分类号\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 密级\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

UDC \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 学号\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



毕业设计（论文）

|  |  |
| --- | --- |
| 论文题目 | 基于SSM国家逃犯网络在线查询系统的 设计与实现 |
| **Thesis Topic** | Design And Implementation of Fugitive Criminal Query System Based on SSM |

|  |  |
| --- | --- |
| 学生姓名 | 刘磊 |
| 学 号 | 2015005973 |
| 所 在 院 系 | 软件学院 |
| 专 业 班 级 | 软件1516班 |
| 导师姓名职称 | 张玲 讲师 |
| 完成日期 | 2019年5月8日 |

2019年 5月 8日

太 原 理 工 大 学

太 原 理 工 大 学

毕业设计（论文）任务书

|  |
| --- |
| 毕业设计（论文）题目：  基于SSM国家逃犯网络在线查询系统的设计与实现  毕业设计（论文）要求及原始数据（资料）：  1．综述Web应用开发技术现状；  2．深入了解Java平台的web应用开发技术；  3．熟练掌握SSM框架组合的使用方式；  4．训练软件工程过程规范和文档书写能力；  5．使用SSM框架组合实现逃犯在线查询系统；  6．训练检索文献资料和利用文献资料的能力；  7．训练撰写技术文档与学位论文的能力。 |

|  |
| --- |
| 毕业设计（论文）主要内容：  1．综述现代web应用开发技术；  2．了解查询系统的设计和实现方式；  3．熟悉Java平台的web应用程序开发；  4．熟悉SSM框架的概念和使用方法；  5. 使用SSM框架组合实现逃犯在线查询系统；  6. 完善软件开发过程中的主要文档  7. 完成软件测试及其结果分析。  学生应交出的设计文件（论文）：  1．内容完整、层次清晰、叙述流畅、排版规范的毕业设计论文；  2．包括毕业设计论文、源程序等内容在内的毕业设计电子文档及其它相关材料。 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 主要参考文献（资料）：   1. [Guillaume Chau. Vue.js 2 Web Development Projects[M]. Britain: Packt, 2017.](https://www.packtpub.com/web-development/vuejs-2-web-development-projects) 2. Ravi Sharma, Shipra Ravi Kumar. Strategies for Web Application Development Methodologies [R]. ICCCA, 2016 . 3. [Rick Osowski . Introduction to microservices[J/OL]. IBM Developer, 2015](https://developer.ibm.com/tutorials/cl-ibm-cloud-microservices-in-action-part-1-trs/) . 4. [Craig Walls, 张卫滨. Spring 实战(第四版) [M]. 北京: 人民邮电出版社, 2014](https://www.manning.com/books/spring-in-action-fourth-edition) . 5. 郝佳.Spring源码深度解析(第二版)[M].北京: 人民邮电出版社, 2019. 6. [章仕锋,潘善亮.Docker技术在微服务中的应用[J/OL].电子技术与软件工程,2019(04)](http://kns.cnki.net/kcms/detail/10.1108.TP.20190304.1523.228.html). 7. Martin R C. Agile software development: principles, patterns, and practices[M]. Prentice Hall, 2002. 8. [郭丞乾,蔡权伟,林璟锵,刘丽敏.单点登录协议实现的安全分析[J].信息安全究,2019,5(01)](http://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ&dbname=CJFDTEMP&filename=XAQY201901008&v=MDIwOTFvRkNybVc3ektQU3phZDdHNEg5ak1ybzlGYklSOGVYMUx1eFlTN0RoMVQzcVRyV00xRnJDVVJMT2ZaT1o=) . 9. [魏春来,付永振.基于微服务的DevOps研究与实现[J].网络安全和信息化,2018(11):54-56.](http://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFD&dbname=CJFDLAST2018&filename=WAXX201811039&v=MzA2NzFOcm85R2JZUjhlWDFMdXhZUzdEaDFUM3FUcldNMUZyQ1VSTE9mWk9ab0ZDcm5VYjdKTWl6VGRyRzRIOW4=) 10. [Building a RESTful Web Service[J/OL]. Spring.io](https://spring.io/guides/gs/rest-service/)  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 专业班级 | 软件1516班 | | | | | 学生 | 刘磊 | | 要求设计（论文）工作起止日期 | | | | | 2019年3月18日~2019年6月21日 | | | | 指导教师签字 | |  | | | | 日期 | 2019年3月15日 | | 教研室主任审查签字 | | | |  | | 日期 |  | | 系主任批准签字 | | |  | | | 日期 |  | |

基于SSM国家逃犯网络在线查询系统  
的设计与实现  
摘要

SSM 即Spring、SpringMVC、MyBatis，是Java平台的现代Web应用开发最成熟的程序开发框架组合之一。Spring即SpringCore提供了面向对象程序设计中通过依赖注入实现控制反转的基本核心库。SpringMVC则提供了对JavaEE平台上的Servlet技术的高层封装，提升开发效率；同时正如其名MVC(Model, View, Controller)， SpringMVC提供了基于模型-视图-控制器模式的web开发功能，实现了web应用开发过程中的架构分层，同时新版本SpringMVC也跟随现代web应用开发的最新技术，提供了对RESTful API开发的支持。MyBatis则是目前java平台流行的ORM（对象关系映射）框架，通过定制化SQL配置将POJO映射到数据库关系模式，且其提供的动态SQL可以非常灵活的根据参数状况改变SQL成分，解决了原生JDBC开发中拼接SQL遇到的格式化难的问题甚至根本不再需要直接接触任何JDBC代码。

本文通过基于上述技术设计和实现一个“逃犯在线查询系统”的过程来展现现代Java平台web应用开发的前沿技术，并通过实现一个单点登录模块来简要体现微服务架构的web应用设计，另外通过使用Kubernetes（Docker based）部署项目来介绍容器化应用部署对于微服务架构的重要性。

**关键词** SSM；SpringBoot；微服务；容器化；查询系统

Design And Implementation of Fugitive Criminal Query System  
Based on SSM  
Abstract

SSM, namely Spring, SpringMVC, and MyBatis, is one of the most mature program development frameworks for modern web application development on the Java platform. Spring, SpringCore, provides a basic core library for control inversion through dependency injection in object-oriented programming. SpringMVC provides high-level packaging of Servlet technology on the JavaEE platform to improve development efficiency. At the same time, as its name is MVC (Model, View, Controller), SpringMVC provides web development function based on model-view-view controller mode. The architecture layering in the web application development process, while the new version of SpringMVC also follows the latest technology of modern web application development, providing support for RESTful API development. MyBatis is the popular ORM (Object Relational Mapping) framework of the current Java platform. It maps POJOs to database relational schema through customized SQL configuration, and the dynamic SQL provided by it can flexibly change the SQL components according to the parameter status and solve the native JDBC. The problem of formatting difficulties encountered in splicing SQL in development does not even require direct contact with any JDBC code.

This paper demonstrates the technology of modern Java platform web application development by designing and implementing a " Fugitive Criminal Query System" based on the above technologies and realizes the web application design of microservice architecture by implementing a single sign-on module. Use the Kubernetes (Docker-based) deployment project to introduce the importance of containerized application deployment to the microservices architecture.

**Keywords**: SSM; SpringBoot; microservices; containerization; query system

目 录

[摘要 1](#_Toc9026253)

[Abstract 1](#_Toc9026254)

[1. 绪论 1](#_Toc9026255)

[1.1. 课题背景 1](#_Toc9026256)

[1.2. 本文研究内容 1](#_Toc9026257)

[1.2.1. 逃犯查询系统 1](#_Toc9026258)

[1.2.2. 现代web应用开发技术 1](#_Toc9026259)

[1.3. 本文结构 2](#_Toc9026260)

[2. 开发与运行环境介绍 3](#_Toc9026261)

[2.1. 开发环境 3](#_Toc9026262)

[2.1.1. 概述 3](#_Toc9026263)

[2.1.2. 硬件环境 3](#_Toc9026264)

[2.1.3. 软件环境 3](#_Toc9026265)

[2.2. 线上环境 3](#_Toc9026266)

[2.2.1. 概述 3](#_Toc9026267)

[2.2.2. 云计算资源详情 4](#_Toc9026268)

[2.2.3. Kubernetes 集群管理 4](#_Toc9026269)

[2.3. 开发过程辅助工具 5](#_Toc9026270)

[2.3.1. 版本控制系统 – Version Control System 5](#_Toc9026271)

[2.3.2. 依赖管理工具 – Dependency Management Tool 7](#_Toc9026272)

[2.4. 其他编程环境 8](#_Toc9026273)

[2.4.1. SpringBoot 8](#_Toc9026274)

[2.4.2. Java 11 8](#_Toc9026275)

[2.4.3. Lombok 8](#_Toc9026276)

[2.4.4. Golang 9](#_Toc9026277)

[2.4.5. PostgreSQL 9](#_Toc9026278)

[2.4.6. Redis 9](#_Toc9026279)

[3. 需求分析 11](#_Toc9026280)

[3.1. 可行性分析 11](#_Toc9026281)

[3.1.1. 系统基本流程 11](#_Toc9026282)

[3.1.2. 经济可行性分析 11](#_Toc9026283)

[3.1.3. 技术可行性分析 11](#_Toc9026284)

[3.2. 系统需求分析 12](#_Toc9026285)

[3.2.1. 编写目的 12](#_Toc9026286)

[3.2.2. 数据描述 12](#_Toc9026287)

[3.2.3. 功能需求 13](#_Toc9026288)

[3.2.4. 环境需求 14](#_Toc9026289)

[4. 系统总体设计 15](#_Toc9026290)

[4.1. 引言 15](#_Toc9026291)

[4.1.1. 编写目的 15](#_Toc9026292)

[4.1.2. 读者对象 15](#_Toc9026293)

[4.1.3. 定义 15](#_Toc9026294)

[4.2. 总体设计 16](#_Toc9026295)

[4.2.1. 处理流程 16](#_Toc9026296)

[4.2.2. 识别用户角色 16](#_Toc9026297)

[4.2.3. 用例分析 17](#_Toc9026298)

[4.2.4. 总体结构 17](#_Toc9026299)

[4.2.5. 模块及功能 17](#_Toc9026300)

[4.2.6. 数据结构设计 20](#_Toc9026301)

[5. 系统详细设计 27](#_Toc9026302)

[5.1. 引言 27](#_Toc9026303)

[5.1.1. 编写目的 27](#_Toc9026304)

[5.1.2. 读者对象 27](#_Toc9026305)

[5.1.3. 本文档的结构 27](#_Toc9026306)

[5.2. 接口规范约定 27](#_Toc9026307)

[5.2.1. 全局响应数据示例 27](#_Toc9026308)

[5.2.2. 状态码定义 27](#_Toc9026309)

[5.3. 后端系统实现技术方案 28](#_Toc9026310)

[5.3.1. 项目结构说明 28](#_Toc9026311)

[5.3.2. SpringMVC开发RESTful API说明 28](#_Toc9026312)

[5.3.3. 逃犯查询系统API权限校验方式 29](#_Toc9026313)

[5.3.4. 单点登录系统实现技术方案 29](#_Toc9026314)

[5.3.5. 构建全国行政区划数据库 33](#_Toc9026315)

[5.3.6. 构建文件存储服务 35](#_Toc9026316)

[5.3.7. 部门管理模块实现技术方案 36](#_Toc9026317)

[5.3.8. 用户管理模块实现技术方案 36](#_Toc9026318)

[5.3.9. 逃犯信息模块实现技术方案 38](#_Toc9026319)

[5.4. 前端实现技术方案 41](#_Toc9026320)

[5.4.1. 前端技术概述 41](#_Toc9026321)

[5.4.2. Vue插件和第三方工具 42](#_Toc9026322)

[5.4.3. 前端程序主体结构 43](#_Toc9026323)

[5.4.4. 页面样式 44](#_Toc9026324)

[5.4.5. 部门选择器组件 45](#_Toc9026325)

[6. 系统测试 47](#_Toc9026326)

[6.1. 测试任务及目标 47](#_Toc9026327)

[6.2. 测试方案 47](#_Toc9026328)

[6.2.1. 概述 47](#_Toc9026329)

[6.2.2. 使用Postman测试RESTful API 47](#_Toc9026330)

[6.2.3. 测试用例 48](#_Toc9026331)

[结论 50](#_Toc9026332)

[参考文献 51](#_Toc9026333)

[致谢 52](#_Toc9026334)

[外文原文 53](#_Toc9026335)

[中文翻译 60](#_Toc9026336)

1. 绪论
   1. 课题背景

从1990年第一个网页浏览器诞生以来，web应用开发已经有了将近30个年头的历史，从最初的完全静态页面，到后来支持数据持久化和动态计算与显示。后来使用JavaEE平台来开发web应用成为最广泛的方案，并且在此过程中出现了非常经典的MVC应用架构。但是MVC应用架构是单体应用时代的产物，主要采用的是服务端对数据进行处理并直接填充进最终的html代码中最后返回给浏览器的内容是一切都组装好的。

最近的几年web应用的开发出现了更大的变化，MVC的方式正在逐渐成为过去，更新的方式是前后端完全分离。后端应用编写时不用再考虑视图的问题，只需要通过HTTP协议对外提供数据接口即可；使用JSON[[1]](#footnote-1)作为数据序列化格式，web API 采用REST[[2]](#footnote-2)风格设计，然后前端应用使用MVVM[[3]](#footnote-3)框架来构建单页面应用，全部的数据交互都是通过发起HTTP异步请求的方式。

* 1. 本文研究内容
     1. 逃犯查询系统

拿到题目后通过多种途径搜索资料关于逃犯查询系统的信息少之又少，几乎没有相关的文献或项目。对于逃犯查询系统的相关功能，在公开资料中本文没有可借鉴资源。所以对于功能需求，将仅实现基本的逃犯信息的增删查改。

* + 1. 现代web应用开发技术

本论文的研究重点一方面是设计实现逃犯查询系统，更重要的一方面是在这个过程中来尽可能多的使用web应用软件开发领域的最新技术来开发、架构和运行一套web应用软件并以此宏观的介绍web前端开发、web后端开发以及web应用实施部署方案的最新技术。主要包括web前端开发技术中的MVVM框架--Vue、微服务web应用架构、后端开发方案--RESTful API、使用Docker容器技术来部署各种服务以及使用Kubernetes在集群中进行容器编排与负载均衡。

* 1. 本文结构

本文主体是一个web应用软件的开发文档，在软件开发过程的每个阶段对应的文档的适当位置来叙述现代web应用开发使用的新技术。

第一章绪论介绍项目背景和研究内容；第二章介绍逃犯查询系统开发和部署的各种软件工具和环境；第三章为逃犯查询系统的需求说明文档；第四章为逃犯查询系统的总体设计文档；第五章依据总体设计文档对系统实现进行描述，作为详细设计说明文档；第六章为系统测试文档；最后附上本论文的参考文献。

1. 开发与运行环境介绍
   1. 开发环境
      1. 概述

在开发环境使用直接使用Docker来运行PostgreSQL和Redis等持久化和中间件工具，通过定义容器镜像文件来锁定各种软件版本；在线上环境部署应用时直接使用定义好的镜像和软件配置来创建容器实例。如此可消除由于开发环境与生产环境的软件差异带来的部署和调试困难的问题，另外也可以大大加快应用部署速度。

* + 1. 硬件环境

表 2.1 开发环境硬件环境

|  |  |
| --- | --- |
| 硬件名称 | 硬件信息 |
| CPU | Intel(R) Core(TM) i5-7360U CPU @ 2.30GHz |
| 内存 | 16 GB 2133 MHz LPDDR3 |
| 显示 | 13.3－英寸 (2560 x 1600)  Intel Iris Plus Graphics 640 1536 MB 图形卡 |

* + 1. 软件环境

表 2.2开发环境软件环境

|  |  |  |
| --- | --- | --- |
| 用途 | 软件项目 | 版本 |
| 操作系统 | MacOS Mojave | 10.14.4 |
| 容器引擎 | Docker | CE 18.09.2 |
| 关系型数据库 | PostgreSQL | 11.2 |
| 缓存数据库 | Redis | 5.0.4 |
| 程序平台 | Java (JDK \ JRE) | 11.0.3 |
| IDE | Intellij IDEA | Ultimate 2019.1 |

* 1. 线上环境
     1. 概述

本文所涉及的软件项目在演示阶段将部署于华为云计算平台，使用华为云提供的“云容器引擎CCE”通过Kubernetes来管理软件应用所有组件的发布和运行。使用Docker官方镜像仓库Docker Hub来分发构建好的容器镜像，再通过华为云CCE拉取镜像构建容器最终运行并对外。具体购买的华为云计算资源详情详见下表。

* + 1. 云计算资源详情

表 2.3云计算资源详情

|  |  |
| --- | --- |
| 项目名称 | 项目信息 |
| 弹性云服务器ECS | 通用计算型|s2.large.2  2核CPU，4GB RAM |
| 操作系统 | Linux 3.10.0 |
| 网络 | 静态BGP, 固定带宽 1Mbps |
| 云容器引擎CCE | Kubernetes 1.11.7-r0  Docker 17.06.0.60 |

* + 1. Kubernetes 集群管理

Kubernetes是一个容器集群编排服务，它用于在集群上部署容器化应用程序、弹性扩展应用、自动更新软件版本（支持持续集成）且无需停机。Kubernetes在集群上部署容器化应用程序可以支持将应用程序示例自动调度到集群中的各个节点，创建应用程序实例后，Kubernetes会持续监视这些实例，如果节点出现了故障，部署控制器也将替换相应的实例，也就是提供了一种机器故障自愈机制。

Kubernetes采用主从节点的方式运行。集群中存在一个Master节点，负责维护集群的目标状态，向Kubernetes下发任务只需要与Master节点进行通讯。集群中的其他节点叫做Node节点，可以是虚拟机或物理机，用来实际承载部署到集群中的应用实例。对Kubernetes集群下发的任务通常是重启容器、调整制定应用的副本数量、创建新的部署等等。

图 2.1 Kubernetes集群工作示意图[[4]](#footnote-4)

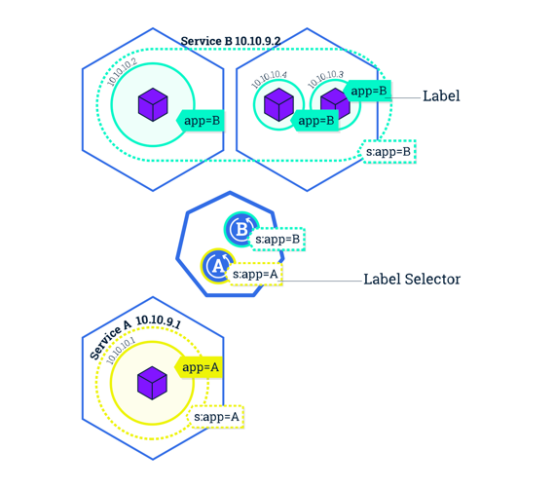
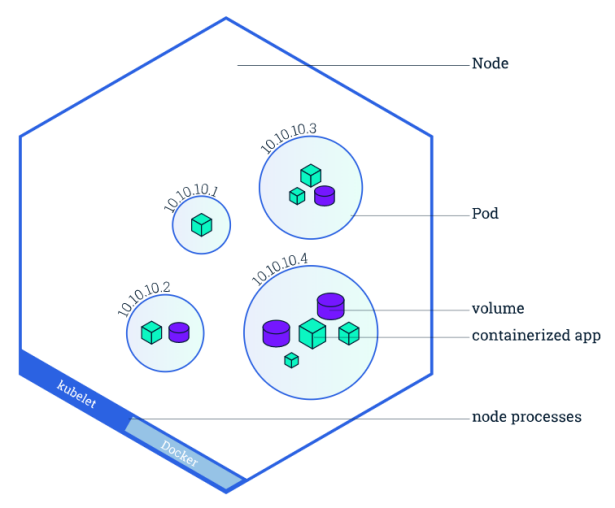


