**写在前面**

**1.Java基本语法知识**

Java新建一个对象A的语法，A a = new A();关键字为new；

**2．什么是框架**

框架提供了一套帮助解决某类问题的库或工具，我们可以通过调用这些库来简化开发任务，使工作注意力集中到核心业务的开发中（对应到本系统就是交易逻辑的开发，而不用关心对象和数据库的对应，类的创建和管理等），同时框架也提供了一个符合软件开发设计模式的大致架构，我们只需要按照框架的要求进行开发，即可以开发出一个符合设计模式的，易于维护的大型系统（例如SpringMVC是一个MVC框架，我们只需要按照框架的规定进行开发，就可以实现MVC的三层系统结构，而不用自己去实现繁琐的MVC分层逻辑）。

**3.什么是Java的集合**

Java的集合是JDK提供的一套包含常用的数据结构（类似于在C语言中自己创建的数组就是一个数据结构，用来存取数据）的类，通过使用Java集合，可以很轻松便捷地对数据进行存取，例如ArrayList是一个列表，其基本特性类似于C语言的数组，可以通过下标对元素进行访问，但是数组的容量大小是不变的，但是ArrayList的容量是动态的，当快要存满的时候，会自动扩大容量，这些细节都是JDK默认实现的，所以我们使用集合类可以很快速便捷地进行开发。

ArrayList.add()：添加一个元素

ArrayList.remove()：删除一个元素

ArrayList.size():取得list的容量大小

Java中常用的集合还有Map和Set，在本系统中都没有用到，不做详述。

**4.什么是正则表达式**

参考网页：http://deerchao.net/tutorials/regex/regex.htm

在编写处理字符串的程序或网页时，经常会有查找符合某些复杂规则的字符串的需要。正则表达式就是用于描述这些**规则**的工具。换句话说，**正则表达式就是记录文本规则的代码**。即是可以通过正则表达式的规范语法，描述正常的字符串应该是什么样子的，然后用各语言提供正则表达式的验证功能对字符串是否和定义的文本规则相符进行验证。对于正则表达式的验证过程，即是首先定义文本规则的代码，然后通过JavaScript或者是Java提供的正则表达式验证功能，对相关字符串进行验证即可。其中JavaScript的验证代码如下：

图 JavaScript的正则表达式验证代码

**4.什么是Spring**

1.首先Spring是一个框架，什么是框架见2，Spring提供了一个**控制反转**和**对象生命周期**的管理功能，并且可以很方便地和其他框架组合起来使用，拥有很好的可扩展性，对应到本系统就和Hibernate框架进行联合使用。

**5.什么是控制反转**

参考网页：

<http://www.cnblogs.com/xishuai/p/3666276.html>

控制反转实际上是一种设计模式，是为了解决大型程序开发中的**耦合性**问题而提出的。Spring提供了一个容器，容器会在初始化的时候新建（类似通过问题1的new的方式）新建了其所有管理的对象，那么哪些对象是受Spring框架管理的呢，我们通过配置的方式指定哪些程序包受spring管理，对应到程序中即是com.gxx.naquhua.entity,com.gxx.naquhua.service, com.gxx.naquhua.api包下面的所有类都是受spring管理的。那么什么叫做代码包呢，代码包是一个集合概念，其实就是一个文件夹，里面存放了一系列我们定义的类文件，拥有相似的功能，我们就从逻辑上将其存放在一个包下面，方便我们后面找到相应的代码文件，类似于你定期清理相同功能的文件夹，只不过我们是在一开始就建立相应文件夹了。那么我们回到最初的问题，spring容器初始化了管理的所有对象，我们之前要使用对象的时候都是通过1所述的new的方式去新建一个对象，现在spring容器在初始化的时候建立了我们要是使用的大部分对象，我们可以直接从容器中获得对象，这样如果后面对象有什么更改，比如改了名字或者功能什么的，就不用去更改引用到他的所有代码了，因为我们是直接从容器中获得对象的，容器会自动感知到这些变化，我们正常调用就行了。这就是我们所说的控制反转，因为讲对象调用和初始化的权利交给了容器，控制权转换了，即反转了。

