



本科毕业论文(设计)

|  |
| --- |
| EMA数据安全及真实保障系统的设计与实现 |
|  |

吴松朗

201627010526

|  |  |
| --- | --- |
| 指导教师 | 梁早清 高级工程师 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学院名称 |  | 软件学院 | 专业名称 |  | 软件工程 |
| 论文提交日期 |  | 年  月 日 | 论文答辩日期 |  | 年  月 日 |

摘 要

随着中国互联网的高速发展，信息和大数据时代纵横交错的今天，数据信息化存储和管理已成

主流，与此同时，成千上万的数据良莠不齐，为此要将有用的、真实的数据信息化存储，剔除伪造的、不合格的数据以保障数据的安全和真实。其中经济利益驱动型食品掺假（EMA）数据尤为注重数据安全及真实。

近年来，从我国的地沟油、问题胶囊、 “三聚氰胺事件”，到欧洲的马肉风波和今年的巴西牛肉问题，全球都存在食品掺假的问题，食品掺假事件频繁发生。这些信息无时无刻充斥着人们的眼球，打击消费者的购买信心。由于利欲熏心和经济利益的诱惑，以次充好，以假乱真已成为当前食品安全保障的一个重要议题。这些事件都是故意污染带来的严重食品安全问题，不仅会引发令食品行业和消费者担忧的经济问题，还会带来潜在的人身健康安全隐患。但众多食品掺假事件里也不全然是真实事件因此要做到保障ema数据的安全和真实。

剔除不需要的数据主要的途径是人为审核提交的数据，该途径基于shiro权限管理实现。权限管理使提交数据事件任务化，即不同的账号以各自的权限发布、审核记录实现提交数据事件高并发、透明化。高并发是指用户可以提交多次数据，且提交事件之间互不干扰。透明化指拥有不同权限的两个账号互不干涉，各自只需完成自身签收的任务。

英文摘要

With the rapid development of China's Internet and the crisscross of information and big data era, data information storage and management has become a reality.

Mainstream, at the same time, tens of thousands of data are mixed in quality. Therefore, useful and real data should be stored informatization, and forged and unqualified data should be eliminated to ensure the safety and authenticity of data. Among them, economic interest-driven food adulteration (EMA) data pays special attention to data security and authenticity.

In recent years, from our country's illegal cooking oil, problem capsules, "melamine incident", to Europe's horse meat storm and this year's Brazilian beef problem, there are food adulteration problems all over the world, and food adulteration incidents occur frequently. This information fills people's eyes all the time, hitting consumers' confidence in buying. Due to the temptation of greed and economic interests, it has become an important issue of food safety protection to substitute inferior goods for genuine ones. These incidents are serious food safety problems caused by intentional pollution, which will not only cause economic problems that worry the food industry and consumers, but also bring potential risks to personal health and safety. However, many food adulteration incidents are not entirely true, so ema data must be guaranteed to be safe and true.

The main way to eliminate unnecessary data is to manually audit the submitted data, which is based on shiro permission management. Permission management makes the submission of data events task-oriented, that is, different accounts publish and audit records with their own permissions to achieve high concurrency and transparency of the submission of data events. High concurrency means that users can submit data multiple times without interference between submission events. Transparency means that two accounts with different permissions do not interfere with each other and each need only complete the task of signing for it.

1. 前言
   1. 研究背景

21世纪以来，经济全球化的发展不断加快，我国经济取得了举世瞩目的成就，我国的iT行业随之快速发展，在此大背景下，数据成为了一种有用的资源。虽然数据的数量足够庞大，但迎面而来的问题十分严峻。大量数据被浏览，无人保证其中数据是真实的。所谓真实，也就是在现实世界中真实发生后被记录的，而虚拟网络的数据是可以由人为伪造的，因此数据存在真伪。

互联网无孔不入，传统的Java Web开发中往往会忽略对数据的保护，即使是存在数据库中的数据，用户不登录也可以通过SQL注入攻击获取其中的数据，用户个人隐私无法得到保证。

* 1. 研究目的

信息化是当代主流，各行各业都将自身工作中数据保存到各自的数据库中，因此数据的安全和真实尤为重要。本次的课程设计主要针对基于SpringBoot+Shiro+mybatis框架的ema安全和真实性保障系统进行研究。

食品掺假事件的数据真实性十分重要，一个假的食品掺假事件发布就如果一次造谣，严重的可能引起社会恐慌。

所以需要一个ema数据安全和真实性保障系统来协助ema事件的审核与存储。而此次使用的Shiro框架并且结合对应的业务处理能够很好的控制数据的存储和显示。

从整体的系统实现技术来看，基于SpringBoot和mybatis技术让项目清晰和轻便，前端又用Vue.js和layui作为框架，便于前后端分离开发，为往后开发总结经验和模板。

Vue.js框架采用了MVVM的开发模式，MVVM本质上是MVC的改进，但它将其中的View的状态和行为抽象画，让视图UI和业务逻辑分开，简单的讲就是HTML文件注重UI设计，而js文件作为HTML和后台通讯桥梁，则负责复杂的数据交互和数据处理。

1. 相关技术

Eam数据安全和真实性保障系统，为实现数据的安全和真实性，选用了Apache Shrio负责安全相关的模块。Shiro是一个强大且易用的安全框架，可以管控前端到Dao层。

前端选用了Vue.js和layui作为框架，以freemarker作为模板引擎。

* 1. Vue.js

首先是Vue.js,它是一套用于构建用户界面的渐进式[JavaScript](https://baike.baidu.com/item/JavaScript/321142" \t "https://baike.baidu.com/item/Vue.js/_blank)框架，Vue 的核心库只关注视图层，并且非常容易学习，非常容易与其它库或已有项目整合。另一方面，Vue 采用的MVVM模式使得系统的前后端完全分离，这样能够使前段专注于样式和脚本开，而后端则注重于逻辑业务的处理。

* 1. SpringBoot

SpringBoot能够快速搭建后台SSM框架，其主要特点是开箱即用和约定由于配置，不像Spring需要使用XML文件进行大量的配置，而是在application文件中设置自己对系统的约定即可使用Spring，SpringMVC，mybatis的功能。SpringBoot中存在许多好用的注释，在类或方法上使用注释即可完成配置，十分的方便。

* 1. Mybatis-plus

Mybatis一款优秀的持久层框架，它支持定制化 SQL、存储过程以及高级映射。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集，可以使用简单的 XML 或注解来配置和映射原生信息，将接口和 Java 的 POJOs映射成数据库中的记录。Mybatis简单易学、灵活、解除sql与程序代码的耦合、支持编写动态sql，但mybatis编写sql语句时工作量很大，尤其是字段多、关联表多时，同时sql语句依赖于数据库，导致数据库移植性差，不能更换数据库。

