# 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。

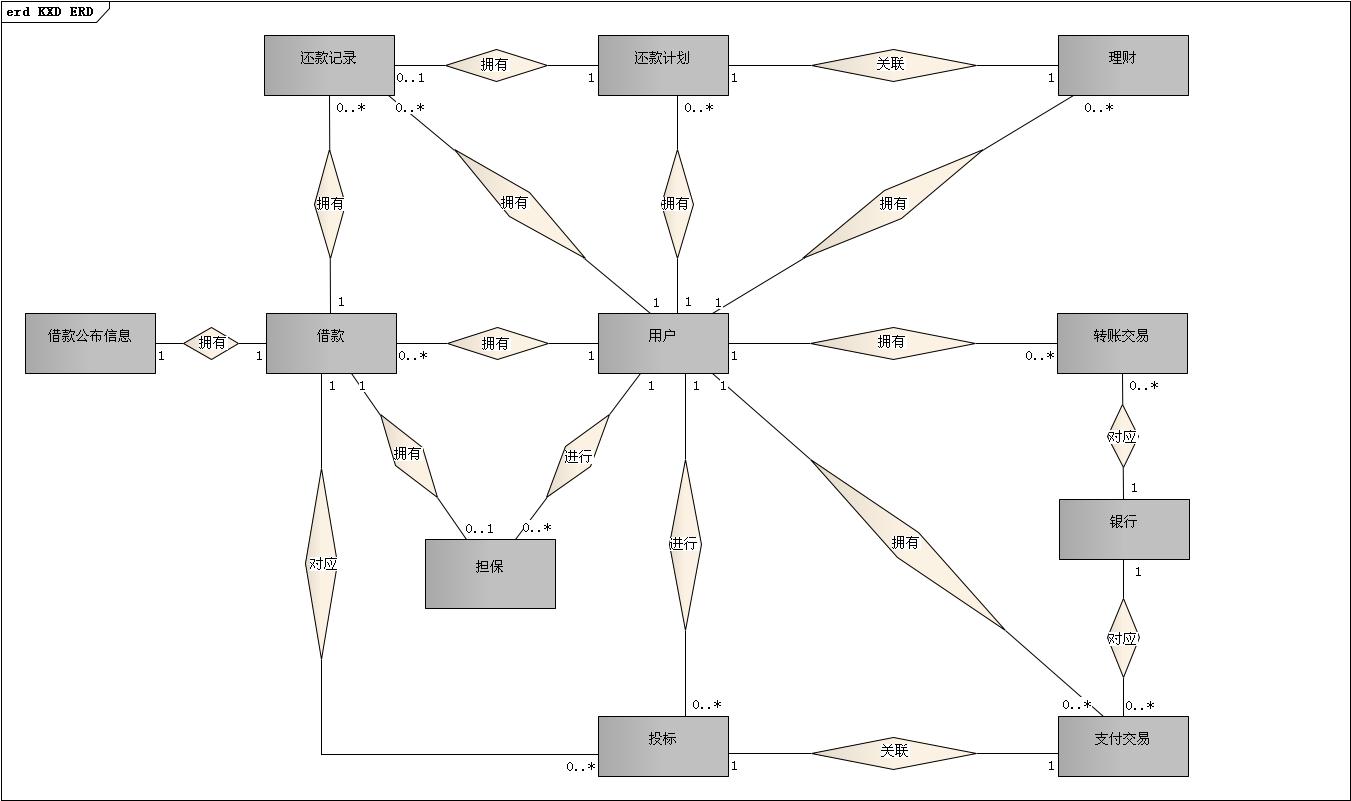
## 4.2 借款模块的详细设计

### 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。

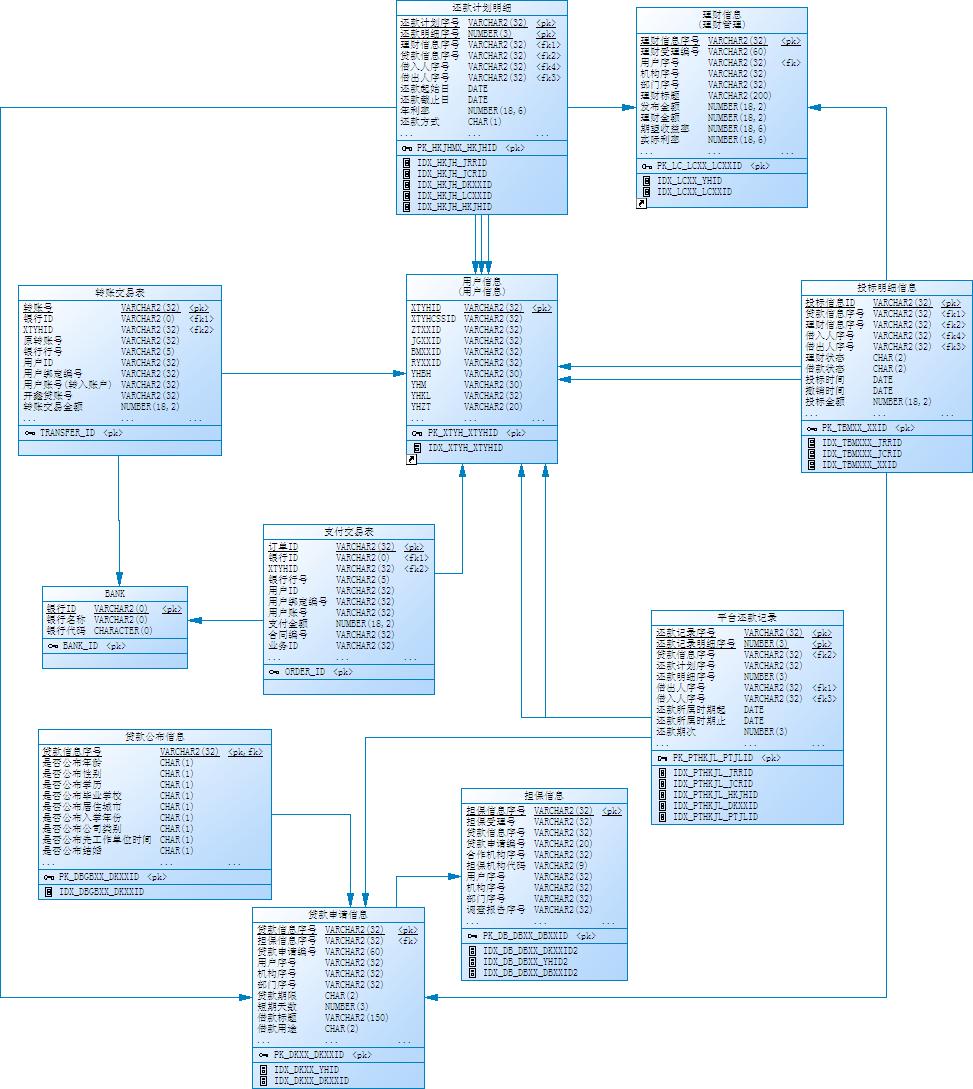


图4.2借款模块相关数据库表结构图

**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**

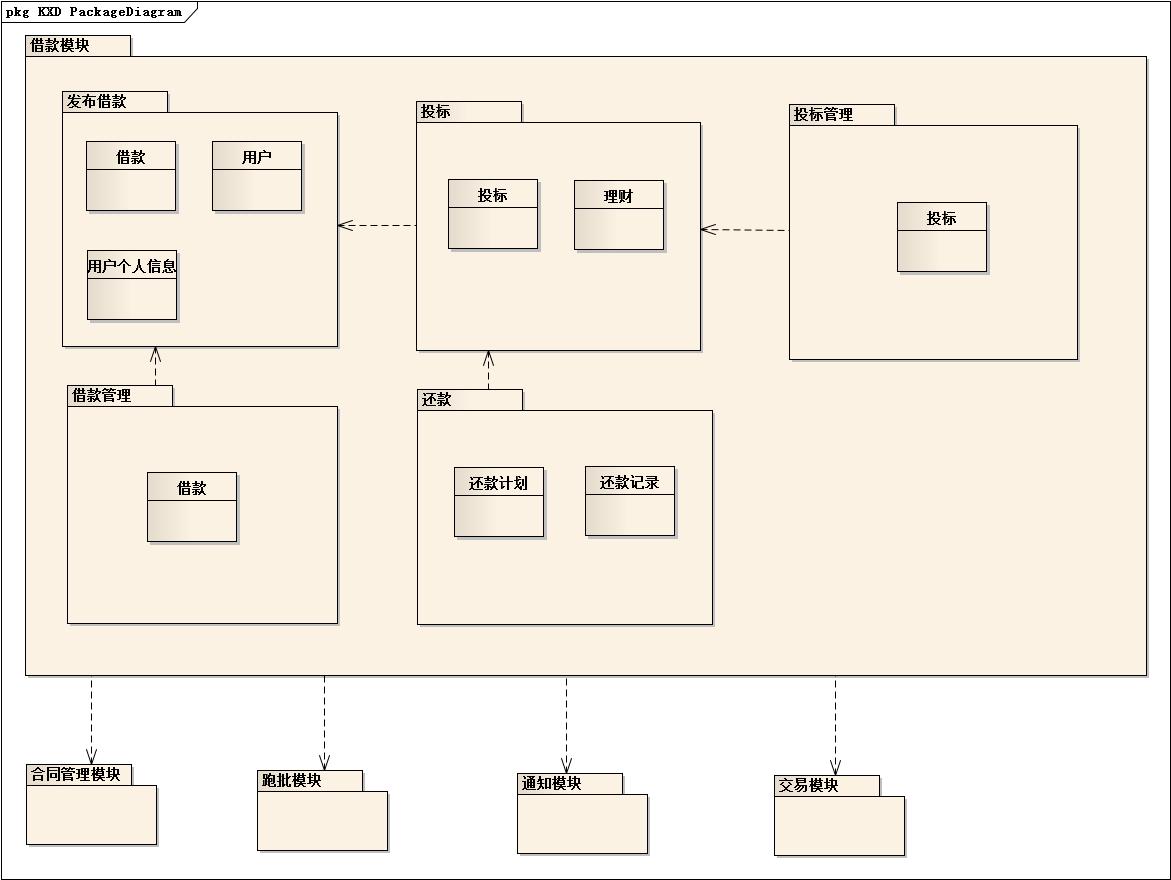