**6.什么是Spring中的对象生命周期**

我们在5中说到，Spring容器会在初始化的时候新建一系列类对象，这些对象拥有生命周期，这个名词的概念我们理解为一个人的出生到死亡。当Spring容器启动的时候，这些类对象是初始化的生命周期，在容器启动后，对象处于运行中的状态，容器关闭的时候，对象处于销毁的周期。

**7.什么是耦合**

软件工程中对象之间的耦合度就是**对象之间的依赖性**。对象之间的耦合越高，维护成本越高。因此对象的设计应使类和构件之间的耦合最小。

**8.什么是Hibernate**

Hibernate是一个将Java对象和关系型数据库对应起来的框架，运用该框架我们不用编写复杂的sql代码和对象进行对应，框架帮我们完成了这一过程。使得Java程序员可以随心所欲的使用对象编程思维来操纵数据库。

**9.Hibernate的使用**

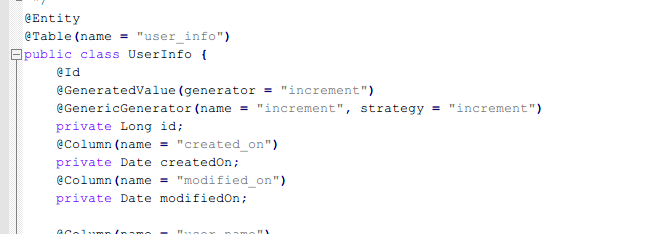
系统使用Hibernate框架建立起了对象模型和关系型数据库的关系模型的联系，编码是采用的是注解的方式进行标注，例如在一个JavaBean的类名前标注@Entity表示这个JavaBean是会通过Hibernate框架建立对象关系模型的，UserInfo对象对应的数据库里面的表名为

图 通过注解@Entity的方式配置hibernate对象关系实体

user\_info，对象里面的各种属性，如id会对应user\_info表里面的id属性。Hibernate框架中，从数据库中新增一个新对象的操作为hibernateTemplate.save(object),

删除一个对象为hibernateTemplate.delete(object),

查找一个对象为hibernateTemplate.find(hql),

更改一个对象hibernateTemplate.update(object)，

通过hibernate框架，可以避免编写繁琐的JDBC代码，将工作注意力转移到业务代码逻辑的编写上，大大减轻了开发任务。

**9.JSON字符串格式**

系统当中的所有服务器和前台的信息交换都是通过符合json格式的字符串形式进行交换的。json只是一种数据展现的格式而已，满足json格式的字符串，我们可以很方便地对其进行存取操作。在网络请求传输的时候，避免使用字节流传输造成的接收方不好辨认数据格式的尴尬。

首先我们有一个对象person，包含连个属性name=‘郭欣欣’，年龄=“永远18岁“，我们将这个对象转换为json格式的字符串，就是person[name:’郭欣欣’,’永远18岁’]的形式，在网络传输的时候，我们不用直接传输对象字节流，而是传输字符串，接收端在接受到json字符串后，就可以很方便地渠道其name和age属性做相应的展示了。

***系统所有的app端展示服务器获得的相应信息的流程都是：服务器将对象转换为json字符串🡪app接收到json字符串,获取字符串对应属性展示到相应地方***

***系统的json字符串的转换也不用关心细节，我们使用谷歌的一个开源方案实现的，叫做gson***

**10．关于cordova和web app技术**

cordova只是一个工具和平台，我们按照cordova的工程要求建立相关的HTML5网页文件样式，通过cordova工具的一条命令cordova compile就可以把这些HTML5代码打包成一个安卓应用，这个web网页🡪应用安装包的技术就叫做web app技术。

**11.为什么选用cordova和web app**

因为原生安卓的界面样式编写和优化需要专业和系统地学习，本系统应该把注意力放在交易相关模块的开发上，选用cordova和web app可以使用HTML5代码开发移动应用，能够很好地**节省时间**和**快速开发**。