图 2.2 Kubernetes节点概念图



* 1. 开发过程辅助工具
     1. 版本控制系统 – Version Control System
        1. 版本控制解决什么问题

软件开发过程通常是一个不断改进和迭代的过程，随着需求的不断完善和细化以及系统设计的改进和重构，软件代码需要不断的满足要求而发生变化。

对于代码量很小的项目可能由单个开发者凭借对项目的熟悉即可熟知每一个变更细节，掌控全局代码；但是对于一个代码量比较大，功能模块较多的复杂软件系统而言，代码的管理变得非常困难；在代码作出修改版本进行演进之后很可能会出现bug，这时无法追踪代码的改变会使得代码的修复变得比开发还困难。

另一方面，通常一个复杂的系统不是由单个程序员完成开发的，需要有一个团队甚至数个团队的通力配合来共同完成一套系统的代码实现。如何将所有成员的开发成果进行集成、运行、调试、测试也是一个非常重要的问题。

另一个问题是，对于现代软件工程项目，通常是线上运行着一个稳定版本，而开发团队在不断研发着好多新特性，也会有不同的小团队来负责不同的特性开发；这些由不同的小组负责的新功能特性开发工作常常是同步进行的。这样就可能导致为了不同的开发任务编辑或修改了软件产品的同一处代码实现从而产生了代码冲突，此时如果没有一个强大的工具来管理代码，似乎新的几个功能根本没办法同时集成进已有系统。

* + - 1. 版本控制系统如何解决问题

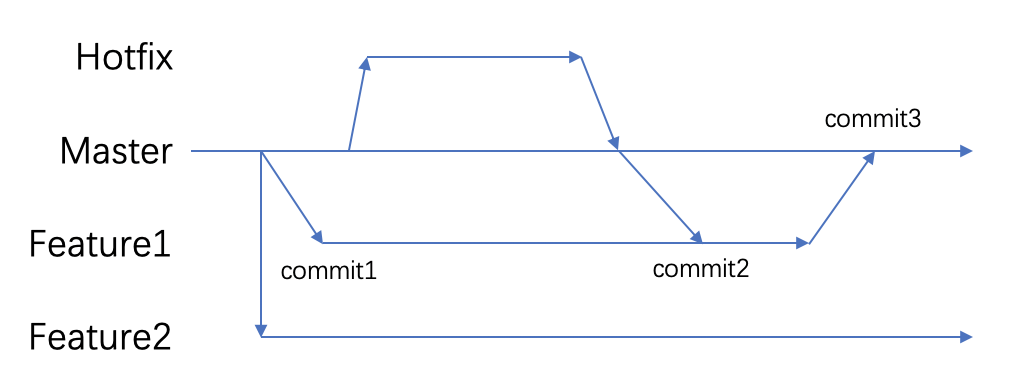
首先VCS的最基本任务就是记录代码的修改。使用版本控制系统来追踪(Track)源代码文件，进行过一些修改后通过提交命令(commit)来提交本次修改，这样版本控制系统就会记录下这次修改的内容；随后可能发生了很多次commit，版本控制系统都会记录下来。价值就在于对每一次commit，随时都可以找回(checkout)查看并重新编辑之前发布的内容，每一个commit记录都是“时光机”的跳跃点。

VCS的另一个重要作用就是解决线上线下多个软件版本在多个软件开发小组之间的协作问题。VCS通过“分支”(branch)机制来管理处在不同开发进度的多个软件版本；对于每一个正在研发的新功能特性都可以建立一个分支，这些分支和主版本之间的关系就像树枝与树干的关系，各自可以有自己的版本演进(commit)；而强大之处就在与这些分支可以随意的进行合并(merge)，只要能解决不同分支之间的冲突[[5]](#footnote-5)(conflict)即可。

对于团队协作，VCS发挥了不可忽略的作用，所有的开发人员不需要考虑使用什么方式将自己的代码共享给其他人，大家通过VCS就可以看到所有分支的所有commit状况[[6]](#footnote-6)。

另外VCS在现代web应用的维护中还担任着持续集成(CI: Continuous Integration)与持续交付(CD: Continuous Delivery)过程中的代码资源库的角色。对于其他VCS的作用将不再深入讨论。

图 2.3产品分支模型



* + - 1. VCS的选择

本文中的示例项目将使用目前最流行的版本控制工具GIT来进行版本控制，并使用GitHub来作为远程git仓库来保存代码；项目将在本论文通过所有审核后变为开放访问权限。

GitHub地址：

后端项目：<https://github.com/linkinghack/criminalquery>

前端项目：<https://github.com/linkinghack/CriminalQueryApp>

* + 1. 依赖管理工具 – Dependency Management Tool
       1. 依赖问题

在软件开发过程中时常需要调用一些第三方库来利用开源成熟解决方案解决软件开发中的某些问题。事实上这是一个发生非常频繁的事件，并且依赖项目之间也有相互的依赖关系。

* + - 1. 解决方案

在本文所述的“逃犯查询系统”中，将使用Java平台比较流行的Apache Maven来管理依赖组件。

Maven可以用来管理Java软件项目开发过程中的依赖jar包，也可以用来进行软件构件(build)，还可以通过模块化的方式来拆分项目，并解决模块间的依赖调用；配合IDE的提示功能在模块间的依赖代码也可以获得完整的代码提示。Maven对于依赖项的上层依赖和重复依赖都能自动解决。

图 2.4 Maven打包命令示例

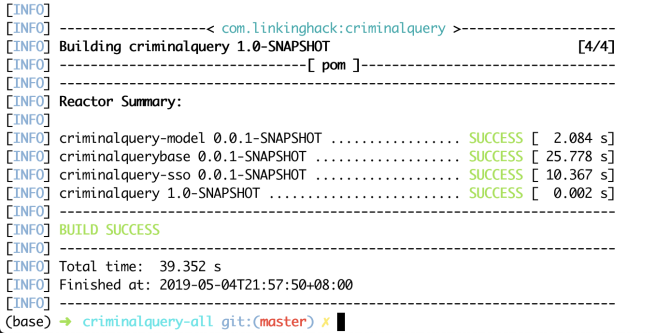
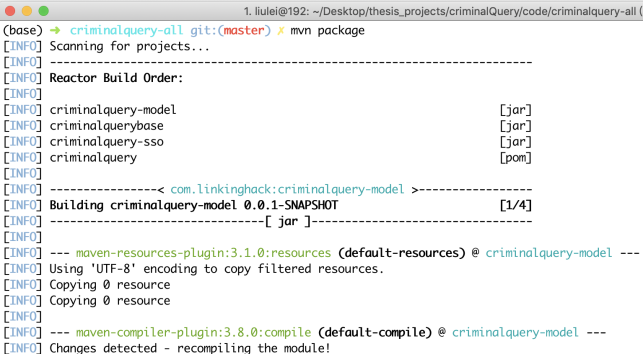
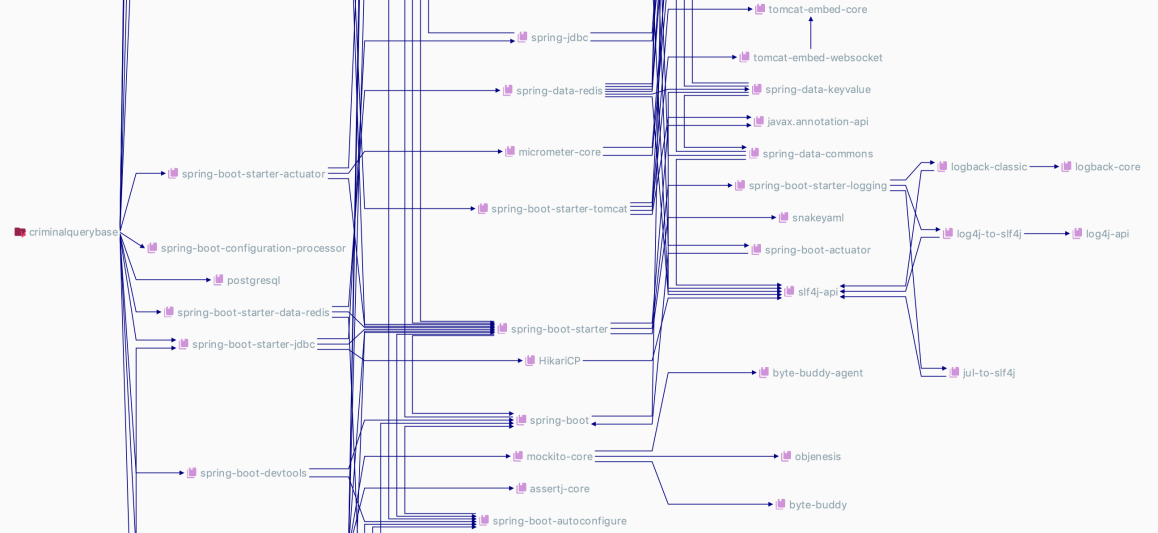


图 2.5项目Maven依赖树局部



* 1. 其他编程环境
     1. SpringBoot

Spring套件虽然强大，但是其繁杂冗长的配置文件让开发效率变得非常低，查文档补配置需要花费大量的时间，并且与Spring集成的各种第三方组件的配置通常是固定的。为了解决配置困难问题以及适应微服务的开发，SpringBoot提供了解决方案。

SpringBoot通过一系列Maven(或Gradle)的starter项目来预置一些经常使用的项目配置方案，在pom中引入这些starter项目就可以直接开始开发业务相关的功能。在配置方面，SpringBoot提供了全局配置方案，在项目classpath中直接编辑application.yml或application.properties文件即可对Spring和其他支持的第三方库进行统一配置。另外SpringBoot集成了内置的Tomcat，可以将整个项目打包成可执行jar，无需再单独安装Servlet容器。

* + 1. Java 11

虽然目前用户最多的Java版本是Java 8，但是自从Oracle宣布Java的半年期迭代计划以来，已经有四个主要版本更新，目前最新为2019年3月下旬发布的 Java 12。Java 11是目前的LTS(Long Term Support)版本，为了兼顾新特性和稳定性，故在本论文所涉及的项目中使用Java 11（包括JDK和JRE）。

* + 1. Lombok

Lombok是一个帮助创建POJO的小插件。在Web应用开发过程中通常需要定义许多领域模型，它们仅仅用来表示数据结构，除了必要的属性定义外，剩下的都是重复性的Getter和Setter方法。为了使代码更加简洁，节约由无意义的劳动浪费掉的时间，本项目使用Lombok来自动完成Getter和Setter的添加，只需要在POJO类上使用@Data注解修饰即可。

* + 1. Golang

Golang是Google开发的一款强类型编译型语言，其语法简单，核心库完善，已经成为现代web应用开发的新秀。越来越多的公司将Golang作为业务实现的主要语言；具有代替JavaEE的潜力。

在本论文所涉及的项目中使用Golang完成了全国行政区划数据库的数据导入工作。

* + 1. PostgreSQL

PostgreSQL是一个开源的关系型数据库管理系统，起源于伯克利大学的POSTGRES项目。PostgreSQL支持非常丰富的数据类型，具有不错的性能表现。本系统使用PostgreSQL作为主要的数据持久化后端。在构建全国行政区划数据库时行政区层级路径边用到了PostgreSQL的数组类型。

程序与数据库建立连接使用了SpringBoot集成的Hikari Pool。它是一个性能表现优异的数据库连接池工具。数据库连接池将保持几个数据库连接不断开，可以一直等待服务，在并发访问量增加时也能自动增加连接数量，动态平衡网络资源。通过设置活动连接数量来保证程序需要数据库访问时可以立即得到结果，节省了因为建立TCP连接而用掉的时间。

PostgreSQL的部署，本系统使用Docker来部署PostgreSQL，postgres将作为一个服务在docker-compose中启动，在开发环境通过暴露5432端口到宿主机来支持开发测试，在生产环境将仅允许docker网络内部访问，保护数据库安全。

* + 1. Redis

Redis是一个内存数据库，使用Key-Value的方式保存数据，支持String、List、Set、Sorted Set、Hash、Bit array、HyperLogLog、Stream八种Value数据类型；而Key的数据类型则是支持使用任意的二进制序列，甚至可以用一张图片的二进制序列作为Key。

在本系统中将Redis作为单点登录系统用于会话控制的工具，得益于Redis提供的数据有效期功能，单点登录系统在缓存Token和用户数据时将设置一个过期时间，此过期时间也就成为了Token对应的登录态的有效期；过期后Token访问无效，以此保护用户账号安全。

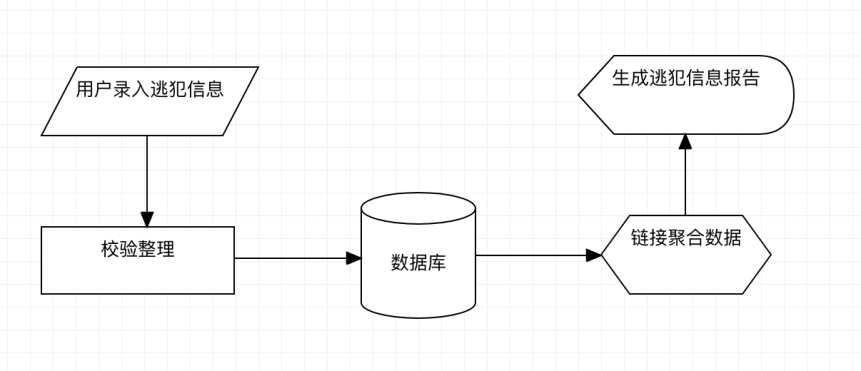
Redis也作为一个服务使用docker部署。与redis的连接使用SpringBoot提供的StringRedisTemplate，不需要池化连接。

1. 需求分析
   1. 可行性分析
      1. 系统基本流程

逃犯查询系统最基本功能便是录入逃犯相关信息，然后随时可以查询已经录入过的逃犯信息。目标用户是公安部门机关内部人员。

目标系统要求可以供全国公安部门使用，数据统一管理，保证一致性。另外要求应用部署简单，具有一定的可扩展性和较强的可维护性；同时对于客户端的硬软件要求尽量降到最低，做到随时随地可用。

图 3.1系统流程图



* + 1. 经济可行性分析

开发此系统总体来说难度不大，属于三到五人团队一个月的工作量。在部署实施方面，可以直接使用公有云计算业务提供的基础计算和存储能力；由于此系统是内部系统，服务的用户数量比较少比较稳定，不会具有突然增加的性能问题，所以计算资源的年支出费用将比较固定。

* + 1. 技术可行性分析

由于本项目将完全重新开发，不需要考虑是否有就项目的整合问题，所以在技术选型上没有约束，选择最新的架构和开发方式。

为了降低对客户端的硬软件要求，采用B/S架构的基本方式，可以提供以web页面的方式访问系统全部功能的用户接口。

对于可扩展性的要求，可以采用后端基于最通用的HTTP协议提供JSON REST API的方式，如此可以放宽对客户端软件的支持范围。最初可以仅支持标准web页面访问，随后可以在后端逻辑只做很少甚至不需要改动的情况下迅速接入其他客户端类型，比如直接支持手机App、微信小程序等等。

对于易维护性，推荐技术架构首选微服务方式，不同的模块之间耦合非常低，一个服务出现故障不至于影响整个系统。微服务架构方式配合容器化应用部署可以做到灵活的服务伸缩，如果系统用户的确不断增长也可以很快的增加计算能力。

* 1. 系统需求分析
     1. 编写目的

本需求规格说明书旨在说明“逃犯查询系统”需要具有的功能特征，分析系统的性能需求、环境需求以及接口需求和用户界面需求；最终确定系统的数据需求，使用有效的数据建模方法，产出数据流图、数据字典与实体关系图。

* + 1. 数据描述
       1. 静态数据

**全国行政区划数据**：需要建立一个全国三级行政区划数据库，包括省、市、县/区以及所有这些级别分区之间的层级关系。

* + - 1. 动态数据

**逃犯基本信息**：逃犯姓名、逃犯性别、逃犯身高、逃犯生日(计算年龄)、逃犯籍贯、逃犯身份证号码。

**逃犯附加信息**：逃犯主要识别图像、逃犯教育背景、逃犯工作种类、逃犯工作单位、逃犯曾住址、逃犯曾用通讯方式、逃犯其他特征描述

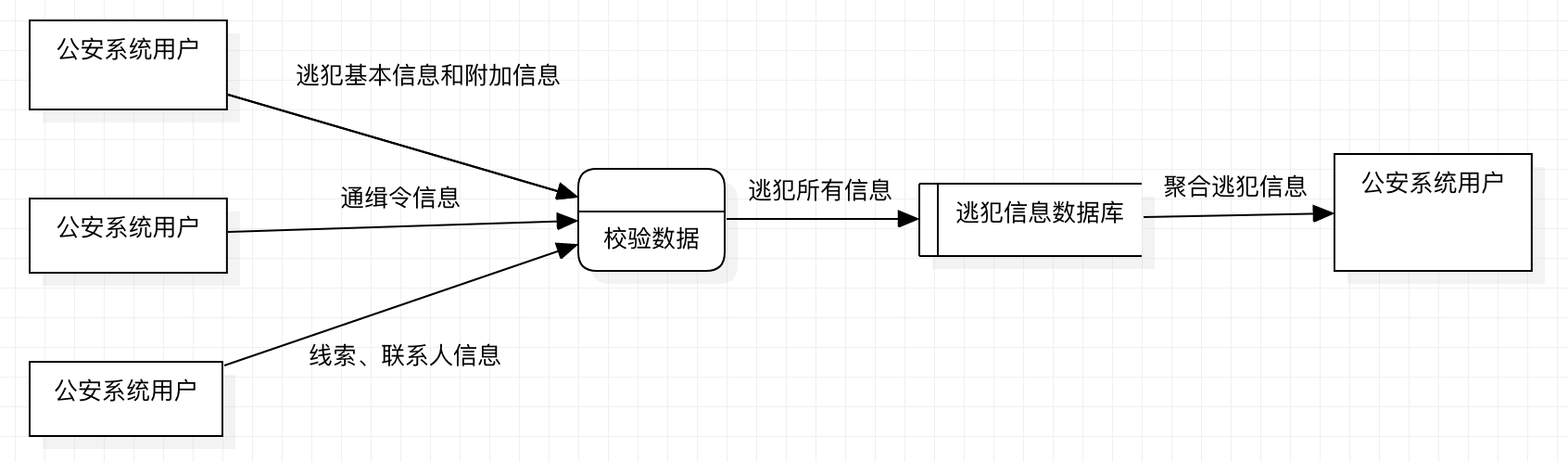
**逃犯相关联系人信息**：联系人姓名、联系人性别、联系人与逃犯关系、联系人电话、联系人地址、联系人生日(计算年龄)

