# 第四章借款模块的详细设计与实现

## 4.1 借款模块概述

开鑫贷网贷系统的借款模块是为了帮助借入人，借出人以及担保机构在网上完成借款申请、投标借出、合同签订以及最终还款而设计的模块，是开鑫贷网站的核心模块。借款模块的主要功能有：借入人发布借款，借出人投标，担保机构对借款进行担保，审核专员审核借款，三方订立合同，查询借款，借入人还款等。借款模块涉及到前台和后台，前台主要是借入人、借出人和担保机构操作的部分，比如发布借款、投标等等；后台主要是开鑫贷的客服、审核人员操作的部分，比如查询借款、审核借款等等。通过借款模块，借入人、借出人和担保机构三方可以在网上完成P2P借贷的业务，将线下的借款交易完全转移到线上，提高了借款业务的效率，使得借款相关的数据可以更易保存和查询。

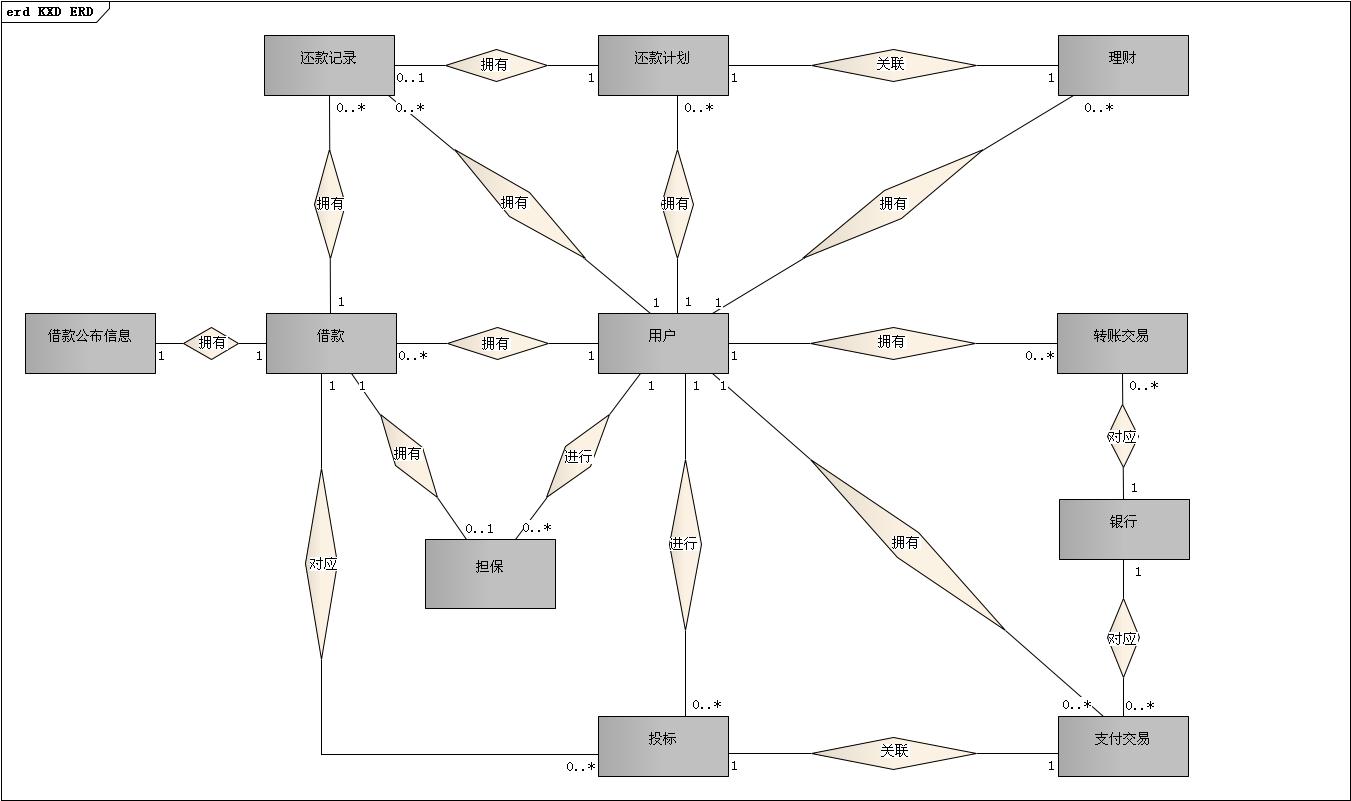
## 4.2 借款模块的详细设计

### 4.2.1 数据库设计

**数据库E-R图**

根据前面的分析，借款模块涉及到的概念实体有：用户，借款，投标，担保，理财，还款，支付交易，转账交易，银行。其中还款可以分成还款计划和还款记录，还款计划是在借入人还款之前的计划表，还款记录是借入人真实还款的记录表。借入人发布借款信息时需要选择哪些信息可以披露出去，因此还可以派生出一个弱实体，即借款公布信息。投标与借款具有对应关系，一个借款可以有零个到多个投标；担保与借款是拥有关系，一个借款拥有零个或者一个担保；理财与还款计划相关联，一笔理财对应一个还款计划。用户依然是整个借款模块数据的核心，多个实体都与用户相关联。借款和投标数据是借款模块的关键数据，因为它们记录了借出人借了多少钱，期限是多少个月，担保机构是哪家，年化利率是多少，借入人投了多少钱等等借款相关的关键信息。

经过以上分析，得出借款模块相关的数据库的实体关系图，如图4.1所示。

图4.1 借款模块相关数据库E-R图

**数据库表结构图**

经过上面对实体关系的分析，确定出借款模块相关的数据库的表，它们是：T\_XTYH，T\_DK\_DKXX，T\_DK\_DKGBXX，T\_DB\_DBXX，T\_LC\_LCXX，T\_DK\_TBMXXX，T\_DK\_PTHKJL，T\_DK\_HKJHMX，T\_TRADE\_TRANSFER，T\_BANK，T\_TRADE\_ORDER。其中，表T\_XTYH表示系统用户表，表T\_DK\_DKXX表示借款申请信息表，表T\_DK\_DKGBXX表示借款公布信息表，表T\_DB\_DBXX表示担保信息表，表T\_LC\_LCXX表示理财信息表，表T\_DK\_TBMXXX表示投标明细信息表，表T\_DK\_PTHKJL表示平台还款记录表，表T\_DK\_HKJHMX表示还款计划明细表，表T\_TRADE\_TRANSFER表示平台支付交易表，表T\_BANK表示银行表，表T\_TRADE\_ORDER表示平台转账交易表。

借款、担保、投标、支付、转账、理财、还款计划、还款记录这些表中有一个用户表的外键，记录着交易对应的用户。投标、还款记录、借款公布信息、担保这些表中有一个借款表的外键，记录这些信息对应的借款项。还款计划中有理财表的外键，记录还款计划是针对那一笔理财的。支付交易表中有投标表的外键及银行的外键，记录每笔支付交易对应哪个投标以及哪家银行的银行卡。转账交易中有银行的外键，记录每笔转账交易对应哪家银行的银行卡。以上就是借款模块相关的表的结构，如图4.2所示。

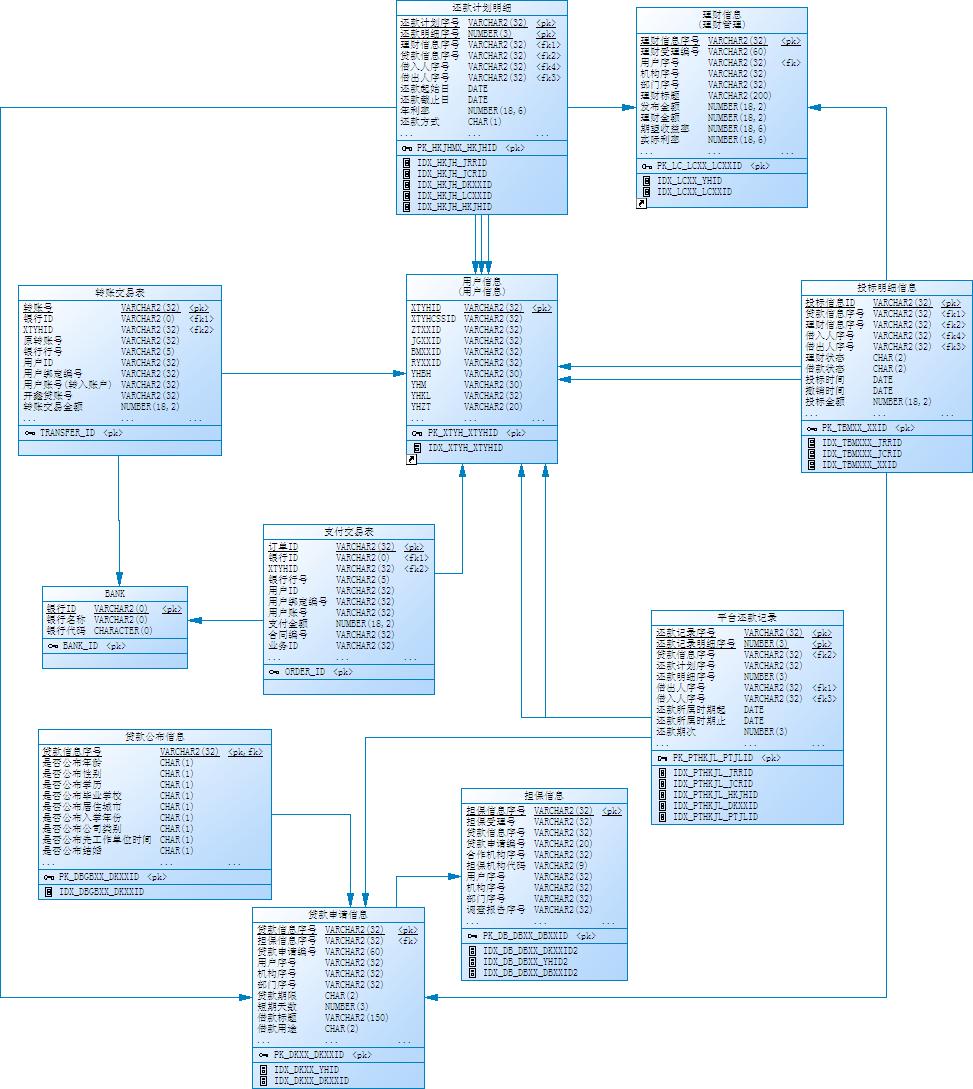


图4.2 借款模块相关数据库表结构图

**数据库表的详细结构**

借款模块相关的数据库表共有十一个，其中借款申请信息表最为重要，这里只详细列出表T\_DK\_DKXX的详细结构，包括表的列名，类型，用途，是否可以为空，是否是主键等。表T\_DK\_DKXX的详细结构如表4.1所示。

