中国科学院大学

University of Chinese Academy of Sciences

大作业结题报告

高考志愿帮——基于智能问答的志愿填报助手

学生姓名： 王凯锐、戢启瑞、郑莘

课程名称： 高级软件工程

指导教师： 罗铁坚

学 院： 计算机科学与技术学院

2021 年 12 月

目 录

第一章 项目背景

1.1 项目意义

随着人工智能、互联网等技术的发展，聊天机器人在多个领域都有了较为广泛的应用。实践证明，聊天机器人的使用不仅能够为用户提供更加人性化的服务，而且能够明显提高用户解决问题的效率。

合理填报志愿对于考生来说意义十分重大。有这样一句话来形容高考志愿填报的重要性，“三分成绩，七分志愿。”志愿填写比高考本身更重要。过去学校包分配时，只要能进入大学校园，就有一个好的起点。而现在，在分数相同的情况下，一旦选错专业，做错决策，那将影响一个考生以后的就业之路。填报志愿实际上是考生与院校之间的一种“双向选择”：一方面考生通过填报志愿，表达自己的愿望即向往何种院校、喜欢什么专业等；另一方面，各普通高校又以考生填报的志愿为其录取的基本依据，从众多的报考者中择优选拔合格的新生。

1.2 研究现状

当前网上有不少类似的高考志愿帮助网站，但是都或多或少存在一些问题，以下列举了一些相似工程的不足之处：

系统 A：

不足之处：

① 有地域限制

② 无机器人问答服务

系统 B：

不足之处：
① 广告繁杂
② 无机器人问答服务

系统 C：

不足之处：
① 数据只更新到 2011 年
② 无机器人问答服务

1.3 研究方向

从实际情况来看，目前的各种志愿填报网站仍然无法满足高校学生等用户的需求，因此本设计计划实现一个基于智能问答的志愿填报助手网站，基于用户的实际需求和查询需要，为用户提供足够的信息，从而帮助用户做出适合自己的志愿选择。

第二章 可行性和需求分析

2.1 可行性分析

可行性分析是通过对项目的主要内容和条件，从技术、经济、工程等方面进行调查研究和分析比较，为项目决策提供依据的一种综合性的系统分析方法。可行性分析应具有预见性、公正性、可靠性、科学性的特点。软件开发设计人员一般会在软件需求定义之前，对软件开发的可行性进行全方位考虑，以便后期各项开发与维护工作的开展。

2.1.1 技术可行性

在本次课题设计中，技术可行性主要从智能问答机器人的实现以及外部数据的获取和网站平台的搭建几个方面进行考虑。

对于一款实用性要求较高的平台来说，具有较为准确的数据源是十分关键的。本课题所需的数据大都可以从互联网上获取，并且从平台技术实现角度来看，对于上文中所提到的各项技术，也均在其余各类研究或者实践中得到了广泛的应用，均不存在技术层面的困难。因此本平台在技术实现层面具备较强的可行性。

2.1.2 经济可行性

在本平台的构造过程中，为了实现部署要求，租用了阿里云服务器，会产生少量可以承受的开销。而就软件成本而言，本系统主要基于开源版本或者免费版本，不存在经济额外支出。因此虽然在课题的实现过程中会产生部分经济开销，但都在可承受范围内，因此本次开发在经济上具有较好的可行性。

