项目名称 Project Name		密级 Confidentiality Level
合同管理系统		仅供收件方查阅
项目编号 Project ID	版本 Version	文档编号 Document Code
BP	1.0	BP

Contract Software System Design Specification
合同管理系统设计说明书

Prepared by 拟制		Date 日期	
Reviewed by 评审人		Date 日期	
Approved by 批准		Date 日期	

目　　录

Keywords 关键词：

合同　　流程　　权限

Abstract　摘　要：

本系需求规格说明书，说明了合同管理系统的功能结构，详细功能介绍、数据字典和性能、接口需求等。通过系统用例图、活动图和功能结构图等描述，讲解了合同管理系统的的详细需求，系统包含用户注册，登录，合同管理，查询统计，基础信息管理和系统管理等功能。

List of abbreviations 缩略语清单：

Abbreviations缩略语	Full spelling 英文全名	Chinese explanation 中文解释
MVC	Model-View-Controllor	模型-视图-控制器
DAO	Data Access Objects	数据访问对象
B/S	Browser/Server	浏览器/服务器结构

1 Introduction 简介

1.1 Purpose 目的

该需求规格说明书是关于合同管理系统的功能需求和性能需求的描述，该说明书的预期读者为：

用户；

项目管理人员；

测试人员；

设计人员；

开发人员。

重点描述了流程管理和权限管理两个模块的功能需求，明确所要开发的软件应具有的功能、性能与界面，使系统分析人员及软件开发人员能清楚地了解用户的需求。

1.2 Scope 范围

1.2.1 Name 软件名称

合同管理系统

1.2.2 Functions 软件功能

参考《合同管理系统需求规格说明书》

1.2.3 Applications软件应用

合同管理系统，从收集合同基础参数开始，进行合同起草，与用户进行谈判、会签并修订合同，签订合同，记录合同执行过程，进行合同执行跟踪管理直到合同终止，贯穿合同管理全过程。

1.3 软件系统上下文定义

合同管理系统为一个B/S结构的系统，系统的访问结构如下：

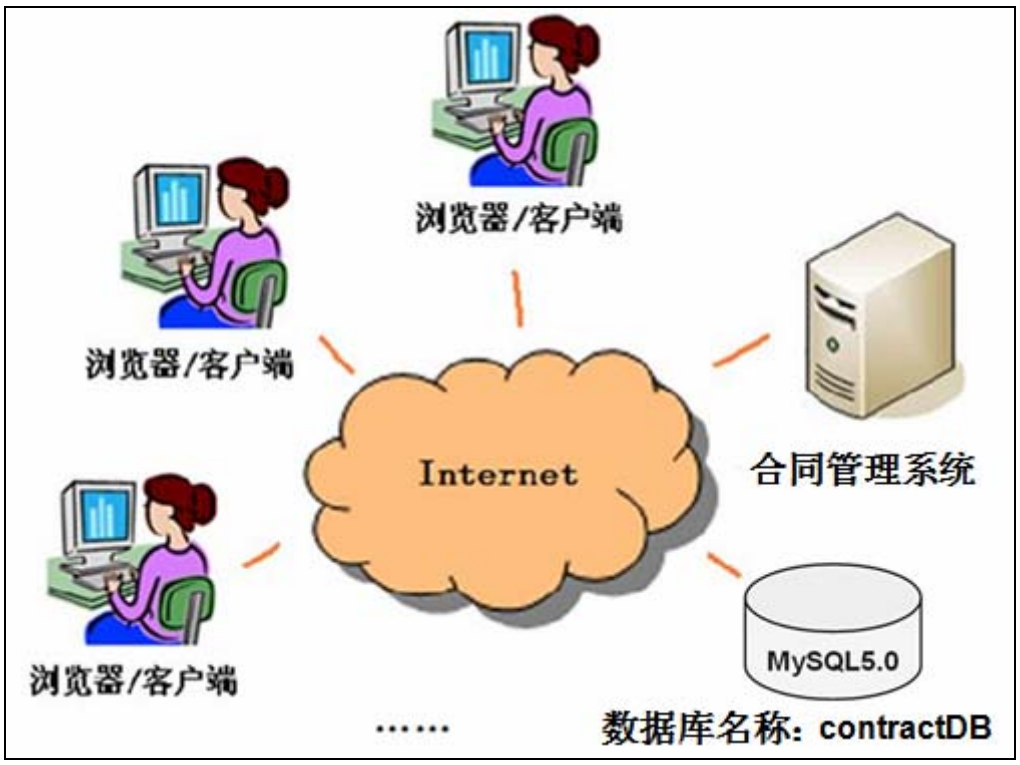

图1-1 系统上下文

1.4 Design Considerations设计思路

1、系统的三层架构，包的结构图如下

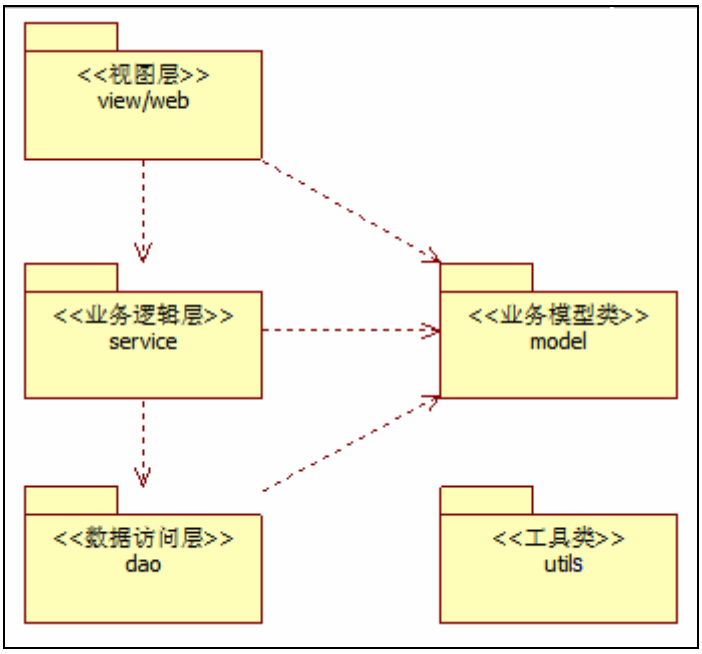

图 1-2 系统包结构

系统分为表示层，业务逻辑层，数据访问层，分别对应包(com.ruanko.web)，

(com.ruanko.service)，(com.ruanko.dao)，他们通过调用业务模型类(即com.ruanko.model包中的类)来完成数据的传递，其中有一部分类会被多处公用，就像工具一样,，这时把他们统一放在com.ruanko.utils包中。

2、程序的开发模式和三层结构

系统使用Model 2开发模式：JSP+Servlet+JavaBean(MVC)，它与三层架构的结合中，Servlet(控制器)和JSP页面(视图)充当表示层，JavaBean即模型(包含业务逻辑，数据访问和业务模型类等)充当业务逻辑层和数据访问层。

3、三层结构与Model 2结合的包层次结构

程序架构	文件与目录结构
表示层	JSP文件
	com.ruanko.web
业务逻辑层	com.ruanko.service
数据访问层	com.ruanko.dao com.ruanko.dao.impl
模型类	com.ruanko.model

文件夹目录

文件夹名	说明
css	存放样式文件
images	存放图片
js	存放脚本文件
upload	存放上传的文件

2 Level 1 Design Description 第1层设计

2.1 System Architecture系统结构

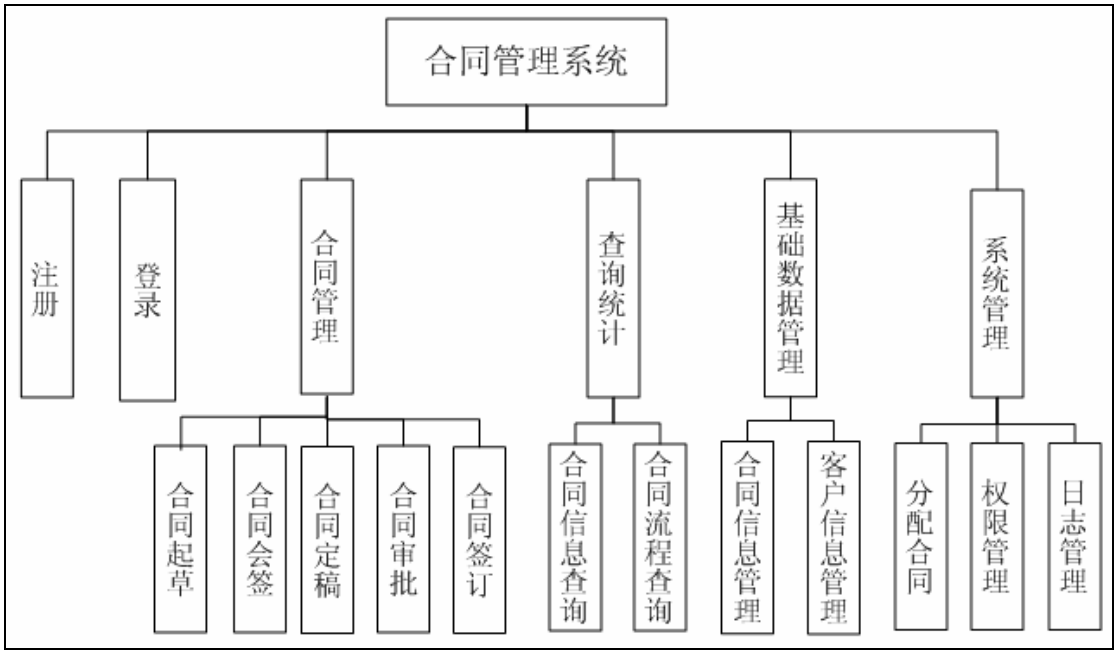

图 2-1 系统功能结构

1、合同管理

合同操作员登录后，可以进行起草操作；点击其中的待会签合同，可以对分配给自己的合同进行会签操作；点击待定稿合同，可以对合同进行定稿操作；点击待审批合同，可以对合同进行审批操作；相应的，也可以对合同进行签订操作。

2、查询统计

管理员登录系统后，可以对合同基本信息进行查询，如根据名称等进行查询，除此之外，也可以跟踪合同流程的操作步骤，查询合同所处的状态。

3、基础数据管理

管理员进入系统后，能够对客户信息进行新增、查询、修改、删除。也可以对合同进行新增、查询、修改、删除。

4、系统管理

管理员登录系统后，可以对起草的合同进行人员分配，对系统的功能模块进行权限分配，除此之外，对于系统中数据库的操作，应建立操作日志，必要时可返回单次或批量操作以前的状态，日志操作是伴随用户操作功能是自动发生并入库。

2.2 Representation of the Business Flow业务流程说明

1、合同管理流程：

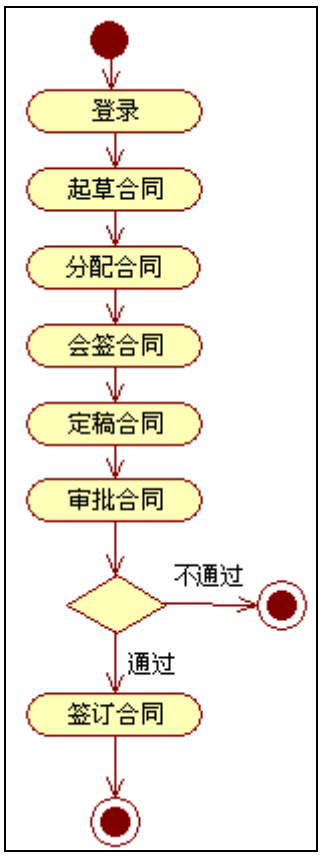

图 2-2 合同管理流程

2.3 Decomposition Description分解描述

2.3.1 注册

1、功能设计描述

在注册页面填写注册信息，要求填写用户名（必填，4~20个汉字、字母、数字或下划线组成的字符串，且不能有重名注册），密码（必填，6~12个字符），确认密码（和密码保持一致），进行注册。

提交注册信息时，有必填项未填写或格式不正确时，提示用户正确填写相应项，当注册失败时在注册页面显示“注册失败！”。当所有信息填写正确，提交表单，注册成功跳转到登录页面，如果程序出现异常则跳转到异常页面。

2、 功能实现

用户提交注册请求，RegisterServlet从注册页面中取得用户注册信息，调用业务逻辑层UserService的register()方法处理注册业务。

UserService调用数据访问层UserDaoImpl的isExist()处理同名验证逻辑，不存在同名用户则调用add()方法保存用户的注册信息。

各层之间通过实体类User对象传递参数。

2.3.2 登录/登出

1、功能设计描述

在登录页面填写用户名、密码，点击“登录”按钮进行登录。要求用户名和密码不能为空，为空则在登录页面给出错误提示。登录失败时在登录页面提示 “用户名或密码错误！”；系统异常，跳转到异常页面；登录成功跳转到新用户页面，并显示当前登录用户名。