表4.1 T\_DK\_DKXX的详细结构表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 列名 | 类型 | 用途 | 是否可以为空 | 是否为主键 |
| DKXX\_ID | VARCHAR2(32) | 借款申请信息序号，表示借款的主键。 | 否 | 是 |
| DBXX\_ID | VARCHAR2(32) | 担保信息序号，表示借款对应的担保。 | 是 | 否 |
| DKSL\_ID | VARCHAR2(60) | 借款申请编号 | 否 | 否 |
| YH\_ID | VARCHAR2(32) | 用户编号，表示借款的借入人ID。 | 否 | 否 |
| DKQX | CHAR(2) | 借款期限，表示多少个月之后还款。 | 否 | 否 |
| DKBT | VARCHAR2(150) | 借款标题 | 否 | 否 |
| DKYT | CHAR(2) | 借款用途 | 否 | 否 |
| DKCJJE | NUMBER(18,2) | 借款成交金额，表示借款实际成交的金额。 | 是 | 否 |
| DBFY | NUMBER(18,2) | 担保费用 | 是 | 否 |
| DKJE | NUMBER(18,2) | 借款金额，表示借入人的意向借款金额。 | 否 | 否 |
| NLL | NUMBER(18,6) | 年利率 | 否 | 否 |
| HKFS | CHAR(1) | 还款方式，有“提前还款”和“正常还款”两种。 | 是 | 否 |
| HKR | NUMBER(2) | 还款日 | 是 | 否 |
| DKMS | VARCHAR2(1000) | 借款描述，借入人发布借款时要输入借款描述。 | 否 | 否 |
| SFDB | CHAR(1) | 担保机构是否担保 | 是 | 否 |
| DBJG\_DM | VARCHAR2(9) | 担保机构代码 | 否 | 否 |
| DBGS\_ID | VARCHAR2(40) | 担保机构序号 | 否 | 否 |
| DBDCQX | NUMBER(2) | 担保调查期限，表示担保机构在哪一天之前完成对借入人的线下审核。 | 否 | 否 |
| FB\_SJ | DATE | 借款的发布时间 | 否 | 否 |
| CJQMR | DATE | 筹集期满日，表示筹集阶段的截止日期 | 否 | 否 |
| DKQSRQ | DATE | 借款起始日(计息开始日) | 否 | 否 |
| DKDQRQ | DATE | 借款到期日(计息结束日) | 否 | 否 |
| DKCLRQ | DATE | 借款成立日期 | 否 | 否 |
| SFMECL | CHAR(1) | 是否不满额成立，借入人发布借款时选择，如果同意不满额成立则只要筹集到百分之八十以上的借款就可以成立。 | 否 | 否 |
| SYJE | NUMBER(18,2) | 剩余金额，表示还需要多少钱才能满额。 | 否 | 否 |
| CXRQ | DATE | 撤销日期 | 是 | 否 |
| CXLY | DATE | 撤销理由 | 是 | 否 |
| DKZT | CHAR(2) | 借款状态 | 否 | 否 |
| DBHTH | VARCHAR2(20) | 担保合同号 | 否 | 否 |
| HTDJF | NUMBER(18,2) | 合同登记费 | 否 | 否 |

### 4.2.2类的设计

**借款模块的子模块设计**

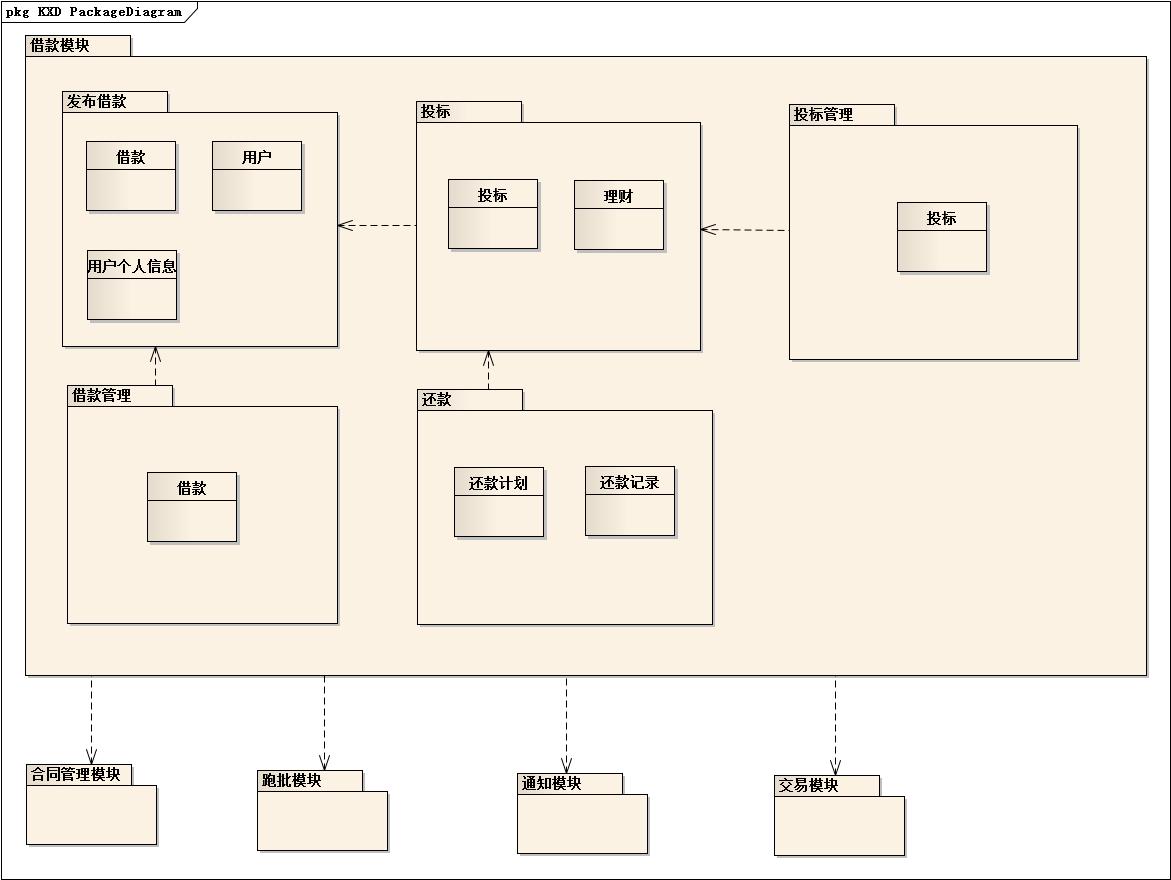
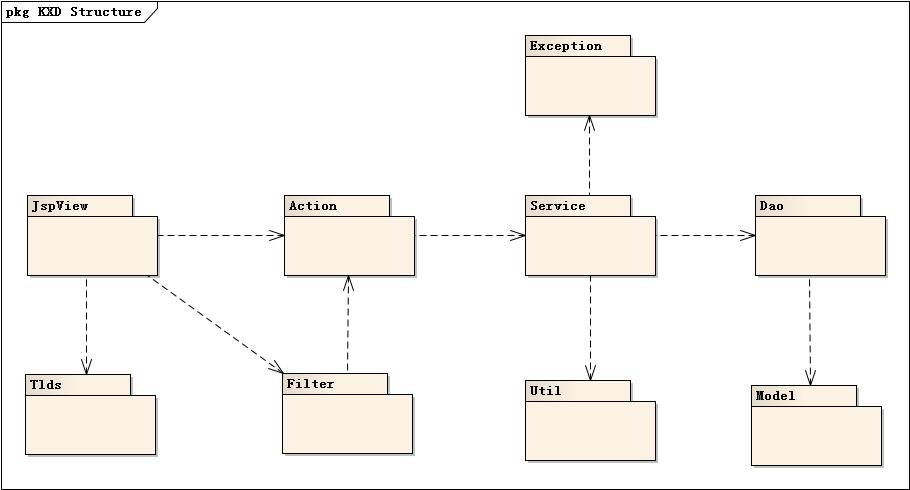
**** 借款模块按照功能可以分为发布借款、投标、还款、投标管理、借款管理这些子模块，发布借款、投标、还款子模块在前台，借款管理和投标管理在后台。投标子模块和借款管理子模块依赖于发布借款模块，只有发布借款之后才能投标并对借款进行管理。还款子模块和投标管理子模块依赖于投标模块，只有投标之后才能进行还款和投标管理。在进行借款流程中的合同预览及合同签订步骤时，需要依赖合同管理模块的功能；在发送通知时需要依赖通知管理模块的功能；在投标及还款支付时，需要调用交易模块的功能；借款的筹集期满和成立时都需要跑批模块跑批之后才能进行状态的改变。如上所述，借款模块的包图如图4.3所示。