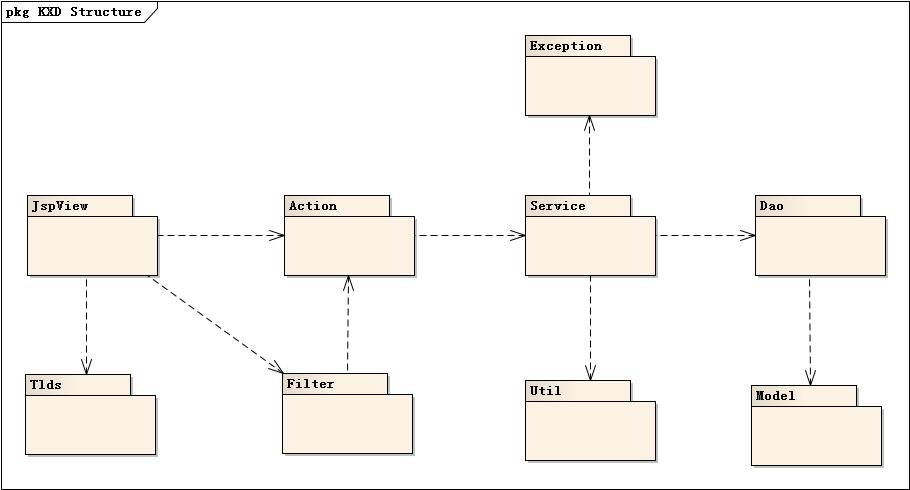
**** 借款模块按照功能可以分为发布借款、投标、还款、投标管理、借款管理这些子模块，发布借款、投标、还款子模块在前台，借款管理和投标管理在后台。投标子模块和借款管理子模块依赖于发布借款模块，只有发布借款之后才能投标并对借款进行管理。还款子模块和投标管理子模块依赖于投标模块，只有投标之后才能进行还款和投标管理。在进行借款流程中的合同预览及合同签订步骤时，需要依赖合同管理模块的功能；在发送通知时需要依赖通知管理模块的功能；在投标及还款支付时，需要调用交易模块的功能；借款的筹集期满和成立时都需要跑批模块跑批之后才能进行状态的改变。如上所述，借款模块的包图如图4.3所示。