**12.加密和权限认证**

后台服务会对客户端的每一次请求进行权限认证，具体认证流程为：

用户在登录成功后，服务器会返回一串32为的随机字符串，我们称之为access\_token，access\_token和用户的userId一起作为一对键值对保存在redis缓存中，并且有一定的有效期，对应于账户登录的有效期概念。而后用户的每一次请求，对需要在请求参数中带上access\_token的值，服务器会根据access\_token的值在redis中查找登录的userId，如果找不到则拒绝资源请求。如果找到，则说明请求合法，准许进行相关操作。通过这种方式实现了对OPEN API系统调用的权限认证。

密码的MD5加密过程：

首先通过HTTP请求服务器收到注册的密码明文，通过MD5将明文加密成密文字符串，再将这串密文字符串存储到mysql数据库中。

用户登录即为上述过程的逆过程，接受到用户名和密码，在数据库中查找对应用户名的用户信息，得到其密码（注意数据库中存储的是密码的加密后字符串），将登录请求中的密码明文通过同样的MD5算法进行加密，将密文和数据库中查找得到的密文进行比对，比对相等登录成功，比对失败登录失败。

这里的MD5加密算法是JAVA JDK中提供的默认实现，直接说用的JDK中的默认实现。

**11.系统开发中用到的技术**

Java基础语法（其中用到了Java的集合对象对数据进行存取），正则表达式验证数据合法性，Spring， Hibernate框架的编程，MySQL数据库相关操作（数据的增删改查）。

系统开发过程中的**每一个模块**都**全部**用到了以上的开发技术，相关技术细节上面已经做了综述。

**开发详解**

**1.系统功能结构划分综述**

系统分为**APP**和**后台服务器**两部分。

**2.APP端的主要功能**

1）APP端主要通过后台提供的API接口，**获得一系列的信息**，例如借款详情信息，用户账户信息等，**展示在APP的界面上**。

2）同时APP也是**信息提交的渠道入口**，例如我们在APP的注册页面的一系列输入框，APP会在信息提交到服务器后台之前对信息进行一次合法性的校验（JavaScript正则表达式校验，例如在输入用户名邮箱的时候，邮箱是否符合格式，密码长度是否符合要求，在要输入数字的地方是否输入了非法字符等），而后通过**HTTP**请求的方式，提交到后台相应的接口。

**3.后台服务器的主要功能**

1）后台提供了一系列的**服务接口**，例如提供给登录验证的接口，注册的接口，这些接口都通过HTTP服务调用的方式开放给了使用者。例如在注册功能中，我们填写相应的注册信息，在APP进行了首次数据合法性的验证后，后台API接口会对接受到的用户名密码等信息做二次验证，通过JAVA的正则表达式匹配的方式，如果验证通过，则会进行下一步后台流程，注册功能中就对应的是，向mysql数据库中插入一条新的用户信息，通过hibernate框架提供的方法，具体是hibernateTemplate.save()方法。

2）后台提供了对一系列**数据实体的增删查改的操作**，比如用户信息实体，借款合同实体，账户实体等。怎么实现对数据实体的操作的，这一切都交给了hibernate框架进行管理，我们为了快速开发不用关心其里面的细节，hibernate框架提供了相应的方法来对数据进行存取，详见第一部分hibernate的相关操作。

**4.redis的使用**

**1）为什么选用redis**

我们选用redis主要是为了尝试一个新技术，且redis拥有**极快的存取速度**，可以很好地满足我们系统中的验证码和登录信息的存取要求，这些信息具有短时高流量和过期失效的特性，所以我们选用redis。

