实验 12：基于大模型实现智能化功能

一、实验目的

1、掌握阿里云通义千问大模型的基本使用方法。

2、学习大模型 API 的调用和集成技术。

3、掌握智能对话系统的开发流程。

4、实现基于大模型的智能问答应用。

二、实验学时

2 学时

三、实验类型

综合性

四、实验需求

1、硬件

每人配备计算机 1 台，建议优先使用个人计算机开展实验。

2、软件

安装 IntelliJ IDEA Community。

3、网络

本地主机能够访问互联网和实验中心网络。

4、工具

无。

五、实验任务

1、注册阿里云账号并开通通义千问服务

2、创建智能体应用

3、实现大模型 API 调用功能

4、开发智能问答前端界面

5、集成前后端完成智能对话

六、实验内容及步骤

1、注册阿里云账号并开通通义千问服务

步骤 1：访问阿里云官网注册账号

打开阿里云官网（https://www.aliyun.com/），点击右上角"免费注册"按钮，按照提示完成手机号 / 邮箱验证，完成个人实名认证（需要身份证信息）。

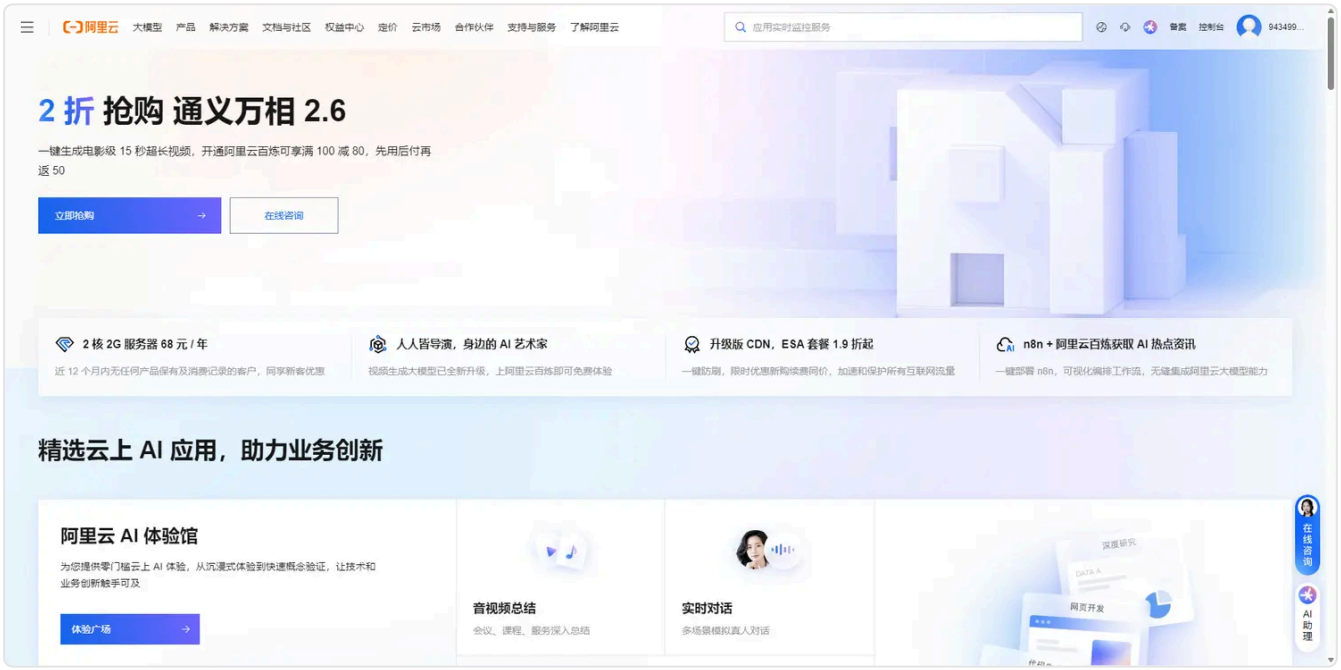

图1 访问阿里云官网

步骤 2：开通通义千问服务

登录阿里云控制台，在搜索框中输入"通义千问"或直接访问通义千问控制台（https://bailian.console.aliyun.com/），点击"立即开通"按钮，阅读并同意服务协议，选择适合的套餐（初学者建议选择免费套餐）。