图4.3 借款模块子模块包图

**类的总体结构设计**

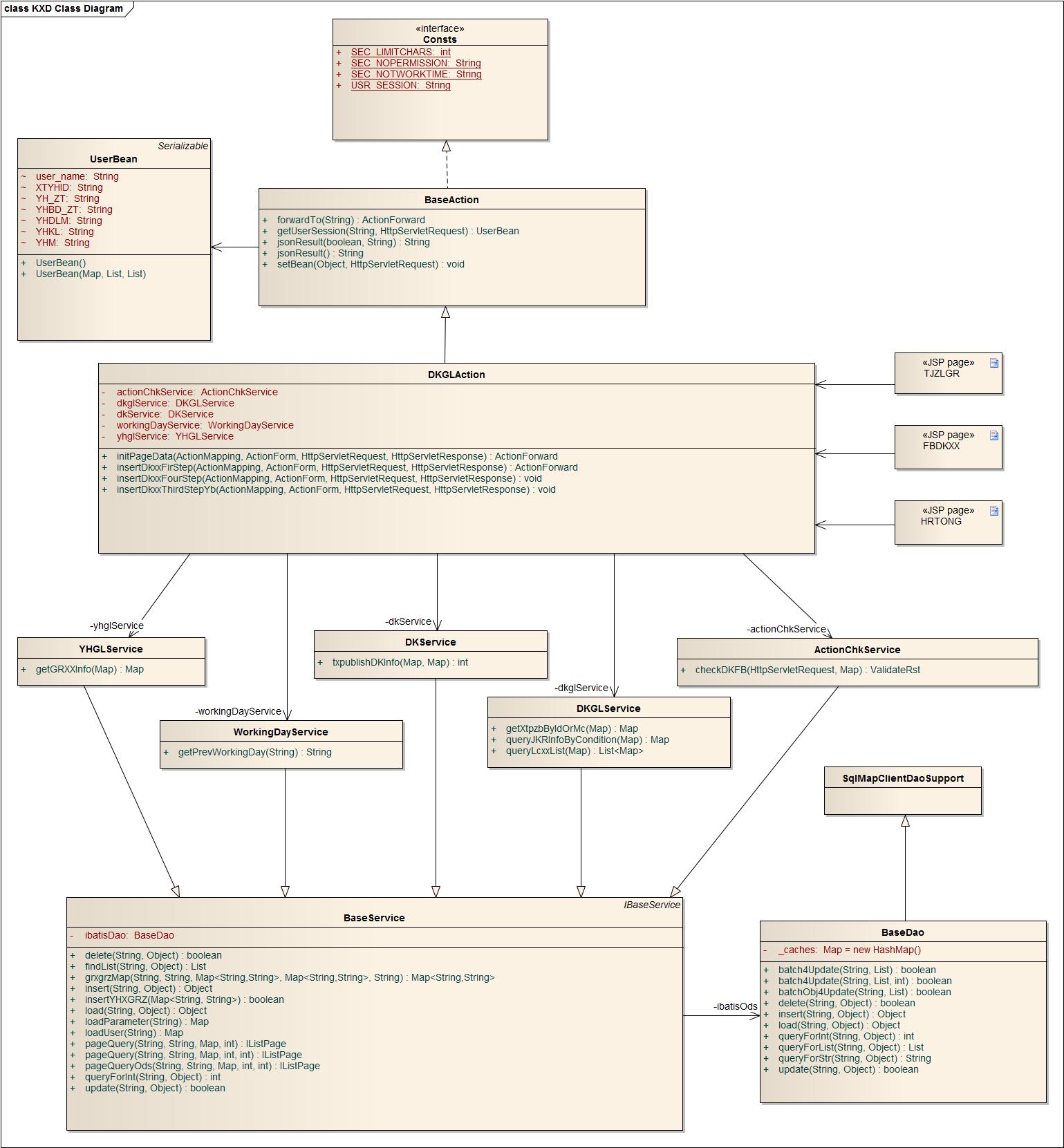
借款模块的类的总体结构设计遵循了“高内聚，低耦合”的设计思路，将表现层，业务逻辑层，数据访问层相分离。本模块的类的结构大概可以分为四层：数据持久层，业务逻辑层，控制层，表现层。表现层大多是JSP文件，由于JSP文件在运行时会被编译成Servlet，这里把JSP也当做类来看待。表现层侧重于页面的展现，使用DIV+CSS的方式布局，在一些需要在页面进行逻辑处理的地方使用了自己实现的标签。控制层主要是Struts1的ActionServlet和一些Action类以及一些Filter类，Struts1通过配置文件来决定用户访问的URL调用到哪个Action以及响应的URL对应到哪一个页面，请求参数通过ActionForm来封装，请求跳转通过ActionForward来封装。业务逻辑层包括一些Service类和一些工具类以及一些异常类等等，通过配置文件，Service类被注入到需要使用它的Action中，表现层的请求被Action转给Service去处理，并将Service返回的数据传递给表现层相应的页面来展现。数据持久层包括一些Dao类以及一些Model类，主要通过iBatis框架来实现，在XML配置文件中将Model和Sql语句对应起来。如上所述，类的总体结构图如图4.4所示。

图4.4 借款模块类的总体结构图

**发布借款功能类的设计**

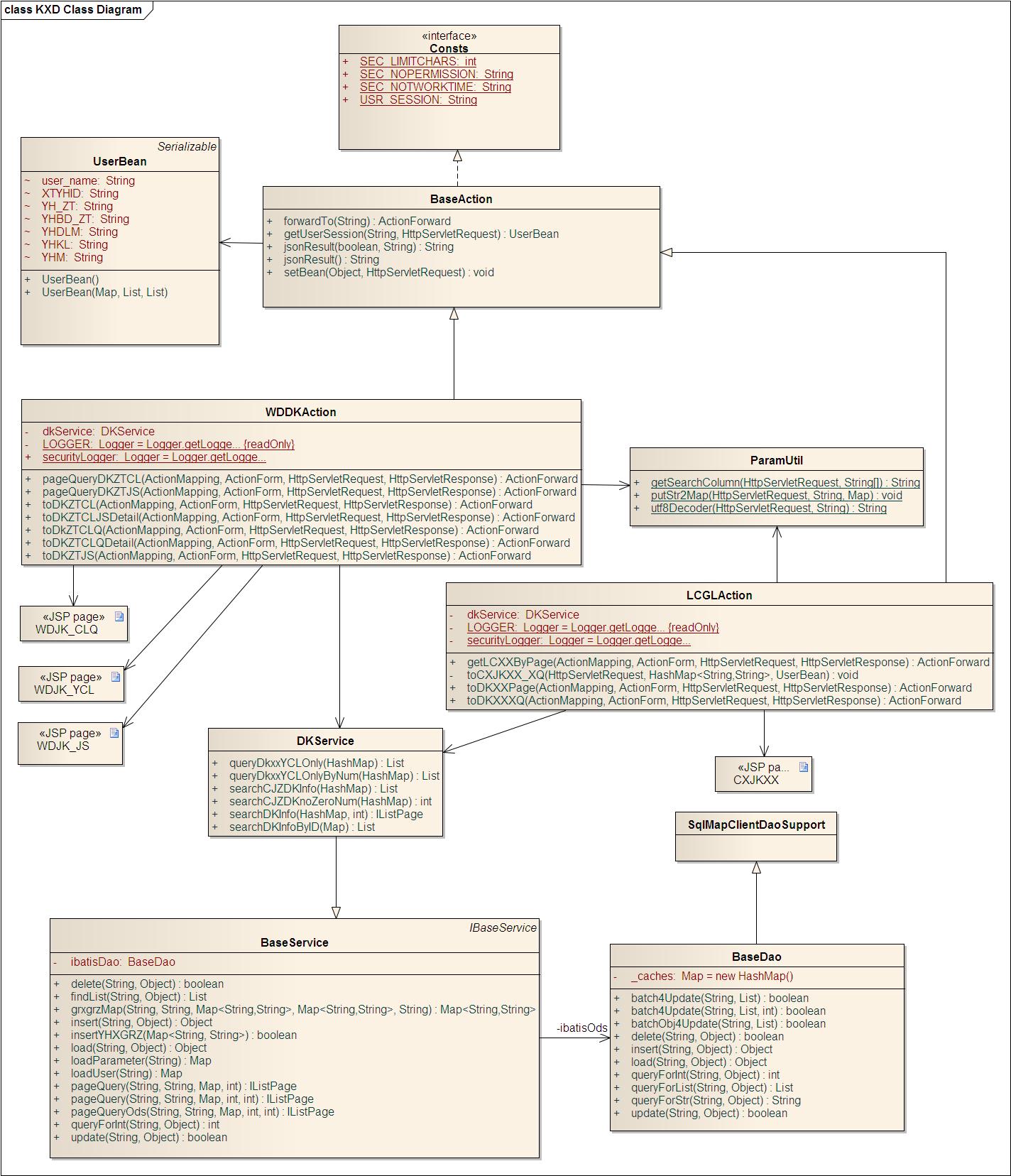
发布借款功能需要借入人首先进入“发布借款申请”页面，填写借款申请基本信息并选择担保机构，接着点击“下一步”进入“上传资料信息”页面，完成资料上传后点击“下一步”进入“合同预览”界面，预览合同后点击“确定”按钮便完成了借款的发布。对借款的流程进行分析，系统需要“发布借款申请”、“上传资料信息”、“合同预览”这三个页面，分别用FBDKXX.jsp，TJZLGR.jsp和HETONG.jsp三个JSP文件来表示。来自这些页面的请求会被Struts1转发到DKGLAction这个类中，因为发布借款是要经过三个页面才能成功发布的，所以每个页面用户输入的信息都要临时保存起来，可以通过Map来保存，这样用户如果要回到上一步他之前输入的信息也可以保存下来，无需再重新输入。在处理用户信息的时候DKGLAction需要调用YHGLService，在处理工作日相关的信息时DKGLAction需要调用WorkingDayService，在获取担保信息等借款相关的信息的时候DKGLAction需要调用DKGLService，在后台对表单进行验证的时候DKGLAction需要调用ActionChkService，最后发布借款的时候DKGLAction调用DKService的txpublishDKInfo方法。DKService等Service继承了BaseService，BaseService中有BaseDao成员，通过调用BaseDao的insert方法将借款信息添加到数据库中。

如上所述，发布借款功能的类图如图4.5所示。

图4.5 发布借款功能类图

**查看借款功能类的设计**

查看借款功能指的是借入人可以查看自己的成立前的、已成立的和已结束的借款，投资人可以查看所有可以投标的借款。这一类查看借款的功能实现的方法大致是一样的，只不过查询的条件不同，因此提供查询服务的Service类可以公用，供不同的Action类调用。查看借款功能包括查看借款列表和查看某个借款的详细信息，这些功能都由DKService中的方法去实现，WDDKAction接收借入人的查看借款的请求，LCGLAction接收借出人的查看借款的请求。查看借款功能的类图如图4.6所示。