**通缉令**：通缉逃犯、通缉原因、通缉区域

**逃犯线索**：资料图片、线索描述

* + - 1. 数据流图

图 3.2数据流图



* + - 1. 数据字典

**逃犯基本信息和附加信息** = 逃犯姓名+逃犯性别+逃犯身高+逃犯生日+逃犯籍贯+逃犯身份证号码+逃犯主照片+教育背景+工作种类+工作单位+通讯方式+地址+描述

**通缉令信息** = 通缉时间+通缉原因+通缉区域+通缉对象

**线索** = 相关逃犯对象+线索描述+相关图片

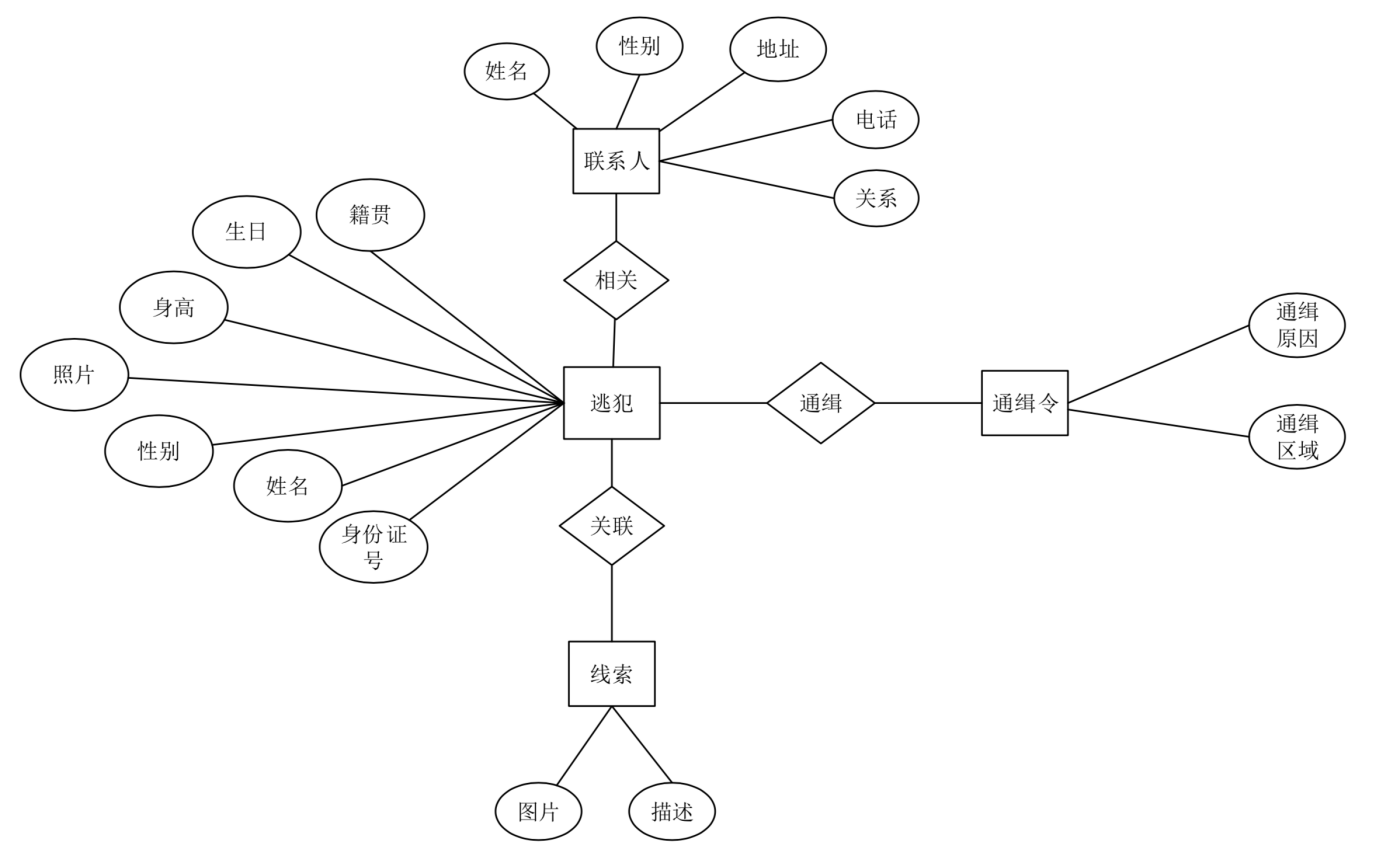
**联系人信息** = 姓名+年龄+性别+关系+联系方式+地址

**逃犯所有信息** = 逃犯基本信息+逃犯附加信息+逃犯通缉令信息+线索+联系人

**聚合逃犯信息** = 逃犯基本信息+与此逃犯相关的所有通缉令+与此逃犯相关的所有线索+与此逃犯相关的所有联系人

* + - 1. 实体联系图

图 3.3实体联系图



* + 1. 功能需求

**添加逃犯**：系统最基础功能之一，添加逃犯基本信息，可记录项目参考数据描述。

**通缉逃犯**：系统可以对逃犯发起通缉，通常在第一次添加逃犯时就发起第一个通缉；随后系统中的其他用户可以对逃犯进行二次通缉。

**添加逃犯线索**：系统需要具有一定的记录逃犯线索的能力，对于每一个逃犯可以添加数条线索。

**添加逃犯联系人信息**：对于每一个逃犯，可以添加其相关联系人信息，以供公安部门调查使用。

**编辑逃犯信息**：对于已经加入到系统中的逃犯信息可以进行修改。

**修改通缉令状态**：可以对通缉令的开关状态进行控制。

**删除通缉令**：管理员可以删除通缉令。

**发布广范围通缉**：用户发起通缉时仅能在所在部门管辖区域内发布通缉信息，后续可以提供将通缉令发布至更大区域，由对应区域的管理员进行审核方可发布。

**用户登录**：仅限公安系统内部使用，除登录、注册之外的所有功能都与要有用户权限校验。

**用户注册**：为方便添加系统用户，允许公安系统用户自主申请账号，注册之后账号处于冻结状态，无法登录；等管理员核对通过后账号方可使用。

部门管理：

**添加部门**：使用系统的公安部门灰度加入此系统，系统中对用户安部门进行简单管理。部门类似“公安部”、“xx省公安厅”、“xx市公安局”、“xx县/区公安局”，对应部门信息中记录部门管辖区域，影响部门成员可以发布通缉令的通缉范围。

**删除部门**：可以对部门进行删除操作。

用户管理：

**新用户审批**：管理员对于新注册的处于冻结状态的账号可以进行激活操作，账号核对任务应该人工完成。

**新用户驳回**：管理员可以拒绝新用户的注册请求。

**用户删除**：对于系统内处于激活状态的用户也可进行删除操作。

**用户识别**：系统仅供内部使用，需要有身份校验机制。

**权限控制**：对于一些敏感操作，如删除数据，仅管理员可进行操作。

* + 1. 环境需求

系统应该可以在CentOS 7 / Java11 环境中运行。

1. 系统总体设计
   1. 引言
      1. 编写目的

系统总体设计说明书用于概括逃犯查询系统如何完成需求中指定的功能，对系统架构进行设计，以及对系统模块划分以及每个模块应该处理的任务进行说明。

* + 1. 读者对象

本系统的软件开发人员。

* + 1. 定义
       1. 微服务

微服务架构是 Martin Fowler 和 James Lewis 定义的一种架构风格。他们将这种风格描述为"一种使用小型服务构建系统的架构方法，每个服务都在自己的进程中，它们通过轻量型协议进行通信"。每个服务都是独立开发和部署，每个微服务专注一个自己擅长的小任务。[11]

微服务具有技术中立、松散耦合的特点，每个微服务的开发团队可以使用自己熟悉的编程语言，仅需确保各个服务之间可以通过约定的通信协议进行通信即可。

* + - 1. SSO

SSO 即Single Sign On 单点登录系统。用户仅需要登录一次，就可以使用相关系统的全部服务，即使这些服务并没有部署在同一台服务器甚至不再同一个地点。单点登录系统为系统内所有受信任的服务提供用户身份校验功能，其他服务只需要关注本身的业务逻辑，只需要很少的开发就可以完成与单点登录系统的整合。

* + - 1. REST

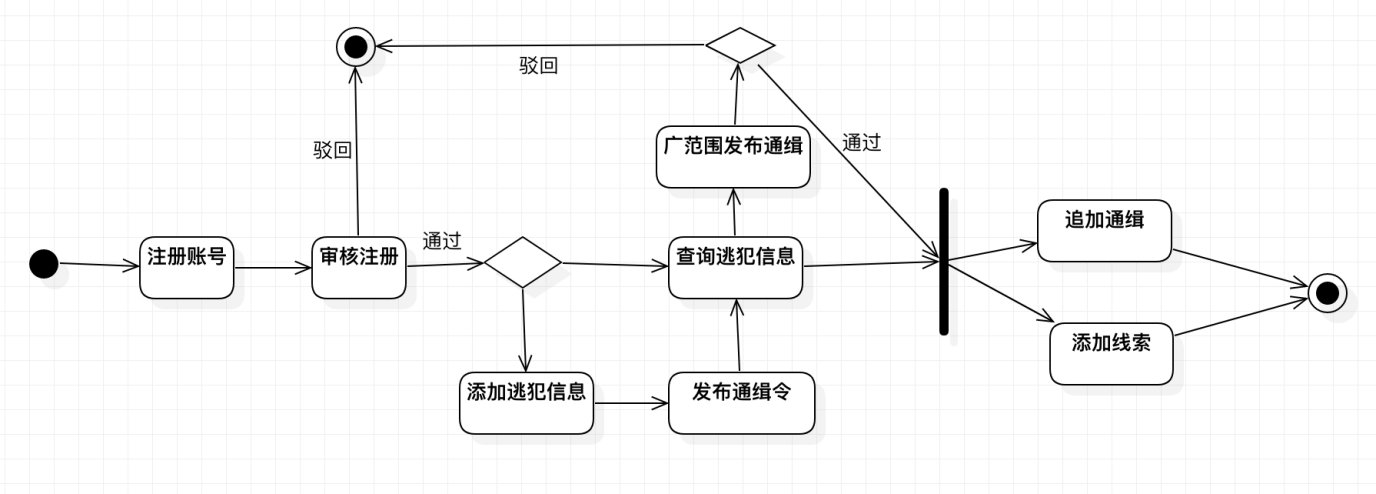
REST即Representational State Transfer 表征状态转移，是一种软件架构风格。对于Web服务发开来说，RESTful Web服务推荐关注资源状态的流转，每一个URI对应一个资源，充分利用HTTP协议的方法（GET、POST、PUT、DELETE、PATCH等）来表示对资源的操作。每个资源对象公开一个URI表示自己的地址，然后在对此地址进行相应的操作。比如使用POST方法新建一个对象，使用GET方法获取一个对象，使用PUT或PATCH对资源进行更新，使用DELETE方法删除资源；这样就分别对应了信息管理系统中最常进行的增、查、改、删功能。

* + - 1. 参考资料

逃犯查询系统需求分析说明书

* 1. 总体设计
     1. 处理流程

图 4.1逃犯查询系统活动图



* + 1. 识别用户角色

**普通用户**：使用本系统的最普遍用户，属于公安系统内部用户，具有添加逃犯信息、添加逃犯线索、添加逃犯联系人信息等基本功能的权限，也可以编辑逃犯信息。具有查询系统内逃犯信息的权限。

**管理员用户**：管理员用户在每个部门中都可以存在，主要职责是审核新用户和广范围发布通缉的请求。除具有普通用户的所有权限外，还具有删除通缉令、删除逃犯信息、删除用户、新建部门、删除部门等高级权限。

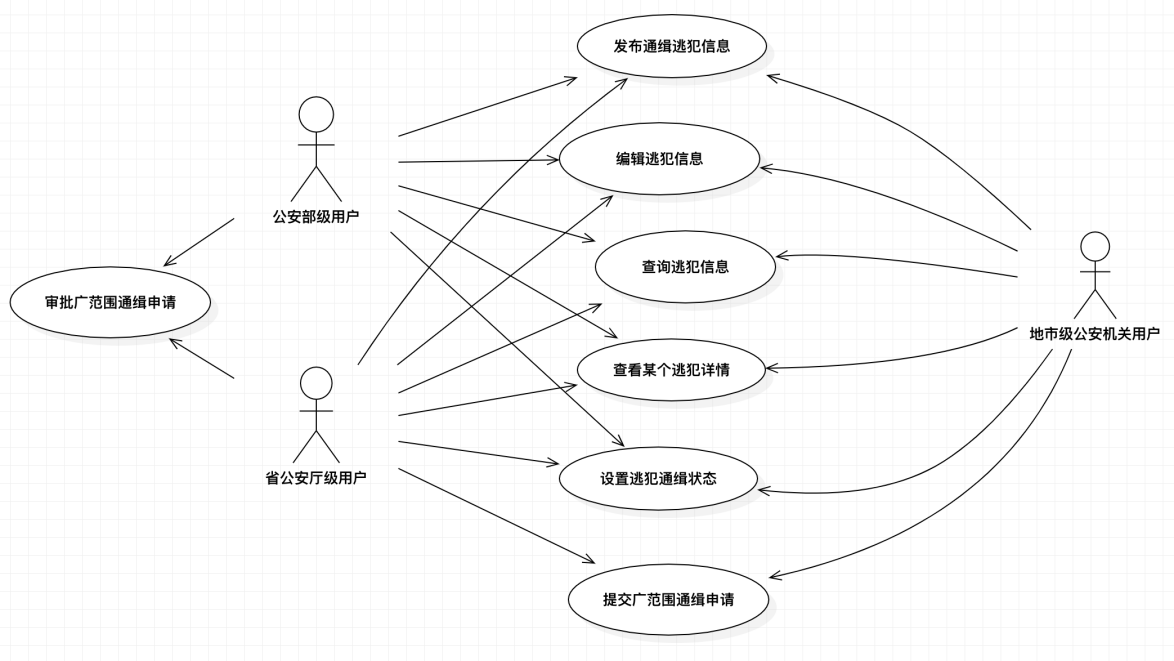
**地市级公安机关用户**：这个概念是从管理范围来讲，所属部门的管辖的行政区域为地市级公安单位。此级别的用户可以在本区域内发布通缉令，可以查看通缉范围大于等于本区域的所有通缉令以及相关逃犯信息。

**省公安厅级用户**：此类用户所属部门的管辖区域为省级，管理员具有审核下级所属部门发起的广范围通缉请求的权限。

**公安部级用户**：此类用户数量应该很少，此级别用户的所属部门“公安部”为系统内置，系统应内置一个此级别的管理员用户，相当于超级管理员。具有操作系统内所有数据和所有功能的权限。

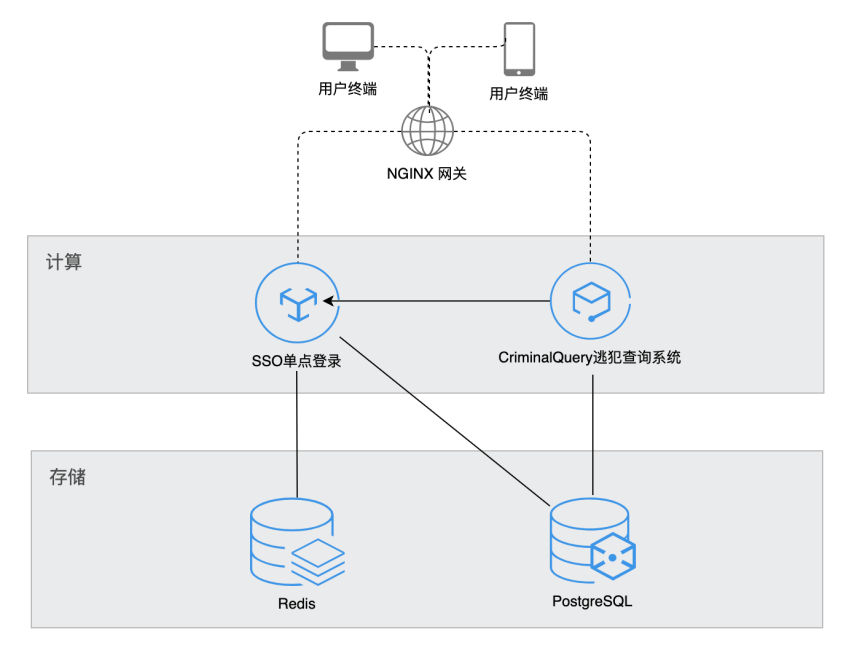
* + 1. 用例分析

图 4.2 逃犯查询系统用例图



* + 1. 总体结构

图 4.3系统架构图



* + 1. 模块及功能
       1. 单点登录服务

单点登录系统作为系统中最重要模块之一，采用微服务的方式设计，主要提供用户单点登录功能和内部服务API保护功能。

工作方式：RESTful API

通讯协议：HTTP 1.1

数据序列化格式：JSON

功能：

1. 登录：客户端要访问系统的功能，首先要与单点登录服务进行身份校验，校验方式使用传统的用户名和密码方式。用户名和密码校验通过后生成一个**全局唯一且不可枚举**的加密的Token，此Token作为当前登录用户在当前客户端上的令牌，用于随后进行API访问权限校验。登录成功后向客户端返回用户信息和Token，随后客户端访问系统内其他服务有权限控制的API时需要携带此Token信息。
2. 用户Token校验服务：系统内的其他服务如果需要对用户的操作进行鉴权，首先应该读取用户请求中包含的Token信息，然后向单点登录系统询问此Token是否有效；此时单点登录系统对Token进行校验，若Token对应了一个用户登录状态，则向其他服务返回此Token对应的用户信息，若Token无效则通知其他系统此Token无效；是否继续提供服务由各微服务自己决定。
3. 登出：提供一个登出接口，由用户主动使Token失效以保证账号安全。
   * + 1. 逃犯查询服务

逃犯查询系统作为系统第一版本阶段首要提供服务的模块，采用微服务方式设计。主要包括实现逃犯信息的增删查改、公安部门的增删查改、用户管理功能（注册、审批注册、删除）。