图4.3借款模块子模块包图

**类的总体结构设计**

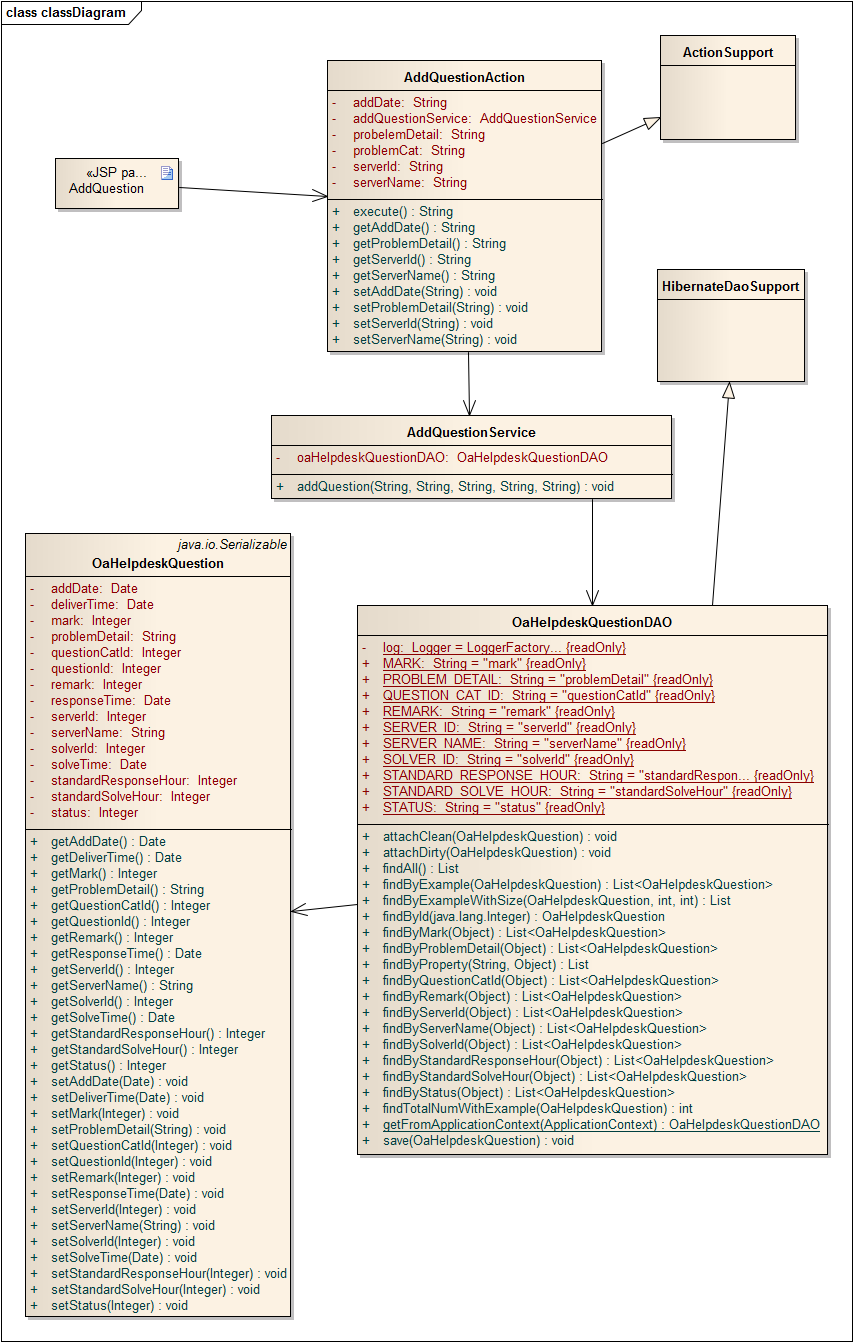
借款模块的类的总体结构设计遵循了“高内聚，低耦合”的设计思路，将表现层，业务逻辑层，数据访问层相分离。本模块的类的结构大概可以分为四层：数据持久层，业务逻辑层，控制层，表现层。表现层大多是JSP文件，由于JSP文件在运行时会被编译成Servlet，这里把JSP也当做类来看待。表现层侧重于页面的展现，使用DIV+CSS的方式布局，在一些需要在页面进行逻辑处理的地方使用了自己实现的标签。控制层主要是Struts1的ActionServlet和一些Action类以及一些Filter类，Struts1通过配置文件来决定用户访问的URL调用到哪个Action以及响应的URL对应到哪一个页面，请求参数通过ActionForm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**

添加提报功能需要获取客服的提报信息，因此需要一个AddQuestion.jsp文件将信息获取，然后传入AddQuestionAction，这个Action继承了Struts2提供的ActionSupport，AddQuestionAction调用AddQuestionService，AddQuestionService再调用继承了Hibernate提供的HibernateDaoSupport类的OaHelpdeskQuestionDAO完成提报的添加。