Mybatis-Plus（MP）在 MyBatis 的基础上只做增强不做改变，简化开发、提高效率。它其中封装了mybatis的标签，以java方法的形式使用，使业务代码和持久层紧密贴合，同时减少了许多重复的代码。

* 1. jQuery与Ajax

jQuery是一个简洁的JavaScript框架，它的宗旨是“write Less，do More”，即它提倡用更少的代码完成更多的事情。它封装JavaScript常用的功能代码，提供一种简便的JavaScript设计模式，优化HTML文档操作、事件处理、动画设计和Ajax交互。jQuery核心是独特的链式语法和多接口功能，具有高效灵活的css选择器，拥有便捷的插件扩展机制和丰富的插件。

其中，Ajax技术是结合了Java、XML以及JavaScript 等编程技术，让开发人员构建基于Java技术的Web应用，Ajax 技术使用了非同步的HTTP请求，在浏览器和服务端之间传递数据，使得浏览器只更新部分网页内容，而不需要重新载入整个页面，即Web页面可以动态更新，而不是打断与服务器的连接与交流。

jQuery与Ajax的关系是：jQuery是JavaScript的框架，它是基于JS语言的。但重要的是，它集合了Ajax技术开发出来的JS库和封装了Ajax的功能，向外可提供函数接口，这样一来，大大简化了Ajax，JS的操作。

* 1. Layui

事实上，layui更多是面向于后端开发者，所以在组织形式上毅然采用了几年前的以浏览器为宿主的类 AMD 模块管理方式，却又并非受限于 CommonJS 的那些条条框框，它拥有自己的模式，更加轻量和简单。layui 定义为“经典模块化”，并非是刻意强调“模块”理念本身，而是有意避开当下 JS 社区的主流方案，试图以尽可能简单的方式去诠释高效！它的所谓经典，是在于对返璞归真的执念，它以当前浏览器普通认可的方式去组织模块！ layui 认为这种轻量的组织方式，仍然可以填补 WebPack 以外的许多场景。所以它坚持采用经典模块化，也正是能让人避开工具的复杂配置，重新回归到原生态的。其次是layui，它提供的样式和模块能够将后台的业务逻辑很好的在前段展示，同时有着良好的用户体验。

* 1. maven

这些框架都需要由jar包来支持，而jar包的管理和系统业务之间的管理尤为重要，因此需要用到maven进行项目的管理。使用maven其中主要的文件是pom.xml，项目通过配置maven的坐标能被其他项目引用，因此我将项目分解为了moudel子项目，common子项目以及server子项目。这样将整个项目分解成三个的子项目便于java类的管理，同时也使不同业务的代码清晰可见。

* 1. Freemarker

它是一款[模板引擎](https://baike.baidu.com/item/%E6%A8%A1%E6%9D%BF%E5%BC%95%E6%93%8E/907667" \t "https://baike.baidu.com/item/freemarker/_blank)： 即一种基于模板和要改变的数据， 并用来生成输出文本（[HTML](https://baike.baidu.com/item/HTML" \t "https://baike.baidu.com/item/freemarker/_blank)网页、[电子邮件](https://baike.baidu.com/item/%E7%94%B5%E5%AD%90%E9%82%AE%E4%BB%B6/111106" \t "https://baike.baidu.com/item/freemarker/_blank)、[配置文件](https://baike.baidu.com/item/%E9%85%8D%E7%BD%AE%E6%96%87%E4%BB%B6/286550" \t "https://baike.baidu.com/item/freemarker/_blank)、[源代码](https://baike.baidu.com/item/%E6%BA%90%E4%BB%A3%E7%A0%81/3969" \t "https://baike.baidu.com/item/freemarker/_blank)等）的通用工具。 它不是面向最终用户的，而是一个Java类库，是一款程序员可以嵌入他们所开发产品的组件。

1. 系统分析
   1. 实现目标
2. 实现一个ema信息管理系统的基本CRUD功能；
3. 使用Shiro给系统添加按权限访问资源的功能；
4. 使用vue.js和layui前端框架优化用户操作。
   1. 需求分析
      1. 用户管理

主要是用于超级管理员管理每个用户的个人信息，其中包括查看用户列表，添加用户，删除用户，编辑用户，查询用户的操作。如图用户管理用例图：



用户管理用例图

1. 查看用户列表：主要是现实使用系统的用户的基本信息，统计用户数量。
2. 添加用户：在用户拥有超级管理员权限时，通过系统提供的管理功能进行添加用户。
3. 删除用户：只有拥有超级管理员的权限的用户才可以删除用，同时运行批量删除。
4. 编辑用户：在用户基本信息不符合实际情况时，允许超级管理员用户编辑用户。
   * 1. 事件管理

事件管理主要是展示ema事件的基本信息，任何用户都运行查看，并获取报表，其中包括了添加事件，删除事件，编辑事件，查询事件，审核通过事件和审核不通过事件的操作。如图x数据管理用例图：



事件管理用例图

1. 添加事件：任何用户都允许添加ema事件，在添加ema事件时，可选择预定义的数据进行填写，如果预定义的数据不存在下拉框中，需要在其对应的管理系统中添加数据再返回添加事件。
2. 删除事件：删除事件只有超级管理员才能有该权限，一般的用户不允许进行事件删除。
3. 编辑事件：在添加ema事件后，ema事件会进入未审核状态，ema事件有三个状态，分别为未审核状态，审核通过状态和审核未通过状态，只有在未审核状态和审核未通过状态的ema事件允许编辑，并且审核未通过状态的ema事件修改过后会变为未审核状态，允许审核人员再次审核。
4. 查询事件：事件可以通过事件名称精确的查询某一条记录，同时也允许事件名称的个别字进行模糊查询某些记录。
5. 审核通过事件：该功能只有拥有专家权限的用户才能够使用，专业人士在审核时，并认为事件为真实的即可审核通过该ema事件。
6. 审核不通过事件：专家在审核ema事件时，如果发现ema事件与事实不符，或者存在信息模糊的情况，可以将事件设置为审核不通。
   * 1. 食品管理