工作方式：RESTful API

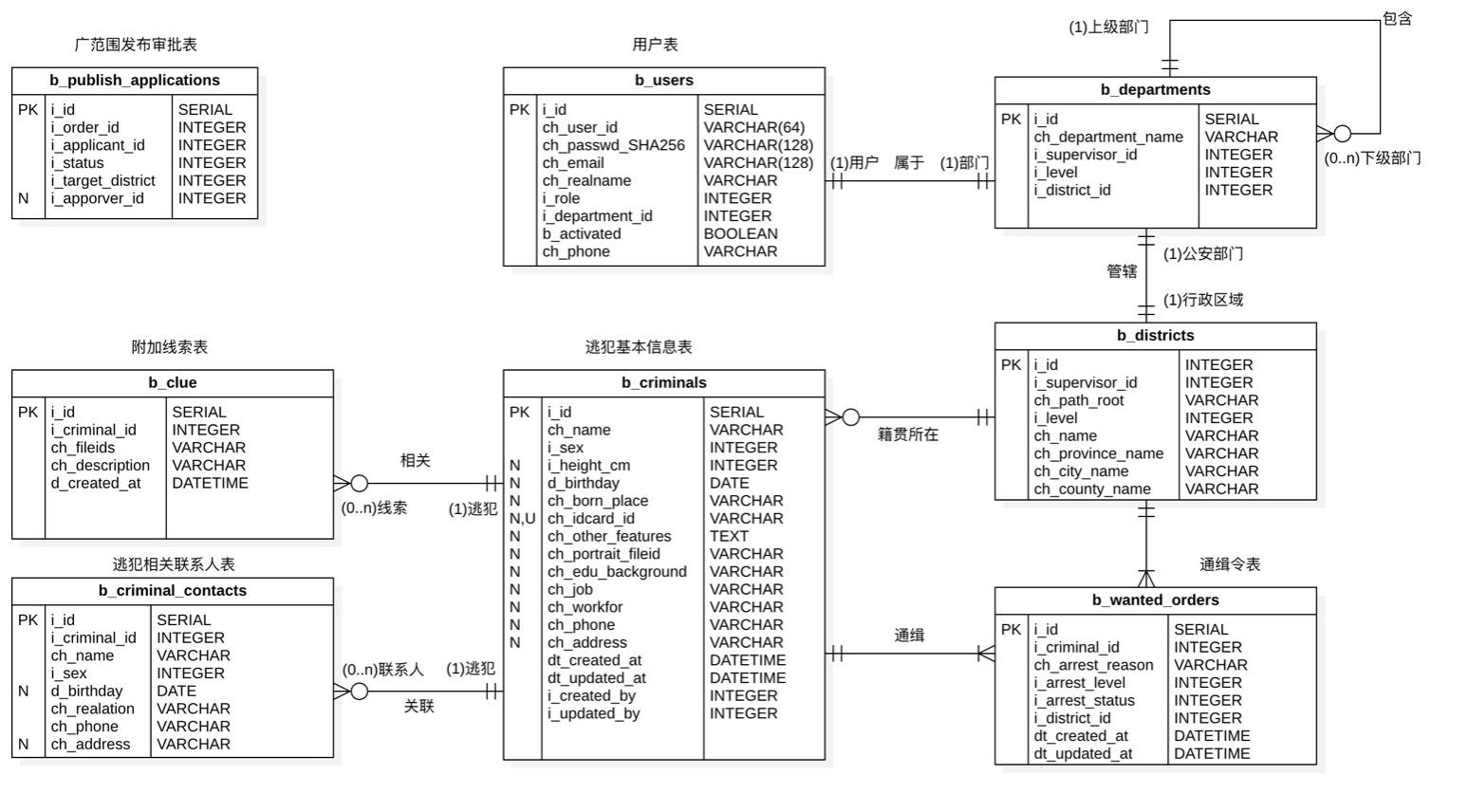
通讯协议：HTTP 1.1

数据序列化格式：JSON

模块/功能：

1. 部门管理模块：
   1. 添加新部门：仅限管理员访问。用户输入部门名称，选择部门管辖区域，设置上级部门后提交创建请求。收到请求后首先向SSO验证Token，通过后判断用户是否管理员。最后向数据库加入新部门数据并返回结果。
   2. 列出部门列表：属于基本服务，在用户注册、创建新部门时都需要列出已有部门列表，不需要管理员权限。
   3. 删除部门：部门管理的基本功能，仅限管理员访问。鉴权过程同添加过程一样。
2. 用户管理模块：
   1. 用户注册：不需要权限校验，不需要登录态。用户填写登录用户名、密码、校验密码、邮箱、电话、真实姓名、申请所属部门、申请角色（管理员/普通用户）等信息，发起创建用户请求。用户管理模块校验用户登录名的唯一性，通过后将用户数据插入用户表，账号状态置为未激活，等待具有权限的管理员进行下一步操作（驳回或通过）。
   2. 列出用户列表：需要管理员权限。用于在用户管理页面查看所有可用的账户。
   3. 删除用户：需要管理员权限。列出用户列表之后可以对账号进行删除操作，敏感操作需要管理员权限。
   4. 按登录名或姓名查询用户：需要管理员权限。为应对用户账号较多的时候管理功能执行困难，增加一个聚合搜索框，可以按照用户姓名或用户登录名进行模糊搜索。
3. 逃犯信息模块：
   1. 添加逃犯信息：需要登录态，不需要管理员权限。在通缉逃犯前需要对逃犯基本信息进行录入，基本信息内容请参考需求分析-数据定义。
   2. 发起通缉/追加通缉：需要登录态，不需要管理员权限。对已添加进系统的逃犯进行发起通缉操作。逃犯与通缉令是一对多关系。
   3. 修改通缉令状态：需要登录态，不需要管理员权限。通缉令具有“正在通缉”和“停止通缉”两个状态，修改通缉令状态功能应该可以切换某条通缉令的状态。
   4. 删除通缉令：需要管理员权限。删除某条通缉令。
   5. 搜索通缉令：需要登录态，不需要管理员权限。逃犯查询系统最基础功能，支持使用逃犯基本信息中的大部分条目以及通缉令中的大部分条目任意进行组合搜索条件进行组合搜索。默认无条件，显示所有结果，支持分页查询。搜索的结果用于展示一个包含逃犯概要信息的表格，若要查看某个逃犯的详细信息（比如线索、联系人、通缉令等），应使用另一个专用的接口。
   6. 添加线索：需要登录态，不需要管理员权限。对于已经存在系统中的逃犯增加线索信息，线索支持上传图片信息和文本描述信息。上传图片后由前端处理图片文件fileID，随后正式提交线索记录时一并将fileID提交并由后端记录在数据库。推荐使用公有云计算服务商提供的对象存储服务。
   7. 添加联系人信息：需要登录态，不需要管理员权限。对已存在系统中的逃犯添加相关联系人信息，逃犯于联系人信息为一对多关系，可以添加多条。
   8. 获取逃犯详细信息：需要登录态，不需要管理员权限。用于获取某个逃犯的所有相关信息，以展示一个关于单个逃犯的聚合页面。返回逃犯的基本信息、此逃犯的所有通缉令信息、此逃犯的所有联系人信息、此逃犯的所有线索信息。后端程序通过SQL 连接查询或多次查询准备所有信息一次性返回。
      1. 数据结构设计
         1. 逻辑结构：

图 4.4系统数据逻辑结构



* + - 1. 物理结构/数据字典：

系统采用关系型数据库PostgreSQL作为主要RDBMS，以下使用PostgreSQL的SQL方言来描述本系统所有数据结构的物理结构，数据字典请参考comments。

CREATE TABLE public.b\_users (

i\_id SERIAL NOT NULL,

ch\_user\_id varchar(64) NOT NULL UNIQUE,

ch\_password\_sha256 varchar(128) NOT NULL,

ch\_email varchar(128) NOT NULL,

ch\_realname varchar NOT NULL,

i\_role integer NOT NULL,

i\_department\_id integer NOT NULL,

b\_activated boolean NOT NULL,

ch\_phone varchar NOT NULL,

PRIMARY KEY (i\_id)

);

COMMENT ON COLUMN public.b\_users.ch\_user\_id

IS '用于用户登录的用户名';

COMMENT ON COLUMN public.b\_users.ch\_password\_sha256

IS 'SHA256加密的密码';

COMMENT ON COLUMN public.b\_users.ch\_email

IS '用户邮箱';

COMMENT ON COLUMN public.b\_users.ch\_realname

IS '用户姓名';

COMMENT ON COLUMN public.b\_users.i\_role

IS '0-普通用户,1-管理员用户,用于区分是否具有编辑权限';

COMMENT ON COLUMN public.b\_users.i\_department\_id

IS '用户所属部门id';

COMMENT ON COLUMN public.b\_users.b\_activated

IS '是否已激活';

COMMENT ON COLUMN public.b\_users.ch\_phone

IS '用户电话';

CREATE TABLE public.b\_departments (

i\_id SERIAL NOT NULL,

ch\_department\_name varchar NOT NULL,

i\_supervisor\_id integer NOT NULL,

i\_level integer NOT NULL,

i\_district\_id integer NOT NULL,

PRIMARY KEY (i\_id)

);

COMMENT ON TABLE public.b\_departments

IS '部门表';

COMMENT ON COLUMN public.b\_departments.i\_id

IS '部门id';

COMMENT ON COLUMN public.b\_departments.ch\_department\_name

IS '部门名称';

COMMENT ON COLUMN public.b\_departments.i\_supervisor\_id

IS '上级部门id';

COMMENT ON COLUMN public.b\_departments.i\_level

IS '部门级别:1-国家级,2-省部级,3-市级别,4-县/区级';

COMMENT ON COLUMN public.b\_departments.i\_district\_id

IS '辖区id';

CREATE TABLE public.b\_criminals (

i\_id SERIAL NOT NULL,

ch\_name varchar NOT NULL,

i\_sex integer NOT NULL,

i\_height\_cm integer,

d\_birthday date,

ch\_born\_place varchar,

ch\_idcard\_id varchar UNIQUE,

ch\_other\_features text,

ch\_portrait\_fileid varchar,

ch\_edu\_background varchar,

ch\_job varchar,

ch\_workfor varchar,

ch\_phone varchar,

ch\_address varchar,

ts\_created\_at timestamp default now() NOT NULL,

ts\_updated\_at timestamp default now() NOT NULL,

i\_created\_by integer NOT NULL,

i\_updated\_by integer NOT NULL,

PRIMARY KEY (i\_id)

);

COMMENT ON COLUMN public.b\_criminals.ch\_name

IS '逃犯姓名';

COMMENT ON COLUMN public.b\_criminals.i\_sex

IS '0-未知,1-男,2-女';

COMMENT ON COLUMN public.b\_criminals.i\_height\_cm

IS '逃犯身高,单位厘米';

COMMENT ON COLUMN public.b\_criminals.d\_birthday

IS '逃犯生日';

COMMENT ON COLUMN public.b\_criminals.ch\_born\_place

IS '籍贯';

COMMENT ON COLUMN public.b\_criminals.ch\_idcard\_id

IS '身份证号';

COMMENT ON COLUMN public.b\_criminals.ch\_other\_features

IS '自然语言描述的逃犯特征';

COMMENT ON COLUMN public.b\_criminals.ch\_portrait\_fileid

IS '逃犯主识别图像的文件id';

COMMENT ON COLUMN public.b\_criminals.ch\_edu\_background

IS '教育背景';

COMMENT ON COLUMN public.b\_criminals.ch\_job

IS '工作种类';

COMMENT ON COLUMN public.b\_criminals.ch\_workfor

IS '工作单位';

COMMENT ON COLUMN public.b\_criminals.ch\_phone

IS '逃犯曾用电话';

COMMENT ON COLUMN public.b\_criminals.ch\_address

IS '住所地址';

COMMENT ON COLUMN public.b\_criminals.ts\_created\_at

IS '逃犯信息入库时间';

COMMENT ON COLUMN public.b\_criminals.ts\_updated\_at

IS '逃犯个人信息更新日期时间';

COMMENT ON COLUMN public.b\_criminals.i\_created\_by

IS '逃犯信息创建人id';

COMMENT ON COLUMN public.b\_criminals.i\_updated\_by

IS '最后一个更新逃犯信息人id';

CREATE TRIGGER t\_updatetime\_criminal BEFORE UPDATE ON b\_criminals

FOR EACH ROW

EXECUTE PROCEDURE update\_on\_modified();

CREATE TABLE public.b\_criminal\_contacts (

i\_id SERIAL NOT NULL,

i\_criminal\_id integer NOT NULL,

ch\_name varchar NOT NULL,

i\_sex integer NOT NULL,

d\_birthday date,

ch\_relation varchar NOT NULL,

ch\_phone varchar NOT NULL,

ch\_address varchar,

PRIMARY KEY (i\_id)

);

COMMENT ON TABLE public.b\_criminal\_contacts

IS '逃犯相关联系人';

COMMENT ON COLUMN public.b\_criminal\_contacts.i\_id

IS '表主键业务无关';

COMMENT ON COLUMN public.b\_criminal\_contacts.i\_criminal\_id

IS '对应逃犯id';

COMMENT ON COLUMN public.b\_criminal\_contacts.ch\_name

IS '联系人姓名';

COMMENT ON COLUMN public.b\_criminal\_contacts.i\_sex

IS '联系人性别';

COMMENT ON COLUMN public.b\_criminal\_contacts.d\_birthday

IS '联系人生日';

COMMENT ON COLUMN public.b\_criminal\_contacts.ch\_relation

IS '联系人与逃犯关系';

COMMENT ON COLUMN public.b\_criminal\_contacts.ch\_phone

IS '联系人电话';

COMMENT ON COLUMN public.b\_criminal\_contacts.ch\_address

IS '联系人地址';

CREATE TABLE public.b\_wanted\_orders (

i\_id SERIAL NOT NULL,

i\_criminal\_id integer NOT NULL,

ch\_arrest\_reason varchar NOT NULL,

i\_arrest\_level integer NOT NULL,

i\_arrest\_status integer NOT NULL,

i\_district\_id integer NOT NULL,

i\_created\_by integer NOT NULL,

ts\_created\_at timestamp default now() NOT NULL,

ts\_updated\_at timestamp default now() NOT NULL,

PRIMARY KEY (i\_id)

);

COMMENT ON TABLE public.b\_wanted\_orders

IS '通缉令表';

COMMENT ON COLUMN public.b\_wanted\_orders.ch\_arrest\_reason

IS '通缉原因,罪行描述';

COMMENT ON COLUMN public.b\_wanted\_orders.i\_arrest\_level

IS '通缉级别';

COMMENT ON COLUMN public.b\_wanted\_orders.i\_arrest\_status

IS '通缉状态:0-停止通缉,1-正在通缉';

COMMENT ON COLUMN public.b\_wanted\_orders.i\_district\_id

IS '统计范围行政区划id';

create index orders\_level\_district\_id\_reason\_index

on b\_wanted\_orders (i\_arrest\_level, i\_district\_id, ch\_arrest\_reason);

CREATE TABLE public.b\_districts (

i\_id integer NOT NULL,

i\_supervisor\_id integer NOT NULL,

ia\_path\_root integer[] NOT NULL,

i\_level integer NOT NULL,

ch\_name varchar NOT NULL,

ch\_province\_name varchar NOT NULL,

ch\_city\_name varchar NOT NULL,

ch\_county\_name varchar NOT NULL,

PRIMARY KEY (i\_id)

);

COMMENT ON TABLE public.b\_districts

IS '行政区划';

COMMENT ON COLUMN public.b\_districts.i\_id

IS '区划id';

COMMENT ON COLUMN public.b\_districts.i\_supervisor\_id

IS '父级区划id';

COMMENT ON COLUMN public.b\_districts.ia\_path\_root

IS '回溯id路径,逗号分隔,从省id开始';

COMMENT ON COLUMN public.b\_districts.i\_level

IS '区划等级:0-国家,1-省/直辖市,2-市,3-县/区';

COMMENT ON COLUMN public.b\_districts.ch\_name

IS '区划名称';

COMMENT ON COLUMN public.b\_districts.ch\_province\_name

IS '所属省或直辖市名称';

COMMENT ON COLUMN public.b\_districts.ch\_city\_name

IS '所属市名称,省级区划为空';

COMMENT ON COLUMN public.b\_districts.ch\_county\_name

IS '所属县区名称,省和市级区划此项为空';

CREATE TABLE public.b\_clue (

i\_id SERIAL NOT NULL,

i\_criminal\_id integer NOT NULL,

ch\_fileids varchar,

ch\_description varchar NOT NULL,

ts\_created\_at timestamp default now() NOT NULL,

PRIMARY KEY (i\_id)

);

COMMENT ON TABLE public.b\_clue

IS '逃犯线索表';

COMMENT ON COLUMN public.b\_clue.i\_id

IS '表主键,业务无关';

COMMENT ON COLUMN public.b\_clue.ch\_fileids

IS '逗号分隔的资源文件id';

COMMENT ON COLUMN public.b\_clue.ch\_description

IS '线索描述';

COMMENT ON COLUMN public.b\_clue.ts\_created\_at

IS '线索上传时间';

CREATE TABLE public.b\_publish\_applications (

i\_id SERIAL NOT NULL,

i\_order\_id integer NOT NULL,

i\_applicant\_id integer NOT NULL,

i\_status integer NOT NULL,

i\_target\_district integer NOT NULL,

i\_apporver\_id integer,

PRIMARY KEY (i\_id)

);

COMMENT ON COLUMN public.b\_publish\_applications.i\_id

IS '表主键业务无关';

COMMENT ON COLUMN public.b\_publish\_applications.i\_applicant\_id

IS '申请者id';

COMMENT ON COLUMN public.b\_publish\_applications.i\_status

IS '审批状态:0-待审批,1-审批通过,2-拒绝发布';

COMMENT ON COLUMN public.b\_publish\_applications.i\_target\_district

IS '目标通缉区域id';

COMMENT ON COLUMN public.b\_publish\_applications.i\_apporver\_id

IS '审批人id';

1. 系统详细设计
   1. 引言
      1. 编写目的

逃犯查询系统详细设计说明书将详细描述概要设计中指定的所有已确定设计方案的具体实现；本说明书时本系统实际开发人员编写的主要开发文档。

* + 1. 读者对象

逃犯查询系统详细设计读者对象为所以与本系统开发工作直接相关的人员（包括所有开发小组成员、系统实施与部署人员、未来对本系统进行修改和维护的人员）以及需要关注本系统的开发与实施过程的所有人员。

* + 1. 本文档的结构[[7]](#footnote-7)

本文档将按照与软件架构方式相似的结构进行组织；基本单位是每一个功能的实现方案（包括功能/目标、输入输出标准、接口、关键算法描述、程序逻辑、关键数据库操作），描述的顺序先给出后端实现细节，最后给出前端实现细节，其中后端实现细节又按照划分的微服务分为单点登录系统实现细节和逃犯查询系统实现细节，以此类推。

* 1. 接口规范约定

除特别说明外，系统所有接口返回数据都是JSON格式，且具有统一的结构样式，包括状态、返回数据、提示信息三个数据域。下文中若无特别说明，返回数据都是本模版的data数据域的内容。

* + 1. 全局响应数据示例

{

"status": 200,

"data": {} | [],

"msg": "成功"

}

* + 1. 状态码定义

200：接口工作正常

400：用户原因导致错误

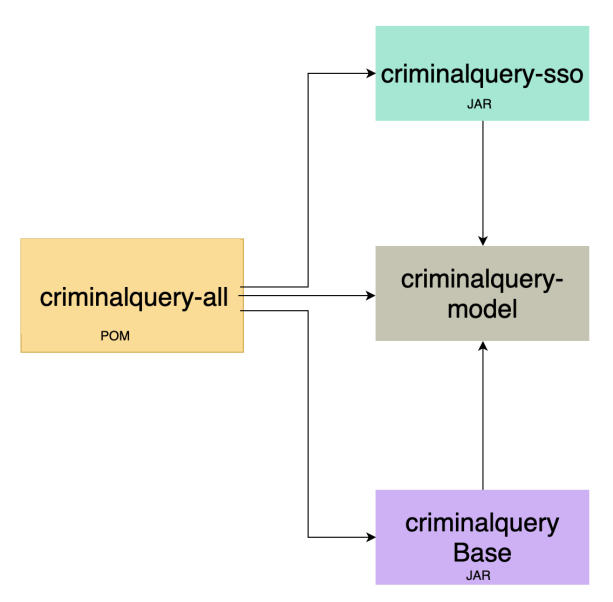
500：服务端异常导致错误

* 1. 后端系统实现技术方案
     1. 项目结构说明
        1. 项目切分

后端系统使用Maven做项目模块和依赖关系管理，首先将系统内的领域数据模型全部提出到一个单独的包中，独立成为一个Maven模块，所有其他服务都引用此模块，以此来保证各种POJO在各微服务和模块之间都是一致的。