2.2 项目软件使用价值

①高三考生：通过聊天机器人获取相关院校以及专业信息，结合系统推荐院校专业以及自身的判断做出志愿选择。

②还未参与高考的高中生：根据对自己成绩的预估进行院校筛选,明确自身定位，同时进一步树立自己的目标。

③高中生家长：根据平台提供的信息为自己孩子的志愿选择提供自己的意见与看法。

2.3 客户端功能性需求分析

根据本项目的目标用户以及上述提出的软件使用价值，我们提出了以下功能需求：

1、可以查询相关院校的基本信息。

2、可以查询不同专业各自的基本信息。

3、可以查询不同考区各个院校的录取分数线情况。

4、智能咨询功能：智能机器人可以根据用户的提问进行回答。

5、为用户提供高校、专业等基本信息展示

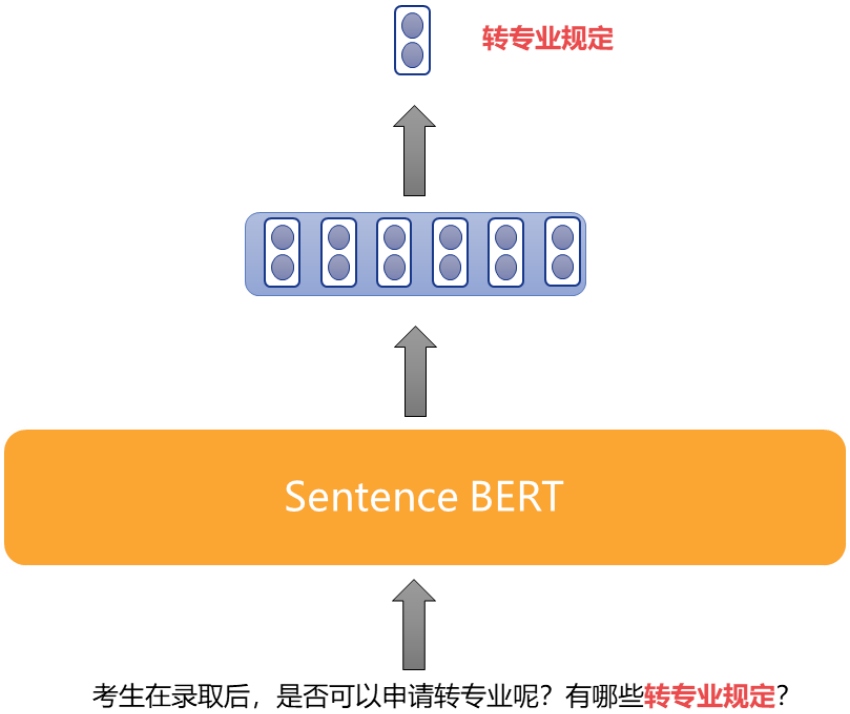
第三章 核心算法设计与实现

3.1 算法描述

本算法基于已有的问题答案对进行检索，返回与询问最接近的答案。本算法无监督地使用预训练语言模型 Sentence BERT 的向量表示对自然语言编码，并使用向量检索引擎 FAISS 实现快速向量匹配。

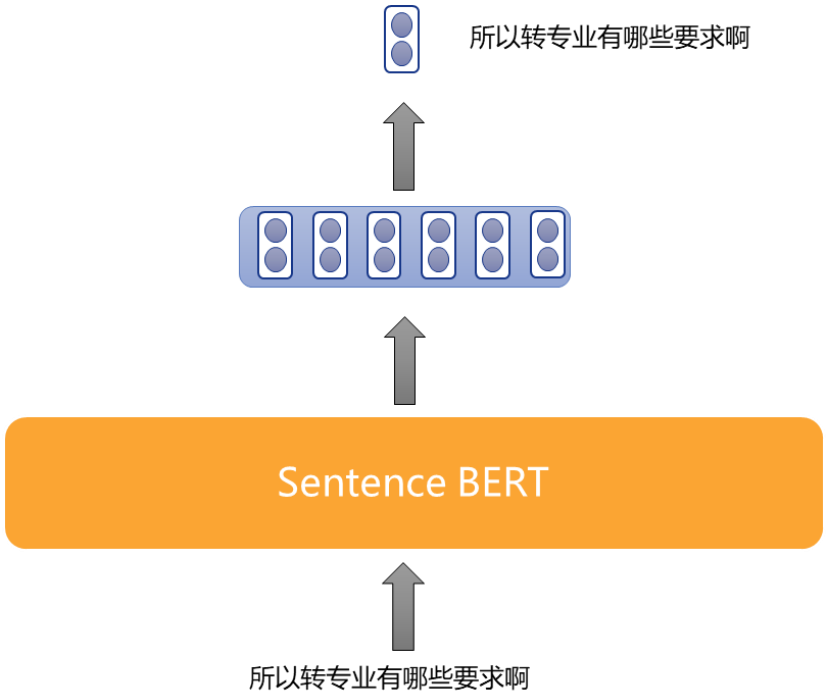
本算法分为三步：预处理部分，用模型对所有问题的关键词进行编码；给定提问，用模型对提问编码；最后，寻找已知问题中与提问最接近的，返回其对应答案。