食品是ema事件的主体，存储食品相关信息能够对ema事件作出详细的分析，因此系统提供了添加食品，修改食品，删除食品，查询食品的功能，如图x食品管理用例图。

1. 添加食品：在发布ema事件时，用户可以通过下拉框查看系统中是否存在涉事的食品，如果没有则需要通过添加食品来完成ema事件的发布。
2. 修改食品：系统已存在的食品中可能会随着时间的推移而出现变化，如果发现系统的食品记录与事实不符，可以修改食品的相关信息进行更新迭代。
3. 删除食品：已有的食品随着时间的推移可能已经不存在现实中了，因此可以通过删除食品，取消该食品记录的显示，一般是不需要删除食品记录的，但为了使系统操作方便，因此添加了删除食品的功能。
4. 查询食品：在众多的记录中，如果一页一页的浏览难免麻烦，因此需要查询找到快速的找到相应的某些或某条记录，而查询食品的功能支持用户通过食品的名称进行模糊查询。
   * 1. 与事件和食品相关的信息管理

事件和食品是系统的主体，但这两个个主体涉及的信息有多，并且之间也有关联，例如事件相关的信息有uri、涉事主体和食品，食品相关的信息有食品类型、商家、品牌、味道、期限和保存方法，而期限与之相关的信息有单位，类型与之相关的信息有父类型，因此需要为uri、涉事主体、商家、品牌、味道、期限、保存方法以及单位这些信息实习增删查改。

1. 系统设计
   1. 结构设计

本系统主要功能模块包括：登陆，注册，事件管理，食品管理，用户管理，授权。图1是系统的总体结构图。

* 1. 分层结构介绍

目前为止，大部分企业或个人都以分层机构进行项目的开发和管理。分层结构即将整体项目划分为各个子模块，每个子模块专注于完成自身功能或业务，并为上一层的结构提供服务。典型的分层结构应用是三层结构。即分为数据访问层、业务逻辑层以及表示层。

数据访问层实现了数据的持久化，业务逻辑层对逻辑的实现及处理，实际上不可能在表示层对数据不做任何处理，但是尽可能的将逻辑分为一层，表示层是数据的展示。这样分层的一定程度是降低了一个程序的耦合度，一个类中有可能超过一千行的代码可以转化为多个类中各自一两百行代码，每个类实现自己的功能，慢慢的就形成了一种标准化的趋势，更加的适应于面向接口编程，注重“开闭原则”的实现，对修改关闭，而对扩展开发，扩展的功能定义为接口，程序员进行实现，在现有的代码中组合进新功能的实现类对象，从而实现新功能的添加。

* 1. 项目架构设计
     1. Maven子项目关联

目前，分层结构的软件结构广泛地被企业或其他开发者使用。而采用分层结构开发的目的是为了开发过程中能够将功能不同或业务不同的类分类存放，这样一来项目整体清晰明了，代码管理也十分方便。但在一个项目中分包管理，类的权责还不够清楚，而且如果多人开发项目，相互之间干扰验证，使得开发效率大大降低，即1+1<2。为了尽可能的使开发效率提高，我们需要将整个项目分为多个子项目，同时使子项目之间能够互相调用，因此需要用到Maven。

Maven除了以程序构建能力为特色之外，还提供了高级项目管理工具。它不只是能够用户jar包的导入与管理，同时能够赋予项目本身Maven坐标，其他项目通过继承和引用该项目Maven坐标来使用它的实现类。下面代码清单为本系统的父项目和各个子项目的pom.xml配置文件。

1. 父项目ema的pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.bill.ema</groupId>

<artifactId>ema</artifactId>

<packaging>pom</packaging>

<version>1.0.1</version>

<modules>

<module>emaCommon</module>

<module>emaModel</module>

<module>emaServer</module>

</modules>

</project>

1. 子项目emaCommon的pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<parent>

<artifactId>ema</artifactId>

<groupId>com.bill.ema</groupId>

<version>1.0.1</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>emaCommon</artifactId>

<packaging>jar</packaging>

</project>

1. 子项目emaModel的pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<parent>

<artifactId>ema</artifactId>

<groupId>com.bill.ema</groupId>

<version>1.0.1</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>emaModel</artifactId>

<packaging>jar</packaging>

1. 子项目emaServer的pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<parent>

<groupId>com.bill.ema</groupId>

<artifactId>ema</artifactId>

<version>1.0.1</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>emaServer</artifactId>

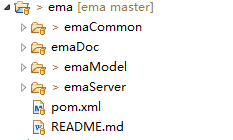
<packaging>jar</packaging>

这样就可以将代码的功能进一步细分，开发效率大大提高。

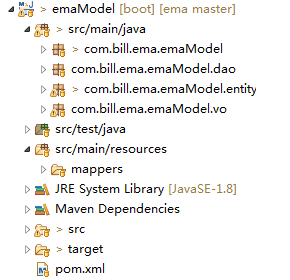
* + 1. 搭建项目架构

从分层结构的分析和Maven子项目的关联，本系统开发的实现方式将采用MVC设计模式并且结合SpringBoot+Shiro+Mybatis-plus+vue.js+layui的框架。持久层主要采用mybatis和mybatis-plus相结合以实现对数据的访问，而业务逻辑层以SpringBoot提供的Spring和mybatis-plus来实现，控制器因为用到了vue.js的MVVM模式，使得控制器专注于API定义。以下是框架搭建细节

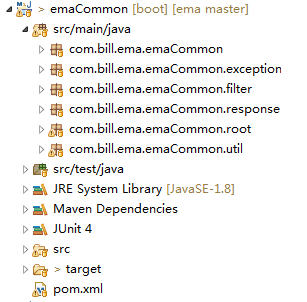
1. Ema父项目结构



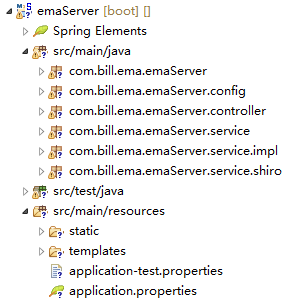
1. emaCommon子项目：该子项目为其他项目提供工具和一些基础类的实现；
2. emaDoc子项目：该子项目是所有子项目文档存放处；
3. emaModel子项目：该子项目集成了数据库与系统交互的所有源文件；
4. emaServer子项目：该子项目存放了业务处理源文件以及前端一切资源。
5. emaModel子项目结构



1. Com.bill.ema.emaModel：该目录为源代码文件根目录；
2. Com.bill.ema.emaModel.dao：该目录存放了以mybatis+MP作为持久层而编写的Dao接口，与对应的xml文件映射；
3. Com.bill.ema.emaModel.entity：该目录存放了数据表映射的实体类，用于项目的业务和数据交互；
4. Com.bill.ema.emaModel.vo：该目录存放了个别页面对应的实体VO即为view object，意思是一个页面或一个表单对应需要的数据；
5. Src/main/resources/mappers：该目录存放了mybatis与Dao接口对应的xml配置文件，主要写自定义的sql与Dao接口方法映射。
6. emaCommon子项目结构