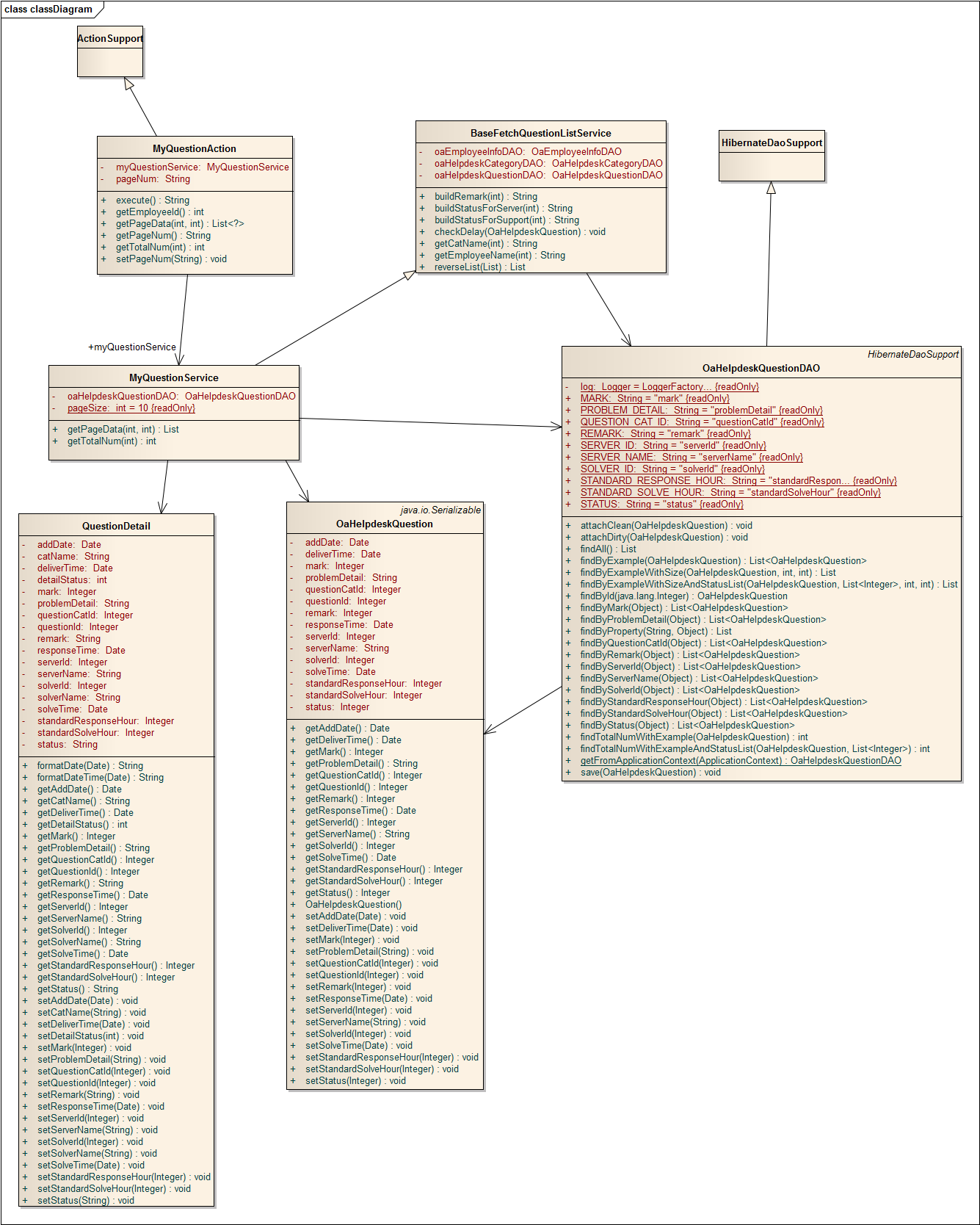
添加提报功能的类图如图4.4所示。

图4.4添加提报功能类图

AdQuestionAction调用AddQuestionService的addQuestion方法，addQuestion方法接收五个参数，构造一个OaHelpdeskQuestion对象，接着传递这个OaHelpdeskQuestion对象作为参数调用OaHeldpeskQuestionDAO的save方法，从而将提报的信息保存到数据库中。

**查看我的提报功能类的设计**

客服查看我的提报这个功能需要将数据库中的相应提报信息取出来显示给客服看，以列表的形式展现，要支持分页查看，每页十条记录。系统根据当前登录的客服的工号去数据库中查找，再将信息显示出来。查看我的提报功能类图如图4.5所示。

图4.5查看我的提报功能类图

MyQuestionAction的execute方法调用getPageData方法，传入参数employeeId和pageNumber，然后调用MyQuestionService的getPageData方法，这个方法调用OaHelpdeskQuestionDAO中的findByExampleWithSize方法，只取出这一页要显示记录，然后用QuestionDetail类对OaHelpdeskQuestion进行包装，以便获取全部所需信息，最后返回一个QuestionDetail实例的列表，放入request对象中，在前端予以显示。

**查看提报的详细信息功能类的设计**

查看提报的详细信息功能指的是使用helpdesk系统的员工进入带有提报列表的页面之后，点击提报号的链接可以查看到提报的详细信息。点击提报号的链接将把提报号作为参数传给相应的action，然后action调用service，service再调用相关的dao将提报信息取出，提报信息取出后还要经过QuestonDetail的包装，然后传入request对象中，在前端通过OGNL表达式把提报的详细信息显示出来。