2）**使用redis的主要流程**

**\* 注册验证码**

用户填写注册请求，点击获取验证码的按钮，后台根据提交的邮箱信息，利用随机函数随机生成一个4位数字的随机值，通过邮件的方式，将4位验证码发送至指定的邮箱，同时会在redis中新增一条验证码记录，并设置其时效性为15分钟（怎么设置，redisTemplate中有相应的方法,set方法，如果老师问到这一层，你就说是工具代码封装好了的，我做系统的，没有对源代码细节进行过研究。）用户收到邮件后输入正确的验证码，提交注册请求，后台校验验证码的正确性（怎么验证，将提交的验证码在redis数据库中进行查找，如果找到该验证码，且该验证码是属于这个用户的，则验证通过。怎么查找，也是代码封装好的，调用get方法就行了。是什么代码封装的，就说是spring中提供的操作redis的模板类，redisTemplate类），如果不正确，返回错误信息给APP，如果正确，注册成功，在mysql数据库中持久化存储一条新用户信息，APP页面跳转到登录页面，用户就可以通过新注册的用户名和密码登录系统了。

**\* 登录信息**

用户每一次登录成功，我们都会返回给其一条唯一的随机字符串token，这个token对应了已登录的用户id信息，用户每一次访问系统都需要提供这个token字符串，token是用户合法性的唯一验证，对应于钥匙的概念。我们将token和用户id作为一个键值对放在redis中，每一次我们都会在redis中根据token查找用户id，如找到，说明token有效，用户登录合法，如否，用户登录不合法。

**3）系统操作redis数据库的方式**

通过spring框架提供的模板类redisTemplate对redis数据库中的数据进行增删查改操作，通过在spring的配置文件中配置redis数据库的用户名密码，ip，端口等信息，spring框架在初始化的时候，就会连接上redis数据库了，而后可以对redis数据库进行系列操作。

**5.spring框架的使用**

spring在本系统中的主要作用是对象生命周期和控制反转的作用，相应概念见第一部分。我们要使用某个类的某方法的时候，直接从spring的容器中取出该对象进行使用即可。这里的容器只是一个逻辑概念，你就想象成一个大碗，里面装的是所有spring的管理的对象，我们 需要使用哪个对象，从碗里去取就行了。

第二个使用到的场景就是我们使用**SpringMVC模块实现了系统的MVC三层模型**。SpringMVC是Spring中的一个模块功能。MVC三层框架，M-（model）模型层，V-（view）视图层，C-（controller）控制层。

**6.hibernate的使用**

在spring的配置文件中，配置hibernate的相关信息，可以通过spring提供的对hibernate进行操作的模板类**hibernateTemplate**对对象实体进行增删改查的操作，对应于**mysql数据库中的数据实体的增删改查**，从而实现了对数据的一系列操作。具体方法名详见第一部分hibernate相关介绍。

比如在新增一个借款请求后，系统会首先验证用户的登录合法性，然后验证用户的信息的合法性，然后是验证用户是否有借款资质，最后通过**hibernateTemplate类提供的save方法**，将一个全新的借款请求实体插入到数据库中，下次用户请求获取所有借款请求合同的时候，就能看到该用户的新增加的借款请求了。

**7.系统注册和登录模块**

系统注册和登录模块主要提供了系统的注册和登录功能。

**登录**

**使用到的技术：**

Java基础语法（其中用到了Java的集合对象对数据进行存取），正则表达式验证数据合法性，Spring， Hibernate框架的编程，MySQL数据库相关操作（数据的增删改查）。

**实现过程：**

用户通过APP的相应的输入框提交用户名和密码，点击登录按钮，APP通过JavaScript中的正则表达式功能验证用户名和密码是否合法（例如用户名是否为空，是否满足邮箱的格式，密码长度是否大于5位数），通过http请求的方式提交给后台服务器的登录API

http请求通过AJAX请求的方式，异步提交大服务器进行验证登录请求，如图：

图 AJAX请求提交代码

后台服务器的对应方法接收到登录请求提交的用户名和密码数据后，会再做一次数据合法性校验，也是通过Java中的正则表达式匹配的方式做的。数据不合法，给APP相应的报错，合法则进行如下流程：在数据库中查找hibernateTemplate.find(“from UserInfo u where u.username=?”)是否有该用户名的用户存在，如不存在，登录失败，返回APP错误信息。如存在，**密码单向加密技术：**将用户提交的用户名进行MD5加密，使用的加密方式是java JDK中实现的MD5加密方法，比对该用户名的用户密码和对提交的用户密码进行MD5加密后的密码是否相等，如果相等，登录成功，反之登录失败，返回给APP相应的登录结果。