1. Com.bill.ema.emaCommon：该目录为子项目java文件的根目录；
2. Com.bill.ema.emaCommon.exception：该目录存放自定义异常类，在框架提供的异常无法满足要求时，需要自定义异常；
3. Com.bill.ema.emaCommon.filter：该目录存放自定义过滤器类，类似SQL注入攻击需要自定义过滤器对其处理；
4. Com.bill.ema.emaCommon.response：该目录存放前后端交互时的消息类；
5. Com.bill.ema.emaCommon.root：该目录存放了一些数据库表实体类的根类，方便工具类夸项目调用；
6. Com.bill.ema.emaCommon.util：该目录存放了自定义工具类和常量类，为了便于修改代码和使代码可读性提高。
7. emaServer子项目结构



1. Com.bill.ema.emaServer：该目录存放服务器启动类和java源代码根类；
2. Com.bill.ema.emaServer.config：该目录存放框架的配置类，引用各个不同的框架需要在此添加配置；
3. Com.bill.ema.emaServer.controller：该目录存放控制器类，即服务器API定义在这可以见；
4. Com.bill.ema.emaServer.service：该目录存放了业务处理的接口类，实现面向接口编程；
5. Com.bill.ema.emaServer.service.impl：该目录存放了业务处理具体实现类；
6. Com.bill.ema.emaServer.service.shiro：该目录存放了shiro的业务处理具体实现类，也就是权限相关的业务处理实现类。
   1. 框架集成及配置
      1. SpringBoot搭建SSM框架

SpringBoot创建的项目中集成了Maven，同时省去了类似于Spring配置的xml文件，转而用application.properties文件代替，即实现了约定大于配置的原则。在创建项目后，只需要在resources文件下的application.properties文件添加配置，同时在pom.xml中加入对应的maven坐标既可以导入框架需要的jar包。这样一来，项目既可以引用SSM框架的类、接口和方法了。

* + 1. Mybatis-plus集成

下面是SpringBoot配置中，集成了mybatis-plus各项功能

mybatis-plus.mapper-locations=classpath:mappers/\*.xml

mybatis-plus.type-aliases-package=com.bill.ema.emaModel.entity

mybatis-plus.global-config.banner=false

mybatis-plus.global-config.db-config.id-type=auto

mybatis-plus.global-config.db-config.field-strategy=not\_null

mybatis-plus.global-config.db-config.column-underline=true

mybatis-plus.global-config.db-config.logic-delete-value=0

mybatis-plus.global-config.db-config.logic-not-delete-value=1

1. mybatis-plus.mapper-locations=classpath:mappers/\*.xml：定义持久层接口映射的xml文件所在目录
2. mybatis-plus.type-aliases-package=com.bill.ema.emaModel.entity：定义数据库表映射的实体类所在目录
3. mybatis-plus.global-config.banner=false：禁止控制台打印mybatis-plus的日志，减轻服务器负担；
4. mybatis-plus.global-config.db-config.id-type=auto：设置主键类型，auto代表数据库ID自增；
5. mybatis-plus.global-config.db-config.field-strategy=not\_null：设置字段更新策略，mybatis更新为null的对象需要设置，not\_null是判断非null；
6. mybatis-plus.global-config.db-config.column-underline=true：数据库表字段映射到实体类中数据的策略，这里true是设置驼峰下划线的转换，例如：数据库字段为“user\_id”，则实体对应的属性为“userId”；
7. mybatis-plus.global-config.db-config.logic-delete-value=0：给表添加deleted字段后，可以设置数据库删除操作是否为逻辑删除，这里等于0，即逻辑已删除值设置为0；
8. mybatis-plus.global-config.db-config.logic-not-delete-value=1：这里是设置逻辑未删除值为1。
   * 1. Shiro集成

Shiro集成不需要在application.properties文件中加约定，而是单独一个类中配置，如下：

@Configuration

public class ShiroConfig {

@Bean

public SecurityManager securityManager(UserRealm userRealm) {

DefaultWebSecurityManager securityManager

= new DefaultWebSecurityManager();

securityManager.setRealm(userRealm);

securityManager.setRememberMeManager(null);

securityManager.setSessionManager(sessionManager());

return securityManager;

}

@Bean

public DefaultWebSessionManager sessionManager() {

DefaultWebSessionManager sessionManager

= new DefaultWebSessionManager();

sessionManager.setGlobalSessionTimeout(1 \* 60 \* 60 \* 1000);

return sessionManager;

}

@Bean("shiroFilter")

public ShiroFilterFactoryBean shiroFilter(SecurityManager securityManager) {

ShiroFilterFactoryBean shiroFilter = new ShiroFilterFactoryBean();

shiroFilter.setSecurityManager(securityManager);

shiroFilter.setLoginUrl("/login.html");

shiroFilter.setUnauthorizedUrl("/login.html");

Map<String,String> filterMap = new LinkedHashMap<>();

filterMap.put("/logout", "logout");

filterMap.put("/login.html", "anon");

filterMap.put("/register.html", "anon");

filterMap.put("/register", "anon");

filterMap.put("/login", "anon");

filterMap.put("/css/\*\*", "anon");

filterMap.put("/lib/\*\*", "anon");

filterMap.put("/verify/\*\*", "anon");

filterMap.put("/fonts/\*\*", "anon");

filterMap.put("/js/\*", "anon");

filterMap.put("/templates/\*", "anon");

filterMap.put("/captcha.jpg", "anon");

filterMap.put("/\*\*", "authc");

shiroFilter.setFilterChainDefinitionMap(filterMap);

Map<String,Filter> customFilterMap = new HashMap();

return shiroFilter;

}

@Bean("lifecycleBeanPostProcessor")

public LifecycleBeanPostProcessor lifcycleBeanPostProcessor() {

return new LifecycleBeanPostProcessor();

}

@Bean

PublicAuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(SecurityManager securityManager){

AuthorizationAttributeSourceAdvisor advisor

=new AuthorizationAttributeSourceAdvisor();

advisor.setSecurityManager(securityManager);

return advisor;

}

}

SpringBoot可以通过@Configuration来配置各个框架需要配置的数据

1. securityManager方法：作为安全器管理，是shiro的核心，管理所有的subject；
2. sessionMnager方法：会话管理，此处可以设置全局session超时的时间；
3. shiroFilter方法：shiro的拦截器，shiro作为权限管理时，需要对网页资源进行权限分配，此方法可以规定各个url需要的各种权限；
4. authorizationAttributeSourceAdvisor方法：注册securityManager；
   * 1. Kaptcha验证码配置