3.1.1 关键词编码



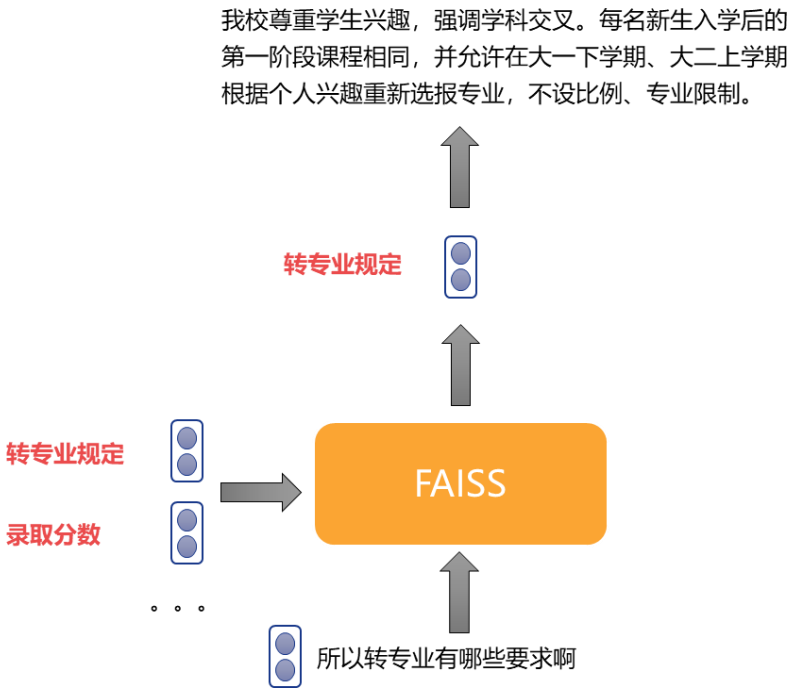
对于已知的每一个问题，用 Sentence BERT 模型，将其输入模型得到整个句子的向量表示。然后，从中提取出关键词对应的几个向量。最后，再对这几个向量取平均得到单个向量，作为该问题的关键词的向量表示。存储关键词向量，并记录与之对应的问题和回答。

3.1.2 提问编码



给定提问，同样用模型编码，得到整个句子的向量表示，再对整个句子的向量取平均得到单个提问向量。

3.1.3 提问检索



给定已知问题的关键词向量，以及提问向量，使用向量检索引擎检索，查询与提问向量最接近的关键词向量，返回与之对应的答案。

3.2 算法测试

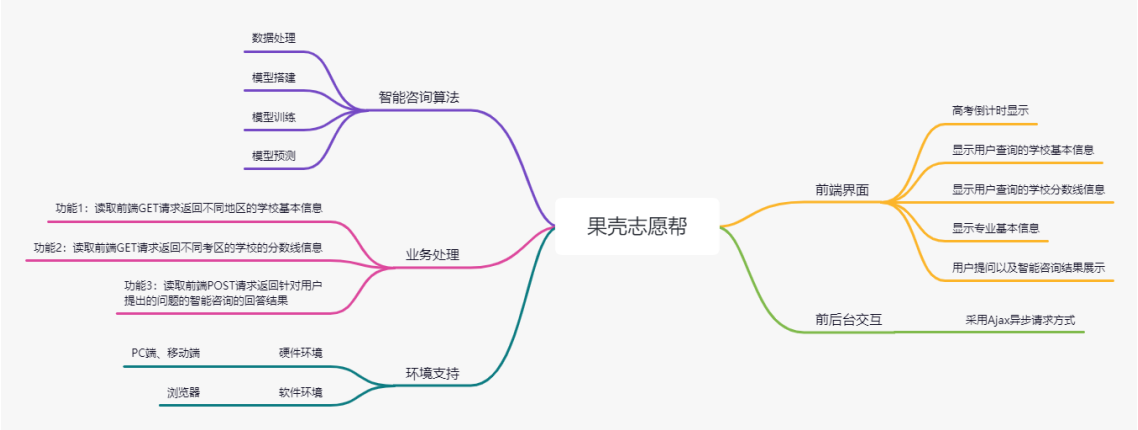
构造算法的自动化测试为：对于已知问题，使用本算法检索，应当返回与之对应的答案。算法返回答案与对应答案一致，计为正确；不一致，计为错误。