图4.6 查看借款功能类图

如图4.6所示，WDDKAction中的六个方法满足不同条件的借款查询，toDKZTCLQ方法使页面跳转至借入人成立前的借款页面，toDKZTCL方法使页面跳转至借入人已成立的借款页面，toDKZTJS方法使页面跳转至借入人已结束的借款页面，pageQueryDKZTCL方法分页查询已成立的借款列表，pageQueryDKZTJS方法分页查询已结束的借款列表，toDKZTCLQDetail方法使页面跳转至成立前的某比借款的详情页面，toDKZTCLJSDetail方法使页面跳转至已成立或已结束的某比借款的详情页面。LCGLAction中的四个方法满足借出人对借款的查询需求，toCXJKXX\_XQ方法是私有方法，可以跳转到借款详情页面，这个私有方法供别的方法调用，toDKXXXQ方法首先对借出人进行权限验证，验证通过后调用toCXJKXX\_XQ方法跳转至借款详情页面，这两个方法共同实现了借出人查看借款详情的功能，getLCXXByPage方法分页显示筹集中的借款，toDKXXPage方法使页面跳转至借出人查询借款页面。DKService提供几个方法供前面两个Action调用，searchDKInfo方法根据传递的参数不同返回满足不同查询条件的借款列表，searchDKInfoById方法根据借款id查询借款信息，DKService的这些方法调用BaseService的findList、pageQuery等方法获得借款数据，而BaseService最终还是调用BaseDao的查询方法来完成对借款的查询。BaseDao继承了iBatis提供的SqlMapClientDaoSupport类，通过继承这个类可以找到配置在XML文件中的Sql语句并把参数传过去，从而完成向数据库的查找。图4.6中还有一个ParamUtil类，该类是一个工具类，会被WDDKAction和LCGLAction调用到，该类的putStr2Map方法可以从HttpServletRequest中获取参数并存放到Map中。

**对借款投标功能类的设计**

借出人查询到筹集中的借款之后，找到想要投资的借款，进入借款详情页面，点击“确认借出”按钮，这个时候系统弹出一个对话框，要求借出人输入投资标题、投资金额、投资描述，然后借出人点击“确定”按钮进入合同预览页面，借出人输入验证码后点击“确认借出”便进入银行支付页面进行支付，支付成功即完成对该笔借款的投标。借出人点击借款详情页面弹出的对话框的“确认”按钮时进入LCGLAction的toYbHtYl方法，在合同预览页面点击“确认借出”后进入LCGLAction的toWYTZHT方法，LCGLAction调用ActionChkService的checkLCYB方法对投标的参数进行验证，调用DKGLService的queryDKTBInfo方法查询已有的投标信息来判断是否是重复应标，调用DKService的searchDKInfoById来获得借款的相关信息，通过KXDTradeConfig类来获得江苏银行支付接口的URL。LCGLAction调用LCGLService的txbitDK进行投标，LCGLService的txGenerateBankOrder方法调用TradeBusinessService的generateTradeOrder方法生成支付订单，LCGLService的txNspublishLCInfo方法在数据库中生成理财信息，LCGLService还会调用UserService中的方法来验证借出人剩余投资额度是否足够，调用HTService中的方法用来生成或查询合同。

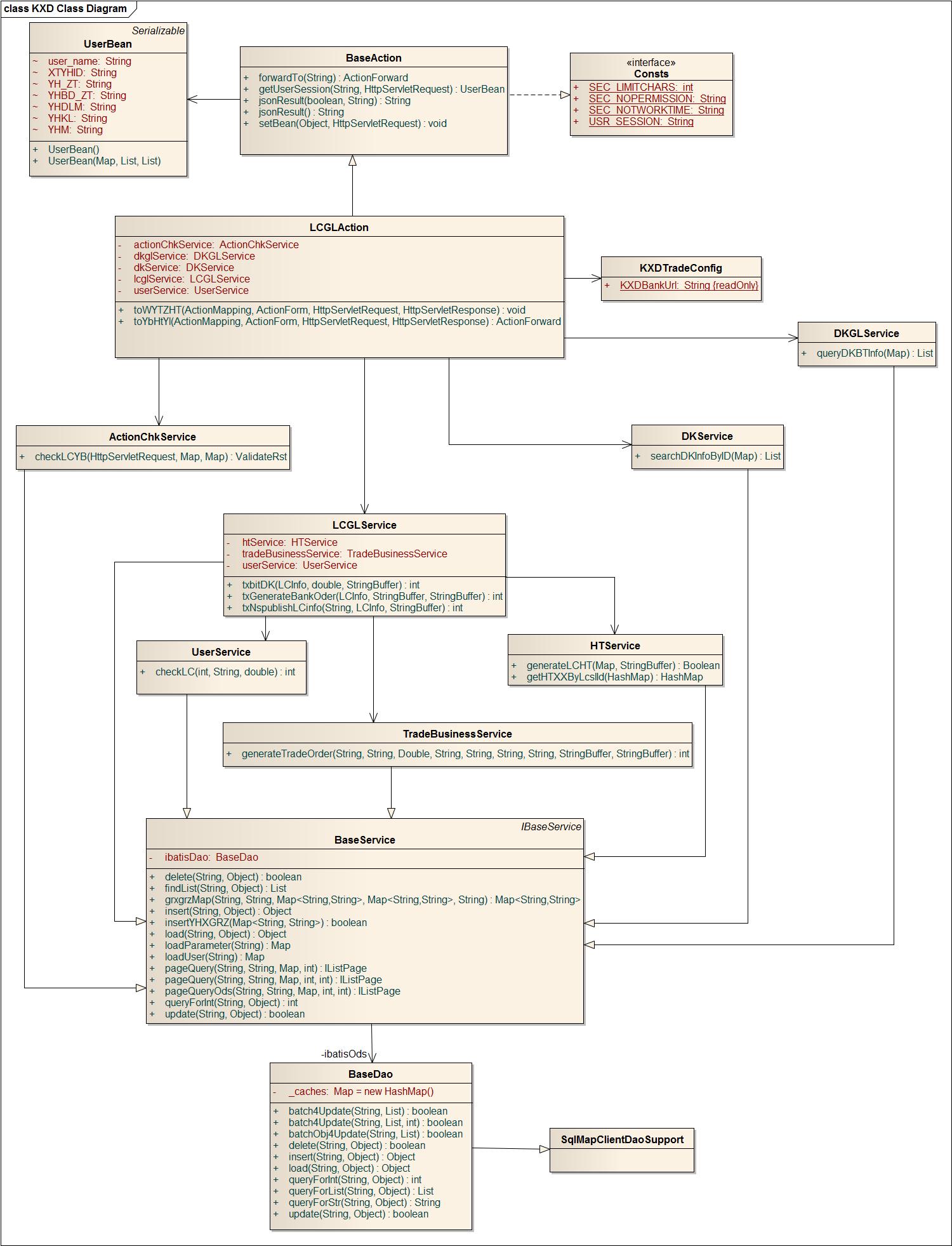
 如上所述，对借款投标功能的类图如图4.7所示。

图4.7 对借款投标功能类图

借出人对投标的支付通过调用交易模块来实现，由于网银支付必须在外网，支付请求必须通过外网让用户跳转到银行支付网关，而支付的返回结果也需要通过网银支付网关传递给开鑫贷的外网，但是出于安全考虑，向银行发送的消息需要经过加密机加密，并且加密解密模块部署在内网，而且不允许任何外网机器直接连接，这就需要解决内外网通信的问题。经过全面的考虑，我们决定使用共享数据库来解决内外网通信的问题，在数据库中建立一个支付中间表，借款模块将用户的支付请求存入这个中间表中，内网的交易模块轮询这张表，对表内的请求逐一处理，通过Bisafe加密机对支付请求进行加密处理，并将加密后的信息保存到中间表中。内网的交易模块没每隔一秒轮询一次交易中间表，借款模块将支付请求写入中间表后先等待三秒，然后查询支付请求是否已经被交易模块处理，如果尚未被处理就再等待六秒，然后再次查询支付请求是否已经被交易模块处理，如果还未被处理则交易失败。考虑到现阶段投标支付的请求的频度，采用先等待三秒再等待六秒的策略应该可以保证支付请求有充足的时间被处理，如果处理失败则可能是Bisafe加密机出了问题或与Bisafe相连的网络出了问题。通过以上分析，决定采用中间表的方式解决支付时内外网通信的问题。

**类的设计总结**

以上三个类图分别是借款模块三个不同功能的类图，不同的功能可能会用到同一个类，只是调用的类的方法不同，为了直观简洁的展示各个功能中各个类的关系，在类图中只列出了与相应的功能相关的属性和方法。图4.5展示了借入人发布借款的功能的类图，图4.6展示了借入人和借出人查看借款的功能的类图，图4.7展示了借出人投标功能的类图。以上三个功能是借款模块的核心功能，借款模块还有诸如借入人还款、借款管理等功能，这些功能的设计与这三个功能是类似的，只是在一些具体细节上有所不同，这里就不把所有功能的类的设计都一一列出了。