已登录的用户，点击新用户页面的“注销登录”链接，清除用户的登录状态。返回到登录页面。

2、功能实现

（1）登录

用户提交登录请求，LoginServlet从登录页面中取得用户登录信息，调用业务逻辑层UserService的login()方法处理登录业务。

UserService调用数据访问层UserDaoImpl.login()方法查询匹配用户信息，返回用户编号。

（2）登出

LogoutServlet接收新用户页面头部的“注销登录”请求，移除session中的用户信息，跳转到登录页面。

2.3.3 合同管理

1、简介

合同操作员登录后，如果拥有起草权限，可以进行起草操作；点击其中的待会签合同，可以对分配给自己的合同进行会签操作；点击待定稿合同，可以对合同进行定稿操作；点击待审批合同，可以对合同进行审批操作；相应的，也可以对合同进行签订操作。

2、功能列表

功能名称	功能描述
起草合同	起草合同时，填写合同信息，可以上传附件。
会签合同	合同分配后，会签人员查看待会签合同列表，可选择某一合同查看其内容，并录入会签意见进行会签。
定稿合同	起草人查看待定稿合同列表，可选择某一合同查看其会签意见，并修改合同进行定稿。
审批合同	审批人员查看待审批合同列表，可选择某一合同查看其内容，并填写审批意见进行审批。
签订合同	签订人员查看待签订合同列表，可选择某一合同查看其内容，并填写签订信息进行签订。

2.3.3.1 起草合同

1、功能设计描述

合同操作员在起草合同页面填写合同名称，开始时间，结束时间，合同内容，客户，并可以上传合同附件。

成功起草合同后，合同状态为“起草”，等待管理员对该合同进行人员分配。

合同的状态：

合同状态	说明
起草	合同操作员成功起草
会签完成	合同对应的会签人员全部会签完毕
定稿完成	合同起草人根据会签意见将合同修订完毕
审批完成	合同对应的审批人员全部审批完毕
签订完成	合同对应的会签人员全部会签完毕

2、 功能实现

用户起草合同，DraftServlet使用doPost()处理用户的起草合同请求，该方法调用ContractService中的draft()方法完成起草合同逻辑。

在draft()方法中，首先生成合同编号，并设置到Contract中，调用ContractDao中add()保存合同信息，如果起草成功，则调用ConStateDao中的add()保存合同状态信息。

2.3.3.2 会签合同

1、功能设计描述

会签人员查看待会签合同列表,可选择某一合同查看其内容,并录入会签意见进行会签。

2、 功能实现

(1) 查看“待会签”合同列表

涉及的会签人员点击管理页面的“待会签合同”超链接，ToDhqhtListServlet获取查询请求，由doPost()方法调用ContractService中的getDhqhtList()处理查询待会签合同请求。

(2) 查看合同详情

会签人员针对某一个合同，点击合同名称超链接，ContractDetailServlet获取查询请求，由doPost()方法调用ContractService中的getContractDetail()方法查询合同详情。

(3) 会签合同

会签人员针对某一个合同，点击“会签”超链接，ToAddHQOpinionServlet获取查询请求，由doPost()方法调用ContractService中的getContract()方法查询合同基本信息。

当会签人员在会签合同页面，输入会签意见并提交时，AddHQOpinionServlet获取会签合同请求，调用ContractService中的counterSign()方法处理会签合同业务逻辑。

2.3.3.3 定稿合同

1、功能设计描述

起草人查看待定稿合同列表，可选择某一合同查看其定稿意见，并修改合同进行定稿。

2、 功能实现

(1) 查看“待定稿”合同列表

起草人点击管理页面的“待定稿合同”超链接，ToDdghtListServlet获取查询请求，由doPost()方法调用ContractService中的getDdghtList()处理查询待定稿合同请求。

(2) 查看会签意见

起草人针对某一个合同，点击“会签意见”超链接，ShowHQOpinionServlet获取查询请求，由doPost()方法调用ContractService中的showHQOpinion()方法查询会签意见。

(3) 定稿合同

起草人针对某一个合同，点击“定稿”超链接，ToDgContractServlet获取查询请求，由doPost()方法调用ContractService中的getContract()方法查询合同基本信息。

当起草人在定稿合同页面，输入合同内容并提交时，DgContractServlet获取定稿合同请求，调用ContractService中的finalize()方法处理定稿合同业务逻辑。

2.3.3.4 审批合同

1、功能设计描述

审批人员查看待审批合同列表，可选择某一合同查看其内容，并填写审批意见进行审批。

2、 功能实现

(1) 查看“待审批”合同列表

涉及的审批人员点击管理页面的“待审批合同”超链接，ToDshphtListServlet获取查询请求，由doPost()方法调用ContractService中的getDshphtList()处理查询待审批合同请求。

(2) 审批合同

审批人员针对某一个合同，点击“审批”超链接，ToAddSHPOpinionServlet获取查询请求，由doPost()方法调用ContractService中的getContract()方法查询合同基本信息。

当审批人员在审批合同页面，输入审批意见并提交时，AddSHPOpinionServlet获取审批合同请求，调用ContractService中的approve()方法处理审批合同业务逻辑。

2.3.3.5 签订合同

1、功能设计描述

签订人员查看待签订合同列表，可选择某一合同查看其内容，并填写签订信息进行签订。

2、 功能实现

(1) 查看“待签订”合同列表

涉及的签订人员点击管理页面的“待签订合同”超链接，ToDqdhtListServlet获取查询请求，由doPost()方法调用ContractService中的getDqdhtList()处理查询待签订合同请求。

(2) 签订合同

签订人员针对某一个合同，点击“签订”超链接，ToAddQDOpinionServlet获取查询请求，由doPost()方法调用ContractService中的getContract()方法查询合同基本信息。

当签订人员在签订合同页面，输入签订意见并提交时，AddQDOpinionServlet获取签订

合同请求，调用ContractService中的sign()方法处理签订合同业务逻辑。

2.3.4 查询统计

2.3.4.1 合同信息查询

1、功能设计描述

根据合同名称分页查询合同信息。

2、 功能实现

管理员点击管理页面的“合同信息”超链接，QueryConInfoServlet获取查询请求，由doPost()方法调用ContractService中的queryContract()处理查询合同信息请求。

2.3.4.2 合同流程查询

1、功能设计描述

根据合同名称分页查询合同流程。

2、 功能实现

管理员点击管理页面的“流程查询”超链接，QueryProcessServlet获取查询请求，由doPost()方法调用ContractService中的queryProcess()处理查询合同流程请求。

2.3.5 基础数据管理

2.3.5.1 合同信息管理

对合同信息进行新增，查询,修改和删除操作。

根据实际需要进行该功能设计。

2.3.5.2 客户信息管理

对客户信息进行新增，查询，修改和删除操作。

根据实际需要进行该功能设计。

2.3.6 系统管理

1、简介

管理员可以分配合同，进行权限管理和日志管理等。

2、功能列表

功能名称	功能描述
分配合同	管理员查看待分配合同列表，可选择某一合同，为该合同指定会签、审批和签订人，完成分配操作。
权限管理	包含用户管理，角色管理，功能操作，分配权限。管理员可以对权限进行配置和管理。
日志管理	对于系统中数据库的操作，除查询、统计外，建立操作日志。

2.3.6.1 分配合同

1、功能设计描述

管理员选择状态为“起草”的待分配的合同，指定参与会签、审批、签订的人员，完成分配。

2、 功能实现

(1) 查看“待分配”合同列表

管理员点击管理页面的“待分配合同”超链接，ToDfphtListServlet获取查询请求，由doPost()方法调用ContractService中的getDfphtList()处理查询待分配合同请求。

(2) 分配合同

管理员针对某一个合同，点击“分配”超链接，ToAssignOperServlet获取查询请求，由doPost()方法调用ContractService中的getContract()方法查询合同基本信息。调用UserService中的getUserListByRoleId()方法查询用户基本信息。

当管理员在分配合同页面，分配完相关的操作人员并提交时，AssignOperServlet获取分配合同请求，调用ContractService中的distribute()方法处理分配合同业务逻辑。

2.3.6.2 权限管理

权限管理包含用户管理，角色管理，功能操作，分配权限。管理员可以对权限进行配置和管理。

本次针对“分配权限”功能进行设计，其他功能根据实际需要进行该功能设计。

1、功能设计描述

管理员登录系统后，查看待分配权限的用户列表，然后，针对某一用户，点击“授权”超链接，转到分配用户权限页面，进行分配操作。

2、 功能实现

(1) 查看用户权限列表

管理员点击管理页面的“配置权限”超链接，ToYhqxListServlet获取查询请求，由doPost()方法调用UserService中的getYhqxList()处理查询待分配权限的用户请求。

(2) 权限分配

管理员针对某一个用户，点击“授权”超链接，ToAssignPermServlet获取查询请求，由doPost()方法调用UserService中的getRoleList()方法查询角色列表信息。

当管理员在分配权限页面，分配完用户角色并提交时，AssignPermServlet获取分配权限请求，调用UserService中的assignPermission()方法处理分配权限业务逻辑。

2.3.6.3 日志管理

对于系统中数据库的操作，建立操作日志，必要时可返回单次或批量操作以前的状态。根据实际需要进行该功能设计。

2.4 Interface Description接口描述

在使用JDBC技术进行数据库访问时，主要涉及Connection、PreparedStatement和ResultSet接口。

1、JDBC Connection对象接口

方法名	描述
prepareStatement	预编译SQL语句
close	关闭数据库连接

2、JDBC PreparedStatement对象接口

方法名	描述
executeUpdate	执行数据库更新操作
executeQuery	执行数据库操作操作
close	关闭查询语句

3、JDBC ResultSet对象接口

方法名	描述
next	定位到记录信息的下一行
getInt(String columnLabel)	获取指定索引位置的信息

close	关闭查询语句

3 Database Design 数据库设计

根据合同管理系统的需求和业务流程，系统数据信息可能来自于界面操作，流程处理，数据存储等过程，对这些过程中涉及到的数据信息进行抽取和分析，进行项目数据库设计。

3.1 概念模型

根据抽样出的数据字典，识别实体，实体的属性和实体间的关系，通过E-R模型进行设计。

3.2 物理模型

以下是基于MySQL的物理模型。

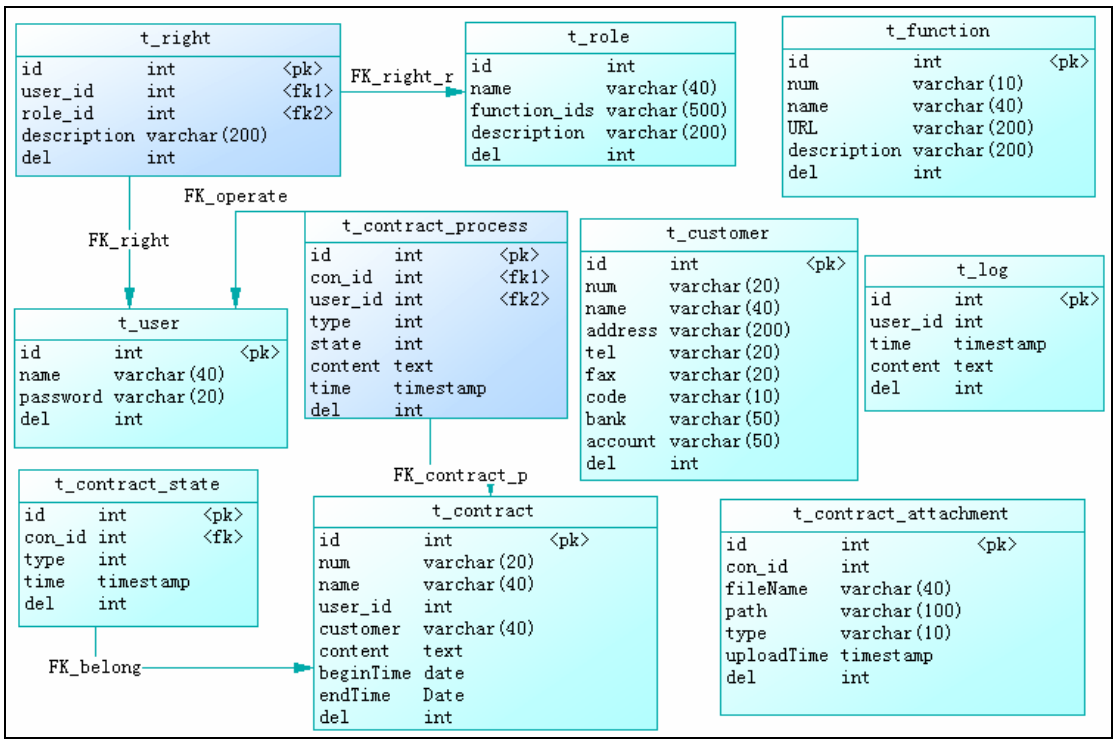

图3-1 基于MySQL数据库物理模型

3.3 数据库表设计

1、用户(t_user)表

Name	Code	Instruction	Type/Length
id	id	int	主键，每次自增1
用户名称	name	varchar(40)	