经测试，本算法的准确率为 100%。

第四章 系统设计与实现

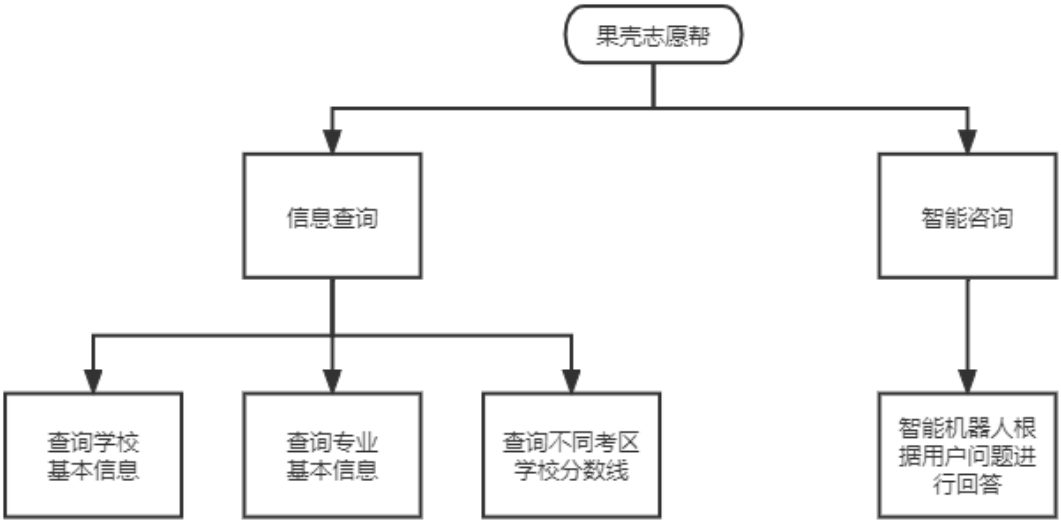
4.1 系统整体架构设计

项目的整体系统框架如下：



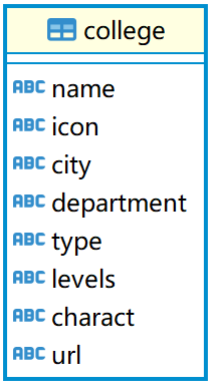
前端框架采用 Vue + View UI，后端框架采用 Java Springboot。前后端之间的交互采用 AJAX，实现网页的异步更新，前端发送 AJAX 请求调用后端的 API 接口并使用 JSON 数据进行交互实现前后端的分离，从而满足软件开发过程中高内聚低耦合的要求。