**这里针对ajax通过http异步提交请求做一个说明，下文的所有请求提交都是通过ajax提交的，将不再做赘述。**

ajax也只是一种技术名词而已，其功能是可以提交数据到服务器，在网页不跳转的情况下。我们在登录页面输入的用户名和密码，点击登录，就是通过ajax异步提交到服务器的，服务器验证登录是否通过，然后返回给app做相应反应。

这里的异步的概念是相对的，一般的数据提交后，对应的网页会跳转到另一个页面，这里我们并没有跳转，所以是在当前页面完成了数据提交，所以是异步的。

**记住，简单回答就是，通过http的方式，将数据提交到服务器了。**

**注册：**

基本过程类似登录过程，首先填写表单，而后APP对表单数据进行合法性校验，而后通过http请求的方式提交给后台API接口，API接受到数据对数据进行合法性校验，而后使用hibernateTemplate对将新数据插入到数据库中，一个注册流程结束。

所以我们能看到，无论什么模块的实现过程就是首先APP对数据合法性进行合法性校验，而后服务器进行数据的二次校验，然后下一层就是操作数据库了（如果是类似登录这个请求就是查找数据库，进行数据比对，如果是注册这类请求，就是将新的数据插入到数据库中）。

前端🡪后端🡪数据库这样的几层，所以不是每一个模块的实现都是不同的，相同的是技术实现方案，不同的是业务逻辑而已。这也是为什么我的第一版文档后面都略写了的原因，因为技术实现细节是一直的，不同的是业务逻辑而已。

**8.个人账户模块**

**账户资质相关：**

账户资质主要实现了对贷款资质的提交功能，这里不同于其他类似登录注册这类信息提交的地方在于，这部分需要对图片文件进行提交，例如用户的身份证和学生证照片。

**资质上传过程如下：信息上传🡪文件另存🡪用户资质信息更新（数据库）**

APP验证填写的信息是否合法，点击提交按钮，会将选择的图片文件以**字节流**的方式通过网络上传到服务器，服务器在接受到这些信息后，会首先验证提交信息的合法性。而后将用户上传的图片文件另存到服务器指定的文件夹（这里我们存在F:/fileupload文件夹），并会通过hibernateTemplate.save保存一条图片的文件信息，里面包含图片名称，图片大小，图片类型等信息，而后将两个资质图片信息和用户进行关联，主要是通过hibernateTem.update更新用户的账户信息中的身份证照片和学生证照片两个字段属性。自此，一次账户资质提交的请求结束，等待管理员审核这一条资质请求。

**管理员审核资质请求:数据库查找用户信息🡪更新设置资质属性的值**

登录进管理员关系系统，进入资质管理功能，会以列表的形式呈现所有等待资质审核的用户信息（怎么以列表形式呈现出来的，利用hibernateTemplate的find方法，查找所有的待审核的用户的资质信息，将信息返回给后台网页界面做展示，怎么返回的，也是通过http请求进行传输的，怎么传输的，通过httpResponse.write()方法写入的，怎么写入的，这是http协议规定的了），也能查看提交的资质照片，管理员审核通过，点击相应的审核通过按钮，会更改用户信息的“是否有借款资质”的属性为1表示能够发布借款请求。自此，一次管理员资质审核流程结束。