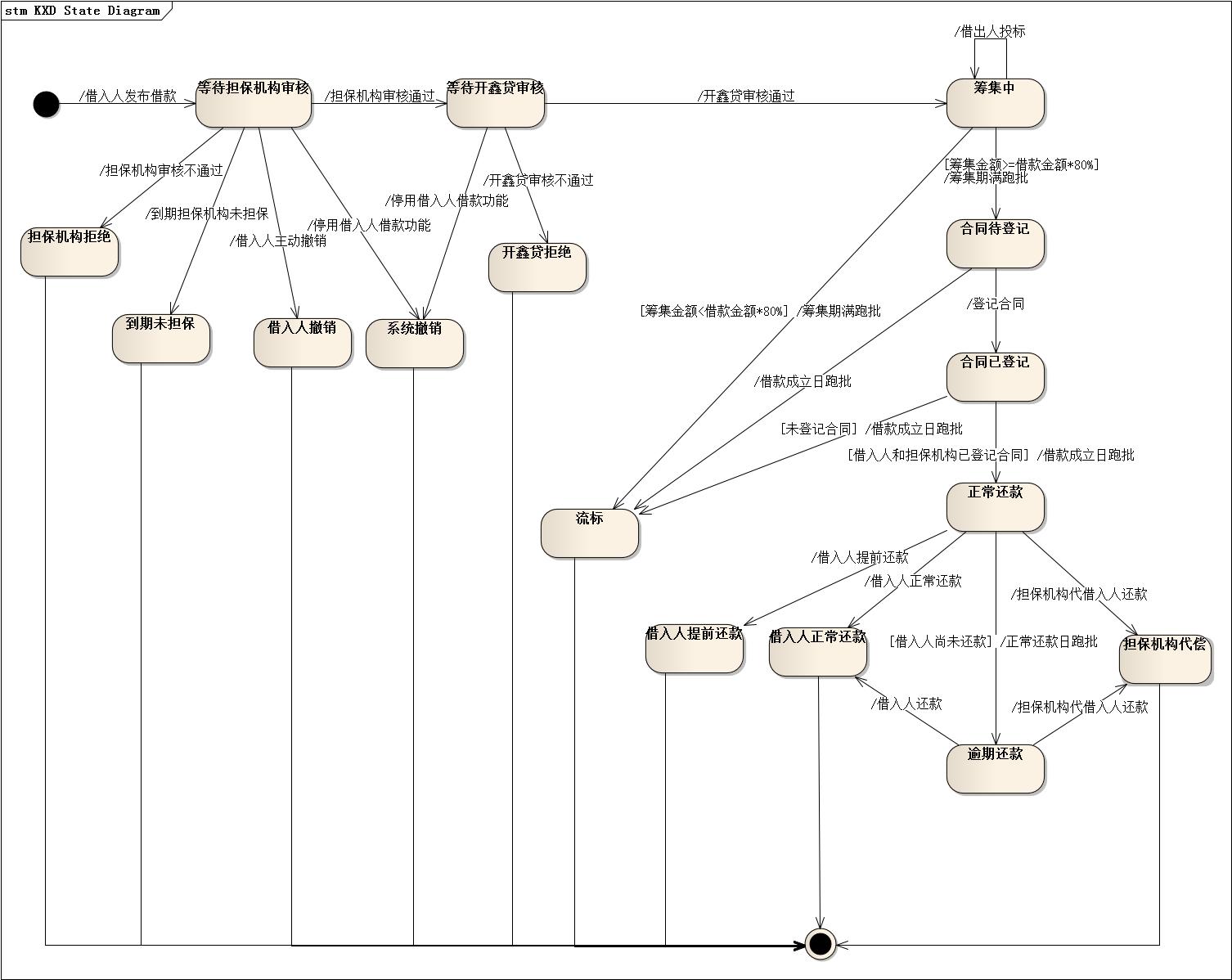
### 4.2.3 借款处理流程的设计

**借款处理流程描述**

从借入人在平台上发布借款到借入人还款之间的流程可以如下概括：首先，借入人打开发布借款的页面，输入借款相关的信息发布借款，此时借款处于“等待担保机构审核”状态。接着担保机构对借入人进行线下审核，审核通过后借款处于“等待开鑫贷审核”状态，如果审核不通过该借款将进入“担保机构拒绝”状态，这笔借款结束，如果到期担保机构未担保，则该借款进入“到期未担保”状态，这笔借款结束。处于“等待开鑫贷审核”状态下的借款会经过开鑫贷审核专员的审核，如果审核通过将进入“筹集中”状态，审核不通过将进入“开鑫贷拒绝”状态，这笔借款结束。处于“筹集中”状态的借款可以被想要投资的借出人看到，借出人对某笔借款投标后，借款还是处于“筹集中”状态，直到筹集期满日跑批，如果达到筹集标准则借款将进入“合同待登记”状态，否则进入“流标”状态，这笔借款结束。借入人和担保机构登记合同后借款进入“合同已登记”状态，借款成立日跑批之后，“合同待登记”状态的借款进入“流标”状态，该笔借款结束，“合同已登记”状态的借款进入“正常还款”状态，表示借款成立。处于“正常还款”状态的借款，如果借入人提前还款将进入“借入人提前还款”状态，该笔借款结束；如果借入人正常还款将进入“借入人正常还款”状态，该笔借款结束；如果担保机构代借入人还款则进入“担保机构代偿”状态，该笔借款结束；如果正常还款日跑批时借入人还未还款则进入“逾期还款”状态。“逾期还款”状态的借款，如果由借入人还款则进入“借入人正常还款”状态，该笔借款结束；如果由担保机构代借入人还款则进入“担保机构代偿”状态，该笔借款结束。处于“等待担保机构审核”状态的借款还可以被借入人撤销，这时候进入“借入人撤销”状态，该笔借款结束。处于“等待担保机构审核”、“等待开鑫贷审核”、“筹集中”状态的借款。如果开鑫贷后台停用借入人的借款功能，则进入“系统撤销”状态，该笔借款结束。以上便是借款处理的流程，描述了不同条件下借款状态的转化，下面将给出借款状态变化的状态图。

**借款状态变化的状态图**

如上面的借款流程描述所述，在开鑫贷系统中借款共有十六中状态，分别是“等待担保机构审核”、“等待开鑫贷审核”、“筹集中”、“合同待登记”、“合同已登记”、“正常还款”、“借入人提前还款”、“借入人正常还款”、“逾期还款”、“担保机构代偿”、“流标”、“担保机构拒绝”、“到期未担保”、“借入人撤销”、“系统撤销”和“开鑫贷拒绝”。使用码表来表示这些状态，在数据库中建立一个码表，存储状态码和状态值，11表示“等待担保机构审核”，12表示“等待开鑫贷审核”，21表示“筹集中”，22表示“合同待登记”，23表示“合同已登记”，31表示“正常还款”，32表示“逾期还款”，41表示“借入人撤销”，42表示“担保机构拒绝”，43表示“借入人正常还款”，44表示“流标”，45表示“担保机构代偿”，46表示“系统撤销”，47表示“到期未担保”，48表示“借入人提前还款”，49表示“开鑫贷拒绝”。“1”开头的状态表示处于开始阶段的借款状态，“2”开头的状态表示处于已经审核阶段的借款状态，“3”开头的状态表示处于已经成立阶段的借款状态，“4”开头的状态表示处于已经结束阶段的借款状态。通过这样按照不同阶段划分状态大的类别，再在每个类别中划分具体的状态的方法，使得在代码中对状态的判断和处理更加简便。借款表中只是存储借款的状态码，查询借款信息时需要将借款的名称显示出来，如果每次都在Sql语句中进行联表查询将会使得Sql语句变得复杂，我们通过自己实现的标签来将码表中的键转化成码表中的值，这样每次需要在页面上显示借款的状态时不需要在查询借款的Sql语句中与码表关联，只需要在界面上使用自己定义的标签即可，如此便提高了开发的效率。如上所述，开鑫贷借款状态变化的状态图如图4.8所示。

图4.8 借款状态变化的状态图

## 4.3借款模块的实现

### 4.3.1 三种框架的配置

本系统使用Struts1+Spring+iBatis作为系统的框架，简化了开发过程，使得系统的扩展更加容易。Struts1担当控制器的角色，负责请求与返回的转发。Spring将三个框架整合起来，管理Action等类的生命周期，通过依赖注入的方式使得系统解耦。iBatis负责数据数据库的访问，支持开发者自己定义Sql语句。系统通过一系列的XML格式的配置文件将三种框架整合起来，系统框架的主要配置文件是web.xml、struts-config.xml、interceptor-config.xml、action-kxd.xml、service-kxd.xml、struts-kxd.xml、sql-map-config.xml、applicationContext-db.xml以及applicationContext-ibatis.xml。

在web.xml中配置了ActionServlet作为启动Servlet，将struts-config.xml和struts-kxd.xml以config参数的形式传递给ActionServlet。在struts-config.xml中配置了两个plug-in，分别是ContextLoaderPlugIn和SAIFSpringPlugin。ContextLoaderPlugIn主要负责加载上下文环境，有一个contextConfigLocation属性配置了所有上下文相关的文件的地址，在开鑫贷系统中这个属性值配置的是“classpath:/spring/applicationContext\*.xml;/WEB-INF/conf-service/service-\*.xml;/WEB-INF/conf-action/action-\*.xml”，表示配置上下文相关的文件满足这些地址模式，其中“\*”表示匹配所有的。SAIFSpringPlugin主要负责配置拦截器，它有一个interceptor-config属性配置了拦截器配置文件的位置，在开鑫贷系统中这个属性的值是“/WEB-INF/interceptor-config.xml”，表示拦截器配置在interceptor-config.xml文件中。struts-kxd.xml中定义了action-mapping的信息，在action-mapping标签中使用action子标签，action有parameter、path、scope和type四个属性，使用“org.springframework.web.struts.DelegatingActionProxy”作为action的类型，使用“action”作为parameter的值，使用“request”作为scope的值，根据不同的场景设置不同的path的值，path的值与action-kxd.xml中配置的bean的name对应，action-mapping将请求URL和对应的处理Action类及方法对应起来。开鑫贷系统struts-kxd.xml中的action标签定义的parameter属性的值为“action”，表示将“path?action=method”形式的请求URL转发至action-kxd.xml中配置的name等于“path”的类的“method”方法来处理。

action-kxd.xml与service-kxd.xml中配置了不同的beans，将name和类对应起来，并且设置了不同bean之间的依赖关系。action-kxd.xml主要配置了作为Action的bean，service-kxd.xml主要配置了作为Service的bean。在这两个配置文件的beans标签中设置default-autowire属性值为“byName”，这样在一个bean中需要依赖其它的bean时我们只要声明一下就可以直接使用了，Spring框架会帮我们自动注入这些依赖的bean。

applicationContext-db.xml和applicationContext-ibatis.xml配置了数据库的连接，applicationContext-db.xml配置了数据源，applicationContext-ibatis.xml中配置了id为“sqlMapClient”的SqlMapClientFactoryBean，这是Spring提供的用于构建iBatis的 SqlMapClient的工厂。SqlMapClientFactoryBean有两个属性，一个是“dataSource”，这个属性链接到applicationContext-db.xml配置的数据源，还有一个是“configLocation”，这个属性的值是“classpath:sqlmap/oracle/sql-map-config.xml”，表示iBatis的各种Sql语句配置文件的地址。在“sql-map-config.xml”文件中有一个“sqlMapConfig”标签，这个标签中有“sqlMap”子标签，“sqlMap”子标签有“resource”属性，“resource”属性的值为一个XML文件的地址，每个这样的XML文件中配置一个数据库表的各类Sql语句。

通过如上所述的这些配置文件，三种框架便可以一起工作了，配置好了三种框架之后，每个功能的开发基本上都是同样的流程，先完成前端的页面，再实现控制器Action类，接着实现需要的Service类，最后在iBatis相关的XML中配置Sql语句并通过BaseDao调用这些Sql语句进行数据库表的增删改查。