项目的整体功能图如下：



4.2 数据库设计

本次我们采用 MySQL 数据库，表结构如下：



→ 包含了所有高校的名称、图标、所属部门等基本信息

→此种类型的表有多张，每张代表了该地区考生近几年的学校分数线情况

4.3 API 接口设计

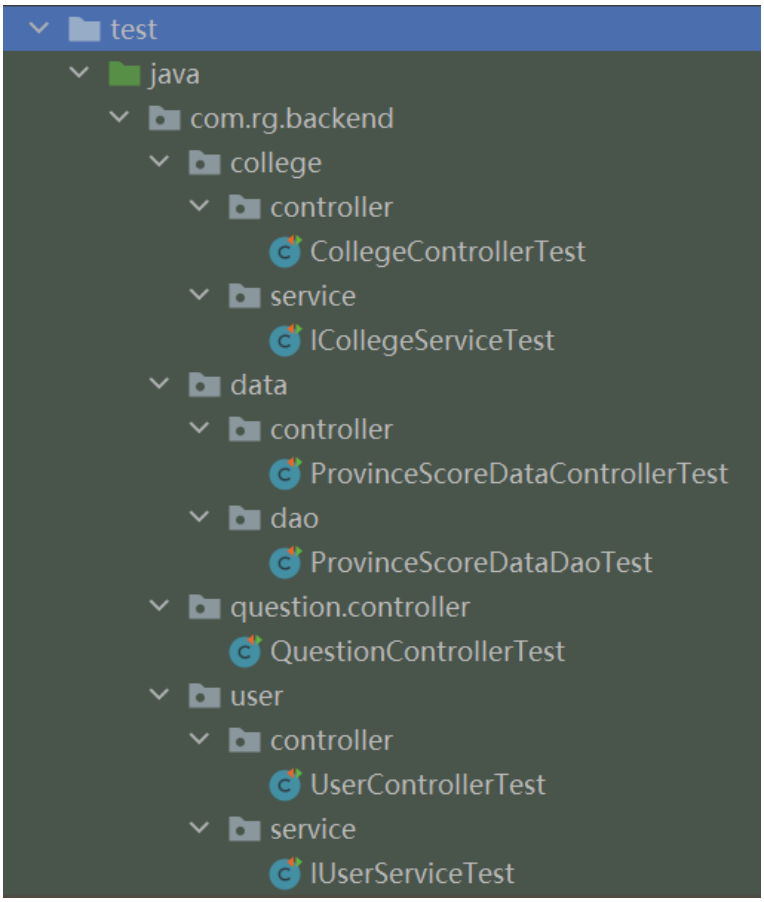
我们的 API 接口设计如下：

第五章 系统测试与部署

5.1 系统测试

5.1.1 单元测试

后端接口采用 Junit 进行单元测试,测试文件目录如下图:



其中 college 和 user 部分是采用的 MybatisPlus 的 ORM 框架,所以不测 Mapper 层，只对 Service 层和 Controller 层进行测试。data 部分由于有很多名称类似的数据表所以测试 Dao 层的 jdbc 和 Controller 层。question 部分将问题回答接口做了一层封装(该接口跑在实验室服务器上,通过端口转发才能访问),所以只测 Controller 层。

Service 层的测试使用@RunWith(SpringRunner.class)和@SpringBootTest 注解，Controller 层的测试在此基础上使用 MockMvc 模拟浏览器环境发送请求验证返回值是否正确。以下为 college 的 Service 和 Controller 测试两个测试函数示例。

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class ICollegeServiceTest {
    @Autowired
    ICollegeService collegeService;

    @Test
    public void getAllCollegeTest(){
        Map<String,Object> map = collegeService.getAllCollege( page: 1);
        long total = (long) map.get("total");
        assertNotNull(total);
        List<College> colleges = (List<College>) map.get("colleges");
        assertNotNull(colleges);
    }
```

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class CollegeControllerTest {

    private MockMvc mvc;
    @Autowired
    private WebApplicationContext wac;

    @Before
    public void init() { mvc = MockMvcBuilders.webAppContextSetup(wac).build(); }
    @Test
    public void queryAllCollege() throws Exception {
        String response = mvc.perform(MockMvcRequestBuilders.get( urlTemplate: "/college/all/1"))
                .andReturn().getResponse().getContentAsString();
        System.out.println(response);
        JSONObject map = (JSONObject) JSON.parse(response);
        String status = map.get("status").toString();
        JSONArray colleges = map.getJSONArray( key: "data");
        assertEquals( expected: "200",status);
        assertNotNull(colleges);
    }
```

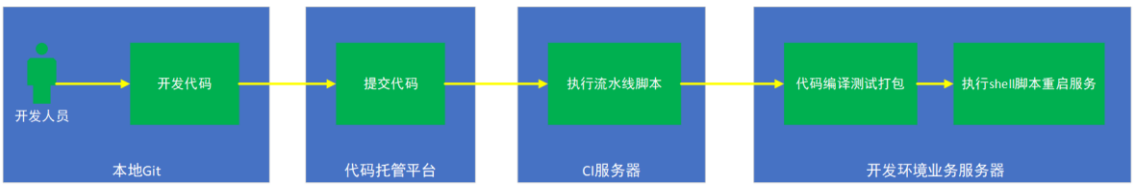
5.1.2 集成测试

使用 SwaggerUI 和 Postman 工具对接口进行集成测试。

5.1.3 系统测试

功能测试：

5.2 持续集成与部署



本项目实现了后端代码的持续集成与一键部署，实现流程如上图。开发人员在本地写完代码后通过 git 提交到代码仓库。然后在 Jenkins 的流水线项目中点击构建，会自动运行 Jenkinsfile 脚本的内容，实现服务器对后端代码的拉取、编译、测

试、打包，并执行重启服务的 shell 脚本实现后端服务的自动部署。CI/CD 的搭建步骤请参考项目部署文档。

5.2.1 代码托管

采用 git + github 的方式：

开发代码都存在小组的 github 仓库 https://github.com/zqrzxwkr-group 中。成员完成某个模块或阶段性的任务后，将代码提交到相应 github 仓库便于管理。