密码	password	varchar(20)	
删除状态	del	int	0 未删除，1 已删除

2、角色(t_role)表

Name	Code	Type/Length	Instruction
id	id	int	主键，每次自增1
角色名称	name	varchar(40)	
角色描述	description	varchar(100)	
功能操作	func_ids	varchar(500)	存储功能操作，多个之间通过逗号“,”进行拼接。
删除状态	del	int	0 未删除，1 已删除

3、权限(t_right)表

Name	Code	Type/Length	Instruction
id	id	int	主键，每次自增1
用户_id	use_id	int	外键，引用用户表id
角色_id	rol_id	int	外键，引用角色表id
描述	description	varchar(100)	
删除状态	del	int	0 未删除，1 已删除

4、功能(t_function)表

Name	Code	Type/Length	Instruction
id	id	int	主键，每次自增1
功能编号	num	varchar(10)	
功能名称	name	varchar(40)	
操作URL	URL	varchar(100)	
功能描述	description	varchar(100)	
删除状态	del	int	0 未删除，1 已删除

5、合同(t_contract)表

Name	Code	Type/Length	Instruction
id	id	int	主键，每次自增1
合同编号	num	varchar(20)	
合同名称	name	varchar(40)	
客户	customer	varchar(40)	
开始时间	beginTime	date	
结束时间	endTime	date	
合同内容	content	text	
用户编号	user_id	int	
删除状态	del	int	0 未删除，1 已删除

6、合同操作流程(t_contract _process)表

Name	Code	Type/Length	Instruction
id	id	int	主键，每次自增1
合同_id	con_id	int	外键，引用合同表id
操作类型	type	int	1 会签，2 审批，3签订
操作状态	state	int	0 未完成，1 已完成，2 已否决
用户_id	use_id	int	外键，引用用户表id
操作内容	content	text	
操作时间	time	datetime	
删除状态	del	int	0 未删除，1 已删除

7、合同操作状态(t_contract _state)表

Name	Code	Type/Length	Instruction
id	id	int	主键，每次自增1
合同_id	con_id	int	外键，引用合同表id
操作类型	type	int	1 起草，2 会签完成，3 定稿完成，4 审批完成，5 签订完成
完成时间	time	datetime	
删除状态	del	int	0 未删除，1 已删除

8、日志(t_log)表

Name	Code	Type/Length	Instruction
id	id	int	主键，每次自增1
操作人id	userid	int	存储用户表中id的值
操作内容	content	text	
操作时间	time	datetime	
删除状态	del	int	0 未删除，1 已删除

9、客户(t_customer)表

Name	Code	Type/Length	Instruction
id	id	int	主键，每次自增1
客户编号	num	varchar(20)	
客户名称	name	varchar(40)	
地址	address	varchar(100)	
电话	tel	varchar(20)	
传真	fax	varchar(20)	
邮编	code	varchar(10)	
银行名称	bank	varchar(50)	
银行账户	account	varchar(50)	
删除状态	del	int	0 未删除，1 已删除

10、合同附件(t_contract _attachment)表

Name	Code	Type/Length	Instruction
id	id	int	主键，每次自增1
合同_id	con_id	int	
附件名称	fileName	varchar(40)	
附件路径	path	varchar(100)	
附件类型	type	varchar(20)	
上传日期	uploadTime	datetime	
删除状态	del	int	0 未删除，1 已删除

3.4 基础数据配置

初始化客户信息。

```
INSERT INTO `t_customer` VALUES (1, 'Cus20131211182300001', 'HB company', 'hubei wuhan', '11111111111', '12121212', '430000', 'Bank Of China', '621661***', 0);

INSERT INTO `t_customer` VALUES (2, 'Cus20131211182300002', 'BJ company', 'beijing', '22222222', '34213467', '100000', 'Agricultural Bank of China', '622848***', 0);

INSERT INTO `t_customer` VALUES (3, 'Cus20131211182300003', 'Jack Wang', 'Shanghai', '14231116', '45678234', '200000', 'Industrial and Commercial Bank of China Limited', '530990***', 0);
```

初始化功能信息。

```
INSERT INTO `t_function` VALUES (1, '001', 'QCHT', null, null, 0);
INSERT INTO `t_function` VALUES (2, '002', 'DGHT', null, null, 0);
INSERT INTO `t_function` VALUES (3, '003', 'CXHT', null, null, 0);
INSERT INTO `t_function` VALUES (4, '004', 'SCHT', null, null, 0);
INSERT INTO `t_function` VALUES (5, '005', 'HQHT', null, null, 0);
INSERT INTO `t_function` VALUES (6, '006', 'SPHT', null, null, 0);
INSERT INTO `t_function` VALUES (7, '007', 'QDHT', null, null, 0);
INSERT INTO `t_function` VALUES (8, '008', 'FPHQ', null, null, 0);
INSERT INTO `t_function` VALUES (9, '009', 'FPSP', null, null, 0);
INSERT INTO `t_function` VALUES (10, '010', 'FPQD', null, null, 0);
INSERT INTO `t_function` VALUES (11, '011', 'LCCX', null, null, 0);
INSERT INTO `t_function` VALUES (12, '012', 'XZYH', null, null, 0);
```

```
INSERT INTO `t_function` VALUES (13, '013', 'BJYH', null, null, 0);
INSERT INTO `t_function` VALUES (14, '014', 'CXYH', null, null, 0);
INSERT INTO `t_function` VALUES (15, '015', 'SCYH', null, null, 0);
INSERT INTO `t_function` VALUES (16, '016', 'XZJS', null, null, 0);
INSERT INTO `t_function` VALUES (17, '017', 'BJJS', null, null, 0);
INSERT INTO `t_function` VALUES (18, '018', 'CXJS', null, null, 0);
INSERT INTO `t_function` VALUES (19, '019', 'SCJS', null, null, 0);
INSERT INTO `t_function` VALUES (20, '020', 'XZGN', null, null, 0);
INSERT INTO `t_function` VALUES (21, '021', 'BJGN', null, null, 0);
INSERT INTO `t_function` VALUES (22, '022', 'CXGN', null, null, 0);
INSERT INTO `t_function` VALUES (23, '023', 'SCGN', null, null, 0);
INSERT INTO `t_function` VALUES (24, '024', 'PZQX', null, null, 0);
INSERT INTO `t_function` VALUES (25, '025', 'XZKH', null, null, 0);
INSERT INTO `t_function` VALUES (26, '026', 'BJKH', null, null, 0);
INSERT INTO `t_function` VALUES (27, '027', 'CXKH', null, null, 0);
INSERT INTO `t_function` VALUES (28, '028', 'SCKH', null, null, 0);
INSERT INTO `t_function` VALUES (29, '029', 'CXRZ', null, null, 0);
INSERT INTO `t_function` VALUES (30, '030', 'SCRZ', null, null, 0);
```

初始化角色信息。

```
INSERT INTO `t_role` VALUES (1, 'admin', 'To implement the system
management and contract management',
'003,004,008,009,010,011,012,013,014,015,016,017,018,
019,020,021,022,023,024,025,026,027,028,029,030', 0);

INSERT INTO `t_role` VALUES (2, 'operator', 'operate contract',
'001,002,003,005,006,007,011,027', 0);
```

初始化用户信息。

```
INSERT INTO `t_user` VALUES (1, 'admin', '123456', 0);
INSERT INTO `t_user` VALUES (2, 'tom', '123456', 0);
INSERT INTO `t_user` VALUES (3, 'lucy', '123456', 0);
INSERT INTO `t_user` VALUES (4, 'lily', '123456', 0);
INSERT INTO `t_user` VALUES (5, 'jack', '123456', 0);
INSERT INTO `t_user` VALUES (6, 'sanri', '123456', 0);
```

初始化权限信息。

```
INSERT INTO `t_right` VALUES (1, 1, 1, 'admin', 0);
INSERT INTO `t_right` VALUES (2, 2, 2, 'operator', 0);
```



```
INSERT INTO `t_right` VALUES (3, 3, 2, 'operator', 0);
INSERT INTO `t_right` VALUES (4, 4, 2, 'operator', 0);
INSERT INTO `t_right` VALUES (5, 5, 2, 'operator', 0);
```