验证码的作用是保护网站安全，一般网站都要通过验证码来防止机器大规模注册，机器暴力破解数据密码等危害，一定程度提高了网站的安全性，配置如下

@Configuration

public class KaptchaCofing {

@Bean

public DefaultKaptcha producer() {

Properties properties = new Properties();

properties.put("kaptcha.border", "no");

properties.put("kaptcha.textproducer.font.color", "black");

properties.put("kaptcha.textproducer.char.space", "5");

properties.put("kaptcha.textproducer.font.names", "Arial,Courier,cmr10,宋体,楷体,微软雅黑");

Config config = new Config(properties);

DefaultKaptcha defaultKaptcha = new DefaultKaptcha();

defaultKaptcha.setConfig(config);

return defaultKaptcha;

}

}

* + 1. Mybatis-plus分页插件配置

分页对于任何系统来说都是十分重要的功能，分页的目的是将数据分批展示给用户，不仅能够给用户良好的体验，同时也提高了系统性能，分页的本质是通过sql中的limit进行数据库表分段查询，而mybatis-plus内置了分页插件将sql的limit进行封装，并且增强了功能，mybatis-plus分页插件配置如下：

@Configuration

public class MybatisPlusConfig {

@Bean

public PaginationInterceptor paginationInterceptor() {

PaginationInterceptor paginationInterceptor

= new PaginationInterceptor();

return paginationInterceptor;

}

}

* + 1. Vue.js+layui配置

每个页面都有着自己的脚步和样式，都存在着异同，因此将相同的样式和需要调用的脚本放在同一文件里，再由各个页面导入，可以节约需要不必要的代码；类似vue.js和layui的导入，可以统一存放在head.html中，而其他页面通过include标签即可导入。

* 1. 系统功能设计
     1. 数据分页功能

1. 分页，是将大量的数据分段显示在页面，但是分页也分真分页和假分页。
2. 假分页是后台将数据库所有的数据获取并发送到客户端后，客户端利用前端技术分段显示。假分页的优点是移植性比较高，缺点是造成系统的性能下降。
3. 真分页是在数据库分页获取数据，一般使用sql中的limit进行获取，但是limit中的offset过大时，数据库的运行时间也会增大，因此需要为数据库表添加索引，并在sql中使用索引来调高查询速率。因此本系统采用的是真分页。
4. Layui的table组件只要设置的page参数为true即可异步获取分页数据进行渲染.
5. Service层调用mybatis-plus和PageUtil类实现分页时，因为实体类与前端显示的数据不完成相同，需要定义vo类将实体类进一步封装。以下是用户表的分页方法。
6. Dao层自定义了分页接口在service层的分页方法中输入的参数是查询时返回的参数，自定义分页接口通过mybatis的xml配置文件，返回按查询参数的分页的数据。
   * 1. 前端路由
7. Hash是url中hash（#）及后面的那部分，常用作锚点在页面内进行导航，改变URL中的hash部分不会引起页面刷新，通过hashchange事件监听URL的变化，改变URL的方式只有这几种：通过浏览器前进后退改变URL、通过标签改变URL、通过window.location改变URL，这几种情况改变URL都会触发hashchange事件。
8. History提供了pushState和replaceState两个方法，这两个方法改变URL的path部分不会引起页面的刷新。History提供类似hashchange事件的popstate事件，但popstate事件有些不同，通过浏览器前进后退改变URL时会触发popstate事件，通过pushState/replaceState或标签改变URL不会触发popstate事件。好在我们可以拦截pushState/replacState的调用和标签的点击事件来检测URL变化，所以监听URL变化可以实现，只是没有hashchange方便。
9. 本系统采用了hash的方式实现单页面浏览的前端路由，hahs能兼容低版本的浏览器。前端路由由三个部分组成，分别是路由器、注册器以及导航。
10. 路由器是在一个独立的脚本文件中实现的，其主要实现了启动路由功能、绑定window的onhashchange事件的回调函数以及路由注册方法；
11. 首先是实现一个路由类Router，它的成员有保存路由的成员routes和获取当前hash的curUrl成员，routes存储已注册的路由。

function Router(){

this.routes={};

this.curUrl='/';

}

路由器是通过routes定位路由url的。curUrl存储了目前用户浏览的url的hash值，用于window的onhashchange事件进行绑定。

1. 启动路由器，实际是为window添加hashchange的事件监听器，并在事件触发后调用绑定window的onhashchage事件的回调函数。

Router.prototype.render = function(){

window.addEventListener('hashchange',this.reloadPage.bind(this))

}

1. 绑定window的onhashchange事件的回调函数，截取URL的锚部分，转换为以‘/’开头的格式，因为routes存储了对应的路由，而curUrl作为key可以调用routes中已注册的路由。

Router.prototype.reloadPage = function(){

this.curUrl = location.hash.substring(1)||'/';

console.log(this.routes);

this.routes[this.curUrl]();

}

1. 路由注册方法，在注册器中，调用该方法，将key和key映射的路由方法返回到方法中，在用routes对象以key-callback的形式保存。

Router.prototype.map = function(key,callback){

this.routes[key] = callback;

}

1. 实现单页浏览和前端路由，还需要用到html中的iframe标签。Iframe元素会创建包含另外一个文档的内联框架，即路由的页面资源会被渲染到iframe中，而iframe外的其他html元素不发生变化。路由注册器实际是利用Router类的路由注册方法map，将已经定义好的url，和iframe元素的设置属性的方法进行绑定。iframe元素的src属性规定了 iframe 中显示的文档的 URL。注册完路由后，调用路由类的启动路由方法render方法即可实现路由初始，启动前端路由功能。

route(){

router.map('/user',function(){

iframe.setAttribute('src','user/user.html')

}),

......

}

1. 导航是用到了layui提供的layui-nav和layui-nav-tree导航元素进行渲染，而用户点击的是超链接a标签，实际连接的是以“#”开头的hash，并不会进行请求，而是有路由器处理再渲染页面。
   * 1. Ema事件审核功能
2. 在数据库中，event表中有一个状态字段，该字段用来保存当前event记录是否已经通过审核，而审核的目的是为了ema数据的真实性。
3. 审核状态有未审核、审核通过和审核未通过三种状态。在事件创建时，默认是未审核状态，按钮审核通过和审核不通过向后台传输消息来设置event表的审核状态。
4. 页面中选择需要审核的事件，点击审核通过或审核未通过按钮对事件进行状态转换。
5. 事件的id返回到service层，通过pass方法或nopass方法改变事件的status字段的数据记录事件的审核状态，同时通过session获取用户的id并为事件设置审核人的id和审核时间。

@Override

@Transactional(rollbackFor = Exception.class)

public Boolean pass(Map<String, Object> param) {

for (Object id : param.values()) {

Event event = getById(Integer.valueOf(id.toString()));

event.setStatus(Constant.PASS);

event.setAuditorId(ShiroUtil.getUserId());

event.setAuditTime(new Date());

if(!updateById(event))

return false;