另外将单点登录系统和逃犯查询系统两个微服务分别作为Maven模块，分别开发；两个微服务都使用SpringBoot。

图 5.1 Maven项目模块结构图



* + 1. SpringMVC开发RESTful API说明

SpringMVC最经典的用法是遵循Model-View-Controller的方式来在后端将数据直接渲染净html模版引擎，开发RESTful API的最明显差别就是不需要在后端进行html页面的数据渲染。SpringMVC提供了几个注解来直接将Controller中的方法返回值转换为响应体，而不是用返回值代表视图名称。

@RestController：在Controller类上使用此注解将此Controller标识为所有方法都提供RESTful API，在此类中所有@RequestMapping修饰的方法返回值都将被转换为JSON数据结构并作为响应体返回给客户端或浏览器。

@ResponseBody：如果不想一个Controller中的所有方法都变成RESTful API，可以仅在需要此功能的方法上添加@ResponseBody注解来实现直接返回响应体。

* + 1. 逃犯查询系统API权限校验方式
       1. 概述

逃犯查询服务中大部分的接口都需要用户登录态，对大量的API进行权限控制是个非常重复性的工作，若在每个接口内部都单独实现身份校验功能无疑是浪费时间和精力的。

SpringMVC提供了拦截器interceptor机制，来对匹配了指定URL模式的请求进行拦截，把登录态校验的逻辑提升到拦截器中可实现高效的代码复用和API权限管理。

* + - 1. 实现方式

图5.2逃犯查询服务API前置鉴权逻辑



* + 1. 单点登录系统实现技术方案
       1. 登录

功能：校验用户登录id和密码，发放临时身份标识Token。

接口：POST /sso/login

输入（http请求）：Content-Type:application/json

{

"userID":<string, required, 用户登录ID>,

"password": <string, required, 用户密码>,

"remember": <boolean, optional, 是否延长登录态有效期>

}

输出（http响应）：Content-Type:application/json

{

"status": 200,

"data": {

"user": {

"id": 1,

"userID": "admin",

"email": "linkinghack@outlook.com",

……

},

"token": "JDEkQjVSY3Z6Z0YkM1k4UnJVdHJjQWx5Rm9ZVGIxN3ZxLg=="

},

"msg": "成功"

}

关键算法1：系统中保存的用户密码并非明文，为保护用户账号安全，密码使用SHA3-256[[8]](#footnote-8)强加密算法计算哈希值后保存。故登录过程需要再次计算用户密码SHA3-256哈希值后再与系统记录值进行比对。

关键算法2：计算Token，使用userID + UUID 计算MD5值。

处理逻辑：参考图5.4所示流程图，其中返回用户数据时已将密码删除。

* + - 1. 用户Token校验

功能：为系统内其他微服务提供基于Token令牌的用户身份校验功能。

接口：GET /sso/auth/{token}

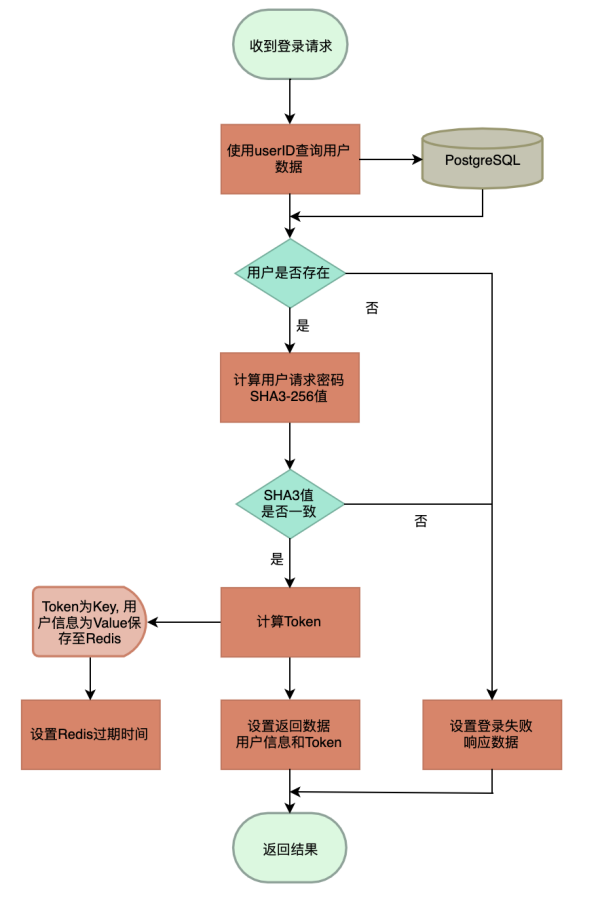
输入：token:<string, required, SSO系统下发的token令牌>

输出：JSON格式的用户数据

图5.3 Token校验返回数据示例

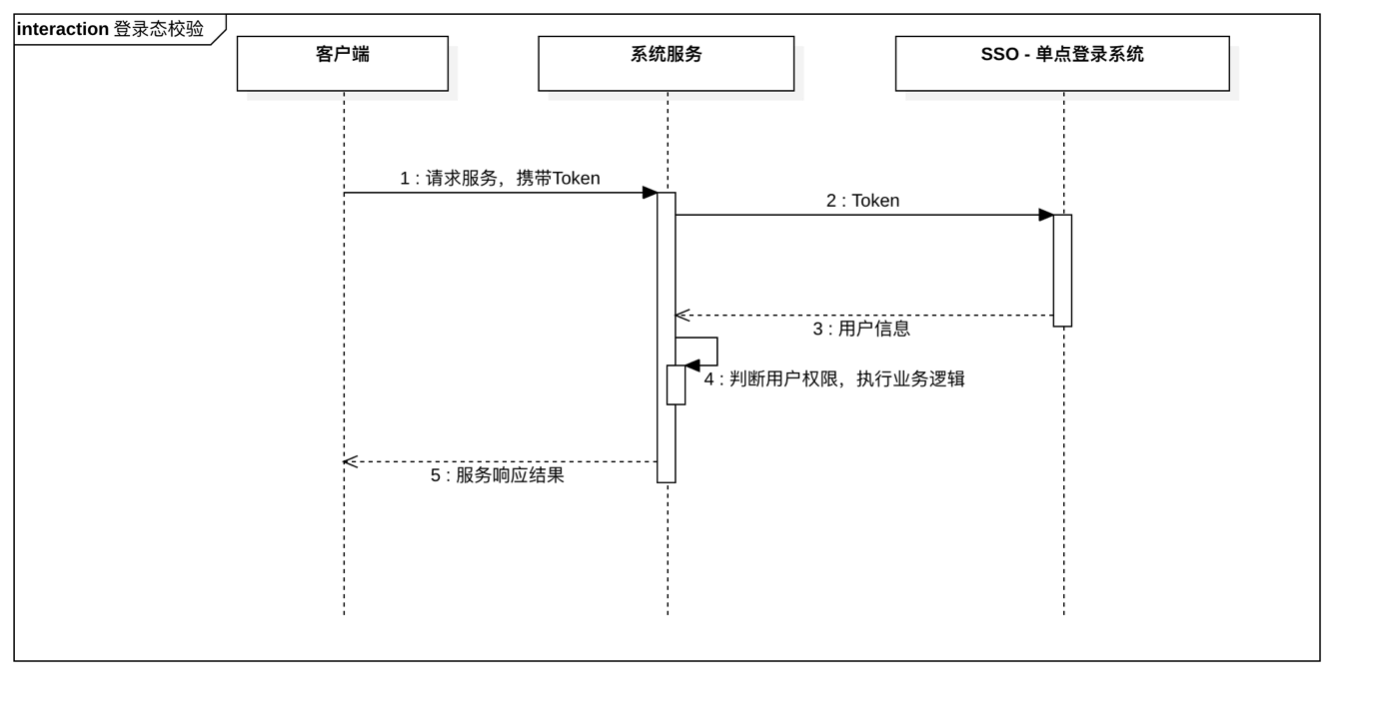


图 5.4 登录过程流程图



处理逻辑：

图 5.5 Token校验时序图



说明：这里的URL参数使用了SpringMVC提供的URL参数解析，在Controller中使用@PathVariable注解即可获得URL参数内容并自动完成数据类型转换（比如有时需要获取整数类型参数）。

实现：

图 5.6 Token校验服务Service层逻辑



图 5.7 用户密码SHA3哈希值计算与比对方式



图5.8 Token校验服务Controller层



* + - 1. 登出

功能：用于用户主动使登录态失效，在此过程中从Redis删除Token对应的信息即可。

* + 1. 构建全国行政区划数据库

目标：建立全国三级行政区划数据库，用于部门模块设置部门管辖区域以及逃犯信息模块设置通缉区域等操作。

功能：提供访问某一个区划级别所有地区列表的接口，提供某个高级行政区下直属子行政区域列表。

接口：

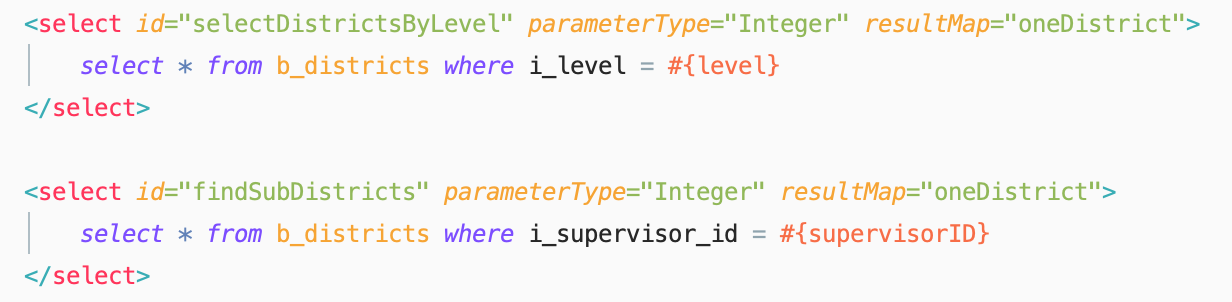
GET /districts/subDistricts/{id} 获取某个行政区划的所有直接子区域

GET /districts/byLevel/{level} 获取某个区划级别的所有区域

关键算法：由于行政区域是具有父子结构关系的树形数据结构，使用类似于并查集的方式来保存和组织数据，在子节点中保存父节点的id来实现数结构关系的记录。

实现方式：每个功能关键部分仅需一条SQL即可实现，以下给出对应MyBatis的Mapper内容。

图 5.9 行政区划服务主要SQL



构建数据库方式：通过搜索引擎查找后从某个公众号下载到了2019年3月最新全国行政区划数据库，数据格式为xlsx，不能直接供本系统使用。此处使用了一小段Golang程序来将xlsx文件中的数据导入到系统数据库。关键代码如下：

*// Prepare PostgreSQL client.*

    connStr := `dbname=postgres user=postgres password=\*6Z,<[9A97Mp;6W sslmode=disable`

    pgdb, err := sql.**Open**("postgres", connStr)

    if err != nil {

        log.**Fatal**("Connot connect to PostgreSQL. err = %v", err)

    }

    txn, err := pgdb.**Begin**()

    if err != nil {

        log.**Fatal**(err)

    }

    stmt, err := txn.**Prepare**(pq.**CopyIn**("b\_districts", "i\_id", "i\_supervisor\_id", "ia\_path\_root", "i\_level", "ch\_name", "ch\_province\_name", "ch\_city\_name", "ch\_county\_name"))

    if err != nil {

        log.**Fatal**(err)

    }

*// 读取xlsx并存入两数据库*

    for \_, sheet := range xlFile.Sheets {

        for i, row := range sheet.Rows {

            if i == 0 { *// 跳过表头*

                continue

            }

            cells := row.Cells

            id, \_ := strconv.**ParseInt**(cells[0].**String**(), 10, 64)

            super, \_ := strconv.**ParseInt**(cells[1].**String**(), 10, 64)

            level, \_ := strconv.**ParseInt**(cells[3].**String**(), 10, 64)

            path := []int64{}

            for \_, v := range strings.**Split**(cells[2].**String**(), ",") {

                value, \_ := strconv.**ParseInt**(v, 10, 64)

                path = **append**(path, value)

            }

            tmpDistrict := District{

                ID: id,

                SupervisorID: super,

                Path: path,

                Name: cells[4].**String**(),

                Level: level,

                Province: cells[7].**String**(),

                City: cells[10].**String**(),

                County: cells[13].**String**(),

            }

            log.**Println**("Row data parsed: ", tmpDistrict)

            districts = **append**(districts, &tmpDistrict)

            \_, err := stmt.**Exec**(tmpDistrict.ID, tmpDistrict.SupervisorID, fmt.**Sprintf**("{%s}", cells[2].**String**()), tmpDistrict.Level, tmpDistrict.Name, tmpDistrict.Province, tmpDistrict.City, tmpDistrict.County)

            if err != nil {

                log.**Println**("One row failed. err = %v", err)

            }

        }

    }

* + 1. 构建文件存储服务
       1. 概述

由于系统中需要提供上传和展示逃犯照片以及线索相关图片的功能，而图片的存储和传输需要稳定的存储机制和较高的带宽资源。通常应用服务器是不具备这些条件的，文件存储应该有专用的服务。目前公有云计算服务商都提供了对象存储服务，本系统将使用阿里云提供的OSS服务来保存和展示图片。

存在OSS上的文件都需要有一个ObjectID来唯一标识文件，在本系统中此ObjectID也叫FileID，由文件存储服务在上传OSS之前生成，上传成功后返回给前端程序；前端程序继续进行原有业务，提交表单时文件一项实际上只是一个FileID字符串。FileID保存在逃犯相关的数据表中。

需要在页面显示某图片时需要使用FileID向文件存储服务兑换临时URL，且此操作需要登录态，如此便可保护系统文件数据的安全。

* + - 1. 接口设计

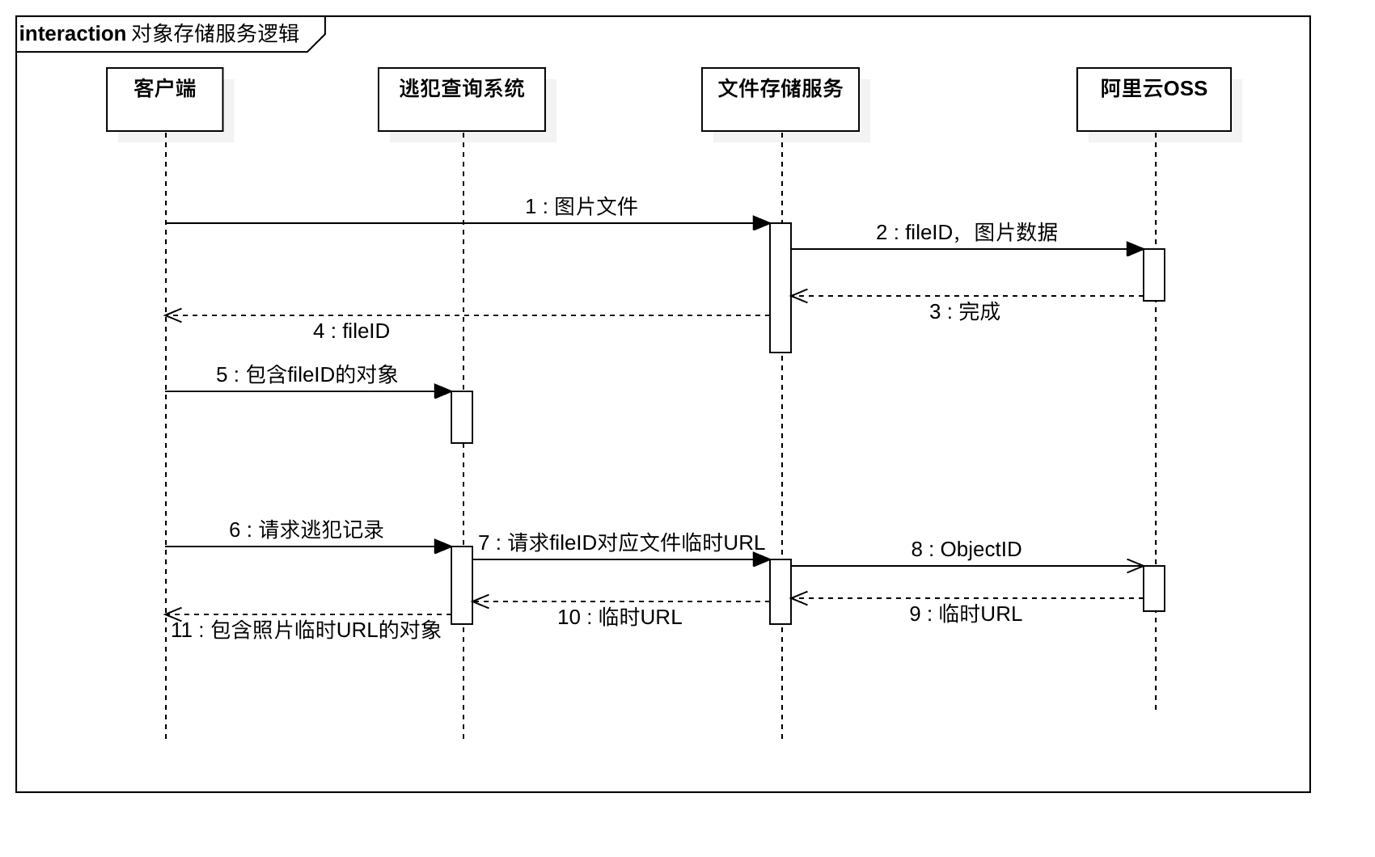
POST /oss/file 上传文件，接受文件参数file，返回FileID。

GET /oss/file/url/{fileID} 获取文件临时URL，需要登录态。返回文件临时URL。

DELETE /oss/file/{fileID} 删除文件，需要登录态。

* + - 1. 整合OSS服务后的图片存储服务逻辑

图 5.10文件存储服务时序图



* + 1. 部门管理模块实现技术方案
       1. 新建部门

功能：增加新的部门记录。

接口：POST /departments/department

参数：

String departmentName 部门名称  
Integer supervisorID 上级部门ID   
Integer districtID 管辖区域行政区划ID