4 UI Design 界面设计

根据需求设计合同管理系统的如下页面：

(1) 注册页面和登录页面。

(2) 合同管理页面。

起草，会签，定稿，审批，签订等页面，合同详情页面，合同列表页面，查看会签意见页面。

(3) 系统管理页面。

分配合同页面，权限管理页面，日志管理页面。

4.1 登录页面

1、界面原图

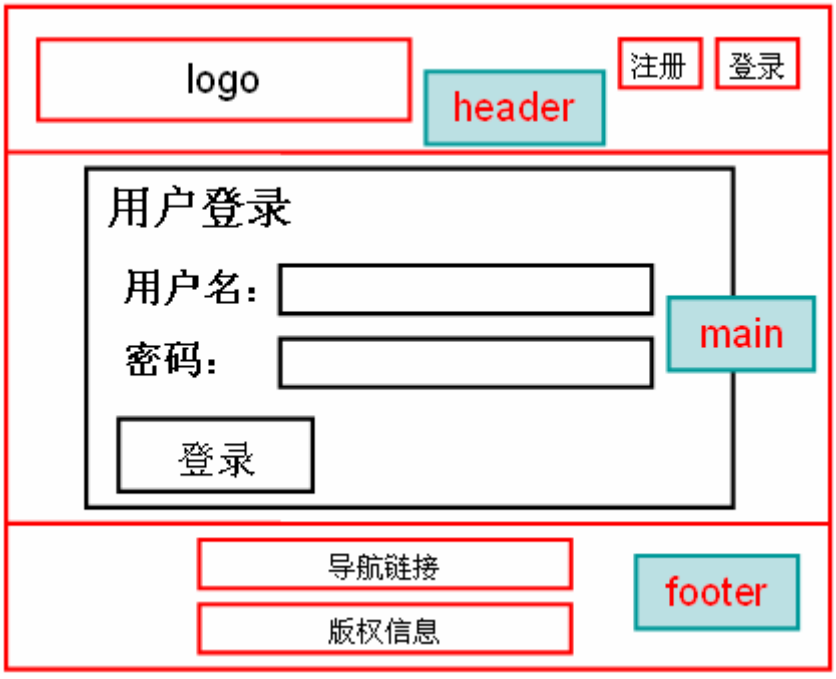

图4-1 登录页面

2、界面说明

主体部分为相应表单，以接收登录信息。

4.2 管理页面

1、界面原图

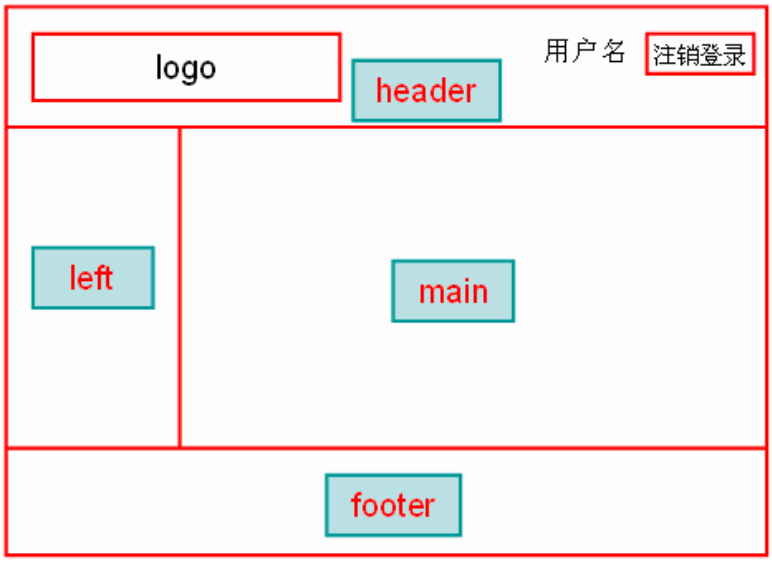

图4-2 管理页面

2、界面说明

管理页面采用HTML框架网页Frame进行布局实现。

主体分为左边和右侧。

点击左侧的管理超链接，在右边加载对应功能页面。

4.3 合同起草页面

1、界面原图

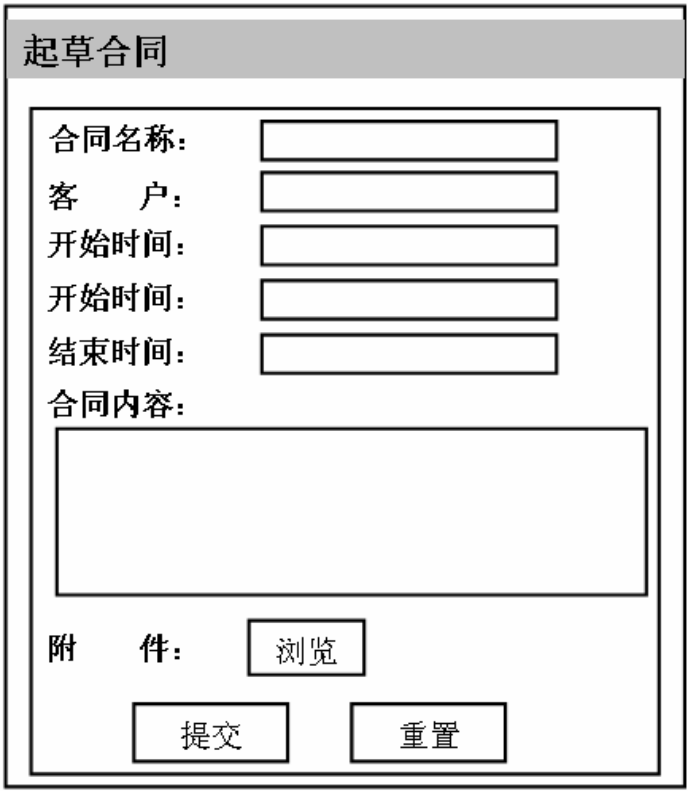

图4-3 合同起草页面

2、界面说明

主体为合同表单，使用<table>进行布局。

4.4 合同列表页面

1、界面原图

图4-4 合同列表页面

2、界面说明

以列表显示合同信息，并能按条件查询合同。

5 Detailed Design 详细设计

5.1 实体类设计

实体类用于层之间传递数据，实体类有一部分与数据库表相对应，有一部分根据实际业务需要,建立业务实体类。

与数据库表对应的实体类：User，Contract，ConAttach，ConProcess，ConState，Customer，Function，Log，Right和Role。

根据业务需要，建立业务实体类：ConBusiModel，ConDetailBusiModel，CsignatureOpinion和PermissionBusiModel等。

在设计实体类时，类主要包括三部分内容：

(1) 每一个类包括私有成员变量。

(2) getXX()和setXX()方法：为public，用于访问类中私有成员变量(属性)，并分别建立与私有变量一一对应的getXX()和setXX()方法。

(3) 无参构造方法：用于初始化私有成员变量，给成员变量赋初值。

实体类存放在com.ruanko.model包中。

5.1.1 Contract

5.1.1.1 简介

合同实体类：用于建立和数据表（合同表）之间的映射关系，声明私有属性和数据库表中的字段相对应，提供初始化构造方法，setter和getter方法供外部调用，在程序中实体类被用作数据传递的载体。

5.1.1.2 类图

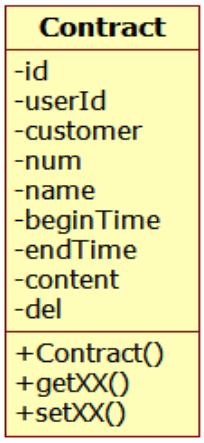

图5-1 合同实体类

5.1.1.3 属性

以上类图和下列表格中，只列出了合同实体类中的部分关键属性。

可见性	属性名称	类型	说明
private	id	int	合同id
private	userId	int	用户编号
private	customer	String	客户
private	num	String	合同编号
private	name	String	合同名称
private	beginTime	Date	开始时间
private	endTime	Date	结束时间
private	content	String	合同内容
private	del	int	删除字段（1：已删除，2：未删除）

5.1.1.4 方法

在实体类中会为这些私有属性提供setter和getter方法供外部调用，而不通过属性名称直接调用。通过setter方法给实体的属性赋值，通过getter方法取实体的属性值。下面我们以合同的id为例来进行说明，其他属性同理。

1、setId()

(1) 方法描述

Prototype 方法原型	public void setId()
Description 功能描述	给合同实体的id属性赋值
Calls 调用方法	无
Input 输入参数	id
Output 输出参数	无
Return 返回值	无
Exception 抛出异常	无

(2) 实现描述

现在给合同实体类的id赋值，假设contract为其一个实例：contract.setId(1)表示将合同的id属性设置为1。

2、getId()

(1) 方法描述

Prototype 方法原型	public int getId()
Description 功能描述	取出合同实体的Id属性值
Calls 调用方法	无
Input 输入参数	无
Output 输出参数	无
Return 返回值	id

Exception 抛出异常	无

(2) 实现描述

现在取出合同实体类的id的值，假设contract为其一个实例：contract.getId()表示取出合同的id的值。

3、Contract()

(1) 方法描述

Prototype 方法原型	public Contract()
Description 功能描述	构造方法，为实体属性赋初始值。
Calls 调用方法	无
Input 输入参数	无
Output 输出参数	无
Return 返回值	无
Exception 抛出异常	无

(2) 实现描述

定义一个实体的构造方法，方法名称和类名相同：public Contract()，构造方法中可以根据需要对属性赋指定值，如下为Contract实体的构造方法。

```
/**
 * 无参构造方法：为对象属性赋初始值
 */
public Contract(){
    this.id = 0;
    this.userId = 0;
    this.customer = "";
    this.num = "";
    this.name = "";
    this.beginTime = new Date();
    this.endTime = new Date();
    this.content = "";
    this.del = 0;
}
```

5.1.2 User

用户实体类：用于建立和数据表（用户表）之间的映射关系，其结构与Contract一致。属性如下：

可见性	属性名称	类型	说明
private	id	int	用户编号

private	name	String	用户名
private	password	String	密码
private	del	int	删除状态，0、未删除，1、已删除

5.1.3 Role

角色实体类：用于建立和数据表（角色表）之间的映射关系。属性如下：

可见性	属性名称	类型	说明
private	id	int	角色id
private	name	String	角色名称
private	description	String	角色描述
private	funcIds	String	功能编号组合
private	del	int	删除状态，0、未删除，1、已删除

5.1.4 Function

功能实体类：用于建立和数据表（功能表）之间的映射关系。属性如下：

可见性	属性名称	类型	说明
private	id	int	功能id
private	num	String	功能编号
private	name	String	功能名称
private	url	String	访问路径
private	description	String	功能描述
private	del	int	删除状态，0、未删除，1、已删除

5.1.5 Right

权限实体类：用于建立和数据表（权限表）之间的映射关系。属性如下：

可见性	属性名称	类型	说明
private	id	int	权限id
private	userId	int	用户id
private	roleId	int	角色id
private	description	String	权限描述
private	del	int	删除状态，0、未删除，1、已删除

5.1.6 ConAttach

附件实体类：用于建立和数据表（附件表）之间的映射关系。属性如下：

可见性	属性名称	类型	说明
private	id	int	附件id
private	conId	int	合同id
private	fileName	String	附件名称
private	path	String	附件路径
private	type	String	附件类型
private	uploadDate	Date	上传日期
private	del	int	删除状态，0、未删除，1、已删除

5.1.7 ConProcess

合同流程实体类：用于建立和数据表（合同流程表）之间的映射关系。属性如下：

可见性	属性名称	类型	说明
private	id	int	合同流程id
private	conId	int	合同id
private	userId	int	用户id
private	type	int	操作类型，1、会签，2、审批，3、签订
private	state	int	操作状态，0、未完成，1、已完成，2、已否决
private	content	String	操作内容
private	time	Date	操作时间
private	del	int	删除状态，0、未删除，1、已删除

5.1.8 ConState

合同状态实体类：用于建立和数据表（合同状态表）之间的映射关系。属性如下：

可见性	属性名称	类型	说明
private	id	int	合同状态id
private	conId	int	合同id
private	type	int	操作类型，1、起草，2、会签完成，3、定稿，4、审批完成，5、签订完成
private	time	Date	操作时间
private	del	int	删除状态，0、未删除，1、已删除

5.1.9 Customer

客户实体类：用于建立和数据表（客户表）之间的映射关系。属性如下：

可见性	属性名称	类型	说明
private	id	int	客户id
private	num	String	客户编号
private	name	String	客户名称
private	address	String	客户地址
private	tel	String	客户电话
private	fax	String	客户传真
private	code	String	客户邮编
private	bank	String	银行名称
private	accout	String	银行账号
private	del	int	删除状态，0、未删除，1、已删除

5.1.10 Log

日志实体类：用于建立和数据表（日志表）之间的映射关系。属性如下：

可见性	属性名称	类型	说明
private	id	int	日志id
private	userId	int	操作人id
private	content	String	日志内容
private	time	Date	操作时间
private	del	int	删除状态，0、未删除，1、已删除

5.1.11 ConBusiModel

合同业务模型类：在程序实现过程中，通常会封装业务实体模型，将不同实体中的相关属性组装到一起以方便数据的传递和显示。

在本工程中，获取合同名称、合同id、起草时间 等合同信息时，需要查询合同表t_contract以获取合同名称、合同id，并级联查询合同状态表t_contract_state以获取起草时间，从合同实体类和合同流程实体类中抽取对应的属性重新组合成业务实体类ConBusiModel，以接收并传递合同信息。

属性如下：

可见性	属性名称	类型	说明
private	conId	int	合同id
private	conName	String	合同名称
private	drafTime	Date	起草时间

5.1.12 ConDetailBusiModel

合同详情业务模型类：封装原理同合同业务模型，在查看合同详情时，需要获取合同名称、合同编号、合同内容、客户、起草人等信息，需要查询合同表t_contract以获取合同名称、合同编号、客户、合同内容，并级联查询客户表t_user以获取起草人名称，从合同实体类和用户实体类中抽取对应的属性重新组合成业务实体类ConDetailBusiModel，以接收并传递合同详情。

属性如下：

可见性	属性名称	类型	说明
private	id	int	合同id
private	draftsman	int	起草人
private	customer	int	客户
private	num	String	合同编号
private	name	String	合同名称
private	content	String	合同内容
private	beginTime	Date	开始时间
private	endTime	Date	结束时间
private	del	int	删除状态，0、未删除，1、已删除

5.1.13 CSignatureOpinion

会签意见业务模型类：封装原理同合同业务模型，在查看会签意见时，需要获取合同编号、会签意见和会签人姓名等信息，需要查询合同流程表t_contract_process以获取合同编号、会签意见，并级联查询用户表t_user以获取会签人名称，从合同流程实体类和用户实体类中抽取对应的属性重新组合成业务实体类CSignatureOpinion，以接收并传递会签意见。

属性如下：

可见性	属性名称	类型	说明
private	conId	int	合同编号
private	csOperator	String	会签人
private	opinion	String	会签意见

5.1.14 PermissionBusiModel

权限业务模型类：封装原理同合同业务模型，在用户权限列表信息时，需要获取用户编号、用户名、角色编号和角色名称等信息，需要查询用户表t_user以获取用户编号、用户名，并级联查询角色表t_role以获取角色编号、角色名称，从用户实体类和角色实体类中抽取对应的属性重新组合成业务实体类PermissionBusiModel，以接收并传递用户权限信息。

属性如下：

可见性	属性名称	类型	说明

private	userId	int	用户编号
private	roleId	int	角色编号
private	userName	String	用户名称
private	roleName	String	角色名称