}

return true;

}

@Override

@Transactional(rollbackFor = Exception.class)

public Boolean noPass(Map<String, Object> param) {

for (Object id : param.values()) {

Event event = getById(Integer.valueOf(id.toString()));

event.setStatus(Constant.NO\_PASS);

event.setAuditorId(ShiroUtil.getUserId());

event.setAuditTime(new Date());

if(!updateById(event))

return false;

}

return true;

}

* + 1. 编辑功能

1. 编辑数据时，会存在多选项，多选项在关系型数据库中一般是一对多的关系。因此这里会用到集合操作。编辑时，多选项会出现三种变化，分别是在原选项上添加选项、将原选项减少、减少原选项添加新选项，如图：
2. 编辑功能算法分析
3. 设A为原集，B为目标集，C为A与B的交集，为简化模型将散列简化为有序列，即A，B的元素有序连续；
4. 在原选项上添加选项时，，，，因此需要添加的项
5. 将原来的选项减少时，，，，因此需要移除的项；
6. 减少原选项添加新选项时，，，，因此需要移除的项，需要添加的项

由以上分析可知，在三种情况下需要移除的项和需要添加的项结果相同，但是作用域不同，因此只需要通过作用域进行判断编辑时的状态是添加项还是移除项，

1. 将方法写在CollectionUtil类中，在service层需要用到编辑时调用即可，降低代码耦合度和使代码可重复利用。

public class CollectionUtil {

public static void edit(List<? extends Object> newC,List<? extends Object> oldC) {

List<Object> tmp = new ArrayList<Object>();

tmp.addAll(newC);

tmp.retainAll(oldC);

newC.removeAll(tmp);

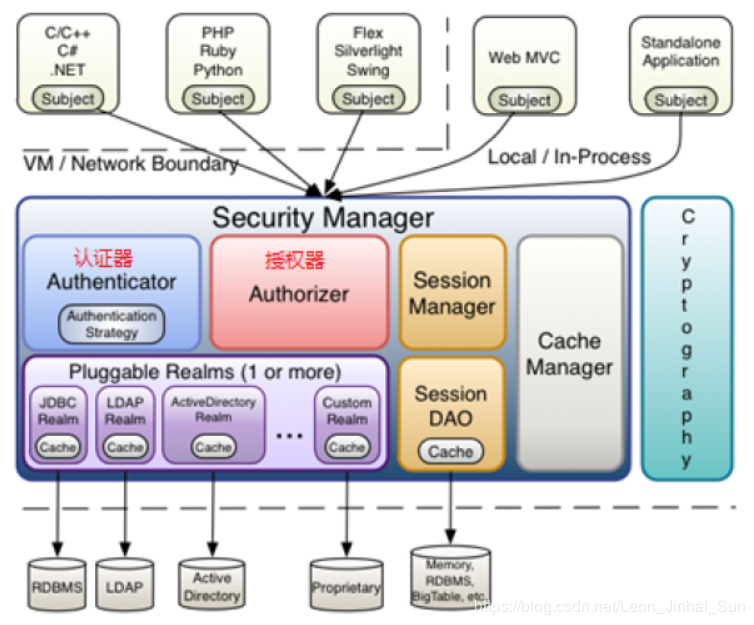
oldC.removeAll(tmp);

}

}

* + 1. 登陆注册功能

为确保系统的安全性，登陆和注册功能是必须的，本系统使用的是shiro作为安全框架。Shiro的体系结构如下。



1. 登陆功能设计
2. 登陆页面使用简约风格，并用layui的form表单进行表单的提交和校验，主要校验用户各个信息的格式是否错误；
3. Service层实现用户的认证，生成token并用session保存用户信息，用户信息设计到用户的基本信息、用户角色和用户的权限；
4. 通过过滤器重定向登录成功的页面。

public R authentication(Map<String, Object> param) {

String kaptcha = ShiroUtil.getKaptcha(Constants.KAPTCHA\_SESSION\_KEY);

Integer retry = ShiroUtil.getRetryNum();

Subject subject = SecurityUtils.getSubject();

try {

if(param.get(Constant.CAPTCHA)==null)

throw new NullCaptchaException();

if(param.get(Constant.USER\_NAME)==null)

throw new NullUsernameException();

if(param.get(Constant.PASSWORD)==null)

throw new NullPasswordException();

if (!(kaptcha.equals(param.get(Constant.CAPTCHA).toString())))

return R.ERROR(Statuscode.InvalidCaptcha);

if (!subject.isAuthenticated()) {

UsernamePasswordToken token =

new UsernamePasswordToken(param.get(Constant.USER\_NAME).toString(),

param.get(Constant.PASSWORD).toString());

subject.login(token);

}

} catch (UnknownAccountException e) {

return R.ERROR(Statuscode.AccountNotExist);

} catch (IncorrectCredentialsException e) {

return R.ERROR(Statuscode.AccountPasswordIncorrect);

} catch (LockedAccountException e) {

return R.ERROR(Statuscode.AccountHasBeenLocked);

} catch (DisabledAccountException e) {

return R.ERROR(Statuscode.AccountHasBeenActive);

} catch (UnsupportedTokenException e) {

return R.ERROR(Statuscode.CurrUserNotPermission);

} catch (AuthenticationException e) {

return R.ERROR(Statuscode.AccountValidatedFail);

}catch(NullCaptchaException e) {

return R.ERROR(Statuscode.CaptchaIsNull);

}catch(NullUsernameException e) {

return R.ERROR(Statuscode.UsernameIsNull);

}catch(NullPasswordException e) {

return R.ERROR(Statuscode.PasswordIsNull);

}

if (subject.isAuthenticated()) {

Integer id =

userService.getByUsername(param.get(Constant.USER\_NAME).toString()).getId();

ShiroUtil.setSessionAttribute(Constant.ROLES, TransformUtil.nameSet(roleService.listByUserId(id)));

ShiroUtil.setSessionAttribute(Constant.PERMISSIONS, TransformUtil.nameSet(permissionService.listByUserId(id)));

ShiroUtil.setSessionAttribute(Constant.USER,userService.getById(id));

log.info(new Date()+

" 用户"+param.get(Constant.USER\_NAME).toString()+"登陆成功。");

}

return R.OK("登录成功");