**账户充值提现**

1.我们的充值和体现只是一个虚拟的功能，并没有相关资源做真正的支付验证。

2.账户充值和提现首先需要绑卡。

用户点击进入账户页面的时候，服务器会首先查找该用户是否有绑定银行卡（怎么查找的，hibernate操作数据库查找的，所有要获取的相关信息，只要是从mysql数据库中获取的，都是通过hibernate框架帮助我们完成的，那几个方法详见第一部分），如果没有，会给出提示，提示其进行绑卡操作，并跳转到绑卡页面。绑卡页面同样是输入银行卡号和预留的手机号码等基本信息，app数据验证，服务器数据验证（验证什么，验证数据的卡号是不是符合卡号的格式呀，卡号不能有汉字，卡号最低我们假设15位），数据库持久化。而后在mysql数据库中新增一条绑定在该用户名下的银行卡数据。如果绑定有银行卡，则用户有权限进行相应的充值和体现操作。

**虚拟充值功能**，用户输入相应的充值金额，后台产生一条在该银行卡下的扣款记录，怎么产生，新建一个扣款记录对象，设置其对应的金额和卡号信息，然后通过hibernate将记录保存在mysql数据库中。而后设置账户余额增加相应金额，怎么设置，首先通过hibernate在数据库中取出该账户基本信息，将余额这个属性设置为最新的值，然后通过hibernate将这个对象重新更新到数据库中，这样该账户余额增加相应的金额。

**虚拟提现功能**，用户输入相应的提现金额，后台首先验证金额是否超过余额，怎么验证，首先通过hibernate取出账户对象，得到其目前余额是多少，然后我们能拿到通过http请求提交过来的提现金额，两个金额做一个大小比较，怎么比较if(提现金额>账户余额) 提示超限，一条代码语句比较的。如是，给出相应的提示，如否，则在该用户该银行卡下产生一条相应的提现记录（怎么产生的，在虚拟充值里面我已经做了详细介绍，不做赘述），用户账户余额减少相应金额。

**投资相关**

**借款**

用户如果需要在平台上发起一条借款请求，后台在用户打开借款信息填写页面之前，会验证用户是否有相应的借款资质（怎么验证，通过hibernate取到用户信息对象，查看器借款资质属性是否为有借款资质，怎么查看，这个属性里面写着的就是是否有借款资质，如果属性值为1有资质，为0无资质），如果没有，则给出相应提示，跳转至借款资质申请填写页面。如果有，则顺利打开借款请求申请页面，用户填写相关的借款信息，同样是分APP前端和后台服务器两步验证（怎么验证，正则表达式验证，正则表达式是什么，上文做过详细介绍了）。而后同样是通过hibernateTemplate类在数据库的Agreement表中新插入一条数据，里面保存了用户此次借款的相关信息，自此，一次借款请求完成。

**投资**

用户可以在平台上针对某一借款进行投资相应的金额，投资过程如下。

用户查看相应的借款详情，借款详情通过hibernateTemplate的find方法，从mysql数据库中查找到相应的合同详情信息，返回给APP进行展示，怎么返回http协议网络传输至app。挑选自己决定投资的借款，输入相应的投资金额，点击确认，后台会验证用户账户余额是否充足，如金额超限（怎么验证超限，和虚拟提现功能中验证是一样的，首先通过hibernate获得账户信息，然后获得余额，然后通过if语句比较两个金额的大小得出，所以我第一版没写，因为技术细节都是一样的，不一样的是业务需求），给出相应提示。验证该借款请求的待筹集金额是否小于投资金额，如是，给出提示。如不是则根据账户-合同模型执行如下流程：

\*产生一条投资类型的合同请求并保存（怎么产生：new一个合同请求对象，里面设置相应的合同信息，然后通过hibernate保存到mysql数据库中，下面的四条都是一模一样的方式，后面将不做赘述），对应于投资者看中且投资的合同；

\*产生一条投资者用户的账户扣款请求并保存，对应投资金额，用户账户余额减少相应金额（怎么减少，首先通过hibernate从数据库中获取其账户对象，将其余额加上相应的金额，怎么加：之前余额+新增加的量=新余额，再通过hibernate更新到数据库中即可，***下面账户金额变动都是通过这种方式，将不做赘述***）；