5.1.15 PageModel

5.1.15.1 简介

分页模型：封装了分页查询的基本属性，上一页、下一页、当前页、页大小、总记录数、总页数和当前页集合，并为属性提供setter和getter方法，供分页查询时使用。

5.1.15.2 类图

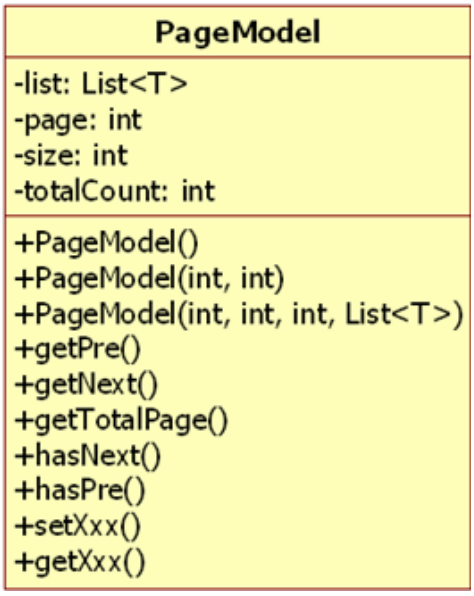

图5-2 分页模型类

5.1.15.3 属性

可见性	属性名称	类型	说明
private	totalCount	int	总记录数
private	list	List	数据集合
private	size	int	页大小
private	page	int	当前页码

5.1.15.4 方法

在分页模型中，会有有参数和无参数构造方法，他们都是给属性赋初值的。一些属性的getter方法会有简单的逻辑运算，如总页数，可以通过总记录数和页大小计算得到；上一页和下一页，可以通过当前页计算获得。

基本的setter和getter方法同合同实体，这里主要描述分页模型的两种构造方法，总页数、上一页和下一页的getter方法，以及判断是否有上一页和下一页的业务方法。

1、PageModel(int page, int size, int totalCount, List<T> list)

(1) 方法描述

Prototype 方法原型	public PageModel()
Description 功能描述	有参构造方法，设置当前页、页大小、总记录数和数据集合。
Calls 调用方法	对应属性的setter方法。
Input 输入参数	page、size、totalCount、list
Output 输出参数	无
Return 返回值	无
Exception 抛出异常	无

(2) 实现描述

定义一个实体的构造方法，方法名称和类名相同：public PageModel()，构造方法中可以根据需要对属性赋指定值。

定义分页模型的有参构造方法，设置当前页、页大小、总记录数和数据集合，调用对应属性的setter方法完成设置。

```
/**
 * 构造方法
 * @param page  当前页的页码
 * @param size    每一页上显示的数量
 * @param totalCount    总记录数
 * @param data  数据
 */
 public PageModel(int page, int size, int totalCount, List<T> list) {
     setPage(page);
     setSize(size);
     setTotalCount(totalCount);
     setList(list);
 }
```

2、PageModel(int page, int size)

(1) 方法描述

Prototype 方法原型	public PageModel()

Description 功能描述	有参构造方法，设置当前页和页大小。
Calls 调用方法	PageModel(int page, int size, int totalCount, List<T> list)
Input 输入参数	page、size
Output 输出参数	无
Return 返回值	无
Exception 抛出异常	无

(2) 实现描述

定义分页模型的有参构造方法，设置当前页和页大小，并设置总记录数为0。

```
/**
 * 构造方法
 * @param page  当前页的页码
 * @param size    每一页上显示的数量
 */
 public PageModel(int page, int size) {
      this(page, size, 0, null);
      //调用PageModel(int page, int size, int totalCount, List<T> list)
}
```

3、PageModel()

(1) 方法描述

Prototype 方法原型	public PageModel()
Description 功能描述	无参构造方法，为分页模型属性赋初始值。
Calls 调用方法	PageModel(int page, int size)
Input 输入参数	无
Output 输出参数	无
Return 返回值	无
Exception 抛出异常	无

(2) 实现描述

PageModel实体的无参构造方法，是将页大小赋值为10，将当前页设置为1，也就是默认显示第一页，每页显示10条记录。

```
/**
*无参构造方法，为对象属性赋初始值
*/
public PageModel() {
      // 默认显示第1页，每页10条数据
      this(1, 10);   // 调用  PageModel(int page, int size)
}
```

4、getTotalPage()

(1) 方法描述

Prototype 方法原型	public int getTotalPage()
Description 功能描述	计算总页数
Calls 调用方法	无
Input 输入参数	无
Output 输出参数	无
Return 返回值	int
Exception 抛出异常	无

(2) 实现描述

通过总记录数和页大小计算总页数，我们知道总记录数和每页的记录数后，总记录数和每页记录数进行取模，如果余数不为0，也会增加一页显示剩余的记录，如果取模余数为零正好整除，则直接取总记录数除以每页记录数。

```
/**
 * 计算总页数
 */
public int getTotalPage() {
   int totalPage = totalCount / size;
    // 若总记录数 % 每页数量 > 0，表示末尾还有一页。
    if (totalCount % size > 0) {
          totalPage++;
     }
     return totalPage;
}
```

5、hasPre()

(1) 方法描述

Prototype 方法原型	public boolean hasPre()
Description 功能描述	判断是否有上一页
Calls 调用方法	无
Input 输入参数	无
Output 输出参数	无
Return 返回值	int
Exception 抛出异常	无

(2) 实现描述

若当前页已经是第1页，那就没有上一页。

```
/**
 * 判断是否有上一页
 * @return true:有  false:没有
 */
```

```
public boolean hasPre() {
    // 若当前页已经是第1页，那就没有上一页。
    return page > 1;
}
```

6、hasNext()

(1) 方法描述

Prototype 方法原型	public boolean hasNext()
Description 功能描述	判断是否有下一页
Calls 调用方法	无
Input 输入参数	无
Output 输出参数	无
Return 返回值	int
Exception 抛出异常	无

(2) 实现描述

若当前页已经为尾页时，则没有下一页。调用getTotalPage()获取总页数(尾页)。

```
/**
 * 判断是否有上一页
 * @return true:有  false:没有
  */
 public boolean hasNext() {
     // 若当前页已经为尾页时，则没有下一页
     return page < getTotalPage();
 }
```

7、getPre()

(1) 方法描述

Prototype 方法原型	public int getPre ()
Description 功能描述	根据当前页计算上一页
Calls 调用方法	hasPre()
Input 输入参数	无
Output 输出参数	无
Return 返回值	int
Exception 抛出异常	无

(2) 实现描述

通过当前页计算上一页，当前页为第一页时，设置上一页为1，当前页大于1时，上一页为当前页减去1，确保在第一页时不能继续向上翻页。

```
/**
* 获取上一页（当前页为首页时，设置上一页为1，否则设置为当前页减1）
```



```
*/
public int getPre() {
    int pre = hasPre() ? page - 1 : page;
    return pre;
}
```

8、getNext()

(1) 方法描述

Prototype 方法原型	public int getNext()
Description 功能描述	根据当前页计算下一页
Calls 调用方法	hasNext()
Input 输入参数	无
Output 输出参数	无
Return 返回值	int
Exception 抛出异常	无

(2) 实现描述

通过当前页计算下一页，当前页为最后一页时，则设置下一页为最后页，当前页小于总页数时，设置下一页为当前页加1，确保在最后一页时不能继续向下翻页。

```
/**
* 获取下一页（当前页为尾页时，设置下一页为尾页，否则设置为当前页加1）
*/
public int getNextPage() {
    int nextPage = hasNext() ? page + 1 : page;
    return nextPage;
}
```

5.2 表示层类设计

表示层，调用业务逻辑层方法，用以处理用户的请求，包含处理用户，合同等操作请求的Servlet，处理页面跳转的Servlet。

Servlet在表示层中充当控制器，Servlet类存放在com.ruanko.web包中。

5.2.1 IndexServlet

处理系统首页的跳转。

5.2.2 ToRegisterServlet

处理注册页面的跳转。

5.2.3 ToErrorServlet

处理异常页面的跳转。

5.2.4 ToLoginServlet

处理登录页面的跳转。

5.2.5 LoginServlet

处理用户登录请求。

5.2.6 RegisterServlet

处理用户注册的请求。

5.2.7 LogoutServlet

处理用户登出请求。移除session中的用户信息，跳转到登录页面。

5.2.8 ToAdminServlet

处理管理员中心页面的跳转。
从session中，提取用户信息进行验证，用户已登录，则跳转到用户中心页面；用户未登录，则跳转到登录页面。

5.2.9 ToOperatorServlet

处理合同操作员中心页面的跳转。
从session中，提取用户信息进行验证，用户已登录，则跳转到用户中心页面；用户未登录，则跳转到登录页面。

5.2.10 ToNewUserServlet

处理新用户中心页面的跳转。

从session中，提取用户信息进行验证，用户已登录，则跳转到用户中心页面；用户未登录，则跳转到登录页面。

5.2.11 ToAddHQOpinionServlet

处理会签页面的跳转。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到会签页面；系统异常则跳转到异常页面。

5.2.12 ToAddQDInfoServlet

处理签订页面的跳转。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到签订页面；系统异常则跳转到异常页面。

5.2.13 ToAddSHPOpinionServlet

处理审批页面的跳转。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到审批页面；系统异常则跳转到异常页面。

5.2.14 ToAssignOperServlet

处理人员分配页面的跳转。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到人员分配页面；系统异常则跳转到异常页面。

5.2.15 ToAssignPermServlet

处理权限配置页面的跳转。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到权限配置页面；系统异常则跳转到异常页面。

5.2.16 ToDdghtListServlet

处理待定稿合同页面的跳转。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到待定稿合同页面；系统异常则跳转到异常页面。

5.2.17 ToDfphtListServlet

处理待分配合同页面的跳转。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到待分配合同页面；系统异常则跳转到异常页面。

5.2.18 ToDgContractServlet

处理定稿合同页面的跳转。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到定稿合同页面；系统异常则跳转到异常页面。

5.2.19 ToDhqhtListServlet

处理待会签合同页面的跳转。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到待会签合同页面；系统异常则跳转到异常页面。

5.2.20 ToDqdhtListServlet

处理待签订合同页面的跳转。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到待签订合同页面；系统异常则跳转到异常页面。

5.2.21 ToDraftServlet

处理起草合同页面的跳转。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到起草合同页面；系统异常则跳转到异常页面。

5.2.22 ToDshphtListServlet

处理待审批合同页面的跳转。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到待审批合同页面；系统异常则跳转到异常页面。

5.2.23 ToYhqxListServlet

处理用户权限列表页面的跳转。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到用户权限列表页面；系统异常则跳转到异常页面。

5.2.24 AddHQOpinionServlet

处理会签合同请求。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到待会签合同页面；系统异常则跳转到异常页面。

5.2.25 AddQDInfoServlet

处理签订合同请求。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到待签订合同页面；系统异常则跳转到异常页面。

5.2.26 AddSHPOpinionServlet

处理审批合同请求。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到待审批合同页面；系统异常则跳转到异常页面。

5.2.27 AssignOperServlet

处理分配合同请求。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到待分配合同页面；系统异常则跳转到异常页面。

5.2.28 AssignPermServlet

处理配置权限的请求。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到用户权限列表页面；系统异常则跳转到异常页面。

5.2.29 ContractDetailServlet

处理合同详情页面的跳转。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到合同详情页面；系统异常则跳转到异常页面。

5.2.30 ShowHQOpinionServlet

处理查看会签意见的请求。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到合同会签意见列表页面；系统异常则跳转到异常页面。

5.2.31 DgContractServlet

处理定稿合同的请求。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到待定稿合同页面；系统异常则跳转到异常页面。

5.2.32 DraftServlet

处理起草合同的请求。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到起草合同页面；系统异常则跳转到异常页面。

5.2.33 QueryConInfoServlet

处理分页查询合同基本信息的请求。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到合同信息列表页面；系统异常则跳转到异常页面。

5.2.34 QueryProcessServlet

处理分页查询合同流程信息的请求。

用户未登录，则跳转到登录页面；用户已登录并且正确执行，跳转到合同流程列表页面；系统异常则跳转到异常页面。

5.3 业务逻辑层类设计

调用数据访问层方法，处理用户，合同等操作逻辑，为表示层提供服务。包含UserService，ContractService等。

逻辑层类存放在com.ruanko.service包中。

5.3.1 UserService

5.3.1.1 简介

用户业务逻辑类，调用用户数据访问层(UserDao/UserDaoImpl)中的方法，实现用户的注册、登录和分配用户权限等业务逻辑。

5.3.1.2 属性

可见性	属性名称	类型	说明
private	userDao	UserDao	用户数据访问层实现类实例
private	roleDao	RoleDao	角色数据访问层实现类实例
private	rightDao	RightDao	权限数据访问层实现类实例

5.3.1.3 方法

1、register()

(1) 方法描述

Prototype 方法原型	public boolean register(User user)
Description 功能描述	处理用户注册业务逻辑
Calls 调用方法	userDao.isExist(); userDao.add(user);
Input 输入参数	user 用户对象
Output 输出参数	无
Return 返回值	boolean：操作是否成功。true为成功，false为失败
Exception 抛出异常	AppException