### 4.3.2发布借款的实现

发布借款的实现过程如下。

首先，借入人点击“发布借款信息”进入FBDKXX.jsp页面，输入标题、用途、金额、期限、年利率等借款信息并选择担保机构信息之后点击“下一步”，这个时候在前端对借款信息这个表单进行验证，这里使用jQuery的“jquery.validate.js”插件进行表单验证，在javascript中配置好各个表单控件的验证规则，这个插件会帮助我们实现验证的功能并显示提示的消息。

接着，在借入人点击“下一步”之后，请求交给DKGLAction类的insertDkxxFirStep方法来处理，该方法获取页面表单传递的参数，放入一个名为dksqxxMap的HashMap中。该方法调用WorkingDayService的getPrevWorkingDay方法获取借款筹集到期日，筹集到期日是借款成立日的前一个工作日，用户提交的表单中只有借款成立日，需要在后台算出借款筹集到期日。参数获取完毕后，调用CommonService的searchImgInfo方法从MongoDB中获取用户附件信息，包括最高学历附件、结婚证扫描件、房产证扫描件、驾驶证扫描件、户口本扫描件、银行流水号账单等等，将获取到的用户附件信息存入名为imgList的List<Map<String, Object>>中。接着使用HttpServletRequest的setAttribute方法将imgList放入request对象中，使用HttpSession的setAttribute方法将dksqxxMap放入session对象中。最后通过ActionMapping 的findForward 方法将页面跳转至上传资料信息页面TJZL\_GR.jsp。

然后，借入人在TJZL\_GR.jsp页面上完善各种证件的扫描件信息，并且选择允许公开给借入人的信息，然后点击“下一步”按钮。上传证件的扫描信息的地方会提示用户之前是否已经上传过，如果上传过则显示“已上传”，否则显示“选择文件”按钮，这是通过之前存入request中的imgList实现的，通过JSTL标签库在页面上进行判断。用户点击“下一步”按钮后，通过jQuery的“jquery.validate.js”插件进行表单验证，验证用户上传的附件是否满足格式、大小等要求。

接着，如果附件满足条件则请求交给DKGLAction的generateHTYL方法来继续处理。在generateHTYL方法中获取界面上传来的允许公布给借出人的信息参数，并将它们存入一个名为dkgbxxMap的HashMap中。通过ActionForm的getMultipartRequestHandler方法获得MultipartRequestHandler，这是Struts用来处理多附件上传的类，需要在TJZL\_GR.jsp页面的表单中将enctype属性设为“multipart/form-data”，然后通过它的getFileElements方法获得存放文件的Hashtable，将这个存放文件的Hashtable存入名为fileMap的HashMap中，再将fileMap存入session中，以供后面取用。接下来从dksqxxMap中获取合同需要的信息，将这些信息放入一个Map中，将这个map作为参数调用HtService的getHT方法获取合同内容，并把获取到的合同内容存入request中。接着通过mapping.findForward("toHTSign")将页面跳转至合同预览界面HT\_YL.jsp。

最后，在合同预览界面显示合同的文本给借出人看，通过EL表达式“${requestScope.context}”获取之前存入request中的合同文本并展示在合同预览页面上。借入人预览合同无异议之后，点击“确认”按钮，请求便被传递给DKGLAction的insertDkxxFourStep方法来处理。这个方法首先调用ActionChkService类的checkDKFB方法来对借款相关的信息格式进行验证，之所以在前台验证后还需要在后台验证，是为了防止跳过前台页面直接向服务器发送Http请求来进行攻击。后台验证没问题之后，调用DbService的getDbjgStatus方法获得担保机构的状态，检测该担保机构是否存在并且是否处于绑定状态，如果不是则通过返回json格式的信息提示“该担保用户不存在或者已被停用”，如果该担保机构存在并且处于绑定状态，则从session中获取存放用户附件信息的tpxxMap，然后调用CommonService的insertImgInfo方法将用户的图片附件信息存入MongoDB中，接着获取借款申请信息dksqxxMap、借款公布信息dkgbxxMap、担保信息dbxxMap以及担保合同信息dbhtMap，然后调用DKService的txpublishDKInfo方法，传入以上四个Map，将借款申请信息、借款公布信息、担保信息以及合同信息写入数据库，并且更新该借入人的剩余借款额度，然后调用TzdxyjService的txZdfbjkxx方法向借入人和担保机构发送短信和邮件通知，最后通过返回json格式的消息提示用户借款发布成功。

以上就是借入人发布借款信息的实现过程，由于DKService中的txpublishDKInfo方法会先后保存借款申请信息、借款公布信息、担保信息以及合同信息到四个不同的数据库表中，因此需要使用事务管理来保证这四个信息或者都保存成功，或者都不保存。这里我们使用Spring框架的事务管理功能来实现，通过Spring的aop:config标签和tx:advice标签来配置，tx:advice标签配置事务的传播特性，通过aop:config标签配置哪些方法使用事务。开鑫贷系统的事务配置如图4.9所示。

|  |
| --- |
| <!-- 配置事务传播特性 -->  <tx:advice id="tx-ibatis" transaction-manager="tm-ibatis">  <tx:attributes>  <tx:method name="tx\*" propagation="REQUIRED" isolation="SERIALIZABLE"/>  <tx:method name="untx\*" propagation="REQUIRES\_NEW" timeout="20" />  </tx:attributes>  </tx:advice>  <tx:advice id="tx-ibatisfordefault" transaction-manager="tm-ibatis">  <tx:attributes>  <tx:method name="tx\*" propagation="REQUIRED" isolation="READ\_COMMITTED"/>  <tx:method name="untx\*" propagation="REQUIRES\_NEW" timeout="20" />  </tx:attributes>  </tx:advice>  <!-- 配置哪些方法使用事务 -->  <aop:config proxy-target-class="true">  <aop:advisor pointcut="execution(\* com.kxd.service.impl.LCGLServiceImpl.\*(..))" advice-ref="tx-ibatisfordefault" />  <aop:advisor pointcut="execution(\* \*..common.\*ServiceImpl.\*(..))" advice-ref="tx-ibatis"/>  <aop:advisor pointcut="execution(\* \*..service.impl.\*ServiceImpl.\*(..)) and not execution(\* com.kxd.service.impl.LCGLServiceImpl.\*(..))" advice-ref="tx-ibatis"/>  <aop:advisor pointcut="execution(\* \*..jn.trade.\*Impl.\*(..))" advice-ref="tx-ibatis"/>  <aop:advisor pointcut="execution(\* \*..service.impl.\*FlowInterface.\*(..))" advice-ref="tx-ibatis"/>  </aop:config>  <!-- 配置事务管理器 -->  <bean id="tm-ibatis" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  <property name="dataSource" ref="dataSource" />  </bean> |

图4.9 开鑫贷系统事务的配置

如图4.9所示，所有满足“\*..common.\*ServiceImpl.\*(..)”格式的类的方法都会采用“tx-ibatis”策略，在“tx-ibatis”策略下，所有以tx开头的方法的传播方式是“REQUIRED”，隔离级别是串行化。DKServiceImpl是在“com.jn.kxd.common”这个包中，满足“\*..common.\*ServiceImpl.\*(..)”这一格式，因此DKServiceImpl类的txpublishDKInfo方法会以事务的方式执行，这样便可以确保数据库不会出现由于部分更新而导致的数据不一致的情况。

如上所述，借入人发布借款过程的顺序图如图4.10所示。DKService在向数据库存储借款信息时调用的是BaseService中的insert方法，而BaseService会调用BaseDao的insert方法，最终由iBatis的SqlMapClientDaoSupport类来完成向数据库的信息插入。由于所有的Service类向数据库中保存信息基本上都是这个调用过程，因此在图4.10以及后续的顺序图中省略BaseService之后的调用过程。

图4.10 借入人发布借款的顺序图

### 4.3.3查看借款的实现

查看提报列表类型的操作包括许多种，比如客服查看自己的提报，技术支持经理查看所有的提报，技术支持组成员查看等待自己解决的提报等等，这些操作的基本原理都是差不多的，这些操作的业务逻辑层的service类都继承了一个BaseFetchQuestionListService类，这个类实现了很多返回提报列表类型操作的公共方法，比如getCatName方法用来获取提报类别的名字，reverseList方法用来倒转列表等等。

这里用“查看已响应未解决提报”为例子进行介绍，其余的这一类型的操作的实现是类似的。

查看已响应未解决的提报信息是技术经理可以进行的操作，它的实现过如下。

首先，技术支持经理点击“已响应未解决提报”链接，这个链接通过struts框架的机制将请求传递给BeginNotSolveQuestionAction，这个Action有一个属性，叫pageNum，用来对列表进行分页显示，初始时传递给BeginNotSolveQuestionAction的pageNum参数的值是1。BeginNotSolveQuestionAction有一个getPageData方法，以pageNum为参数，返回一个list，这个方法调用BeginNotSolveQuestionService的getPageData方法。

BeginNotSolveQuestionService的getPageData方法同样以pageNum为参数，返回一个list，该方法首先新建一个OaHelpdeskQuestion对象，设置这个对象的mark属性为“1”，然后新建一个状态列表，状态列表中加入状态码3和4，表示状态为“已开始”和“解决超时”，接着计算去数据库中取数据时的数据区间，算出开始的记录号和结束的记录号。

然后，在getPageData方法中调用OaHelpdeskQuestionDAO的findByExampleWithSizeAndStatusList方法，将新建的OaHelpdeskQuestion对象，状态列表，开始的记录号和结束的记录号作为参数传过去。findByExampleWithSizeAndStatusList方法返回状态在状态列表中的，mark值为1的，位于开始的记录号和结束的记录号之间的记录列表。

接着，BeginNotSolveQuestionService对返回的提报记录列表进行遍历，将每个OaHelpdeskQuestion对象封装成QuestionDetail对象，并加入一个list中，然后调用reverseList方法对这个list进行倒转操作，使后加入的记录显示在前面。生成的这个list被返回给BeginNotSolveQuestionAction之后，这个Action调用BeginNotSolveQuestionService的getTotalNum方法，得到能够显示的记录总数。

最后，提报list，总数以及pageNum被加入到request中，用于在jsp页面中的分页显示。

如果整个过程中没有异常产生，则BeginNotSolveQuestionAction返回SUCCESS，否则返回INPUT。返回SUCCESS对应的页面是BeginNotSolveQuestion.jsp，返回INPUT对应的页面是error.jsp。BeginNotSolveQuestion.jsp页面通过OGNL表达式将提报列表显示出来，通过“<s:iterator>”标签遍历list，通过“<s:if>”，“<s:elseif>”，“<s:else>”等条件标签通过不同的条件决定显示不同的内容，通过“<s:property>”标签将属性值显示出来。