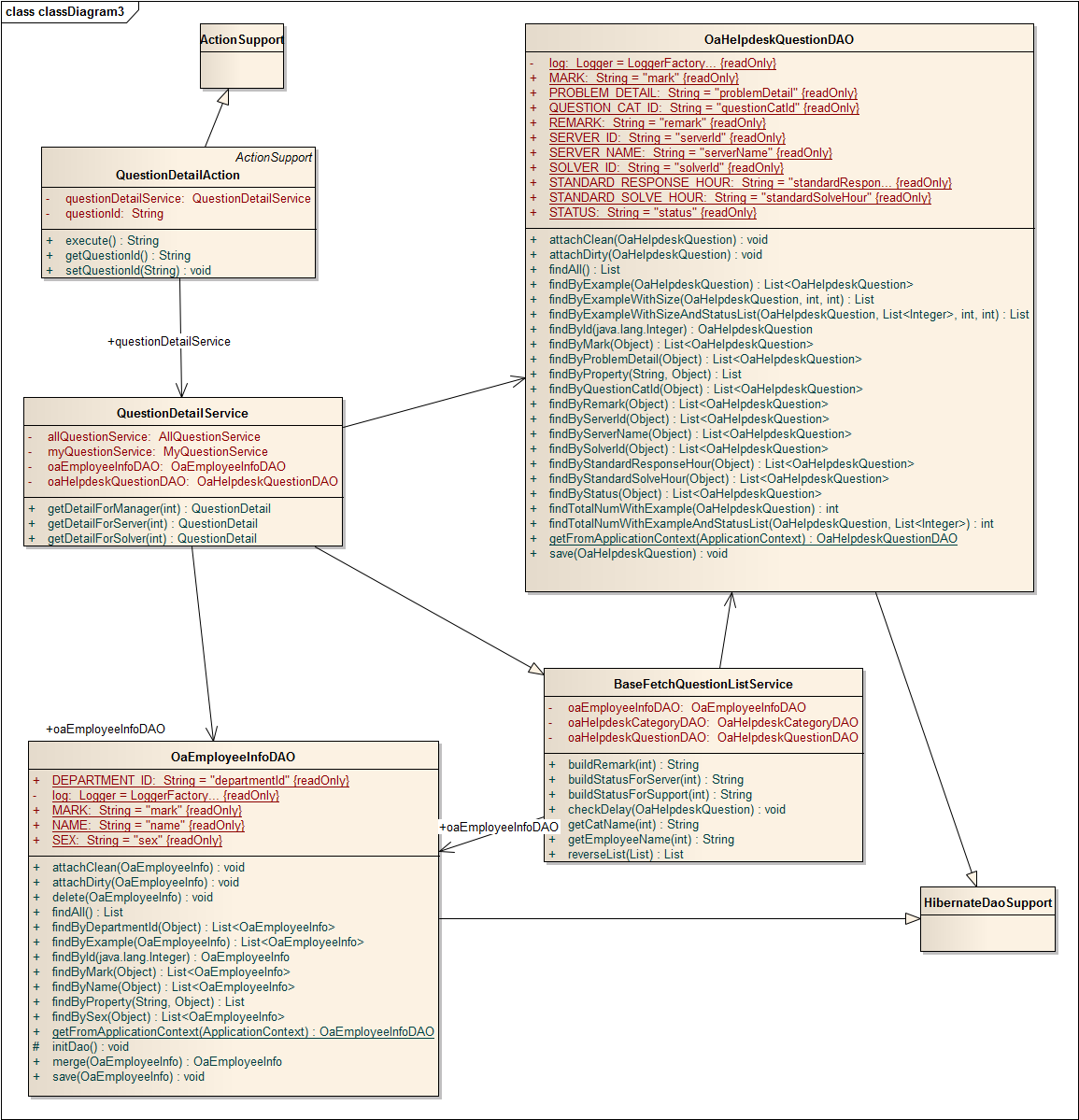
 查看提报的详细信息功能类图如图4.6所示。

图4.6查看提报的详细功能类图

**类的设计总结**

以上三个类图分别代表了系统中的三种不同的访问情景。图4.4代表的是向数据库中添加提报的信息，图4.5代表的是返回提报的列表，图4.6代表的是返回某个提报的详细信息。Helpdesk模块的所有使用人员的所有操作都大概可以分成以上三种情况，它们的设计都是类似的，只不过在一些细节上有所不同，因此在这里就不把所有的功能类的设计都一一列出了。