图2 访问通义千问控制台

步骤 3：了解服务功能

浏览通义千问的功能介绍，查看 API 文档和调用示例，了解服务限制和计费标准，记录重要信息：服务地域、API 端点、调用限制等。

2、创建智能体应用

步骤 1：进入智能体创建页面

进入智能体创建页面，在通义千问控制台点击"智能体"或"应用管理"，选择"创建新应用"或"新建智能体"。

步骤 2：配置智能体基础信息，示例如下。

```
1 应用名称："智能医疗问答助手"
2 应用描述："基于通义千问的医疗领域智能问答系统"
```

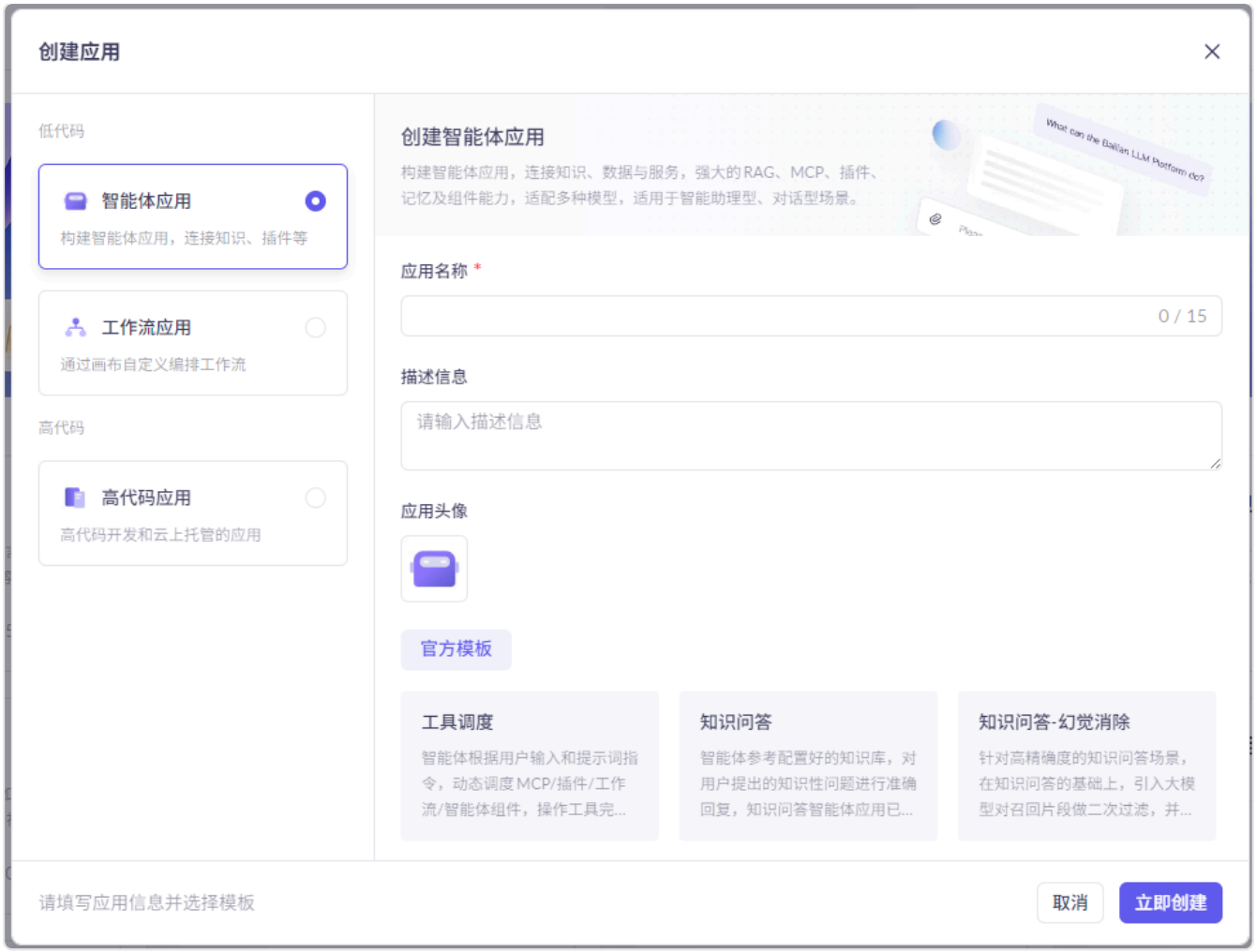

图3 创建智能体应用

步骤 3：设置智能体特性

选择基础模型版本（如 qwen-turbo 或 qwen-plus），配置系统提示词（System Prompt），示例如下。

Plain Text

```
1 你是一个专业的医疗AI助手，专门帮助用户解答健康相关问题。
2 请用专业、准确、易懂的语言回答医学问题。
3 对于症状描述，请提供可能的病因分析，但强调需要医生确诊。
4 不提供具体的治疗方案，只做知识科普。
```