关于分页的实现，在BeginNotSolveQuestion.jsp中引入了一个名字为Pager的javabean，Pager中有totalPage，currentPage，totalRecord，currentRecord以及pageSize属性，分别表示总的页数，当前页，总的记录数，当前显示到的记录数，每页显示的条目数量。在BeginNotSolveQuestion.jsp对这些属性进行计算和赋值，并根据不同的条件决定页面控制栏显示的内容。

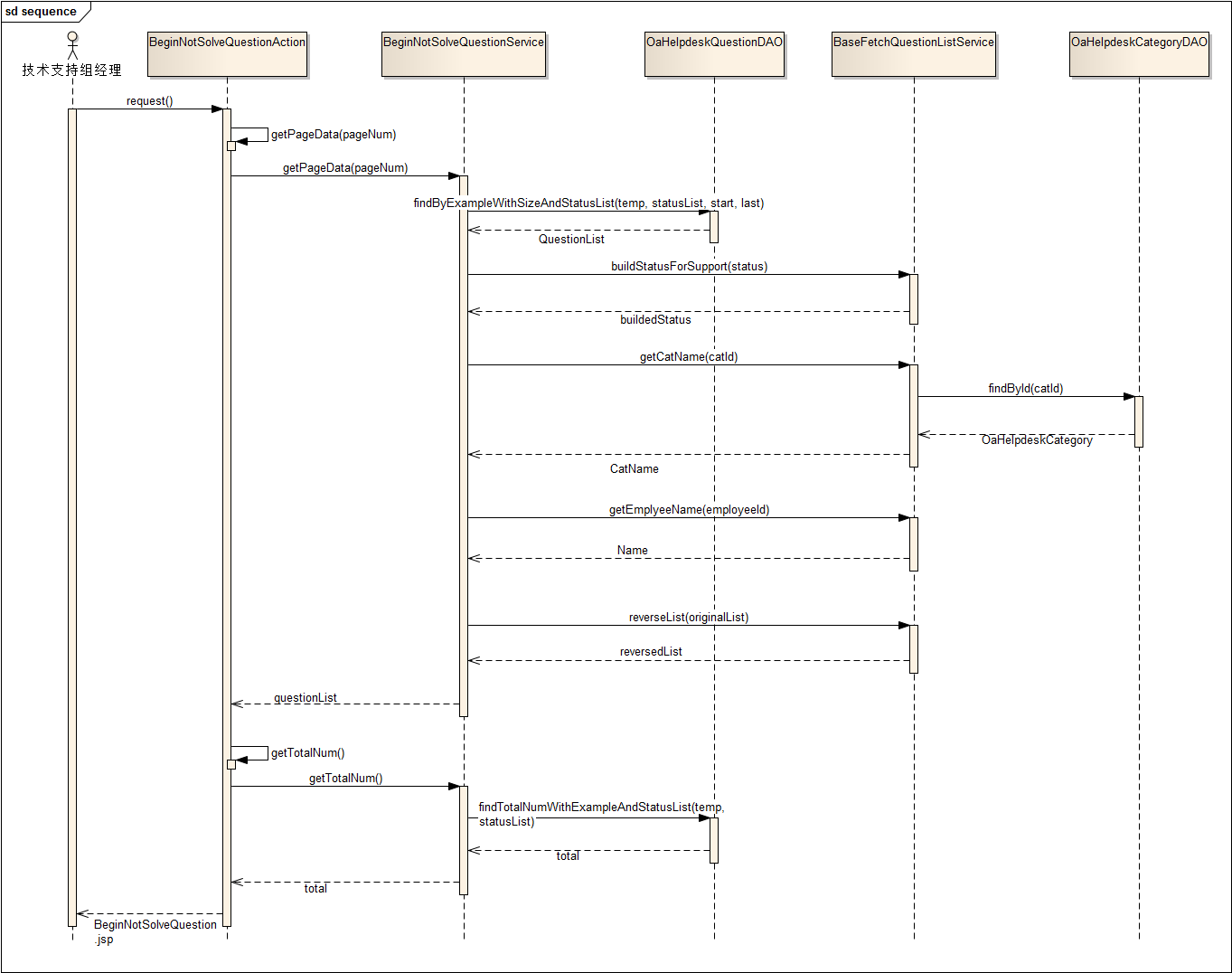
 查看提报列表的实现的顺序图如图4.9所示。

图4.9 查看提报列表顺序图

BeginNotSolveQuestion.jsp中分页控制栏的代码如图4.10所示。

|  |
| --- |
| <%  int total = (Integer)request.getAttribute("totalNum");  intpageNum = (Integer)request.getAttribute("pageNum");  intcurrentRecord = 10;  pager.setPageSize(10);  pager.setTotalRecord(total);  pager.setTotalPage(total,pager.getPageSize());  //计算并设置分页的参数  if(request.getParameter("currentRecord")!=null){  currentRecord = Integer.parseInt(request.getParameter("currentRecord"));  pager.setCurrentRecord(currentRecord); pager.setCurrentPage(currentRecord,pager.getPageSize())  }  %>  <span><font size="2">共<%=pager.getTotalRecord()%>条记录|共<%=pager.getTotalPage()%>页|当前<%=pager.getCurrentPage()+1%>页|  <%  //如果已经到了首页就去掉前一页的链接 if(pager.getCurrentRecord()-pager.getPageSize()<0)  out.println("首页|");  else  //上一页就是把当前记录位置减去一页显示记录的条数  out.print("<a href='/TuniuOA/helpdesk/beginNotSolveQuestion?pageNum="+(pageNum-1)+"&currentRecord="+(pager.getCurrentRecord()-pager.getPageSize())+"'>上一页</a>|");  //如果到了尾页就去掉下一页的链接 if(pager.getCurrentRecord()+pager.getPageSize()>=pager.getTotalRecord())  out.println("尾页");  else  //下一页就是把当前记录位置加上一页显示记录的条数  out.print("<a href='/TuniuOA/helpdesk/beginNotSolveQuestion?pageNum="+(pageNum+1)+"&currentRecord="+(pager.getCurrentRecord()+pager.getPageSize())+"'>下一页</a>|");  %>  </font>  </span> |

图4.10 分页控制栏的实现

### 4.3.4对借款投标的实现

查看提报详细信息类型的操作有好多种，比如客服查看提报的详细信息，技术支持组经理查看提报详细信息，技术支持组成员查看提报详细信息等。这些操作的实现过程是类似的，这里以技术支持经理查看提报详细信息为例来做介绍。

技术支持经理在查看某一提报的详细信息的时候点击提报号的链接，这样就把这个提报号作为参数传递给了QuestionAllDetailAction，QuestionAllDetailAction有questionId属性，通过struts的机制自动赋值给questionId。接着QuestionAllDetailAction的execute方法被执行，在execute方法中调用QuestionDetailService的getDetailForManager方法，以questionId为参数，返回QuestionDetail对象。QuestionDetailService的getDetailForManager方法得到questionId之后，调用OaHelpdeskQuestionDAO的getById方法得到OaHelpdeskQuestion对象，再通过QuestionDetail的包装，最后返回一个QuestionDetail对象。QuestionAllDetailAction将得到的QuestionDetail对象加入request对象中，返回SUCCESS，通过struts机制，展现出QuestionAllDetail.jsp页面，在这个页面里面通过OGNL表的式显示一些属性值，通过一些条件判断标签根据不同的条件决定显示的不同内容。

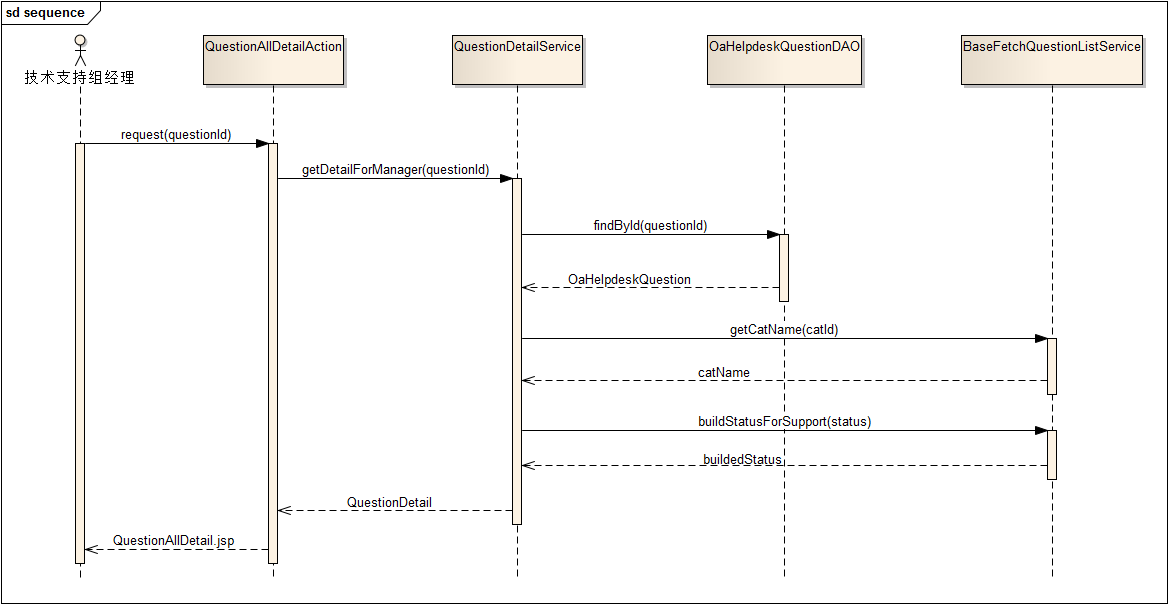
 查看提报的详细信息的顺序图如图4.11所示。

图4.11 查看提报详细信息的顺序图

### 4.3.5更改提报数据的实现

更改提报数据指的是更新数据库中已有提报的信息，在helpdesk模块中这些操作是通过ajax调用实现的，包括技术支持经理设定标准响应时长，技术支持经理设定标准解决时长，技术支持组成员更改提报状态，技术支持组经理更改提报问题类别，客服对提报进行评价等操作。这里以技术支持组成员更改提报状态为例介绍更改提报数据的实现。