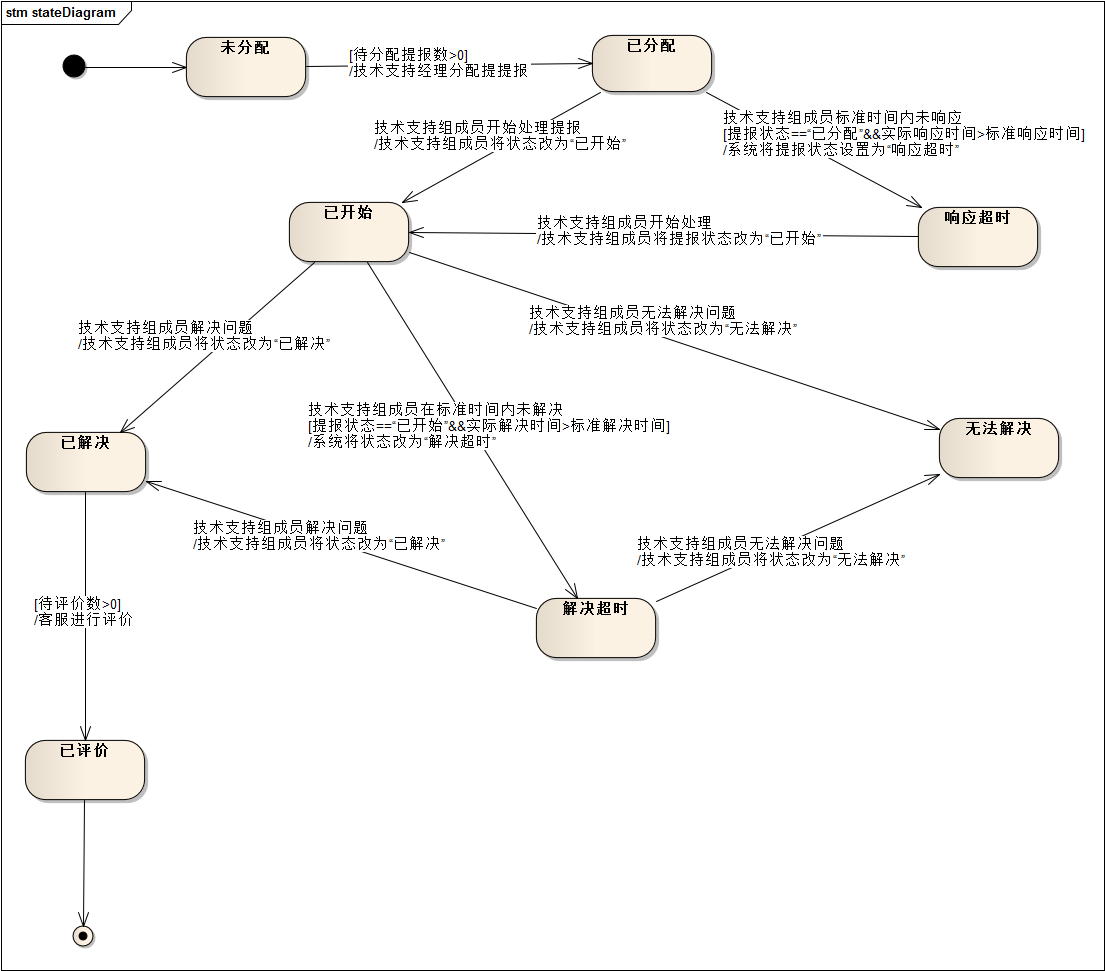
### 4.2.3 提报处理流程的设计

**提报处理流程描述**

从客服遇到问题到提报处理过程得到评价之间的流程可以如下概括：首先，客服打开AddQuestion.jsp页面，填入提报的信息，点击提交。提交的信息分别经过AddQuestionAction和AddQuestionService以及OaHelpdeskQuestionDAO的处理之后保存在数据库中。接着，技术经理登录系统，点击查看待分配的提报发出请求，请求依次经过ToDeliverQuestionAction，ToDeliverQuestionService以及OaHelpdeskQuestionDAO的处理之后，返回QuestionDetail的列表展现给技术支持经理。技术支持经理点击提报号的链接查看提报详细信息并进行分配，设置标准响应时长和标准处理时长，然后把它分配给技术支持组的某个员工。这一步通过ajax实现，依次经过SetSolverAction，SetSolverService以及OaHelpdeskQuestionDAO的处理之后提报就分配给了某个技术支持组成员。分配到该提报的员工登录系统，查看待我解决的提报，请求通过ToSolveByMeQuestionAction，ToSolveByMeQuestionService以及OaHelpdeskQuestionDAO处理之后返回QuestionDetail的列表展示到页面上。分配到该提报的员工查看该提报的详细信息，将提报状态改为“已开始”，标志着提报开始处理。当提报处理完成或者无法处理时，分配到该提报的员工将提报状态改为“已解决”或者“无法解决”。改变状态是通过ajax来完成的，请求依次经过SetStatusAction，SetStatusService以及OaHelpdeskQuestionDAO处理之后提报的状态便被改变。最后，客服登录系统查看待我评价的提报，请求通过ToRemarkAction，ToRemarkService以及OaHelpdeskQuestionDAO处理之后返回QuestionDetail列表显示给客服。客服点击该提报的提报号查看提报的详细信息并对提报进行评价，提供几个评价给客服选择，分别是“满意”，“一般”，“不满意”，请求通过RemarkAction，RemarkService以及OaHelpdeskQuestionDAO处理之后提报评价便被保存在数据库中。至此，整个提报的处理流程便完成了。

**提报状态变化的状态图**

在helpdesk模块中提报的状态共有八种，分别是“未分配”，“已分配”，“已开始”，“响应超时”，“解决超时”，“已解决”，“无法解决”，“已评价”。在helpdesk模块中用一个enum表示提报的八种状态。提报状态以及状态之间转化的状态图如图4.7所示。

图4.7 提报状态变化的状态图

## 4.3 Helpdesk模块的实现

### 4.3.1 SSH框架的配置

本系统使用Struts2+Spring+Hibernate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。本系统的SSH框架配置大量的采用了注解的方式，减少了需要配置的信息，使得配置文件不会过于庞大，同时减少了代码量。系统框架主要的配置文件是applicationContext.xml和struts.xml。

applicationContext.xml配置文件中主要配置了系统需要的各种bean，包括dataSource，sessionFactory，transactionManager等。系统的actionBean，serviceBean，dao等没有在这个配置文件中显示的配置出来，而是通过注解的方式来配置。使用“<context:annotation-config />”配置隐式地向 Spring 容器注册了Autowired/Common/Persistence/Required这四个AnnotationBeanPostProcessor，这样在代码中可以使用“Service”，“Controller”，“Repository”,“Autowired”等注释。使用“Service”表示这个bean处于业务逻辑层，使用“Controller”表示这个bean处于前端控制层，使用“Repository”表示这个bean处于数据持久层，使用“Autowired”注释表示让Spring容器自动进行某个成员变量的装配，不用手动提供getter和setter方法。这种隐式的注入方式使Spring容器在运行时自动的寻找标有“Service”，“Controller”，“Repository”等注解的bean并进行维护，需要给Spring设置一个自动扫描的包的范围，这是通过“<context:component-scan base-package="com.tuniu.action,com.tuniu.service,com.tuniu.dao" />”实现的，这条配置表示让Spring从action,service,dao这三个包中寻找带有“Service”，“Controller”，“Repository”等注解的bean进行维护。系统采用注解式的事务管理方法来实现事务管理，通过“<tx:annotation-driven transaction-manager="transactionManager"/>”这样的配置完成事务管理，在DAO中通过“Transactional”注解让Spring帮助系统进行事务管理。