}

1. 注册功能设计
2. 注册页面同登录页面使用简约风格，同样是用到layui的form表单进行表单提交和校验，但在注册是时表单的项相比于登录表单更多、更详细；
3. Service层需要对用户提交的表单进行校验、和密码明文的加密，页面对表单进行各个信息的格式校验后，还需要校验用户信息是否已经存在于数据库中，如果用户的信息已存在是不允许进行注册的；
4. Dao层怎是将用户信息封装层User实体类，分配默认的角色和权限，再将这些信息保存在数据库中。

public R register(Map<String,Object> param) {

if(userService.getByUsername(

(String)param.get(Constant.USER\_NAME))!=null) {

return R.ERROR(Statuscode.UserNameExist);

}

userService.create(param);

return R.OK(Constant.MESSAGE\_REGISTER\_SUCCESS);

}

@Override

@Transactional(rollbackFor = Exception.class)

public Boolean create(Map<String, Object> param) {

Role role = roleService.getByName(Constant.NORMAL\_USER);

if(role==null) {

role = new Role();

role.setName(Constant.NORMAL\_USER);

roleService.save(role);

boolean isExist = false;

Set<Permission> permissions =

permissionService.listByRoleName(role.getName());

if(permissions.isEmpty()) {

for(Permission entity:permissions) {

if(entity.equals(Constant.PERMISSION\_LOGIN)) {

isExist = true;

}

}

}

if(!isExist) {

Permission permission = new Permission();

permission.setName(Constant.PERMISSION\_LOGIN);

permissionService.save(permission);

Role2Permission r2p = new Role2Permission();

r2p.setPermissionId(permission.getId());

r2p.setRoleId(role.getId());

role2PermissionService.save(r2p);

}

}

User newUser = new User(param);

newUser.setCreateTime(new Date());

this.save(newUser);

User2Role u2r = new User2Role();

u2r.setUserId(newUser.getId());

u2r.setRoleId(role.getId());

return user2RoleService.save(u2r);

}

* + 1. 授权功能

1. 每个用户在注册之后都有一个默认的角色USER，该角色拥有的基本权限是登陆权限LOGIN，用必须拥有LOGIN权限才能够访问系统。系统内部有责许多不同的功能和页面，类似于授权的页面必须是超级管理员或者管理员才能够访问，而如果没有这些角色的用户是无法看到功能键的；
2. 权限管理E-R图如下：

如图可知，角色是用户和权限的桥梁，也就是说，用户是通过角色来进行访问系统的，这样不仅能够隐藏用户的信息，提高系统的安全性，同时使得用户访问系统多元化，即可以用角色来访问系统的资源，也可以通过权限来反问资源。

* + 1. 网页资源限权功能

Shiro 提供了JSP 的一套JSTL 标签，用于做JSP 页面做权限控制的。可以控制一些按钮和一些超链接，或者一些显示内容。Shiro关于freemarker的配置，需要完成三部分导入shiro-freemarker-tags包、推送给前端使用的shiro对象变量以及创建并注册自定义freemaker标签

1. 导入shiro-freemarker-tags在emaServer包的pom.xml文件中添加shiro-freemarker-tags的maven坐标。

<dependency>

<groupId>net.mingsoft</groupId>

<artifactId>shiro-freemarker-tags</artifactId>

<version>0.1</version>

</dependency>

1. 在配置包中创建ShiroVariable类，将shiro对象变量推送给前端使用。

@Component

public class ShiroVariable {

public Boolean hasPermission(String permission){

Subject subject= SecurityUtils.getSubject();

return (subject!=null && subject.isPermitted(permission))? true : false;

}

}

|  |  |
| --- | --- |
| 标签 | 功能 |
| <@shiro.guest> | 游客需要访问 |
| <@shiro.user> | 已经登录，或者记住我登录 |
| <@shiro.authenticated> | 已认证，排除记住我登录的 |
| <@shiro.notAuthenticated> | 识别是否本次操作的登陆。 |
| <@shiro.hasRole name="admin"> | 判断是否拥有该角色 |
| <@shiro.hasAnyRoles name="admin,user,member"> | 判断是否拥有这些角色的其中一个 |
| <@shiro.lacksRole name="admin"> | 判断是否不拥有这个角色 |
| <@shiro.hasPermission name="user/add"> | 判断是否拥有这个权限 |

1. Shiro-freemarker提供了如表格中的标签，而由表格可知，并没有判断是否拥有这些权限功能的标签，因此需要自定义标签hasAnyPermission。创建HashAnyPermissionsTag类对hasAnyPermission标签进行定义。PermissionTag是自定义标签必须继承的类，在showTagBody方法中获取当前用户subject和将前端中标签属性name的字符串转换为权限数组，只要用户拥有的其中一个权限存在于权限数组中及判断为允许访问。

public class HasAnyPermissionsTag extends PermissionTag {

private static final String PERMISSION\_NAMES\_DELIMETER = ",";

protected boolean showTagBody(String permissions) {

boolean hasAnyPermission = false;

Subject subject = getSubject();

if (subject != null)

for (String permission : permissions

.split(PERMISSION\_NAMES\_DELIMETER))

if (subject.isPermitted(permission.trim())) {

hasAnyPermission = true;

break;

}

return hasAnyPermission;}

}

1. 自定义完hasAndPermissions后，需要将标签注册到shiro-freemarker的引擎中。创建一个ShiroTagFreeMarkerConfigurer类，将标签名和自定义标签类进行映射。

@Component

public class ShiroTagFreeMarkerConfigurer {

@Autowired

private FreeMarkerConfigurer freeMarkerConfigurer;

@PostConstruct

public void setSharedVariable() throws TemplateModelException {

ShiroTags tags = new ShiroTags();

tags.put("hasAnyPermissions", new HasAnyPermissionsTag());

freeMarkerConfigurer.getConfiguration().setSharedVariable("shiro", tags);

}

}

1. 做完配置和定义后，需要将标签运用到前端中。将各个页面资源分配对应的权限。使用类似以下的格式将页面的元素包含在shiro标签内即可通过权限限制页面元素的显示。

<@shiro.hasAnyPermissions name="role:edit,ADMIN">

<button class="layui-btn layui-btn-sm" lay-event="update">编辑</button>

[</@shiro.hasAnyPermissions>](mailto:</@shiro.hasAnyPermissions>)

1. 以下是各个权限的明细。

|  |  |
| --- | --- |
| 权限 | 明细 |
| \*:add | 添加的权限 |
| \*:delete | 删除的权限 |
| \*:edit | 编辑的权限 |
| event:audit | 事件审核的权限 |
| test:debug | 测试系统的权限 |
| LOGIN | 登陆系统的权限 |
| ADMIN | 超级管理员的权限 |

1. 以下是角色的明细

|  |  |
| --- | --- |
| 角色 | 明细 |
| ADMIN | 超级管理员 |
| DEBUG | 测试人员 |
| USER | 普通用户 |