(2) 实现描述

声明操作标识，初始值为false 调用userDao.isExist();方法根据用户名查询对应用户是否存在 若不存在则调用userDao.add(user);保存用户信息，并将方法的返回结果赋值给操作标识，否则不进行注册，操作标识为false

2、login()

(1) 方法描述

Prototype 方法原型	public int login(String name,String password)
Description 功能描述	处理用户登录业务逻辑
Calls 调用方法	userDao.login(name, password);
Input 输入参数	name 用户姓名；password 密码

Output 输出参数	无
Return 返回值	用户编号
Exception 抛出异常	AppException

(2) 实现描述

传递用户名和密码，调用userDao.login(name, password);查询匹配用户编号，返回用户编号。

3、getUserRole()

(1) 方法描述

Prototype 方法原型	public Role getUserRole(int userId)
Description 功能描述	获取用户对应的角色信息
Calls 调用方法	rightDao.getRoleIdByUserId(userId); roleDao.getById(roleId);
Input 输入参数	userId 用户编号
Output 输出参数	无
Return 返回值	角色对象
Exception 抛出异常	AppException

(2) 实现描述

传递用户编号，调用rightDao.getRoleIdByUserId(userId); 根据用户编号，获取用户对应的角色编号，调用roleDao.getById(roleId)，根据角色编号，获取角色信息。

4、getUserListByRoleId()

(1) 方法描述

Prototype 方法原型	public List<User> getUserListByRoleId(int roleId)
Description 功能描述	获取对应角色的用户列表
Calls 调用方法	rightDao.getUserIdsByRoleId(roleId); userDao.getById(userId);
Input 输入参数	roleId 角色编号
Output 输出参数	无
Return 返回值	用户列表
Exception 抛出异常	AppException

(2) 实现描述

传递角色编号，调用RightDaoImpl中的getUserIdsByRoleId()获取指定角色的用户id集合，通过遍历该集合，调用UserDaoImpl中的getById()完成用户信息查询。

5、getYhqxList()

(1) 方法描述

Prototype 方法原型	public List<PermissionBusiModel> getYhqxList()
Description 功能描述	获取用户权限列表

Calls 调用方法	userDao.getIds();
	userDao.getById(userId);
	rightDao.getRoleIdByUserId(userId);
	roleDao.getById(roleId);

Input 输入参数	无
Output 输出参数	无
Return 返回值	用户权限列表
Exception 抛出异常	AppException

(2) 实现描述

调用UserDaoImpl中的getIds()，获取用户id集合，通过遍历查询出来的用户id集合，调用UserDaoImpl中的getById()，根据用户id获取用户信息；调用RightDaoImpl中的getRoleIdByUserId()，根据用户id获取角色id；如果存在相应记录，则调用RoleDaoImpl中的getById()，获取角色信息；将查询的信息进行组装，完成查询待分配权限的用户业务。

6、getRoleList()

(1) 方法描述

Prototype 方法原型	public List<Role> getRoleList()
Description 功能描述	获取角色列表
Calls 调用方法	roleDao.getAll();
Input 输入参数	无
Output 输出参数	无
Return 返回值	角色列表
Exception 抛出异常	AppException

(2) 实现描述

调用RoleDaoImpl中的getAll()完成角色列表信息查询。

7、assignPermission()

(1) 方法描述

Prototype 方法原型	public boolean assignPermission(Right right)
Description 功能描述	进行权限配置

Calls 调用方法	rightDao.getRoleIdByUserId(userId);
	roleDao.getById(roleId);
	rightDao.getIdByUserId(userId);
	rightDao.updateById(right);
	rightDao.add(right);

Input 输入参数	right 权限对象
Output 输出参数	无

Return 返回值	boolean：操作是否成功。true为成功，false为失败
Exception 抛出异常	AppException

(2) 实现描述

调用RightDaoImpl中的getRoleIdByUserId()，获取用户对应的角色编号；调用RoleDaoImpl中的getById()，获取角色信息；若用户原来拥有某个角色，则调用RightDaoImpl中的getIdByUserId()，获取用户对应的权限id，然后调用RightDaoImpl中的updateById()更改角色，否则调用RightDaoImpl中的add()，新增角色；完成分配权限功能。

5.3.2 ContractService

合同业务逻辑类，调用合同数据访问层(ContractDao/ContractDaoImpl)、合同流程数据访问层(ConProcessDao/ConProcessDaoImpl)、合同状态数据访问层(ConStateDao/ConStateDaoImpl)和用户数据访问层(UserDao/UserDaoImpl)中的方法，实现合同的起草、分配和流程查询等业务逻辑。

可见性	属性名称	类型	说明
private	contractDao	ContractDao	合同数据访问层实现类实例
private	conStateDao	ConStateDao	合同状态数据访问层实现类实例
private	conProcessDao	ConProcessDao	合同流程数据访问层实现类实例
private	userDao	UserDao	用户数据访问层实现类实例

1、draft()

(1) 方法描述

Prototype 方法原型	public boolean draft(Contract contract)
Description 功能描述	起草合同
Calls 调用方法	contractDao.add(contract); conStateDao.add(conState);
Input 输入参数	contract 合同对象
Output 输出参数	无
Return 返回值	boolean：操作是否成功。true为成功，false为失败
Exception 抛出异常	AppException

(2) 实现描述

调用ContractDaoImpl中的add()，如果起草成功，则调用ConStateDaoImpl中的add()，完成起草合同业务逻辑。

2、getDfphtList()

(1) 方法描述

Prototype 方法原型	public List<ConBusiModel> getDfphtList()
Description 功能描述	查询待分配合同集合

Calls 调用方法	conStateDao.getConIdsByType();
	conProcessDao.isExist(conId);
	contractDao.getById(conId);
	conStateDao.getByConId(conId);

Input 输入参数	无
Output 输出参数	无
Return 返回值	待分配合同列表
Exception 抛出异常	AppException

(2) 实现描述

调用ConStateDaoImpl中的getConIdsByType()，查询合同状态表中状态为“起草”的合同的id集合，通过遍历查询出来的合同id集合，从流程表中，筛选出待分配合同(待分配合同，即为在合同流程表中是否存在分配记录)；如果不存在分配记录，则分别调用ContractDaoImpl中的getById()和ConStateDaoImpl中的getConState()，将查询信息进行组装，完成查询待分配合同业务。

3、getContract()

(1) 方法描述

Prototype 方法原型	public Contract getContract(int id)
Description 功能描述	获取合同实体信息
Calls 调用方法	contractDao.getById(id);
Input 输入参数	合同id
Output 输出参数	无
Return 返回值	合同实体
Exception 抛出异常	AppException

(2) 实现描述

调用ContractDaoImpl中的getById()完成合同信息查询。

4、distribute()

(1) 方法描述

Prototype 方法原型	public boolean distribute(int conId, int userId, int type)
Description 功能描述	分配合同
Calls 调用方法	conProcessDao.add(conProcess);
Input 输入参数	合同id，用户id，操作类型
Output 输出参数	无
Return 返回值	boolean：操作是否成功。true为成功，false为失败
Exception 抛出异常	AppException异常

(2) 实现描述

使用业务实体封装合同流程信息(合同id、合同对应的操作人、操作状态和操作类型)，调用ConProcessDaoImpl中的add()，将合同流程信息进行保存，完成分配

合同功能。

5、getDhqhtList()

(1) 方法描述

Prototype 方法原型	public List<ConBusiModel> getDhqhtList(int userId)
Description 功能描述	查询待会签合同集合

Calls 调用方法	conProcessDao.getConIds(conProcess);
	contractDao.getById(conId);
	conStateDao.getByConId(conId);

Input 输入参数	用户编号
Output 输出参数	无
Return 返回值	待会签合同列表
Exception 抛出异常	AppException

(2) 实现描述

调用ConProcessDaoImpl中的getConIds()，查询合同流程表中操作类型为“会签”，状态为“未完成”的合同的id集合，通过遍历该集合，分别调用ContractDaoImpl中的getById()和ConStateDaoImpl中的getConState()，将查询信息进行组装，完成查询待会签合同业务。

6、counterSign()

（1）方法描述

Prototype 方法原型	public boolean counterSign(ConProcess conProcess)
Description 功能描述	会签合同

Calls 调用方法	conProcessDao.update(conProcess);
	conProcessDao.getTotalCount(conProcess);
	conStateDao.add(conState);

Input 输入参数	合同流程实体
Output 输出参数	无
Return 返回值	boolean：操作是否成功。true为成功，false为失败
Exception 抛出异常	AppException

（2）实现描述

调用ConProcessDaoImpl中的update()，将对应的合同状态更新为已完成，更新成功，调用ConProcessDaoImpl中的getTotalCount()统计待会签的总人数，若待会签总人数为0，则所有会签人已经完成会签，此时调用ConStateDaoImpl中的add()，将“会签完成”状态的合同信息保存，完成会签合同业务逻辑。

7、getContractDetail()

(1) 方法描述

Prototype 方法原型	public ConDetailBusiModel

	getContractDetail(int id)
Description 功能描述	获取合同详细信息
Calls 调用方法	contractDao.getById(id); userDao.getById(userId);
Input 输入参数	合同id
Output 输出参数	无
Return 返回值	合同详情业务实体
Exception 抛出异常	AppException

(2) **实现描述**

调用ContractDaoImpl中的getById()，获取指定合同的详细信息；调用UserDaoImpl中的getById()，获取合同对应的起草人的信息，将查询信息进行组装，完成查询合同详情业务。

8、getDdghtList()

(1) 方法描述

Prototype 方法原型	public List<ConBusiModel> getDdghtList(int userId)
Description 功能描述	查询待定稿合同集合

Calls 调用方法	contractDao.getIdsByUserId(userId);
	conStateDao.isExist();
	contractDao.getById(conId);
	conStateDao.getByConId(conId);

Input 输入参数	用户id
Output 输出参数	无
Return 返回值	待定稿合同列表
Exception 抛出异常	AppException

(2) 实现描述

调用ContractDaoImpl中的getIdsByUserId()，根据用户id查询合同基本信息表合同的id集合，通过遍历查询出来的合同id集合，从合同状态表中，筛选出待定稿合同的id集合(待定稿合同，即为在合同状态表中存在“会签完成”并且不存在“定稿完成”的状态记录)。通过遍历该集合，分别调用ContractDaoImpl中的getById()和ConStateDaoImpl中的getConState()，将查询信息进行组装，完成查询待定稿合同业务。

9、finalize()

(1) 方法描述

Prototype 方法原型	public boolean finalize(Contract contract)
Description 功能描述	定稿合同
Calls 调用方法	contractDao.updateById(contract); conStateDao.add(conState);

Input 输入参数	合同实体
Output 输出参数	无
Return 返回值	boolean：操作是否成功。true为成功，false为失败
Exception 抛出异常	AppException

(2) 实现描述

调用ContractDaoImpl中的updateById()，更新合同内容，更新成功，调用ConStateDaoImpl中的add()，将“定稿完成”状态的合同信息保存，完成定稿合同业务逻辑。

10、showHQOpinion()

(1) 方法描述

Prototype 方法原型	public List<CSignatureOpinion> showHQOpinion(int conId)
Description 功能描述	显示会签意见

Calls 调用方法	conProcessDao.getIds();
	conProcessDao.getById(id);
	userDao.getById(userId);

Input 输入参数	合同id
Output 输出参数	无
Return 返回值	合同会签意见列表
Exception 抛出异常	AppException

(2) 实现描述

调用ConProcessDaoImpl中的getIds()，获取会签合同的流程id集合；通过遍历该集合，分别调用ConProcessDaoImpl中的getById()和UserDaoImpl中的getById()，将查询信息进行组装，完成查询会签意见业务。

11、getDshphtList()

(1) 方法描述

Prototype 方法原型	public List<ConBusiModel> getDshphtList(int userId)
Description 功能描述	查询待审批合同集合

Calls 调用方法	conProcessDao.getConIds(conProcess);
	conStateDao.isExist();
	contractDao.getById(conId);
	conStateDao.getByConId(conId);

Input 输入参数	用户id
Output 输出参数	无
Return 返回值	待审批合同列表
Exception 抛出异常	AppException

(2) 实现描述

调用ConProcessDaoImpl中的getConIds()，查询合同流程表中操作类型为“审批”，状态为“未完成”的合同的id集合，通过遍历查询出来的合同id集合，从被分配的合同中，筛选出待审批合同的id集合(待审批合同，即为在合同状态表中存在“定稿完成”的状态记录)。通过遍历该集合，分别调用ContractDaoImpl中的getById()和ConStateDaoImpl中的getConState()，将查询信息进行组装，完成查询待审批合同业务。

12、approve()