5.2.2 运维配置托管

采用 git + github 的方式。

Jenkins 流水线项目的 Jenkinsfile 脚本文件和重启服务的 restart_jar.sh 脚本与后端代码保存在一个仓库内。配置文件与代码一起进行版本管理，这样在出现错误想回滚到以前的版本时，只用 git 回退到原版本，项目一定能构建成功。

本项目后端的 Jenkinsfile 内容如下：

```
def VERSION = "v0"
def GITREPO = "git@github.com:zqrzxwkr-group/backend.git"
pipeline{
    agent { label 'master' }
    tools {
        jdk 'jdk11'
        maven 'maven3.6.3'
    }
    stages{
        stage("代码克隆"){
            steps{
                sh "cd /var/lib/jenkins/workspace/backend-pipeline && rm -rf ./*"
                git branch: 'main', credentialsId: '1', url: "${GITREPO}"
                echo "代码克隆完成"
            }
        }

        stage("代码构建打包"){
            steps{
                sh "cd /var/lib/jenkins/workspace/backend-pipeline/backend && mvn clean package"
                echo "代码构建打包完成"
            }
```

```
        }

        stage("更新服务"){
            steps{
                sh "cd /var/lib/jenkins/workspace/backend-pipeline/backend"
                sh "chmod +x restart_jar.sh"
                sh "./restart_jar.sh"
                echo "更新服务"
            }
        }
    }
}
```

重启服务的 restart_jar.sh 的脚本内容如下：

```
export JENKINS_NODE_COOKIE=dontkillme
#!/bin/bash
APP_NAME=$(pwd)/backend/target/backend-0.0.1-SNAPSHOT.jar

#检查程序是否在运行
is_exist(){
  pid=`ps -ef|grep $APP_NAME|grep -v grep|awk '{print $2}'`
  #如果不存在返回 1，存在返回 0
  if [ -z "${pid}" ]; then
   return 1
  else
    return 0
  fi
}

#启动方法
start(){
  is_exist
  if [ $? -eq 0 ]; then
    echo "${APP_NAME} is already running. pid=${pid}"
  else
    echo "启动${APP_NAME}"
    nohup java -jar ${APP_NAME} > log.out 2>&1 &
```

```
        echo "启动结束"
    fi
}

#停止方法
stop(){
    is_exist
    if [ $? -eq "0" ]; then
        kill -9 $pid
    else
        echo "${APP_NAME} is not running"
    fi
}

#输出运行状态
status(){
    is_exist
    if [ $? -eq "0" ]; then
        echo "${APP_NAME} is running. Pid is ${pid}"
    else
        echo "${APP_NAME} is NOT running."
    fi
}

#重启
restart(){
    stop
    sleep 5
    start
}

restart
```