* + 1. SQL注入过滤器

SQL注入是一种非常常见的数据库攻击手段，SQL注入漏洞也是互联网中最普遍的漏洞之一。SQL注入其实就是恶意用户通过在表单中填写包含SQL关键字的数据来使数据库执行非常规代码的过程，也就是数据“越俎代庖”做代码才能够的事情。问题的来源是SQL数据库的操作是通过SQL语句来执行的，而无论是执行代码还是数据项都必须写在SQL语句之中，这就导致如果在数据项中加入了某些SQL语句关键字，例如‘SELECT’、‘DROP’等，这些关键字就很可能在数据库写入或读取数据时得到执行。注入问题都是因为执行了数据项中的SQL关键字，那么只要检查数据项中是否存在SQL关键字并过滤有SQL关键的请求即可。

1. 创建SQLFilter类，定义SQL过滤器的规则。在SQL过滤器中中待验证的字符串中‘’’，‘\’，‘;’，‘\\’字符去掉，并把待验证字符串转换为小写形式，最后判断带验证字符串是否包含非法字符，如果有非法字符抛出异常。

public class SQLFilter {

public static String sqlInject(String str){

if(StringUtils.isBlank(str)){

return null;

}

str = StringUtils.replace(str, "'", "");str = StringUtils.replace(str, "\"", "");

str = StringUtils.replace(str, ";", "");str = StringUtils.replace(str, "\\", "");

str = str.toLowerCase();

String[] keywords = {"master", "truncate", "insert", "select", "delete", "update", "declare", "alter", "drop"};

for(String keyword : keywords){

if(str.indexOf(keyword) != -1){

throw new CommonException("有SQL注入风险：包含非法字符："+keyword);

}

}

return str;

}

}

1. 在创建定义SQL过滤器完成后，查询时返回给前端的数据都是分页之后的数据，因此SQL过滤器需要用在查询统一封装的工具类中，为查询时SQL注入今夕过滤。

public class QueryUtil<T> {

public IPage<T> getPage(Map<String, Object> params) {

return this.getPage(params, null, false);

}

public IPage<T> getPage(Map<String, Object> params, String defaultOrderField, boolean isAsc) {

long curPage = 1;

long limit = 10;

if(params.get(Constant.PAGE) != null)

curPage = Long.parseLong((String)params.get(Constant.PAGE));

if(params.get(Constant.LIMIT) != null)

limit = Long.parseLong((String)params.get(Constant.LIMIT));

Page<T> page = new Page<>(curPage, limit);

params.put(Constant.PAGE, page);

String orderField =

SQLFilter.sqlInject((String)params.get(Constant.ORDER\_FIELD));

String order = (String)params.get(Constant.ORDER);

if(StringUtils.isNotEmpty(orderField) && StringUtils.isNotEmpty(order))

if(Constant.ASC.equalsIgnoreCase(order))

return page.setAsc(orderField);

else

return page.setDesc(orderField);

if(isAsc)

page.setAsc(defaultOrderField);

else

page.setDesc(defaultOrderField);

return page;

}

public IPage<T> getQueryPage(Map<String, Object> params) {

long curPage = 1;

long limit = 10;

if (params.get(Constant.PAGE)!=null)

curPage=Long.valueOf(params.get(Constant.PAGE).toString());

if (params.get(Constant.LIMIT)!=null)

limit=Long.valueOf(params.get(Constant.LIMIT).toString());

Page<T> page = new Page<>(curPage, limit);

if(params.get(Constant.ORDER)!=null &&

params.get(Constant.ORDER\_FIELD)!=null){

SQLFilter.sqlInject((String) params.get(Constant.ORDER\_FIELD));

if(Constant.ASC.equalsIgnoreCase(params.get(Constant.ORDER).toString()))

return page.setAsc(params.get(Constant.ORDER\_FIELD).toString());

else

return page.setDesc(params.get(Constant.ORDER\_FIELD).toString());

}

return page;

}

}

* 1. 数据库设计
     1. E-R图



* 1. 系统各功能流程

1. 登陆流程
2. 第一步：用户浏览登陆页面，将用户名、密码和验证码返回到LoginService的authentication方法中，调用subject的login方法进行用户认证。登陆成功后，需要将用户的角色、权限以及用户ID存入session中，便于后续的权限管理。
3. 第二步：在subject调用login后，将用户信息存在了session中，在自定义的Realm调用doGetAuthenticationInfo方法进行认证。认证首先需要判断用户是否已经登录，其次是判断用户是否有登陆的权限，最后将用户信息返回给SecurityManager处理。
4. 第三步：securityManager判断用户允许登陆后，shiroFilter会跳转到对应的url。
5. 以下登陆流程的流程图：



1. 注册流程
2. 第一步：用户浏览到注册用户页面，填写用户表单并通过校验返回用户信息到LoginService的register方法，
3. 第二步：而register方法调用了UserService的create方法进行用户创建，因此用户参数输入到userService的create方法中；
4. 第三步：在create方法中，用户需要分配默认的角色和权限，因此需要调用Role2PermissionsService和User2RoleService保存用户与角色，角色与权限的关系。以下
5. 以下注册流程的流程图：



1. Ema事件审核流程
2. 第一步：创建ema事件，用户登陆系统后可以浏览事件管理模块，进入事件管理模块后点击添加按钮创建ema事件，在创建ema事件前需要创建ema事件对应的信息，如uri信息，网站信息、食品信息以及涉事主体信息；
3. 第二步：创建ema事件后，ema事件处于未审核状态，拥有AUDITOR角色或者AUDIT权限的用户可以访问到审核通过和审核未通过按钮，通过这两个按钮可以对事件进行审核；
4. 第三部：如果ema事件被审核未通过，ema事件进入审核未通过状态，因此需要通过编辑ema事件修改ema事件信息并重新提交，让ema事件再次回到审核未通过状态。
5. 以下是ema审核流程图：



1. 授权流程

因为角色是用户和权限的桥梁，所以授权的功能需要基于角色的，如果没有角色的存在是无法进行授权的。

1. 第一步：进入角色管理页面，添加新的角色，完成角色的各个信息并按确认按钮。
2. 第二步：进入授权页面，选择其中一行记录，点击授权按钮，进入授权界面。
3. 第三步：在授权页面中，可以选择对应的用户进行分配角色，因为用的是多选下拉框，可以选择多个用户分配多个权限。
4. 第四步：完成客户端操作后，将用户id集合、角色id以及权限id集合信息传递回service层中进行业务处理，AuthorizationService中调用user2RoleService和role2PermisisonService的saveOrUpdateByMap方法将用户与角色，角色与权限的关系更新或保存。
5. 以下是授权流程图：



1. 实现效果
   1. 开发工具
   2. 运行效果
2. 总结