(1) 方法描述

Prototype 方法原型	public boolean approve(ConProcess conProcess)
Description 功能描述	审批合同

Calls 调用方法	conProcessDao.update(conProcess);
	conProcessDao.getTotalCount(conProcess);
	conStateDao.add(conState);

Input 输入参数	合同流程实体类
Output 输出参数	无
Return 返回值	boolean：操作是否成功。true为成功，false为失败
Exception 抛出异常	AppException

(2) 实现描述

调用ConProcessDaoImpl中的update()，将对应的合同状态更新为已完成，更新成功，调用ConProcessDaoImpl中的getTotalCount()统计待审批和审批拒绝的总人数，若待审批和审批拒绝的总人数同时为0，则所有审批人已经完成审批，此时调用ConStateDaoImpl中的add()，将“审批完成”状态的合同信息保存，完成审批合同业务逻辑。

13、getDqdhtList()

(1) 方法描述

Prototype 方法原型	public List<ConBusiModel> getDqdhtList(int userId)
Description 功能描述	查询待签订合同集合

Calls 调用方法	conProcessDao.getConIds(conProcess);
	conStateDao.isExist();
	contractDao.getById(conId);
	conStateDao.getByConId(conId);

Input 输入参数	用户id
Output 输出参数	无
Return 返回值	待签订合同列表
Exception 抛出异常	AppException

(2) 实现描述

调用ConProcessDaoImpl中的getConIds()，查询合同流程表中操作类型为“签订”，状态为“未完成”的合同的id集合，通过遍历查询出来的合同id集合，从被分配的合同中，筛选出待签订合同的id集合(待签订合同，即为在合同状态表中存在“审批完成”的状态记录)。通过遍历该集合，分别调用ContractDaoImpl中的getById()和ConStateDaoImpl中的getConState()，将查询信息进行组装，完成查询待签订合同业务。

14、sign()

(1) 方法描述

Prototype 方法原型	public boolean sign(ConProcess conProcess)
Description 功能描述	签订合同
Calls 调用方法	conProcessDao.update(conProcess); conStateDao.add(conState);
Input 输入参数	合同流程实体类
Output 输出参数	无
Return 返回值	boolean：操作是否成功。true为成功，false为失败
Exception 抛出异常	AppException

(2) 实现描述

调用ConProcessDaoImpl中的update()，将对应的合同状态更新为已完成，更新成功，调用ConStateDaoImpl中的add()，将“签订完成”状态的合同信息保存，完成签订合同业务逻辑。

15、queryContract()

(1) 方法描述

Prototype 方法原型	public PageModel<ConBusiModel> queryContract(String searchName, int page, int size)
Description 功能描述	分页查询合同信息

Calls 调用方法	contractDao.getTotalCount(searchName);
	contractDao.getIdsByName(searchName, page, size);
	contractDao.getById(conId);
	userDao.getById(userId);

Input 输入参数	合同名称关键字、当前页、每页显示的记录数
Output 输出参数	无
Return 返回值	分页模型
Exception 抛出异常	AppException

(2) 实现描述

调用ContractDaoImpl中的getTotalCount()，统计合同基本信息表中的合同总记录数，然后调用ContractDaoImpl中的getIdsByName()，分页查询合同id集合，通过遍历该集合，分别调用ContractDaoImpl中的getById()和ConStateDaoImpl

中的getConState()，将查询信息进行组装，并与合同总数目、合同的当前页和每页显示合同的数目一起，封装到分页模型实体中，完成分页查询合同业务。

16、queryProcess()

(1) 方法描述

Prototype 方法原型	public PageModel<ProcessBusiModel> queryProcess(String searchName, int page, int size)
Description 功能描述	分页查询合同流程信息

Calls 调用方法	contractDao.getTotalCount(searchName);
	contractDao.getIdsByName(searchName, page, size);
	contractDao.getById(conId);
	conStateDao.getTypeCountByConId(conId);

Input 输入参数	合同名称关键字、当前页、每页显示的记录数
Output 输出参数	无
Return 返回值	分页模型
Exception 抛出异常	AppException

(2) 实现描述

调用ContractDaoImpl中的getTotalCount()，统计合同基本信息表中的合同总记录数，然后调用ContractDaoImpl中的getIdsByName()，分页查询合同id集合，通过遍历该集合，分别调用ContractDaoImpl中的getById()和ConStateDaoImpl中的getTypeCountByConId()，将查询信息进行组装，并与合同总数目、合同的当前页和每页显示合同的数目一起，封装到分页模型实体中，完成分页查询合同流程业务。

17、generateConNum()

(1) 方法描述

Prototype 方法原型	private String generateConNum()
Description 功能描述	生成合同编号
Calls 调用方法	无
Input 输入参数	无
Output 输出参数	无
Return 返回值	返回一个“年月日时分秒+5位随机数”的合同编号
Exception 抛出异常	AppException

(2) 实现描述

生成合同编号，规则为：年月日时分秒+5位随机数。

5.4 数据访问层类设计

数据访问层通过接口定义数据库操作的方法，由该接口的具体实现类实现方法逻辑，与数据库交互，实现数据的存取操作。

包含用户注册，登录，合同管理和系统管理等的接口和对应实现类：UserDao/UserDaoImpl，ContractDao/ContractDaoImpl，ConProcessDao/ConProcessDaoImpl，ConStateDao/ConStateDaoImpl，RightDao/RightDaoImpl，RoleDao/RoleDaoImpl。

接口存放在com.ruanko.dao包中，相应实现类存放在com.ruanko.dao.impl包中。

5.4.1 UserDaoImpl

5.4.1.1 简介

用户数据库访问层接口(UserDao)定义与用户有关的数据库操作方法，通过访问数据库，向数据库中插入、查询用户信息。并提供给外界调用。由用户数据库访问层实现类(UserDaoImpl)进行具体的实现。

5.4.1.2 方法

1、add()

(1) 方法描述

Prototype 方法原型	public boolean add(User user) throws AppException
Description 功能描述	向数据库表中插入用户信息
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeStatement(psmt); DBUtil.closeConnection();
Input 输入参数	用户对象
Output 输出参数	无
Return 返回值	boolean：操作是否成功，true表示成功，false表示失败
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 定义插入用户信息sql语句 执行插入操作 调用DBUtil.closeConnection()关闭数据库连接，释放资源 返回操作是否成功的标识

2、isExist()

（1）方法描述

Prototype 方法原型	public boolean isExist(String name);
Description 功能描述	验证是否有同名用户存在
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	用户名
Output 输出参数	无
Return 返回值	boolean：操作是否成功，true表示成功，false表示失败
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 定义执行查询的sql语句 执行查询操作 调用DBUtil.closeXxx()关闭数据库操作对象，释放资源 返回结果标识

3、login()

(1) 方法描述

Prototype 方法原型	public int login(String name, String password);
Description 功能描述	根据用户名和密码查询匹配的用户信息
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	name 用户姓名，password 密码
Output 输出参数	无
Return 返回值	用户编号
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据用户名和密码查询匹配的用户信息 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回用户编号

4、getById()

(1) 方法描述

Prototype 方法原型	public User getById(int id)
Description 功能描述	根据用户id查询用户信息
Calls 调用方法	DBUtil.getConnection();

	DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	id 用户id
Output 输出参数	无
Return 返回值	用户对象
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据用户id查询匹配的用户信息 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回用户对象

5、getIds()

(1) 方法描述

Prototype 方法原型	public List<Integer> getIds()
Description 功能描述	查询用户id集合
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	无
Output 输出参数	无
Return 返回值	用户id集合
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 查询用户id集合 调用DBUtil.closeXxx()关闭数据库操作对象，释放资源 返回用户id集合

5.4.2 ContractDaoImpl

合同数据库访问层接口(ContractDao)定义与合同有关的数据库操作方法，通过访问数据库，向数据库中插入、查询或更新合同信息。并提供给外界调用。由合同数据库访问层实现类(ContractDaoImpl)进行具体的实现。

1、add()

(1) 方法描述

Prototype 方法原型	public boolean add(Contract Contract)
Description 功能描述	向数据库表中插入合同信息
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection();
Input 输入参数	合同对象
Output 输出参数	无
Return 返回值	boolean：操作是否成功，true表示成功，false表示失败
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 定义插入合同信息sql语句 执行插入操作 调用DBUtil.closeXxx()关闭数据库操作对象，释放资源 返回操作是否成功的标识

2、getById()

（1）方法描述

Prototype 方法原型	public Contract getById(int id)
Description 功能描述	根据合同id查询合同对象
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	合同id
Output 输出参数	无
Return 返回值	合同对象
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据合同id查询合同对象 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回合同对象

3、getIdsByUserId()

(1) 方法描述

Prototype 方法原型	public List<Integer> getIdsByUserId(int userId)
Description 功能描述	根据用户id查询合同id集合
Calls 调用方法	DBUtil.getConnection();

	DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	用户id
Output 输出参数	无
Return 返回值	合同id集合
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据用户id查询合同id集合 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回合同id集合

4、updateById()

(1) 方法描述

Prototype 方法原型	public boolean updateById(Contract contract)
Description 功能描述	根据合同id更新合同内容，通过实体对象传递参数
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	合同对象
Output 输出参数	无
Return 返回值	boolean：操作是否成功，true表示成功，false表示失败
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据合同id更新合同内容 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回操作是否成功的标识

5、getTotalCount()

(1) 方法描述

Prototype 方法原型	public int getTotalCount(String searchName)
Description 功能描述	获取符合条件的合同总记录数
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	搜索关键字
Output 输出参数	无

Return 返回值	合同总记录数
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据合同名称的关键字，获取符合条件的合同总记录数 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回合同总记录数

6、getIdsByName()

（1）方法描述

Prototype 方法原型	public List<Integer> getIdsByName(String searchName, int page, int size)
Description 功能描述	根据合同名称分页查询合同id集合
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	合同名称的关键字、分页页码、分页查询的数目
Output 输出参数	无
Return 返回值	合同id集合
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据合同名称分页查询合同id集合 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回合同id集合

5.4.3 ConProcessDaoImpl

合同流程数据库访问层接口(ConProcessDao)定义与合同流程有关的数据库操作方法，通过访问数据库，向数据库中插入、查询或更新合同流程信息。并提供给外界调用。由合同流程数据库访问层实现类(ConProcessDaoImpl)进行具体的实现。

1、isExist()

(1) 方法描述

Prototype 方法原型	public boolean isExist(int conId)
Description 功能描述	判断指定合同id的记录在合同流程表中是否存在
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs);

	DBUtil.closeStatement(psmt); DBUtil.closeConnection();
Input 输入参数	合同id
Output 输出参数	无
Return 返回值	boolean：操作是否成功，true表示成功，false表示失败
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据合同id获取总记录数 如果总记录数大于0，将操作标识设为true 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回操作是否成功的标识

2、add()

（1）方法描述

Prototype 方法原型	public boolean add(ConProcess conProcess)
Description 功能描述	新增合同操作流程信息
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	合同流程对象
Output 输出参数	无
Return 返回值	boolean：操作是否成功，true表示成功，false表示失败
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 定义插入合同流程信息sql语句 执行插入操作 调用DBUtil.closeXxx()关闭数据库操作对象，释放资源 返回操作是否成功的标识

3、getConIds()

(1) 方法描述

Prototype 方法原型	public List<Integer> getConIds(ConProcess conProcess)
Description 功能描述	从合同流程表中查询所有符合条件的合同id
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	合同流程对象

Output 输出参数	无
Return 返回值	合同id集合
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 从合同流程表中查询所有符合条件的合同id 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回合同id集合

4、update()

(1) 方法描述

Prototype 方法原型	public boolean update(ConProcess conProcess)
Description 功能描述	根据用户id、合同id和操作类型更新合同流程的操作状态、内容和时间信息
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	合同流程对象
Output 输出参数	无
Return 返回值	boolean：操作是否成功，true表示成功，false表示失败
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 更新合同流程信息 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回操作是否成功的标识

5、getTotalCount()

(1) 方法描述

Prototype 方法原型	public int getTotalCount(ConProcess conProcess)
Description 功能描述	根据合同id、操作类型和类型的处理状态查询符合条件的记录的总记录数
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	合同流程对象
Output 输出参数	无
Return 返回值	符合条件的总记录数
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据合同id、操作类型和类型的处理状态查询符合条件的记录的总记录数 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回符合条件的总记录数

6、getIds()