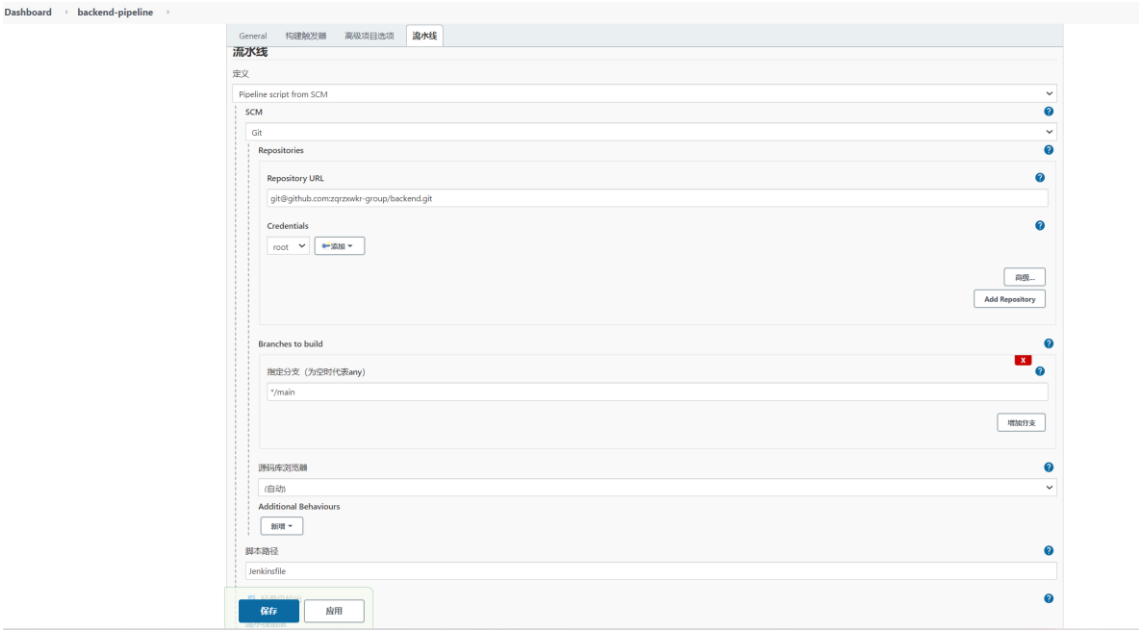
5.2.3 CI 服务器

本项目选用 Jenkins 作为 CI 服务器。

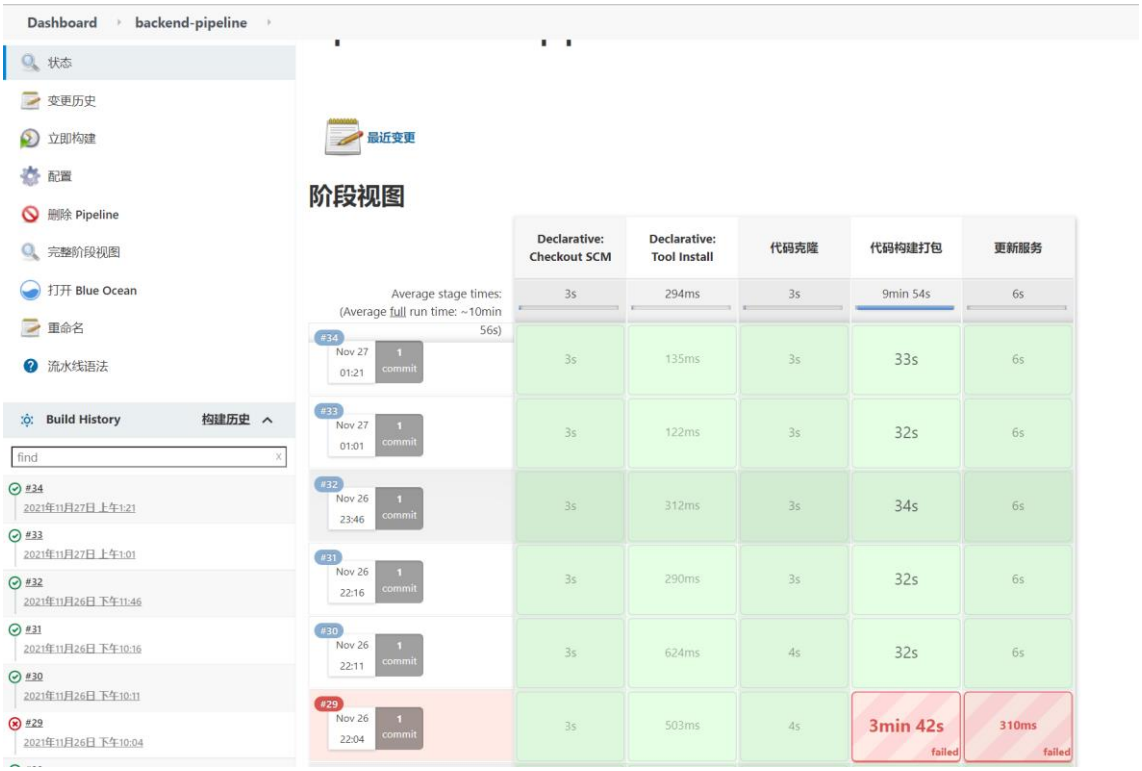
- CI 服务器安装配置：首先在服务器安装 jdk 和 Jenkins，除默认安装 Jenkins

推荐插件外还需安装 SSH、Blue Ocean 等插件。然后在 Jenkins 的全局工具中配置 jdk 和 maven 的路径，以及配置 github 项目的免密登录 Secret。