Struts.xml主要配置了与struts相关的配置信息，定义了各个Action的访问标识，以及对应不同返回值应该访问的界面。Helpdesk模块相关的struts.xml中的配置可以分为两大类，一种是继承struts-default的包，返回类型一般是“dispatcher”，还有一种是继承json-default的包，返回类型是json。第一种根据返回的字符串决定显示的页面，第二种是ajax调用时使用的，返回的是json对象。为了与spring结合，让spring托管struts的action，action的class属性值设置为对应action的类名，第一个字母小写，这时因为在action中使用“Controller”注解action，默认的action的访问名字为首字母小写的类名。

关于hibernate的配置，这里没有使用配置文件，而是使用注解的方式。在POJO中通过“Entity”“Table”等注解配置hibernate的POJO类，实现对象关系映射。

### 4.3.2添加提报的实现

添加提报的实现过程如下。

首先，客服点击“添加提报”链接后进入AddQuestion.jsp页面，页面上显示客服工号，客服姓名，问题类别，问题描述，提报日期五个信息。其中客服工号，客服姓名是根据登录时在session中存储的内容得到的，提报日期是显示的是当天的日期，这些都是不能更改的。问题类别是select框，在里面可以选择提报的类型，这些类型保存在数据库中，由技术支持经理维护问题种类，在这里是通过ajax从数据库中取得的数据并加载到页面中。问题描述是一个textarea框，客服在这里输入提报问题的详细描述。

接着，客服点击提交，提交之后将会通过struts框架的机制将表单参数传递给“AddQuestionAction”，AddQuestionAction的属性有serverId，serverName，problemCat，problemDetail，addDate，这些属性有自己的getter和setter，struts框架会帮助我们将表单的参数传递给这些属性变量，并且会调用AddQuestionAction的execute方法。

然后，AddQuestionAction在execute方法中调用AddQuestionService的addQuestion方法，传递的参数为serverId，serverName，problemCat，problemDetail，addDate，如果抛出异常，execute方法返回“INPUT”，否则返回“SUCCESS”。AddQuestionService的addQuestion方法中新建一个OaHelpdeskQuestion对象，设置以上五个参数的值，再设置初始状态为0，表示“未分配”，设置mark为1，表示该记录有效。

最后调用OaHelpdeskQuestionDAO的save方法，将新建的OaHelpdeskQuestion对象作为参数传过去，OaHelpdeskQuestionDAO对象的save方法将OaHelpdeskQuestion保存在数据库中。