(1) 方法描述

Prototype 方法原型	public List<Integer> getIds(int conId, int type, int state)
Description 功能描述	根据合同id和操作类型type及对应的操作状state查询合同流程id集合
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	合同id，操作类型，操作类型对应的操作状态
Output 输出参数	无
Return 返回值	合同流程id集合
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据合同id和操作类型type及对应的操作状state查询合同流程id集合 调用DBUtil.closeXxx()关闭数据库操作对象，释放资源 返回合同流程id集合

7、getById()

(1) 方法描述

Prototype 方法原型	public ConProcess getById(int id)
Description 功能描述	根据合同流程id查询合同流程信息
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	合同流程id
Output 输出参数	无
Return 返回值	合同流程对象
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接

根据合同流程id查询合同流程信息 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回合同流程对象

5.4.4 ConStateDaoImpl

合同状态数据库访问层接口(ConStateDao)定义与合同状态有关的数据库操作方法，通过访问数据库，向数据库中插入、查询或更新合同状态信息。并提供给外界调用。由合同状态数据库访问层实现类(ConStateDaoImpl)进行具体的实现。

1、add()

(1) 方法描述

Prototype 方法原型	public boolean add(ConState conState)
Description 功能描述	新增合同操作状态信息
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeStatement(psmt); DBUtil.closeConnection();
Input 输入参数	合同状态对象
Output 输出参数	无
Return 返回值	boolean：操作是否成功，true表示成功，false表示失败
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 定义插入合同状态信息sql语句 执行插入操作 调用DBUtil.closeXxx()关闭数据库操作对象，释放资源 返回操作是否成功的标识

2、getConIdsByType()

（1）方法描述

Prototype 方法原型	public List<Integer> getConIdsByType(int type)
Description 功能描述	根据合同类型查询符合条件的合同id集合
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	操作类型
Output 输出参数	无
Return 返回值	合同id集合
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据合同类型查询符合条件的合同id集合 调用DBUtil.closeXxx()关闭数据库操作对象，释放资源 返回合同id集合

3、getByConId()

（1）方法描述

Prototype 方法原型	public ConState getByConId(int conId)
Description 功能描述	根据合同id查询合同状态信息
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	合同id
Output 输出参数	无
Return 返回值	合同状态对象
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据合同id查询合同状态信息 调用DBUtil.closeXxx()关闭数据库操作对象，释放资源 返回合同状态对象

4、isExist()

（1）方法描述

Prototype 方法原型	public boolean isExist(int con_id, int type)
Description 功能描述	根据合同id和类型判断记录在合同表中是否存在
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	合同id，操作类型
Output 输出参数	无
Return 返回值	boolean：操作是否成功，true表示成功，false表示失败
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据合同id和类型判断记录在合同表中是否存在 调用DBUtil.closeXxx()关闭数据库操作对象，释放资源

返回操作是否成功的标识

5.4.5 RightDaoImpl

权限数据库访问层接口(RightDao)定义与权限有关的数据库操作方法，通过访问数据库，向数据库中插入、查询和更新权限信息。并提供给外界调用。由权限数据库访问层实现类(RightDaoImpl)进行具体的实现。

1、getRoleIdByUserId()

(1) 方法描述

Prototype 方法原型	public int getRoleIdByUserId(int userId)
Description 功能描述	根据用户id获取角色id
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection();
Input 输入参数	用户id
Output 输出参数	无
Return 返回值	角色id
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据用户id获取角色id 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回角色id

2、getUserIdsByRoleId()

(1) 方法描述

Prototype 方法原型	public List<Integer> getUserIdsByRoleId(int roleId)
Description 功能描述	查询根据角色id，查询用户id
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	角色id
Output 输出参数	无
Return 返回值	用户id
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接

查询根据角色id，查询用户id 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回用户id

3、getIdByUserId()

(1) 方法描述

Prototype 方法原型	public int getIdByUserId(int userId)
Description 功能描述	根据用户id，获取权限id
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection();
Input 输入参数	用户id
Output 输出参数	无
Return 返回值	权限id
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据用户id，获取权限id 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回权限id

4、updateById()

(1) 方法描述

Prototype 方法原型	public boolean updateById(Right right)
Description 功能描述	根据权限id更新权限内容，通过实体对象传递参数
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeStatement(psmt); DBUtil.closeConnection();
Input 输入参数	权限对象
Output 输出参数	无
Return 返回值	boolean：操作是否成功，true表示成功，false表示失败
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据权限id更新权限内容，通过实体对象传递参数 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回操作是否成功的标识

5、add()

(1) 方法描述

Prototype 方法原型	public boolean add(Right right)
Description 功能描述	保存权限信息
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeStatement(psmt); DBUtil.closeConnection();
Input 输入参数	权限对象
Output 输出参数	无
Return 返回值	boolean：操作是否成功，true表示成功，false表示失败
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 保存权限信息 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回操作是否成功的标识

5.4.6 RoleDaoImpl

角色数据库访问层接口(RoleDao)定义与角色有关的数据库操作方法，通过访问数据库，向数据库中插入、查询和更新角色信息。并提供给外界调用。由角色数据库访问层实现类(RoleDaoImpl)进行具体的实现。

1、getById()

(1) 方法描述

Prototype 方法原型	public Role getById(int id)
Description 功能描述	根据角色id，查询角色信息
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	角色id
Output 输出参数	无
Return 返回值	角色对象
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 根据角色id，查询角色信息 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回角色对象

2、getAll()

(1) 方法描述

Prototype 方法原型	public List<Role> getAll()
Description 功能描述	查询所有角色对象集合
Calls 调用方法	DBUtil.getConnection(); DBUtil.closeResultSet(rs); DBUtil.closeStatement(psmt); DBUtil.closeConnection(conn);
Input 输入参数	无
Output 输出参数	无
Return 返回值	角色对象集合
Exception 抛出异常	AppException

(2) 实现描述

调用DBUtil.getConnection()获取数据库连接 查询所有角色对象集合 调用DBUtil.closeXxx()关闭数据库连接，释放资源 返回角色对象集合

5.5 工具类设计

将公共的方法抽离出来形成工具类，供其他类使用，这里包含数据库工具类DBUtil,常量定义类Constant，自定义异常类AppException。

工具类存放在com.ruanko.utils中。

5.5.1 DBUtil

作为一个数据库的操作的工具类，用来获取数据库的连接，关闭数据库操作对象（数据库连接对象，查询指令对象和结果集对象）。

1、属性

可见性	属性名称	类型	说明
private	url	String	数据库连接字符串，数据库字符集设置
private	user	String	数据库用户名
private	password	String	数据库密码

2、类图

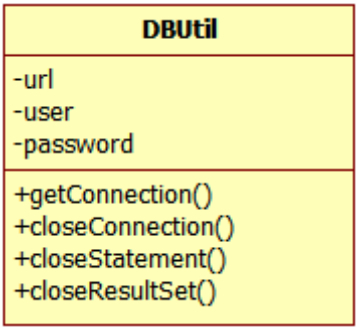

图5-3 DBUtil类

3、方法

(1) getConnection()

1) 方法描述

Prototype 方法原型	public static Connection getConnection()
Description 功能描述	获取数据库连接
Calls 调用方法	DriverManager.getConnection()
Input 输入参数	url、user和password
Output 输出参数	无
Return 返回值	Connection数据库连接对象
Exception 抛出异常	SQLException

2) 实现描述

调用DriverManager.getConnection()创建数据库连接

(2) closeConnection()

1) 方法描述

Prototype 方法原型	public static void closeConnection(Connection conn)
Description 功能描述	关闭数据库连接对象
Calls 调用方法	conn.close()
Input 输入参数	Connection对象
Output 输出参数	无
Return 返回值	无
Exception 抛出异常	处理SQLException

2) 实现描述

```
if(数据库连接对象不为空 && 连接对象没有关闭){
    调用conn.close()关闭数据库连接
    将链接对象conn置空
}
```

(3) closeStatement()

1) 方法描述

Prototype 方法原型	public static void closeStatement(Statement st)
Description 功能描述	关闭数据库查询对象
Calls 调用方法	st.close()
Input 输入参数	Statement对象
Output 输出参数	无
Return 返回值	无
Exception 抛出异常	处理SQLException

2) 实现描述

```
if(数据库查询对象不为空 && 查询对象没有关闭){
    调用st.close()关闭数据库查询对象
    将查询对象st置空
}
```

(4) closeResultSet()

1) 方法描述

Prototype 方法原型	public static void closeResultSet(ResultSet rs)
Description 功能描述	关闭结果集对象
Calls 调用方法	rs.close()
Input 输入参数	ResultSet对象
Output 输出参数	无
Return 返回值	无
Exception 抛出异常	处理SQLException

2) 实现描述

```
if(结果集对象不为空 && 结果集对象没有关闭){
    调用rs.close()关闭结果集对象
    将结果集对象rs置空
}
```

5.5.2 Constant

根据业务需求和数据库表字段，在常量类中定义相关状态常量，以表示具体状态的值。这些状态包括合同状态表中的操作类型、合同流程表中的操作类型和操作状态等。

所有的常量定义为公有的静态常量，即修饰类型为 public static final 。

如下所示。

名称	值	数据类型	说明
STATE_DRAFTED	**1**	int	起草
STATE_CSIGNED	**2**	int	会签完成
STATE_FINALIZED	**3**	int	定稿完成

STATE_APPROVED	**4**	int	审批完成
STATE_SIGNED	**5**	int	签订完成
PROCESS_CSIGN	**1**	int	会签
PROCESS_APPROVE	**2**	int	审批
PROCESS_SIGN	**3**	int	签订
UNDONE	**0**	int	未完成
DONE	**1**	int	已完成
VETOED	**2**	int	已否决

6 Error Design 出错处理设计

6.1 概述

为了统一处理表示层、业务逻辑层、数据访问层的异常，定义 AppException 类作为异常处理的工具类。采用 try，catch，throw 进行异常处理。当底层出现异常，将自定义异常上抛，并包含异常信息，异常信息格式为：出现异常的类.出现异常的方法名称。直至表示层统一处理。

异常结构如下图所示：

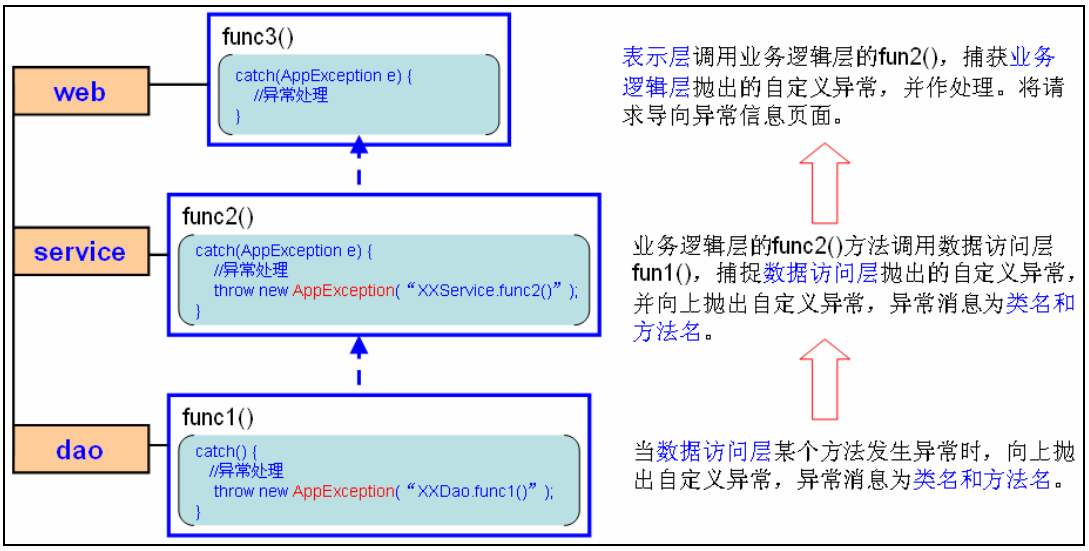

图 6-1 异常结构

6.2 自定义异常类

程序中统一的异常类为 AppException，存放在工程的 com.ruanko.utils 包中。

类的定义如下所示：

```
public class AppException extends Exception {

    private int exceptionCode;       //异常编号
    private String message;          //异常消息

     * 构造方法，设置异常消息
    public AppException(String message) {
        this.message = message;
    }

     * 构造方法，设置异常消息和异常编号
    public AppException(String message,int exceptionCode) {
        this.message = message;
        this.exceptionCode = exceptionCode;
    }

     * 获取异常编号
    public int getExceptionCode() {
        return exceptionCode;
    }

     * 获取详细的异常消息
    public String getMessage() {
        String detailMessage = "Detail message:"
            + exceptionCode + " " + message;
        return detailMessage;
    }
}
```

图 6-2 自定义异常类