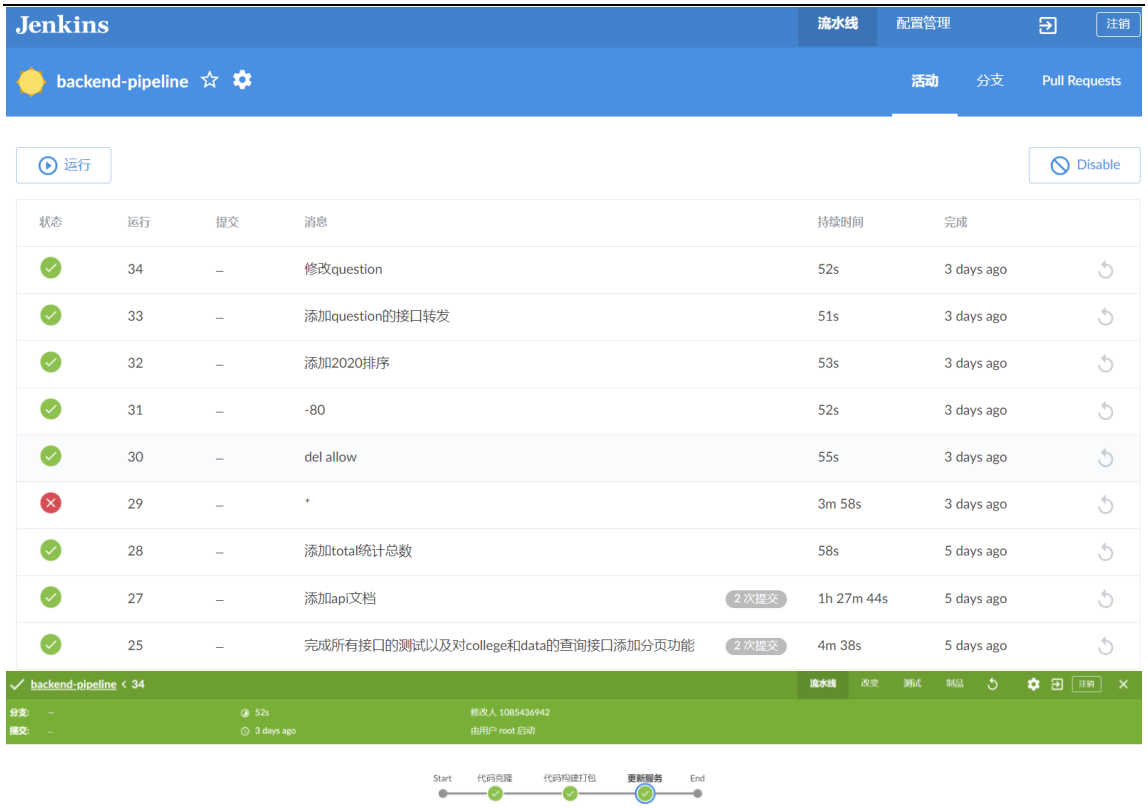
- 创建后端流水线项目 backend-pipeline，配置如下图。



- 每次提交代码后点击构建后会执行 Jenkinsfile 流水线脚本。若出错则中断，可以通过每个 stage 的日志输出来定位错误，若成功则能将新的后端代码部署到服务器上。此功能的界面如下图:



Blue Ocean 插件提供了一个更好看的 UI，界面如下图：



5.2.4 回归测试

在 Jenkins 流水线脚本中会运行 mvn clean package 命令对后端项目进行自动化回归测试，来验证新增的部分是否有 bug 或影响原先部分的结果。

第六章 遇到的问题与解决方案

1.后端 maven 静态资源过滤的问题，在 build 中添加如下部分解决：

```
    <!-- 静态资源过滤 -->
    <resources>
        <resource>
            <directory>src/main/resources</directory>
            <includes>
                <include>**/*.yml</include>
                <include>**/*.properties</include>
                <include>**/*.xml</include>
            </includes>
        </resource>
    </resources>
</build>
```

2.maven-surefire-plugin 插件 2.22 版本自动跳过测试的问题。将版本号 2.10 后解决。
3.Jenkins 执行流水线脚本后自动杀死了创建的后端项目进程的问题。这是 Jenkins 防止僵尸进程的一种机制，通过在 restart_jar.sh 文件的第一行添加以下语句解决：

export JENKINS_NODE_COOKIE=dontkillme

4.前端 Vue 项目部署到 Nginx 之后 CSS 样式出现覆盖问题。配置引用路径，通过相对路径引用和 src 目录同级的 static 静态目录文件。