* + - 1. 列出部门列表

功能：列出所有已存在的部门列表，其中包含父级部门id，以便前端按树形结构展示。

接口：GET /departments/tree/nodes 需要登录态

* + - 1. 删除部门

功能：部门管理的基本数据操作，支持删除一个部门，仅限管理员操作。

接口：DELETE /departments/department/{ID}

* + 1. 用户管理模块实现技术方案
       1. 注册

功能：实现新用户注册，注册加入用户信息表后默认置为未激活状态。此接口不需要登录态。

接口：POST /user/register

输入（HTTP请求）：

{

"userID": <string, reqiured, 用户登录ID>,

"password": <string, reqiured, 用户密码>,

"confirm": <string, required, 确认密码>,

"email": <string, required, 用户邮箱>,

"realName": <string, required, 用户真是姓名>,

"role": <integer, required, 用户申请成为角色[1-管理员,0-普通用户]>

"departmentID": <integer, required, 用户所属部门id>,

"phone": <string, required, 用户电话号>

}

输出：注册成功时返回成功提示

实现：给出关键SQL（MyBatis），详细实现代码请参考源代码附件。

<insert *id*="createUser" *parameterType*="com.linkinghack.criminalquerymodel.data\_model.User" *useGeneratedKeys*="true">  
 insert into b\_users(ch\_user\_id, ch\_password\_sha256, ch\_email, ch\_realname, i\_role, i\_department\_id, b\_activated, ch\_phone)  
 values ( #{userID}, #{password}, #{email}, #{realName}, #{role}, #{departmentID}, #{activated}, #{phone} )  
</insert>

* + - 1. 获取所属部门

接口：GET /user/ofDepartment/{UID}

参数：URL 参数UID，代表用户主键ID（非用户登录ID）

实现：此处给出MyBatis SQL

<select *id*="getUserDepartment" *parameterType*="Integer" *resultMap*="com.linkinghack.criminalquerybase.dao.mapper.DepartmentMapper.department">  
 select b\_departments.i\_id as did, ch\_department\_name, i\_supervisor\_id, i\_level, i\_district\_id  
 from b\_departments left join b\_users on b\_users.i\_department\_id = b\_departments.i\_id  
 where b\_users.i\_id = #{UID}  
</select>

* + - 1. 获取用户列表

功能：返回用户列表，需要管理员权限；提供一个可选参数，用于根据用户姓名或登录ID进行筛选结果，支持分页。此接口在用户管理页面和新用户审批页面使用。

接口： GET /user/activated 已激活用户列表

GET /user/inactivated 未激活用户列表

参数：搜索条件 {nameOrID:<string>, pageSize:Int, page:Int(start from 1)} 可选,模糊搜索用户登录用户名

参数说明：Content-Type：application/x-www-form-urlencoded  
使用标准的HTML表单方式传递参数，无需以JSON数据格式提交。

实现方式：利用MyBatis的动态SQL技术实现参数可选。使用MyBatis Mapper配置文件中支持的<if>标签，通过test属性来探测是否具有筛选条件，以实现可选的参数。

<select *id*="users" *resultMap*="user">  
 select users.i\_id as uid, ch\_user\_id, ch\_password\_sha256, ch\_email, ch\_realname, i\_role, i\_department\_id, b\_activated, ch\_phone,  
 department.i\_id as did, ch\_department\_name, i\_supervisor\_id, i\_level, i\_district\_id  
 from b\_users as users left join b\_departments as department on users.i\_department\_id = department.i\_id  
 where b\_activated = true  
 <if *test*="nameOrID != null"> AND ch\_user\_id like #{nameOrID} or ch\_realName like #{nameOrID}</if>  
 order by ts\_created\_at  
 <if *test*="pageSize != null">limit #{pageSize}</if>  
 <if *test*="offset != null">offset #{offset}</if>  
</select>

* + 1. 逃犯信息模块实现技术方案
       1. 逃犯信息组合查询

功能：提供一个支持多条件组合搜索的接口，所有条件都可选；条件覆盖逃犯基本信息表和通缉令表中的关键字段，字符串类型使用模糊匹配，整数、日期类型使用上下边界值来规定范围；支持分页查询。

接口：GET /criminal/criminals

参数：对应Model项目中的SearchCriminalRequest类定义

*// 逃犯基本信息表相关条件*String name; 逃犯姓名  
Integer sex; 逃犯性别  
Integer heightStart; 身高下限  
Integer heightEnd; 身高上限  
Integer ageStart; 年龄下限  
Integer ageEnd; 年龄上限  
String phone; 电话号  
String eduBackground; 教育背景  
String job; 工作类型  
String workFor; 工作单位  
String address; 居住地址  
String otherFeatures; 其他特征  
  
*// 通缉令表相关条件*String arrestReason; *//通缉原因*Integer arrestLevel; 通缉级别  
Integer arrestStatus; 通缉令状态  
Integer districtID; 通缉区域 LocalDateTime arrestCreateTimeStart; 通缉时间起始  
LocalDateTime arrestCreateTimeEnd; 通缉时间结束  
Integer pageSize; 分页-页大小  
Integer page; 分页-页码（不是offset）  
Integer offset; 分页-offset，不需要前端传递，由后端程序从page算出

实现：

Service层：

*/\*\*  
 \* 多条件模糊查找逃犯  
 \* pageSize默认为10, page传递页码从0开始，将自动转换为offset  
 \** ***@param search*** *搜索条件  
 \** ***@return*** *搜索结果列表  
 \*/  
public* UniversalResponse searchCriminals(SearchCriminalRequest search) {  
 HashMap<String, Object> result = *new* HashMap<>();  
 *// 处理日期条件前端传递不包含时间导致不能搜索精确一天的问题  
 if* (search.getArrestCreateTimeEnd() != *null*){  
 search.setArrestCreateTimeEnd(search.getArrestCreateTimeStart().plusHours(23).plusMinutes(59).plusMinutes(59));  
 }  
 Integer totalCount = mapper.searchResultCount(search);  
 *// 默认页大小  
 if* (search.getPageSize() == *null*)  
 search.setPageSize(Constants.*DefaultPageSize*);  
 *// page 转换为offSet  
 if* (search.getPage() == *null*)  
 search.setPage(0); *// 默认第一页  
 else* search.setPage(search.getPage() - 1); *// 前端返回页码从1开始  
 // 计算offset* search.setOffset(search.getPage() \* search.getPageSize());  
  
 List<Criminal> criminals = mapper.searchCriminals(search);  
 *// 获取照片临时url  
 if* (search.getSyncLoadPortraitURL()) {  
 *for* (Criminal criminal : criminals) {  
 *if* (criminal.getPortraitFileID() != *null*)  
criminal.setPortraitFileURL( fileService.getTempraryURL(criminal.getPortraitFileID(), *null*));  
 }  
 }  
 result.put("totalCount", totalCount);  
 result.put("currentPage", search.getPage());  
 result.put("criminals", criminals);  
 *return* UniversalResponse.Ok(result);  
}

Controller层部分代码：

*/\*\*  
 \* <GET>[/criminal/criminals]  
 \* 模糊查找：按逃犯查询的多条件高级检索方法  
 \** ***@param searchCriminalRequest*** *检索条件  
 \** ***@param request*** *用于获取当前用户  
 \** ***@return*** *{totalCount: 123, criminals: [{},{},...]}  
 \*/*@GetMapping("/criminals")  
*public* UniversalResponse criminalInfo(SearchCriminalRequest searchCriminalRequest,  
 HttpServletRequest request) {  
 String fn = "<GET>[/criminal/criminals]";  
 User user = (User) request.getAttribute("user");  
 logger.info("@Request{} request:{}, User:{}", fn, searchCriminalRequest, user);  
 *// 是否同步加载逃犯主照片URl 的默认值处理  
 if* (searchCriminalRequest.getSyncLoadPortraitURL() == *null*) {  
 searchCriminalRequest.setSyncLoadPortraitURL(*false*);  
 }  
 *// 处理空字符串  
 if* (searchCriminalRequest.getName()!=*null* && searchCriminalRequest.getName().length() < 1)  
 searchCriminalRequest.setName(*null*);  
 *if* (searchCriminalRequest.getAddress()!=*null* && searchCriminalRequest.getAddress().length() < 1)  
 searchCriminalRequest.setAddress(*null*);  
*// 此处省略多条判读空字符串语句*  
 UniversalResponse response = criminalService.searchCriminals(searchCriminalRequest);  
 logger.info("@Response{} response:{}", fn, response);  
 *return* response;  
}

SQL：省略了部分字符串类型参数模糊匹配的条件

<select *id*="searchCriminals" *resultMap*="criminalInfo"  
 *parameterType*="com.linkinghack.criminalquerymodel.transfer\_model.SearchCriminalRequest">  
 select bcri.i\_id as i\_id, ch\_name, i\_sex, i\_height\_cm, d\_birthday,( CURRENT\_DATE - d\_birthday)/365 as age ,  
 ch\_born\_place, ch\_idcard\_id, ch\_other\_features, ch\_portrait\_fileid, ch\_edu\_background, ch\_job, ch\_workfor,  
 ch\_phone, ch\_address, bcri.ts\_created\_at as ts\_created\_at  
 from b\_criminals as bcri  
 left join b\_wanted\_orders as bwant on bcri.i\_id = bwant.i\_criminal\_id  
 <where>  
 <if *test*="name != null">ch\_name like #{name}</if>  
 <if *test*="sex != null">AND i\_sex = #{sex}</if>  
 <if *test*="heightStart != null">AND i\_height\_cm &gt;= #{heightStart}</if>  
 <if *test*="heightEnd != null">AND i\_height\_cm &lt;= #{heightEnd}</if>  
 <if *test*="ageStart != null">AND ( CURRENT\_DATE - d\_birthday)/365 &gt;= #{ageStart}</if>  
 <if *test*="ageEnd != null">AND ( CURRENT\_DATE - d\_birthday)/365 &lt;= #{ageEnd}</if>  
 <if *test*="job != null">AND ch\_job like #{job}</if>  
  
 <if *test*="arrestReason != null">AND bwant.ch\_arrest\_reason like #{arrestReason}</if>  
 <if *test*="arrestLevel != null">AND bwant.i\_arrest\_level = #{arrestLevel}</if>  
 <if *test*="districtID != null">bwant.i\_district\_id in (select ia\_path\_root from b\_districts where i\_id =  
 #{districtID})  
 </if>  
 <if *test*="arrestCreateTimeStart != null">AND bwant.ts\_created\_at &gt;= #{arrestCreateTimeStart}</if>  
 <if *test*="arrestCreateTimeEnd != null">AND bwant.ts\_created\_at &lt;= #{arrestCreateTimeEnd}</if>  
 </where>  
 group by bcri.i\_id  
 order by bcri.ts\_created\_at desc  
 <if *test*="pageSize != null">limit #{pageSize}</if>  
 <if *test*="offset != null">offset #{offset}</if>  
</select>

解决大于号、小于号在XML文件中和标签标识冲突的问题，使用xml转义写法：“&gt;” 代表 “>” ， “&lt;” 代表 “<”。

* 1. 前端实现技术方案
     1. 前端技术概述

目前前端程序开发有三个主流的MVVM框架，一是Google开源的Angular，二是Facebook开源的React，三是一位中国开发者主要贡献的Vue。Vue也是当前目前GitHub上Star数量最多的开源框架项目，本系统中将使用Vue开发用户界面。

* + - 1. 浏览器里的客户端

在过去几年开发web应用习惯上称为B/S应用架构，主要是因为客户端用户界面运行在浏览器中，其次把它和C/S架构区分开的原因是业务逻辑大部分都在后端实现，浏览器仅有渲染HTML这一个任务。目前的开发风格是要充分利用浏览器中的JavaScript引擎来实现部分业务逻辑，把浏览器当做一个功能健全的客户端，同时兼备了网页在浏览器中运行的零成本特性。

如此进行应用开发就需要前端有成熟的应用开发框架的支撑；Vue框架把复杂繁冗的DOM操作封装起来，抽象出组件的概念，并且提供了“双向数据绑定”机制，将页面上的显示内容与JS程序中的数据变量对应起来，自动更新，需要数据动态变化的部分仅需要更新JS中相关的数据变量页面上显示的内容就会同步变更；反向的数据绑定比如表单中的数据，也可以实时的映射到JS程序变量中。

使用MVVM框架开发应用也被称为SPA（Single Page Application）单页应用，因为前端应用编译完成后只有一个index.html页面，后来的内容全都是JS程序动态加载的，在使用过程中也几乎不会有刷新页面重渲染动作。

事实上微信小程序也是受到Vue框架的启发，小程序的开发与Vue非常类似，这也是本系统具有非常好的易扩展性的原因，不需要后端程序作出修改，只要拿到后端技术文档，就可以开发另一种客户端来支持更多的设备访问系统。

* + - 1. 与服务端通信：HTTP协议和HTTP Client

前后端程序的开发完全分离后前后端的通信协议仍然是HTTP；不同之处在于之前用户每做一个操作都需要一个页面跳转，发起HTTP请求的动作是由浏览器自动完成的，接收到的HTTP响应是HTML内容由浏览器直接渲染，而现在与服务端程序通信的动作是使用浏览器JavaScript环境中的HTTP Client -- XMLHttpRequest主动发起的。返回的数据正如上文后端技术文档描述，都是JSON格式，这可以在JavaScript中直接转换为JS对象使用。

* + 1. Vue插件和第三方工具
       1. Vuex 全局状态管理

在本系统开发中，用户登录后需要保存用户登录态Token和用户姓名等信息，这写信息在前端程序中多处都会用到，而且若发生更新需要同步通知所有调用点。Vuex提供了全局状态管理功能，将需要全局同步的数据保存在Vuex对象里即可。

* + - 1. Vue-Router页面路由

使用Vue框架后不会出现全页面重新渲染的情况了，但是为用户提供的功能依然是分开几个页面的，前端页面的跳转需要一个路由引擎，Vue-Router解决了单页Vue应用的页面跳转问题。

* + - 1. Axios

在浏览器中使用JavaScript发起HTTP请求有很多种方式，比如使用浏览器原生XMLHttpRequest对象，或者使用支持HTML5的浏览器提供的fetch API。在本系统中将使用一块开源第三方HTTP Client工具Axios，此工具支持Promise来方便的处理异步的网络请求结果，且提供了发起GET、POST、DELETE、PUT等请求的同名方法，是开发变得更容易。

* + - 1. Ant Design

Ant Design是阿里巴巴开源的一款UI组件库，包含了常用的用户界面元素，比菜单栏、表格、列表、上传组件、表单等等；它适配了React和Vue，完全可以满足本项目需求。

* + 1. 前端程序主体结构
       1. 仅仅11行的HTML页面结构

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width,initial-scale=1.0">

<title>criminalquery-app</title>

</head>

<body>

<div id="app"></div>

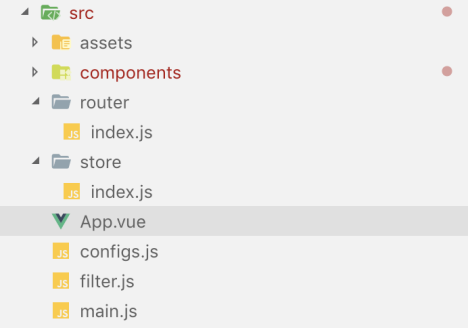
<!-- built files will be auto injected -->

</body>

</html>

在此页面结构中，具有id=“app”属性的<div>标签就是前端应用程序要渲染的位置，由此也可看出单页应用几乎所有内容都是动态生成的。

图 5.11 前端项目结构



图中components目录下存放的就是Vue组件，页面上的各种元素都可以单独开发成一个小组件，以重复使用。

* + - 1. 唯一的Vue对象实例

图 5.12 全局唯一Vue对象实例



前端所有的UI元素都在上图所示的component：App 组件中开发。

* + 1. 页面样式

图 5.13 逃犯查询页面设计

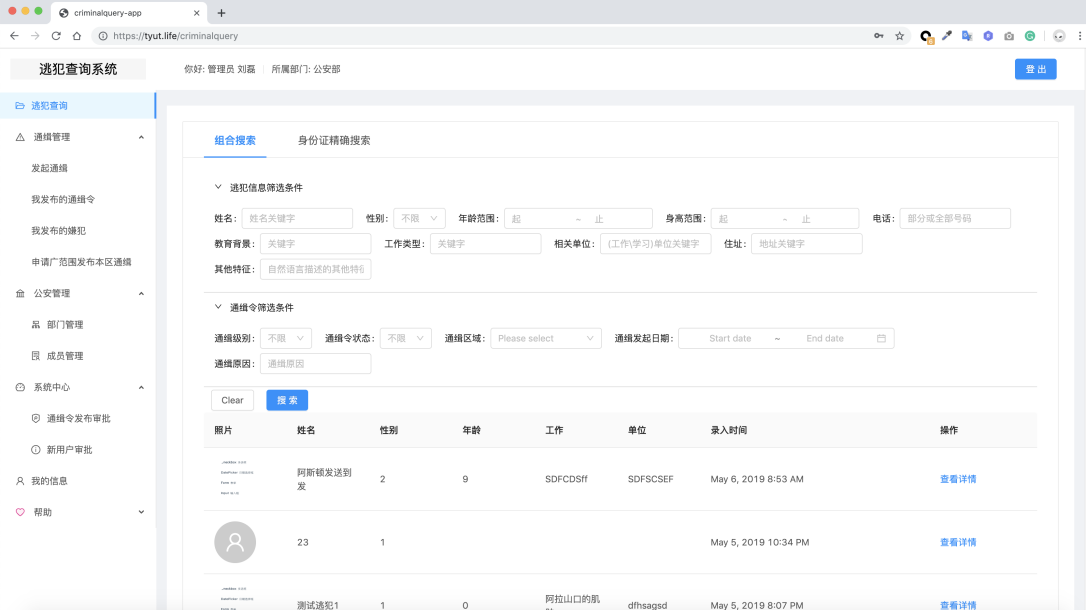
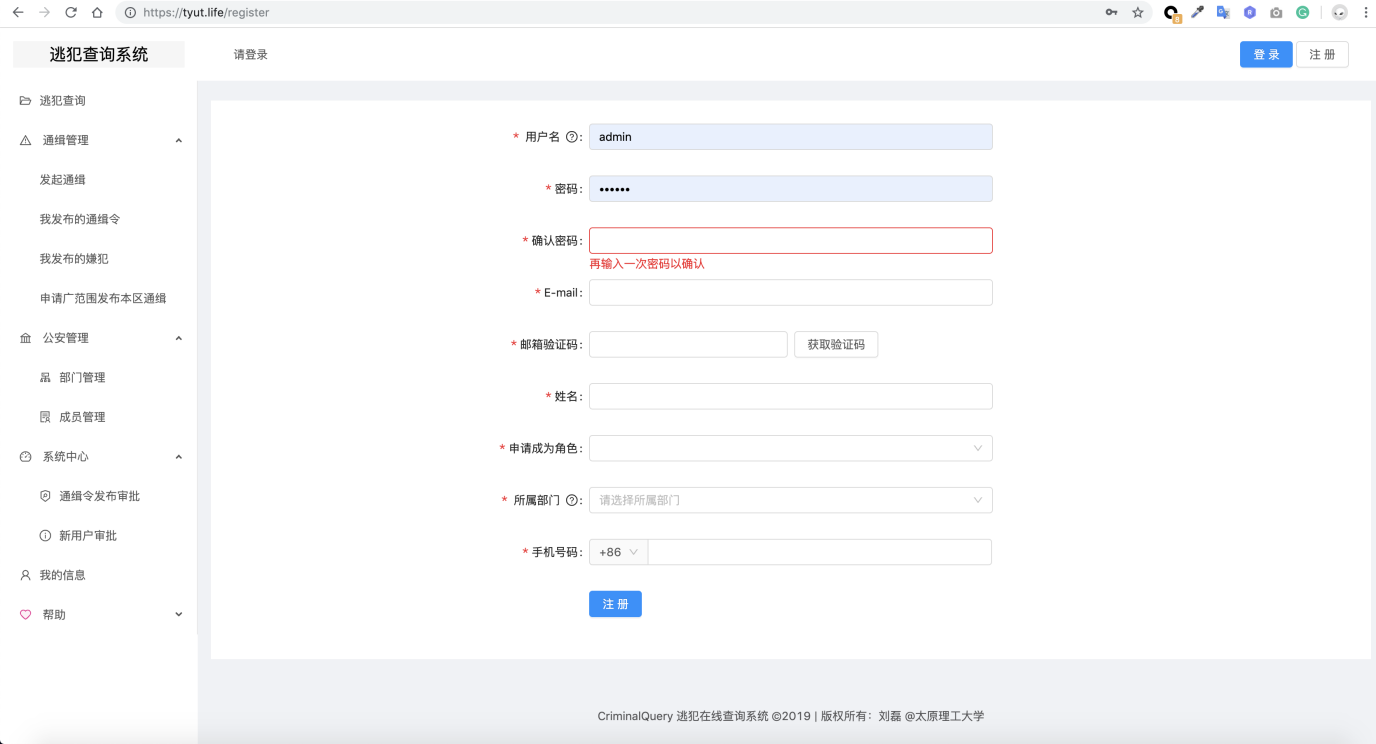