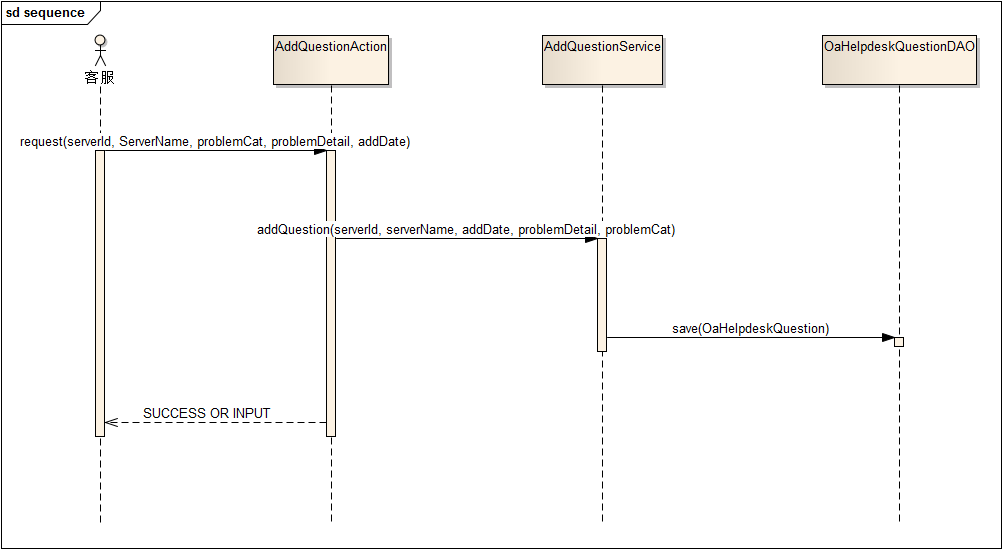
添加提报过程的顺序图如图4.8所示。

图4.8 添加提报的顺序图

### 4.3.3查看提报列表的实现

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。

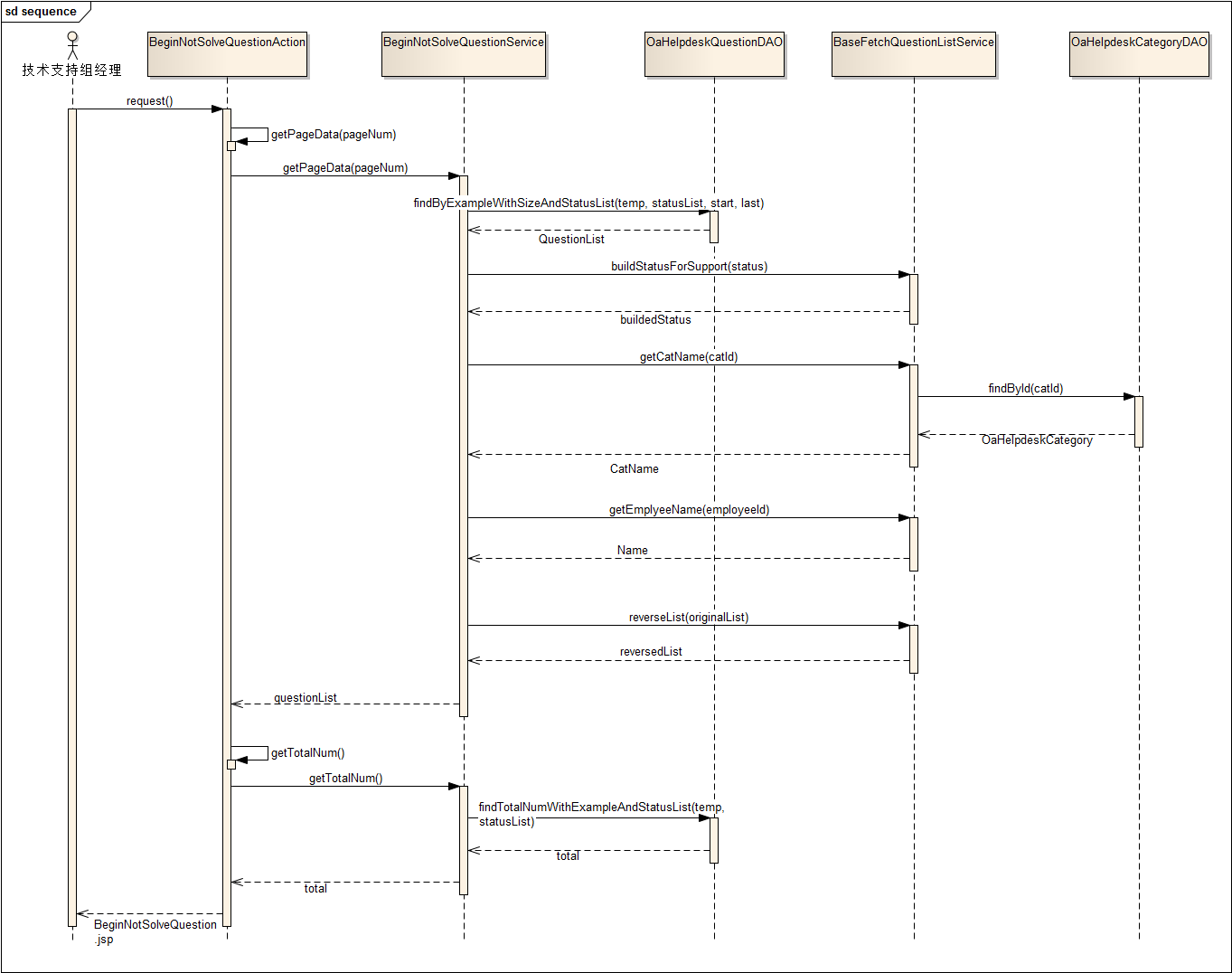
 查看提报列表的实现的顺序图如图4.9所示。

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查看提报详细信息的实现

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。

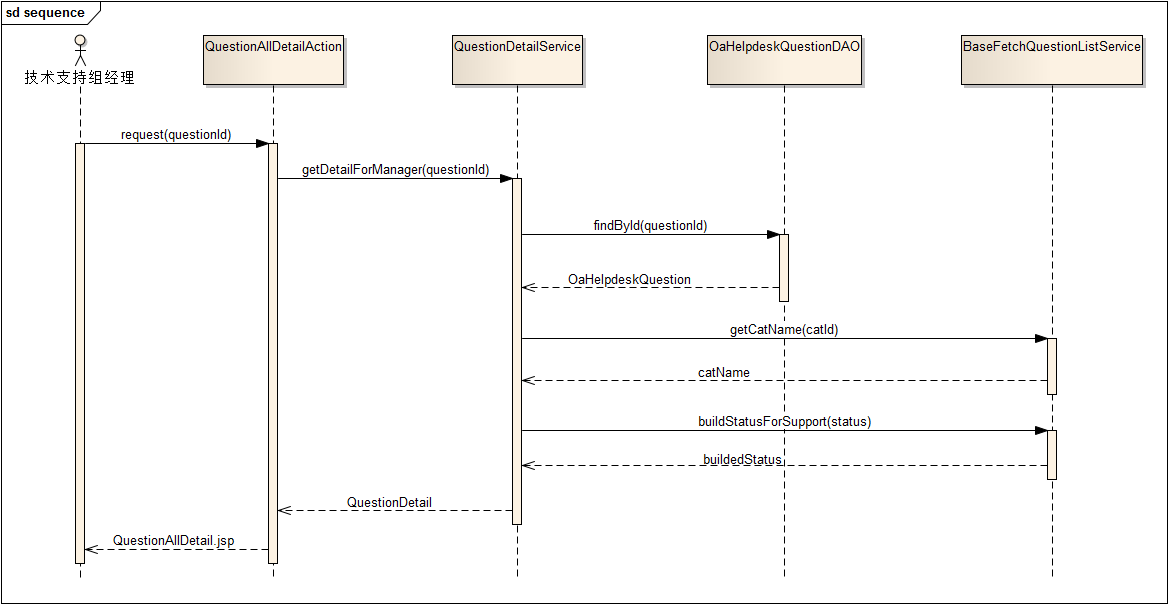
 查看提报的详细信息的顺序图如图4.11所示。

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。

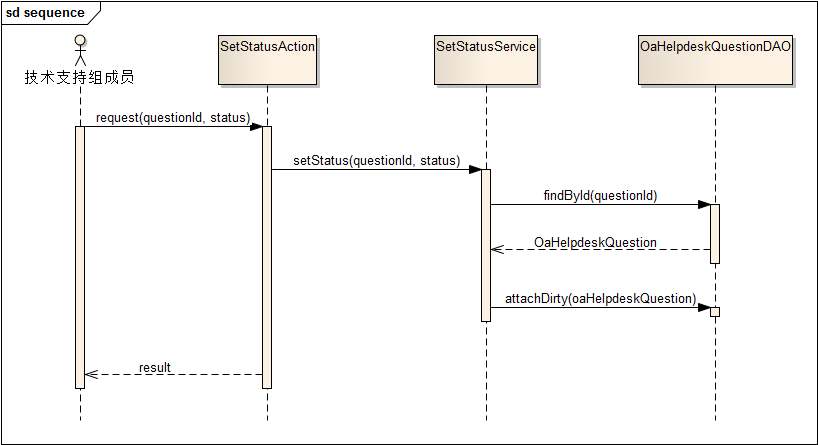
请求发给SetStatusAction之后，SetStatusAction的execute方法将被调用，execute方法中调用SetStatusService的setStatus方法，将questionId和status作为参数传递过去。

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。