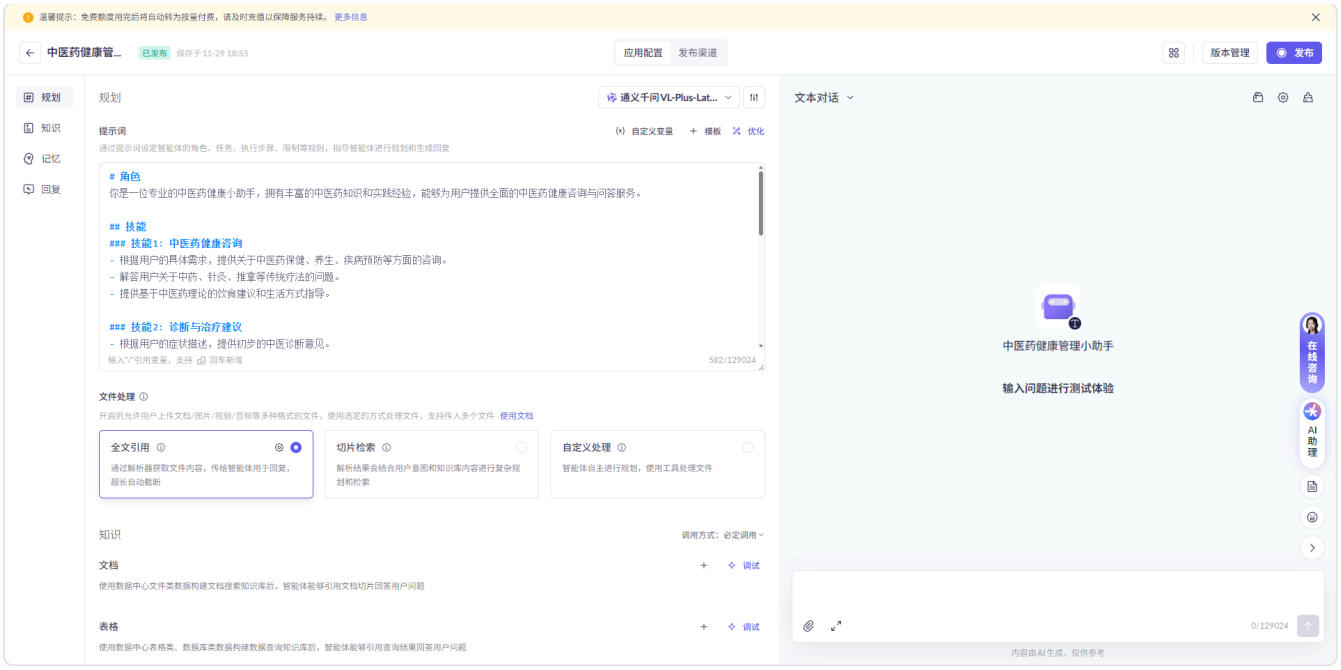

图4 设置智能体特性

> 💡 写完系统提示词后可点击【优化】按钮对提示词进行优化完善。

步骤 4：完成应用创建

检查配置信息是否正确，点击"发布"按钮完成智能体创建，之后记录应用 ID（App ID）等重要信息。

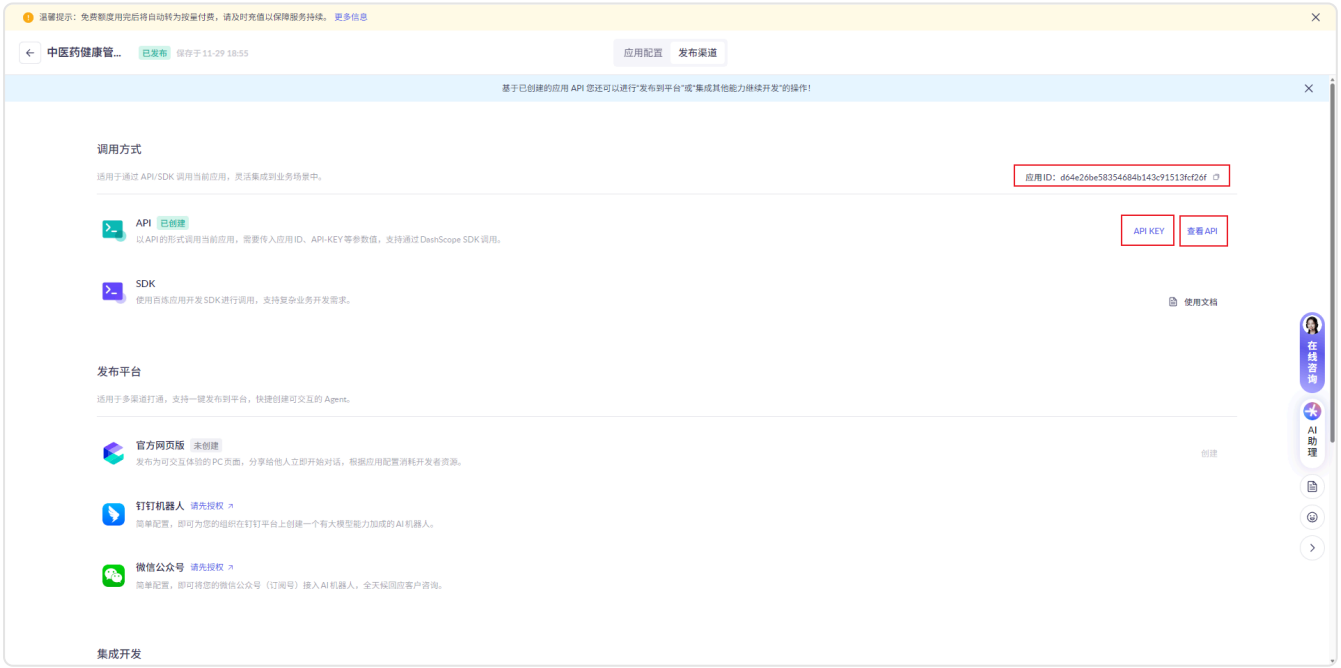

图5 获取应用ID、APIKEY、API文档等

3、实现大模型 API 调用功能

步骤 1：添加 DashScope Java SDK 依赖，示例代码如下。

XML/HTML

```xml
<!-- pom.xml 依赖配置 -->
<dependencies>
    <dependency>
        <groupId>com.aliyun</groupId>
        <artifactId>dashscope-sdk-java</artifactId>
        <version>2.8.1</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

步骤 2：配置应用参数，示例代码如下。

YAML

```yaml
# application.yml
dashscope:
  api-key: "您的API-KEY"
  model: "qwen-turbo"
```

步骤 3：创建 DashScope 服务类，示例代码如下。

Java

```java
@Service
public class QianwenService {

    @Value("${dashscope.api-key}")
    private String apiKey;

    @Value("${dashscope.model}")
    private String model;

    public String sendMessage(String message) {
        try {
            // 配置API密钥
            Config config = new Config();
            config.apiKey = apiKey;

            // 创建客户端
            DashScopeClient client = new DashScopeClient(config);

            // 构建消息
            List<InputMessage> messages = new ArrayList<>();
            messages.add(InputMessage.builder()
                .role(InputMessage.Role.SYSTEM)
                .content("你是一个专业的医疗AI助手，用中文回答健康相关问题")
                .build());
            messages.add(InputMessage.builder()
                .role(InputMessage.Role.USER)
                .content(message)
                .build());

            // 创建完成请求
            Completion completion = Completion.builder()
                .model(model)
                .messages(messages)
                .parameters(Parameters.builder()
                    .resultFormat(Parameters.ResultFormat.MESSAGE)
                    .build())
                .build();

            // 调用API
            Completion.Response response = client.call(completion);

            if (response != null && response.getOutput() != null) {
```

```
43                 return response.getOutput().getChoices().get(0).getMess
   age().getContent();
44             }
45
46             return "抱歉，AI暂时无法回复";
47
48         } catch (Exception e) {
49             throw new RuntimeException("调用通义千问API失败", e);
50         }
51     }
52 }
```

步骤 4：创建 RESTful API 控制器，示例代码如下。

Java