图5.14用户注册页面设计



* + 1. 部门选择器组件

部门选择器组件是个典型的树形选择器，其中的数据内容具有树形数据结构，本小组件使用后端提供的部门列表API来异步加载部门列表数据，在前端依赖AntDesign的树形选择组件显示为树状结构。

图5.15部门选择器组件



* + - 1. 部门选择器核心代码

<template>

<a-tree-select

*style*="width: 100%"

:*dropdownStyle*="{ maxHeight: '400px', overflow: 'auto' }"

*treeDataSimpleMode*

:*treeData*="treeData"

:*placeholder*="placeHolder"

@*change*="departmentSelected"

></a-tree-select>

</template>

<script>

import appConfigs from "../configs";

import Axios from "axios";

export default {

props: ["value", "placeHolder"],

**data**() {

return {

treeData: []

};

},

methods: {

**departmentSelected**(value, label, extra) {

this.**$emit**("change", value);

this.**loadData**();

},

**loadData**() {

**console**.**log**("DeaprtmentSelector: loading data...");

let that = this;

*// 加载省级部门数据*

Axios.**get**(appConfigs.ApiBaseUrl + "/departments/tree/nodes", {

headers: {

Token: localStorage.**getItem**("token")

}

})

.**then**(resp => {

*// 成功*

**console**.**log**("provinces loaded. ", resp.status);

var data = resp.data;

if (data.status == "200") {

that.treeData = data.data;

}

})

.**catch**(resp => {

*// 失败*

that.$message.**error**("无法加载省级部门数据");

});

}

},**mounted**() {

this.**loadData**();

}

};

</script>

上述代码展示了一个Vue单文件组件的全部内容，一个Vue组件主要分为<template />和<script />两部分(还可以加入<style/>)；<template>部分提供组件的html样式，<script>部门提供此小组件的JavaScript逻辑代码。

在组件中可以调用其他组件，比如上述代码中使用了一个<a-tree-select>组件，这显然不是html原生的内容，这是由AntDesign库提供的树形选择器，通过将此组件的treeData属性绑定到本组件定义的treeData变量上以获得动态监听数据变化并自动重新渲染的能力；随后只需要改变treeData变量的内容，页面上内容就会做出相应变化。

loadData()方法中则展示了使用Axios发起HTTP请求的例子，第一个参数是字符串类型的URL；第二个参数中设定了自定义的HTTP请求头Token由此方式传递给后端逃犯查询系统来进行身份校验。

1. 系统测试
   1. 测试任务及目标

对逃犯查询系统的测试任务主要分为REST API 测试和页面逻辑测试两部分。目标是检验API是否可以正常工作，返回数据是否正确，以及页面显示是否与数据匹配，是否可以正常操作并完成业务逻辑。本文档给出部分关键功能的测试流程。

* 1. 测试方案
     1. 概述

对于本系统功能的测试主要是对后端微服务的RESTful API 的接口测试，在测试过程中将使用黑盒测试的方式，根据功能需求说明和接口设计说明，对输入输出数据进行核对；使用的主要工具是Postman。

对于本系统前端软件功能的测试通过测试用例中描述的内容进行操作，查看是否符合测试用例描述，检验各种功能是否正常。

* + 1. 使用Postman测试RESTful API

Postman是一个RESTful API测试工具，主要作用是定制化HTTP请求，模拟客户端的请求操作，并记录http请求和响应的所有细节，可以精确的控制请求方法、请求头、请求体等内容。

在测试过程中使用此工具对所有后端API进行了测试，出现的关于参数名称错误、数据返回格式不正确、返回数据属性名称错误等等问题都得到了修正。

图 6.1 Postman测试用例列表

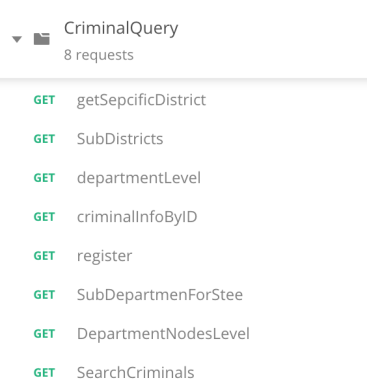
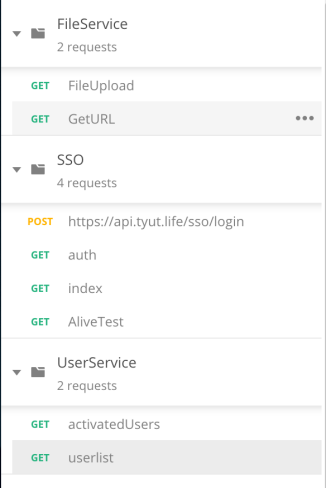
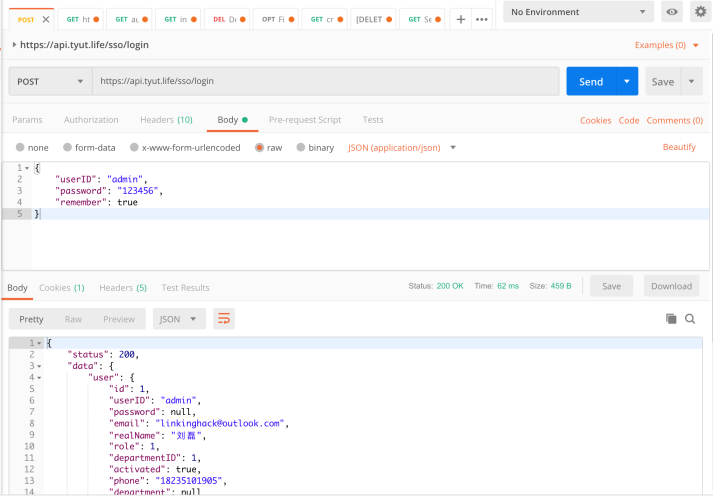


图 6.2 Postman测试示例



* + 1. 测试用例
* 测试1:

名称：注册

目的：用户注册功能是否可用，用户登录ID是否保证唯一性，用户注册后是否直接置为冻结状态，用户密码存入数据库前是否已加密。

内容：使用全新的ID尝试进行用户注册，随后尝试登录，查看是否禁止登录；使用一个已知存在的用户登录ID尝试进行注册操作，查看是否拒绝注册；通过查验数据库来查看用户密码是否加密。

结果：系统可以保证用户登录ID的唯一性，可以保证新用户注册后需要走审批流程才可开通使用的特性，系统在保存密码进数据库前已对密码进行强加密。

* 测试2:

名称：登录

目的：测试使用正确的用户名和密码是否可以登录系统，测试使用错误的用户名或密码是否可以被阻止使用系统，测试正常登录后是否可以访问单点登录系统之外的其他服务。

内容：使用系统中处于激活状态的用户名和明文密码尝试登录系统，查看是否可以正常进入内容页面，从而验证密码加密校验是否正常工作；使用随意输入的错误用户名和密码尝试登录，随后直接尝试访问非公开页面检查是否可阻挡未授权的访问。

结果：使用正确的用户名和明文密码可以进入系统，正常登录后可以访问其他微服务提供的功能；使用错误的用户凭证无法进入系统，无法访问除登录和注册页面之外的任何页面。

* 测试3:

名称：添加新逃犯基本信息

内容：测试添加逃犯基本信息功能是否正常，测试是否可以上传逃犯主识别图像，测试提交信息后是否可以正确记录所填写信息。

结果：执行添加逃犯信息流程后，检查相应数据库表，发现逃犯主识别图像的文件ID正确保存，检查对象存储服务存储桶文件记录可以发现对应的文件记录；其他字段保存正常。

* 测试4:

名称：发起通缉

目的：测试添加逃犯基本信息后第二步页面中直接对其发起通缉功能是否正常，测试通缉范围选择器是否正常加载出行政区划数据。

内容：执行标准添加逃犯基本信息流程，进入发起通缉页面选择通缉区域并填写通缉原因后提交；直接查看数据库表中记录，查看通缉区域ID属性是否保存，通缉原因属性是否保存正确；检查通缉区域ID在行政区划数据表中记录是否对应目标通缉区域。

结果：发起通缉功能可以正常工作，完成发起通缉操作后，检查数据库表相应记录数据均正确；通缉范围的区域ID可以正确对应行政区划数据表中的相关区域。

结论

本文通过编写一个逃犯查询系统的需求分析文档、系统概要设计文档、系统详细设计文档以及软件测试文档，完成了基于SSM框架的逃犯在线查询系统的设计；通过实施本文所述的设计方案，完成编码工作，实现了本文所述逃犯查询系统的大部分功能。

在设计本逃犯查询系统的思路上采用了目前软件开发实际工作中较新的微服务的设计方案，并且采用前后端完全分离开发的方法，后端仅关注REST API的开发，满足了系统可扩展的要求；前端技术使用了目前GitHub上Star数量最多的开源项目Vue，配合Axios作为HTTP Client与后端服务进行交互。最后使用Docker容器来启动项目实现了微服务的快速部署。

本文中所述逃犯系统仅实现了最基本的API权限控制，后续系统应该还需要一套非常完善的可以支持多角色的精确到API级别对各角色任意组合的权限系统；由于权限系统的开发本身也是一个研究方向，本文所述系统没有过多的去涉及；若后续有机会可以继续研究。另一个局限性是本系统还应该加入一个邮件系统，来对用户注册审核结果、通缉令发布结果、新线索、本区域新逃犯等等事件的发生给相关用户发送邮件通知，提高效率；本次开发中没有实现此功能。

参考文献

[1][Guillaume Chau. Vue.js 2 Web Development Projects[M]. Britain: Packt, 2017.](https://www.packtpub.com/web-development/vuejs-2-web-development-projects)

Ravi Sharma, Shipra Ravi Kumar. Strategies for Web Application Development Methodologies [R]. ICCCA, 2016 .

[2][Rick Osowski . Introduction to microservices[J/OL]. IBM Developer, 2015](https://developer.ibm.com/tutorials/cl-ibm-cloud-microservices-in-action-part-1-trs/) .

[3][Craig Walls, 张卫滨. Spring 实战(第四版) [M]. 北京: 人民邮电出版社, 2014](https://www.manning.com/books/spring-in-action-fourth-edition) .

[4]郝佳.Spring源码深度解析(第二版)[M].北京: 人民邮电出版社, 2019.

[5][章仕锋,潘善亮.Docker技术在微服务中的应用[J/OL].电子技术与软件工程,2019(04)](http://kns.cnki.net/kcms/detail/10.1108.TP.20190304.1523.228.html).

[6]Martin R C. Agile software development: principles, patterns, and practices[M]. Prentice Hall, 2002.

[7][郭丞乾,蔡权伟,林璟锵,刘丽敏.单点登录协议实现的安全分析[J].信息安全究,2019,5(01)](http://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ&dbname=CJFDTEMP&filename=XAQY201901008&v=MDIwOTFvRkNybVc3ektQU3phZDdHNEg5ak1ybzlGYklSOGVYMUx1eFlTN0RoMVQzcVRyV00xRnJDVVJMT2ZaT1o=) .

[8][魏春来,付永振.基于微服务的DevOps研究与实现[J].网络安全和信息化,2018(11):54-56.](http://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFD&dbname=CJFDLAST2018&filename=WAXX201811039&v=MzA2NzFOcm85R2JZUjhlWDFMdXhZUzdEaDFUM3FUcldNMUZyQ1VSTE9mWk9ab0ZDcm5VYjdKTWl6VGRyRzRIOW4=)

[9][Building a RESTful Web Service[J/OL]. Spring.io](https://spring.io/guides/gs/rest-service/)

[10]IBM.微服务架构[EB/OL].https://www.ibm.com/developerworks/cn/java/j-cn-java-and-microservice-1st/index.html,2017-01-20

[11]LinuxFoundation.KubernetesBasics[EB/OL].https://kubernetes.io/docs/tutorials/kubernetes-basics/,2019-02-22.

致谢

首先特别感谢张玲老师和李钢老师在我的毕业设计进行过程中给予耐心指导和帮助；在毕设项目需求分析阶段由于题目比较罕见，可搜集的到的资料非常少，两位老师百忙之中抽出时间一同帮我思考系统的内容，这些是我完成此论文和项目的关键。

其次感谢云账户技术（天津）有限公司在我实习和完成毕设过程中为我提供了强大的硬件设备支持、办公环境支持和技术支持。

最后感谢东软睿道的各位老师在我完成毕业设计过程中提供的指导和帮助。

外文原文

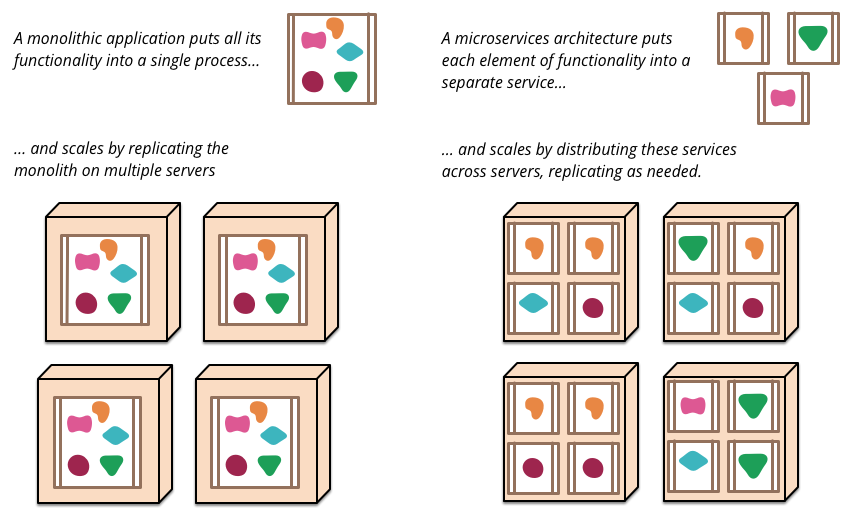
A Definition of Microservice

The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.

To start explaining the microservice style it's useful to compare it to the monolithic style: a monolithic application built as a single unit. Enterprise Applications are often built in three main parts: a client-side user interface (consisting of HTML pages and javascript running in a browser on the user's machine) a database (consisting of many tables inserted into a common, and usually relational, database management system), and a server-side application. The server-side application will handle HTTP requests, execute domain logic, retrieve and update data from the database, and select and populate HTML views to be sent to the browser. This server-side application is a *monolith* - a single logical executable. Any changes to the system involve building and deploying a new version of the server-side application.

Monolithic applications can be successful, but increasingly people are feeling frustrations with them - especially as more applications are being deployed to the cloud . Change cycles are tied together - a change made to a small part of the application, requires the entire monolith to be rebuilt and deployed. Over time it's often hard to keep a good modular structure, making it harder to keep changes that ought to only affect one module within that module. Scaling requires scaling of the entire application rather than parts of it that require greater resource.

Figure.1 Monoliths and Microservices



These frustrations have led to the microservice architectural style: building applications as suites of services. As well as the fact that services are independently deployable and scalable, each service also provides a firm module boundary, even allowing for different services to be written in different programming languages. They can also be managed by different teams .

We do not claim that the microservice style is novel or innovative, its roots go back at least to the design principles of Unix. But we do think that not enough people consider a microservice architecture and that many software developments would be better off if they used it.

Characteristics of a Microservice Architecture

Componentization via Services

Microservice architectures will use libraries, but their primary way of componentizing their own software is by breaking down into services. We define **libraries** as components that are linked into a program and called using in-memory function calls, while **services** are out-of-process components who communicate with a mechanism such as a web service request, or remote procedure call. (This is a different concept to that of a service object in many OO programs.)

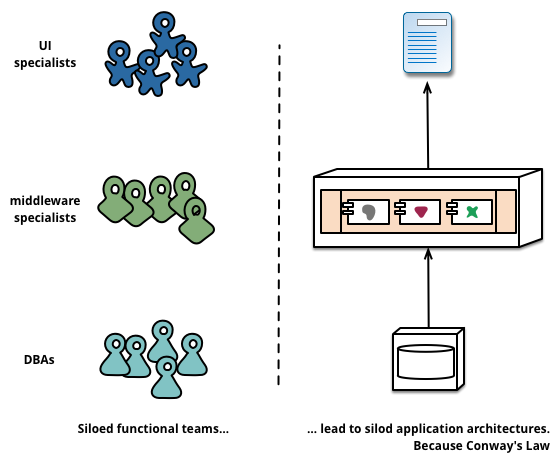
One main reason for using services as components (rather than libraries) is that services are independently deployable. If you have an application that consists of a multiple libraries in a single process, a change to any single component results in having to redeploy the entire application. But if that application is decomposed into multiple services, you can expect many single service changes to only require that service to be redeployed. That's not an absolute, some changes will change service interfaces resulting in some coordination, but the aim of a good microservice architecture is to minimize these through cohesive service boundaries and evolution mechanisms in the service contracts.

Another consequence of using services as components is a more explicit component interface. Most languages do not have a good mechanism for defining an explicit [Published Interface](https://martinfowler.com/bliki/PublishedInterface.html). Often it's only documentation and discipline that prevents clients breaking a component's encapsulation, leading to overly-tight coupling between components. Services make it easier to avoid this by using explicit remote call mechanisms.

Organized around Business Capabilities

When looking to split a large application into parts, often management focuses on the technology layer, leading to UI teams, server-side logic teams, and database teams. When teams are separated along these lines, even simple changes can lead to a cross-team project taking time and budgetary approval. A smart team will optimise around this and plump for the lesser of two evils - just force the logic into whichever application they have access to. Logic everywhere in other words. This is an example of Conway's Law in action.

Figure2 Conway's Law in action



The microservice approach to division is different, splitting up into services organized around **business capability**. Such services take a broad-stack implementation of software for that business area, including user-interface, persistant storage, and any external collaborations. Consequently the teams are cross-functional, including the full range of skills required for the development: user-experience, database, and project management.