技术支持组成员更改提报状态实现如下。

首先，技术支持组成员打开查看提报详细信息的页面QuestionDetailToSolver.jsp，在这个页面中有一个“更改状态为”的label，后面有一个select框，当技术支持组成员选择select框中的选项之后便出发了更新状态的操作。在QuestionDetailToSolver.jsp页面上获取状态值和提报号作为参数提交给SetStatusAction，SetStatusAction有三个属性，分别是status，questionId以及result。传给SetStatusAction的参数会被赋值给status和qustionId属性，result属性存储ajax调用返回的json对象的字符串值。

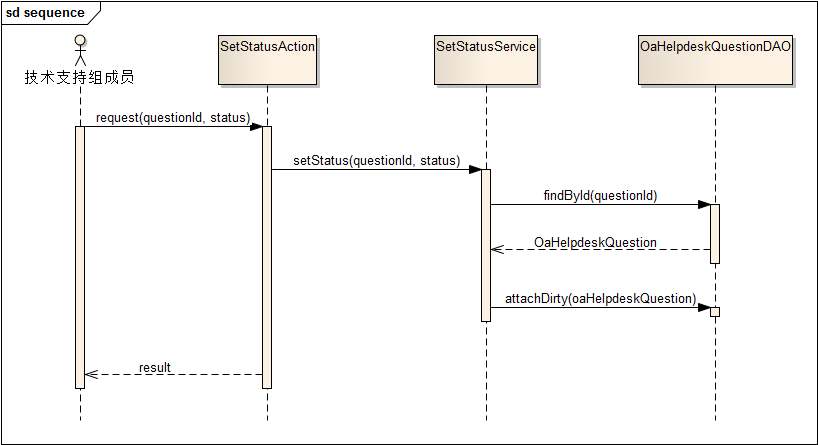
请求发给SetStatusAction之后，SetStatusAction的execute方法将被调用，execute方法中调用SetStatusService的setStatus方法，将questionId和status作为参数传递过去。

SetStatus方法调用OaHelpdeskQuestionDAO的findById方法，得到对应questionId的OaHelpdeskQuestion对象，然后设置该对象的status属性，最后调用OaHelpdeskQuestionDAO的attachDirty方法将对象保存到数据库中。如果整个过程成功，就新建一个hashmap对象，将"状态设置成功！！"这个返回信息加入到新建的map中，接着用JSONObject的fromObject方法将新建的map转化成JSONObject，最后用JSONObject的toString方法将JSONObject转化成字符串赋值给result字符串并返回SUCCESS。在struts.xml中配置的result的type为json，result字符串将会被作为ajax调用中回调函数返回的数据。

在QuestionDetailToSolver.jsp页面中通过javascript的eval方法将返回的字符串解析成JSON数据格式，然后将返回的信息alert出来，提示用户操作成功与否。

以上就是通过ajax方式更改提报信息的整个流程。在action中必须通过JSONObject的toString方法转化成String，这样才能赋值给result字符串，在页面中必须通过javascript的eval方法将字符串重新转化成json对象，这样才能将存在json对象中的值成功取出来。

更改提报数据操作的顺序图如图4.12所示。

图4.12 更改提报数据的顺序图

更改提报数据的ajax调用的实现代码如图4.13所示。

|  |
| --- |
| jsp中的javascript代码：  $("[name='status']").on("change",function(){  var status = $(this).val();  varquestionId = $("#questionId").text();  $.get("/TuniuOA/ajax/setStatus",{status:status,questionId:questionId},function(data){  var result = eval("("+data+")");//将字符串转化成json对象  alert(result.finalResult);  },"json");  });  action中的相关代码：  public String execute(){  ActionContextctx = ActionContext.getContext();  Map map = new HashMap();  try{  setStatusService.setStatus(Integer.parseInt(questionId.trim()), Integer.parseInt(status));  map.put("finalResult", "状态设置成功！！");  JSONObjectobj = JSONObject.fromObject(map);//将map转化成json对象  result = obj.toString();//将json对象转化成字符串  }catch(Exception e){  map.put("finalResult","状态设置失败！！");  e.printStackTrace();  }  return SUCCESS;  }  struts.xml中的配置：  <package name="ajax" namespace="/ajax" extends="json-default">  <action name="setStatus" class="setStatusAction">  <result type="json">  <param name="root">result</param>  </result>  </action>  <package> |

图4.13 更改提报数据ajax调用相关代码

### 4.3.6超时提报高亮显示的实现

超时问题高亮显示指的是当技术支持组经理或技术支持组成员查看提报列表的时候，对于超时的问题要高亮显示出来。超时有两种情况，一种是响应时间超时，一种是解决时间超时。响应时间指的是从提报被分配给某个人，到这个人将提报状态改为“已开始”所用的时间。响应时间超时指的是实际的响应时间超过了标准的响应时间。解决时间指的是从提报状态被改为“已开始”到提报状态被改为“已解决”用掉的时间。解决时间超时指的是实际的解决时间超过了标准的解决时间。这些时间的计算只算工作日的工作时间，单位是小时。

超时提报高亮显示涉及到两方面的操作。一方面是检测超时的提报并把提报状态改为“响应超时”或“解决超时”。另一方面是将超时的提报高亮显示出来。

超时提报的检测是在显示提报列表时完成的。在显示提报列表时，对于状态为“已分配”的提报，检测分配时间至当前时间的小时数有没有超过标准的响应时间，如果超时了就把提报的状态改为“响应超时”。对于状态为“已开始”的提报，检测响应时间至当前时间的小时数有没有超过标准处理时间，如果超时了就把提报的状态改为“解决超时”。

超时提报的检测实现如图4.14所示。

|  |
| --- |
| public void checkDelay(OaHelpdeskQuestionoaHelpdeskQuestion){  if(oaHelpdeskQuestion.getStatus()==1){  if(DateHelper.HourBetweenTwoDay(oaHelpdeskQuestion.getDeliverTime(),new Date())>oaHelpdeskQuestion.getStandardResponseHour()){  oaHelpdeskQuestion.setStatus(2);  oaHelpdeskQuestionDAO.attachDirty(oaHelpdeskQuestion);  }  }  if(oaHelpdeskQuestion.getStatus()==3){  if(DateHelper.HourBetweenTwoDay(oaHelpdeskQuestion.getResponseTime(),new Date())>oaHelpdeskQuestion.getStandardSolveHour()){  oaHelpdeskQuestion.setStatus(4);  oaHelpdeskQuestionDAO.attachDirty(oaHelpdeskQuestion);  }  }  }  //计算两个时间点之间的小时数  //大概思路是：先计算两个时间点之间的工作日数，按每日都是8小时算出总和，由于两个时间点的小时不一定相同，故再计算两个小时之间相差的小时数，用前面算出的总和减去这个小时差即可。  public static intHourBetweenTwoDay(Date firstDate , Date secondDate){  Calendar cal = Calendar.getInstance();  Calendar secondCal = Calendar.getInstance();  cal.setTime(firstDate);secondCal.setTime(secondDate);//设置两个时间点  int hour = cal.get(Calendar.HOUR\_OF\_DAY);int i = 0 ;int sum=0;  while(cal.before(secondCal)){  cal.add(Calendar.DATE, 1);if(cal.get(Calendar.DAY\_OF\_WEEK)!=Calendar.SATURDAY&&cal.get(Calendar.DAY\_OF\_WEEK)!=Calendar.SUNDAY)  i++;//不算周末，只算工作日  }  sum+=8\*i;//一天8小时工作  //下面计算小时差  intsecondHour = secondCal.get(Calendar.HOUR\_OF\_DAY);  hour = cal.get(Calendar.HOUR\_OF\_DAY);  if(hour>=secondHour){  if(secondHour<12){  if(hour<12){sum-=(hour-secondHour);  }else{  if(hour<13){sum-=(12-secondHour);//中午12点至13点休息，不计  }else{sum-=(12-secondHour+hour-13);  }  }  }else{  if(secondHour<13){sum-=(hour-13);  }else{sum-=(hour-secondHour);  }  }  }  if(hour<secondHour){  if(hour<12){sum-=(hour-9);//9点开始上班  }else{  if(hour<13){sum-=3;  }else{sum-=(hour-9-1);  }  }  if(secondHour<12){sum-=(18-secondHour-1);//18点下班  }else{  if(secondHour<13){sum-=5;  }else{sum-=(18-secondHour);  }  }  }  return sum;  } |

图4.14 超时提报检测的实现代码

超时提报高亮显示的实现就是在显示提报列表的页面中通过OGNL表达式判断状态是不是“响应超时”或“解决超时”，如果是的话就把背景设为黄色。超时问题高亮显示的实现如图4.15所示。

|  |
| --- |
| <s:iterator value="#request.data" var="o" status="s">  <s:if test="#o.detailStatus==2||#o.detailStatus==4">  <tr style="background-color:yellow"><!-- 如果超时就把这一行背景设为黄色 -->  </s:if>  <s:else>  <s:if test="#s.odd">  <tr style="background-color:#bbb">  </s:if>  <s:else>  <tr>  </s:else>  </s:else>  <td>  <s:a href="/TuniuOA/helpdesk/questionDetailToSolver?questionId=%{#o.questionId}"><s:property value="#o.questionId"/>  </s:a>  </td>  <td>  <s:property value="#o.formatDate(#o.addDate)"/>  </td>  <td>  <s:property value="#o.serverId"/>  </td>  <td>  <s:property value="#o.catName"/>  </td>  <td>  <s:property value="#o.status"/>  </td>  <td>  <s:property value="#o.formatDateTime(#o.deliverTime)"/>  </td>  <td>  <s:property value="#o.standardResponseHour"/>小时  </td>  <td>  <s:property value="#o.standardSolveHour"/>小时  </td>  </tr>  </s:iterator> |

图4.15 超时提报高亮显示的实现代码

## 4.4 本章小结

本章主要介绍了Helpdesk模块的设计和实现。设计部分从数据库设计，类的设计，流程设计三个方面来展现。实现部分描述了系统ssh框架的配置，添加提报的实现，查看提报列表的实现，查看提报信息的实现以及超时提报检测的实现，描述了各部分的实现过程，顺序图以及部分关键代码。