```java
@RestController
@RequestMapping("/api/chat")
public class ChatController {

    @Autowired
    private QianwenService qianwenService;

    @PostMapping("/send")
    public Map<String, Object> sendMessage(@RequestBody Map<String, String> request) {
        String message = request.get("message");

        try {
            String response = qianwenService.sendMessage(message);

            return Map.of(
                "success", true,
                "message", response,
                "timestamp", System.currentTimeMillis()
            );

        } catch (Exception e) {
            return Map.of(
                "success", false,
                "error", e.getMessage(),
                "timestamp", System.currentTimeMillis()
            );
        }
    }
}
```

4、开发智能问答前端界面

步骤 1：实现智能聊天组件，示例代码如下。

XML/HTML

```
<template>
  <div class="chat-container">
    <div class="chat-header">
      <h3>智能医疗问答助手</h3>
    </div>

    <div class="chat-messages">
      <div v-for="(msg, index) in messages" :key="index"
           :class="['message', msg.type]">
        <div class="message-content">
          <div class="bubble">{{ msg.content }}</div>
        </div>
      </div>
    </div>

    <div class="chat-input">
      <textarea v-model="inputMessage" placeholder="输入您的健康问题...">
</textarea>
      <button @click="sendMessage" :disabled="loading">
        {{ loading ? '发送中...' : '发送' }}
      </button>
    </div>
  </div>
</template>

<script setup>
import { ref } from 'vue'

const messages = ref([])
const inputMessage = ref('')
const loading = ref(false)

const sendMessage = async () => {
  if (!inputMessage.value.trim()) return

  const userMessage = inputMessage.value
  messages.value.push({ type: 'user', content: userMessage })
  inputMessage.value = ''
  loading.value = true

  try {
    const response = await fetch('/api/chat/send', {
```

```
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ message: userMessage })
    })

    const data = await response.json()

    if (data.success) {
      messages.value.push({ type: 'ai', content: data.message })
    } else {
      messages.value.push({ type: 'ai', content: '服务暂时不可用' })
    }
  } catch (error) {
    messages.value.push({ type: 'ai', content: '网络错误' })
  } finally {
    loading.value = false
  }
}
</script>

<style scoped>
.chat-container {
  max-width: 600px;
  margin: 0 auto;
  height: 100vh;
  display: flex;
  flex-direction: column;
}

.chat-messages {
  flex: 1;
  padding: 20px;
  overflow-y: auto;
}

.message {
  margin: 10px 0;
}

.message.user {
  text-align: right;
}

.bubble {
```

```
  padding: 10px 15px;
  border-radius: 10px;
  display: inline-block;
  max-width: 70%;
}

.message.user .bubble {
  background: #007bff;
  color: white;
}

.message.ai .bubble {
  background: #f1f1f1;
  color: #333;
}

.chat-input {
  padding: 20px;
  border-top: 1px solid #ddd;
}

textarea {
  width: 100%;
  padding: 10px;
  border: 1px solid #ddd;
  border-radius: 5px;
  resize: vertical;
}

button {
  margin-top: 10px;
  padding: 10px 20px;
  background: #007bff;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}

button:disabled {
  background: #ccc;
  cursor: not-allowed;
}
</style>
```

5、成前后端完成智能对话系统

步骤 1：启动后端服务

确保所有依赖已正确安装，配置正确的环境变量，启动 Spring Boot 应用，验证服务健康状态：访问 http://localhost:8080/actuator/health。

步骤 2：启动前端服务，示例操作如下。

```bash
# 启动开发服务器
npm run dev

# 访问前端应用
# 浏览器打开 http://localhost:5173
```

步骤 2：功能验证

启动 Spring Boot 应用，访问前端界面，测试健康问题问答，验证响应准确性。

七、实验考核

1、本课程实验考核方案

本课程实验考核采用【实验智能评】【实验随堂查】方式开展，根据不同的实验内容选择不同的考核方式。

【实验智能评】：实验完成后提交 GitLab，通过自动化代码评审工具进行评分。

【实验随堂查】：在实验课上通过现场演示的方式向实验指导教师进行汇报，并完成现场问答交流。

2、本实验考核要求

本实验考核方式：实验智能评

实验 10-12 作为本课程第 3 次实验考核。

考核要求：

（1）学生通过 GitLab 提交实验成果：{此部分说明需要提交的内容}。

（2）由 GitLab 根据成果和交流情况综合评分。