        @Override
        protected Message doInBackground(Void... params) {
        Message msg =new Message();
        try {
            User user = mAppContext.loginVerify(account, passwd);
            msg.what = 1;
            msg.obj = user;
        } catch (Exception e) {
             …….
        }
                return msg;
        }
        }.execute();
```

2、从异步登录的代码中可以看到，代码走到了：mAppContext.loginVerify(account, passwd),从代码中可以看到——通过ApiClient.login(this, account, pwd)请求user信息，如果！=null ,就存储用户信息，否则return;

```
/**
 * 用户登录
 *
 * @param account
 * @param pwd
 * @return
 * @throws AppException
 * @throws IOException
 */
public User loginVerify(String account, String pwd) throws AppException {
	User user = ApiClient.login(this, account, pwd);
	if (null != user) {
		// 保存登录用户的信息
		saveLoginInfo(user);
	}
	return user;
}
```

3、接下来走到了AppContext中的saveLoginInfo(user)：

```
/**
 * 保存登录用户的信息
 *
 * @param user
 */
@SuppressWarnings("serial")
private void saveLoginInfo(final User user) {
	if (null == user) {
		return;
	}
	// 保存用户的信息
	this.loginUid = StringUtils.toInt(user.getId());
	this.login = true;
	setProperties(new Properties() {
		{
			setProperty(XXX, String.valueOf(user.getXXX()));
```

```
                .....
            }
        });
    }
```

4、接下来走到了，AppContext中的setProperties（ps）方法：

```
    public void setProperties(Properties ps) {
        AppConfig.getAppConfig(this).set(ps);
    }
```

5、接下来走到了，AppConfig中的set(ps)方法：

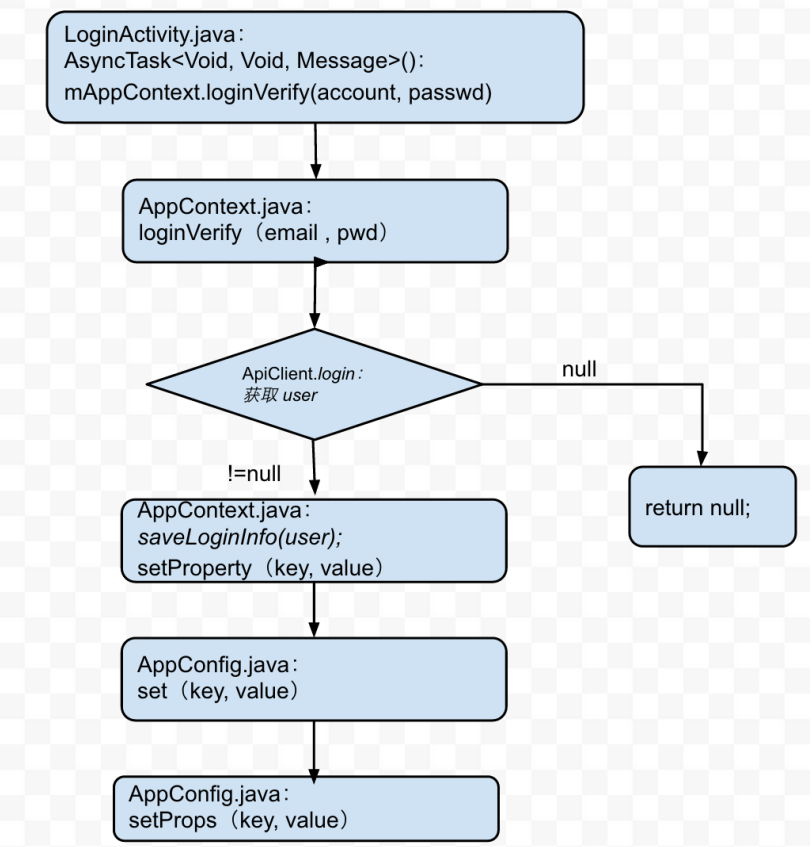
```
public void set(Properties ps) {
        Properties props = get();
        props.putAll(ps);
        setProps(props);
}
```

6、接下来走到了，AppConfig中的setProps（ps）方法中：

```
    private void setProps(Properties p) {
        FileOutputStream fos = null;
        try {
            // 把config建在files目录下
            // fos = activity.openFileOutput(APP_CONFIG, Context.MODE_PRIVATE);
            // 把config建在(自定义)app_config的目录下
            File dirConf = mContext.getDir(APP_CONFIG, Context.MODE_PRIVATE);
            File conf = new File(dirConf, APP_CONFIG);
            fos = new FileOutputStream(conf);

            p.store(fos, null);
            fos.flush();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                fos.close();
            } catch (Exception e) {
            }
        }
    }
```

用户数据的存储流程图：



至此：用户名和密码、从服务器返回的用户信息的配置学习完毕，我觉着我还得在好好学习一下 Properties 这个类的用法，以前在学Java基础的时候，学过但不是很深入！