\*产生一条借款合同所有者的账户充值请求并保存，对应投资金额，用户账户余额增加相应金额；

\*借款合同的已筹集金额和待筹集金额做分别的改变，怎么改变，和上面的红字斜体部分是一样的，做类比。首先取得合同对象，获得已筹集金额和待筹集金额，通过加法设置最新的值，通过hibernate更新至mysql数据库中。

**好友管理模块**

好友模块主要功能为：上传通讯录至服务器，服务器根据通讯录号码匹配平台注册用户的手机号码，如匹配成功，则自动建立好友联系。同时可以在好友页面看到该用户下的好友列表，点击对应好友列表的一项，可以看到此好友的所有借款信息。

**好友通讯录上传功能**

首先通过获取到智能手机的通讯录信息（怎么获取到的，安卓手机提供了相应的功能接口，一句代码就都获取到了），迭代所有联系人（迭代就是一个for循环，C语言基础那种for循环），将每个联系人的电话号码拼接起来，组成一个包含有所有联系人的电话号码信息，以空格形式隔开的字符串，通过HTTP请求的方式传递给服务器，服务器在接受到这串字符串后，会以空格作为分割，将所有的联系人电话号码存入到一个ArrayList列表中，而后对列表进行循环，在数据库中匹配查找系统中是否有对应电话号码的注册用户，如果找到，则新增一条好友关系信息（new一个好友关系对象），通过hibernate的save方法将对象保存至mysql数据库中。

**好友借款请求功能**

通过对应的好友列表中的电话号码，查找好友的账户信息（hibernate查找find），得到其所有的借款请求（hibernate的find，但是这里有两次查找，第一次是通过好友的用户id查找到期对对应的账户，通过账户查找到对该账户下的所有借款请求，但是技术上都是通过find方法查找的），new多个借款请求对象，包含多个借款请求，将其存入到一个java列表ArrayList中，将这个arraylist转换为json字符串，返回给APP进行详情信息的展示，怎么展示，app会根据获取到的json字符串的属性的值，将其获得到的值得相应的属性展示到对应的页面位置上。

**邮件发送功能**

系统的邮件发送时通过使用JavaMail的开源代码实现的，通过配置服务邮箱的服务器地址和用户名密码即可实现邮件的发送功能。具体实现细节可不做讨论，就说是用的开源的JavaMail的功能即可。

**定时扫描超过还款期限的借款功能**

系统可以通过配置该功能的周期运行规则，实现定时地按照配置规则的运行，扫描超过还款期限的借款请求的功能，并执行发送对应邮件进行提醒的功能。

定时任务的实现是通过quartz开源框架的方式实现的，通过简单的配置任务的执行规则和要执行的动作即可。

**怎么判定超过还款期限：**

每一个借款信息中都包含其借款创建日期，借款期限信息，通过简单运算（创建日期顺延期限天数即可得还款日期，如果此刻在还款日期之前，则判定超过还款期限）即可得。

**应还金额的计算公式**

每一个借款合同有可能有N个投资人，所以也应该对应N个还款人，所以在还款时，应该查找该借款合同下的所有投资请求，通过年利率\*期限\*投资额+投资额的方式计算得到每一笔投资请求的应还金额，依次还款到每一个投资人的投资账户。

**信用评级**

信用评级功能是系统评定某用户的信誉度的标尺。

用户初始信用分为0，默认等级为A

用户按期还款一次，信用分+10分

用户逾期还款一次，信用分-20分

信用等级和分数的对应规则如下：

**if** (score >= 0 && score < 50) {  
 **return** "A";  
} **else if** (score >= 50 && score < 100) {  
 **return** "A+";  
} **else if** (score >= 100) {  
 **return** "S";  
} **else if** (score >= -10 && score < 0) {  
 **return** "B";  
} **else if** (score >= -50 && score < -10) {  
 **return** "C";  
} **else if** (score >= -100 && score < -50) {  
 **return** "D";  
} **else if** (score < -100) {  
 **return** "E";  
}