Products not Projects

Most application development efforts that we see use a project model: where the aim is to deliver some piece of software which is then considered to be completed. On completion the software is handed over to a maintenance organization and the project team that built it is disbanded.

Microservice proponents tend to avoid this model, preferring instead the notion that a team should own a product over its full lifetime. A common inspiration for this is Amazon's notion of ["you build, you run it"](https://queue.acm.org/detail.cfm?id=1142065) where a development team takes full responsibility for the software in production. This brings developers into day-to-day contact with how their software behaves in production and increases contact with their users, as they have to take on at least some of the support burden.

The product mentality, ties in with the linkage to business capabilities. Rather than looking at the software as a set of functionality to be completed, there is an on-going relationship where the question is how can software assist its users to enhance the business capability.

There's no reason why this same approach can't be taken with monolithic applications, but the smaller granularity of services can make it easier to create the personal relationships between service developers and their users.

Smart endpoints and dumb pipes

When building communication structures between different processes, we've seen many products and approaches that stress putting significant smarts into the communication mechanism itself. A good example of this is the Enterprise Service Bus (ESB), where ESB products often include sophisticated facilities for message routing, choreography, transformation, and applying business rules.

The microservice community favours an alternative approach: *smart endpoints and dumb pipes*. Applications built from microservices aim to be as decoupled and as cohesive as possible - they own their own domain logic and act more as filters in the classical Unix sense - receiving a request, applying logic as appropriate and producing a response. These are choreographed using simple RESTish protocols rather than complex protocols such as WS-Choreography or BPEL or orchestration by a central tool.

The two protocols used most commonly are HTTP request-response with resource API's and lightweight messaging.

Decentralized Governance

One of the consequences of centralised governance is the tendency to standardise on single technology platforms. Experience shows that this approach is constricting - not every problem is a nail and not every solution a hammer. We prefer using the right tool for the job and while monolithic applications can take advantage of different languages to a certain extent, it isn't that common.

Splitting the monolith's components out into services we have a choice when building each of them. You want to use Node.js to standup a simple reports page? Go for it. C++ for a particularly gnarly near-real-time component? Fine. You want to swap in a different flavour of database that better suits the read behaviour of one component? We have the technology to rebuild him.

Teams building microservices prefer a different approach to standards too. Rather than use a set of defined standards written down somewhere on paper they prefer the idea of producing useful tools that other developers can use to solve similar problems to the ones they are facing. These tools are usually harvested from implementations and shared with a wider group, sometimes, but not exclusively using an internal open source model. Now that git and github have become the de facto version control system of choice, open source practices are becoming more and more common in-house .

Decentralized Data Management

Decentralization of data management presents in a number of different ways. At the most abstract level, it means that the conceptual model of the world will differ between systems. This is a common issue when integrating across a large enterprise, the sales view of a customer will differ from the support view. Some things that are called customers in the sales view may not appear at all in the support view. Those that do may have different attributes and (worse) common attributes with subtly different semantics.

This issue is common between applications, but can also occur within applications, particular when that application is divided into separate components. A useful way of thinking about this is the Domain-Driven Design notion of Bounded Context. DDD divides a complex domain up into multiple bounded contexts and maps out the relationships between them. This process is useful for both monolithic and microservice architectures, but there is a natural correlation between service and context boundaries that helps clarify, and as we describe in the section on business capabilities, reinforce the separations.

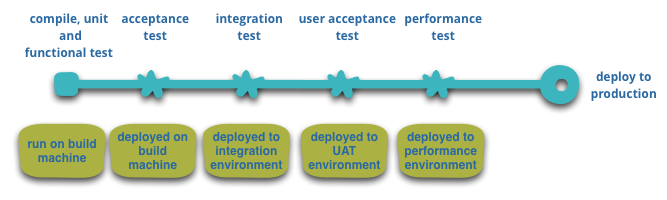
As well as decentralizing decisions about conceptual models, microservices also decentralize data storage decisions. While monolithic applications prefer a single logical database for persistant data, enterprises often prefer a single database across a range of applications - many of these decisions driven through vendor's commercial models around licensing. Microservices prefer letting each service manage its own database, either different instances of the same database technology, or entirely different database systems - an approach called Polyglot Persistence. You can use polyglot persistence in a monolith, but it appears more frequently with microservices.

Infrastructure Automation

Infrastructure automation techniques have evolved enormously over the last few years - the evolution of the cloud and AWS in particular has reduced the operational complexity of building, deploying and operating microservices.

Many of the products or systems being build with microservices are being built by teams with extensive experience of Continuous Delivery and it's precursor, Continuous Integration. Teams building software this way make extensive use of infrastructure automation techniques. This is illustrated in the build pipeline shown below.

Figure3 basic build pipeline



Design for failure

Since services can fail at any time, it's important to be able to detect the failures quickly and, if possible, automatically restore service. Microservice applications put a lot of emphasis on real-time monitoring of the application, checking both architectural elements (how many requests per second is the database getting) and business relevant metrics (such as how many orders per minute are received). Semantic monitoring can provide an early warning system of something going wrong that triggers development teams to follow up and investigate.

This is particularly important to a microservices architecture because the microservice preference towards choreography and event collaboration leads to emergent behavior. While many pundits praise the value of serendipitous emergence, the truth is that emergent behavior can sometimes be a bad thing. Monitoring is vital to spot bad emergent behavior quickly so it can be fixed..

Microservice teams would expect to see sophisticated monitoring and logging setups for each individual service such as dashboards showing up/down status and a variety of operational and business relevant metrics. Details on circuit breaker status, current throughput and latency are other examples we often encounter in the wild.

Evolutionary Design

Microservice practitioners, usually have come from an evolutionary design background and see service decomposition as a further tool to enable application developers to control changes in their application without slowing down change. Change control doesn't necessarily mean change reduction - with the right attitudes and tools you can make frequent, fast, and well-controlled changes to software.

中文翻译

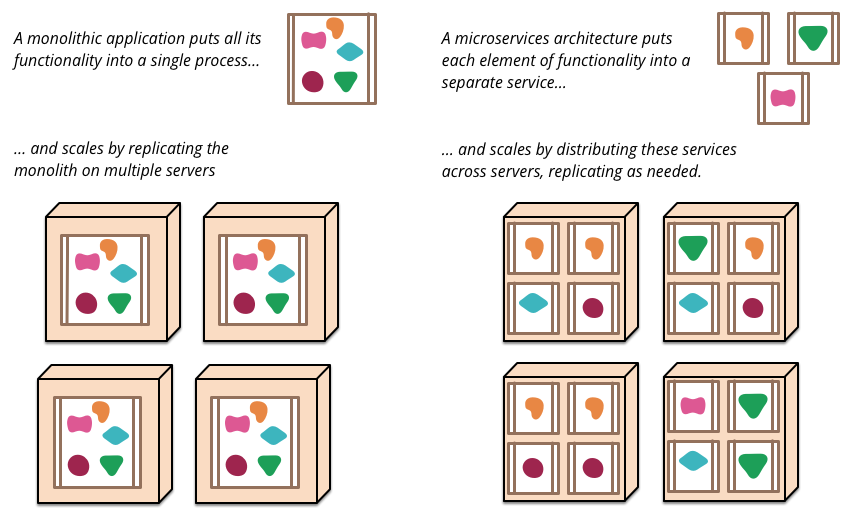
微服务的定义

在过去的几年中，“微服务体系结构”一词已经出现，用来描述将软件应用程序设计为独立可部署服务套件的一种特殊方式。虽然没有对这种体系结构风格的精确定义，但是围绕业务能力、自动化部署、端点中的智能以及语言和数据的分散控制，组织有一些共同的特征。

要开始解释微服务风格，将其与单片式风格进行比较是很有用的：单片式应用程序作为单个单元构建。企业应用程序通常由三个主要部分组成：客户端用户界面（由用户计算机上浏览器中运行的HTML页面和JavaScript组成）、数据库（由插入到公共数据库管理系统（通常是关系数据库管理系统）中的许多表组成）和服务器端应用程序。服务器端应用程序将处理HTTP请求，执行域逻辑，从数据库中检索和更新数据，并选择和填充要发送到浏览器的HTML视图。这个服务器端应用程序是一个单块单逻辑可执行文件。对系统的任何更改都涉及到构建和部署服务器端应用程序的新版本。

单片应用程序可能会成功，但越来越多的人对它们感到失望，尤其是随着越来越多的应用程序部署到云端。变更周期被捆绑在一起——对应用程序的一小部分所做的变更，需要重新构建和部署整个整体。随着时间的推移，很难保持一个好的模块化结构，这使得很难保持只影响该模块中一个模块的更改。扩展需要扩展整个应用程序，而不是部分需要更大的资源。

图1：单体应用和微服务



这些挫折导致了微服务体系结构风格：将应用程序构建为服务套件。除了服务可以独立部署和扩展之外，每个服务还提供了一个坚实的模块边界，甚至允许用不同的编程语言编写不同的服务。它们也可以由不同的团队管理。

我们并不认为微服务风格是新颖或创新的，它的根源至少可以追溯到Unix的设计原则。但我们确实认为，没有足够的人考虑使用微服务架构，而且如果他们使用它，许多软件开发会更好。

微服务体系结构的特点

1通过服务实现组件化

微服务体系结构将使用库，但它们将自己的软件组件化的主要方式是分解成服务。我们将库定义为链接到程序中并使用内存函数调用调用的组件，而服务是与诸如Web服务请求或远程过程调用等机制通信的进程外组件。（这与许多OO程序中服务对象的概念不同。）

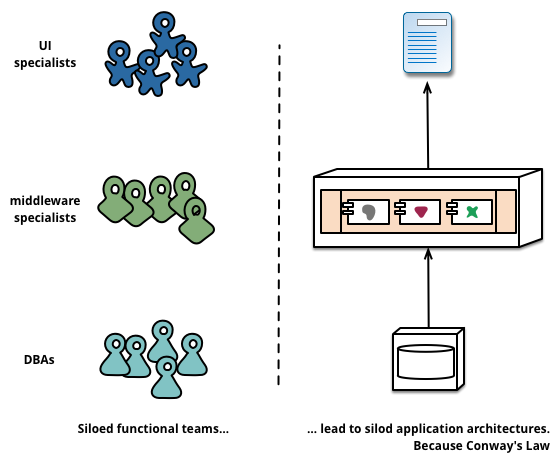
将服务用作组件（而不是库）的一个主要原因是服务可以独立部署。如果您的应用程序在一个进程中由多个库组成，那么对任何单个组件的更改都会导致必须重新部署整个应用程序。但是，如果该应用程序被分解为多个服务，那么您可以期望对单个服务的许多更改只需要重新部署该服务。这并不是绝对的，一些变化会改变服务接口，导致一些协调，但是一个好的微服务架构的目的是通过服务契约中的一致服务边界和演化机制来最小化这些变化。

使用服务作为组件的另一个结果是更显式的组件接口。大多数语言没有良好的机制来定义显式发布的接口。通常，只有文档和规程可以防止客户机破坏组件的封装，从而导致组件之间的耦合过紧。服务通过使用显式的远程调用机制更容易避免这种情况。

2围绕业务能力组织

当要将一个大型应用程序拆分为多个部分时，管理层通常将重点放在技术层上，从而领导UI团队、服务器端逻辑团队和数据库团队。当团队沿着这些线分离时，即使是简单的更改也可能导致跨团队项目需要时间和预算批准。一个聪明的团队将围绕这一点进行优化，并针对两个缺点中较小的一个进行充实——只需将逻辑强制到他们可以访问的任何应用程序中。换句话说，逻辑无处不在。这是康威定律的一个实例。

图2: 康威定律



微服务的划分方法是不同的，分为围绕业务能力组织的服务。这类服务需要为该业务领域实现大量的软件，包括用户界面、持久性存储和任何外部协作。因此，团队是跨职能的，包括开发所需的全部技能：用户体验、数据库和项目管理。

3产品不是项目

我们看到的大多数应用程序开发工作都使用一个项目模型：其目标是交付一些软件，然后将其视为已完成。完成后，软件将移交给维护组织，建立它的项目团队将被解散。

微服务的支持者倾向于避免这种模式，他们更倾向于一个团队应该在其整个生命周期内拥有一个产品的概念。对此，一个常见的灵感来自于亚马逊的“你建立，你运行”的概念，即开发团队对生产中的软件负全责。这使开发人员能够与软件在生产中的行为保持日常联系，并增加与用户的联系，因为他们必须承担至少一部分支持负担。

产品心态，与企业能力的联系紧密。与其将软件视为一组要完成的功能，还存在一种持续的关系，问题是软件如何帮助其用户增强业务能力。

对于整体应用程序来说，没有理由不能采用相同的方法，但是服务的粒度越小，就越容易在服务开发人员和他们的用户之间创建个人关系。

4智能端点和哑管道

在构建不同流程之间的通信结构时，我们看到了许多产品和方法，它们强调在通信机制本身中引入重要的智能。企业服务总线（EnterpriseServiceBus，ESB）就是一个很好的例子，ESB产品通常包括用于消息路由、编排、转换和应用业务规则的复杂工具。

微服务社区支持另一种方法：智能端点和哑管道。从微服务构建的应用程序的目标是尽可能的分离和内聚——它们拥有自己的域逻辑，在传统的Unix意义上更像过滤器——接收请求，适当地应用逻辑并产生响应。这些都是使用简单的restish协议进行编排的，而不是使用中心工具进行复杂的协议（如WS-Choreography或BPEL或编排）。

最常用的两种协议是HTTP请求响应和资源API和轻量级消息传递。

5分散治理

集中治理的后果之一是趋向于在单一技术平台上实现标准化。经验表明，这种方法是紧缩的-不是每个问题都是钉子，不是每个解决方案都是锤子。我们更喜欢为工作使用正确的工具，虽然单片应用程序在某种程度上可以利用不同的语言，但这并不常见。

将整块体的组件分割成服务，我们在构建每个组件时都有选择。您想使用node.js来支持一个简单的报告页面吗？去争取它。C++用于一个特别接近实时的组件？好的。您想换一种更适合一个组件的读取行为的不同风格的数据库吗？我们有技术来重建他。

构建微服务的团队也喜欢使用不同的标准方法。与其使用一套写在纸上的已定义标准，他们更喜欢使用产生有用工具的想法，其他开发人员可以使用这些工具来解决他们所面临的类似问题。这些工具通常是从实现中获得的，并与更广泛的组共享，有时，但不是仅使用内部开放源代码模型。现在Git和Github已经成为事实上的版本控制系统的选择，开源实践在内部变得越来越普遍。

6分散式数据管理

数据管理的分散以多种不同的方式呈现。在最抽象的层面上，它意味着世界的概念模型在系统之间会有所不同。这是跨大型企业集成时的常见问题，客户的销售视图与支持视图不同。在“销售”视图中称为“客户”的某些内容可能根本不会出现在“支持”视图中。那些有可能具有不同的属性和（更糟的）具有细微不同语义的公共属性。

此问题在应用程序之间很常见，但也可能发生在应用程序中，特别是当应用程序被划分为单独的组件时。一种有用的思考方法是有界上下文的领域驱动设计概念。DDD将一个复杂域划分为多个有界上下文，并映射出它们之间的关系。这个过程对于整体和微服务架构都很有用，但是服务和上下文边界之间有一种自然的关联，这有助于澄清，正如我们在业务能力部分中描述的那样，加强了分离。

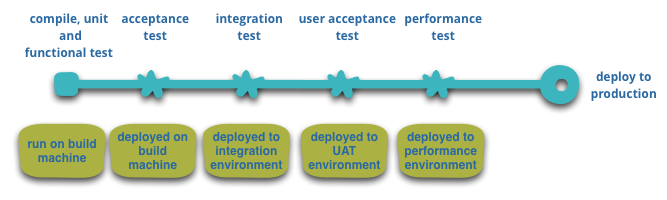
除了分散概念模型的决策之外，微服务还分散数据存储决策。虽然单片应用程序更喜欢单一逻辑数据库来存储持久性数据，但企业通常更喜欢跨一系列应用程序的单一数据库——其中许多决策都是由供应商的商业模型围绕许可做出的。微服务更喜欢让每个服务管理自己的数据库，要么是同一数据库技术的不同实例，要么是完全不同的数据库系统——一种称为polyglot持久性的方法。您可以在一个整体中使用polyglot持久性，但它在微服务中出现的频率更高。

7基础设施自动化

在过去的几年中，基础设施自动化技术已经有了巨大的发展——尤其是云和AWS的发展降低了构建、部署和操作微服务的操作复杂性。

许多使用微服务构建的产品或系统都是由具有广泛连续交付经验的团队构建的，并且是它的先驱、持续集成。以这种方式构建软件的团队广泛使用基础设施自动化技术。这在下面的构建管道中进行了说明。

图3：基本构建管道



8失效设计

由于服务可以在任何时候发生故障，因此能够快速检测故障并在可能的情况下自动恢复服务是很重要的。微服务应用程序非常重视应用程序的实时监控，检查体系结构元素（数据库每秒收到多少请求）和业务相关指标（例如每分钟收到多少订单）。语义监控可以提供出错的预警系统，从而触发开发团队进行跟踪和调查。

这对于微服务体系结构尤其重要，因为微服务对编排和事件协作的偏好导致了紧急行为。虽然许多专家称赞意外发生的价值，但事实是，紧急行为有时是件坏事。监控对于快速发现不良紧急行为至关重要，因此可以对其进行修复。

微服务团队希望看到每个服务的复杂监控和日志记录设置，例如显示上/下状态的仪表盘以及各种与操作和业务相关的指标。有关断路器状态、电流吞吐量和延迟的详细信息是我们在野外经常遇到的其他示例。

9进化设计

微服务从业者通常来自于一个进化的设计背景，他们将服务分解看作是一个更进一步的工具，使应用程序开发人员能够控制他们的应用程序中的更改，而不会减慢更改的速度。变更控制并不一定意味着减少变更——有了正确的态度和工具，您就可以频繁、快速和良好地控制软件变更。

1. JSON即JavaScript Object Notation，一种数据序列化格式，使用key:value的方式来嵌套的定义数据结构 [↑](#footnote-ref-1)
2. 即Representational State Transfer，详见https://zh.wikipedia.org/wiki/表现层状态转换 [↑](#footnote-ref-2)
3. MVVM即Model-View-viewmodel, 详见https://zh.wikipedia.org/wiki/MVVM [↑](#footnote-ref-3)
4. 图片来自kubernetes.io，版权归Linux Foundation所有 [↑](#footnote-ref-4)
5. 冲突即不同的分支修改了同一个文本文件的同一行内容，VCS不知道应该保留那一个分支的修改，此时需要人为决定。 [↑](#footnote-ref-5)
6. 这里的“所有”指的是权限范围内的所有，VCS的商业版本（比如gitlab）有比较完善的权限系统，可以控制代码可见的范围。 [↑](#footnote-ref-6)
7. 本文将只包括非常重要的功能实现内容，其他未详细列出的实现细节为太过于简单的功能，参考项目代码来查看具体实现。 [↑](#footnote-ref-7)
8. SHA3为第三代安全散列算法（Secure Hash Algorithm 3），本系统使用256位输出标准。参考：<https://zh.wikipedia.org/wiki/SHA-3> [↑](#footnote-ref-8)