河南中医药大学信息技术学院（智能医疗行业学院）智能医学工程专业《计算机程序设计》课程

第14章：开发服务软件

王猛

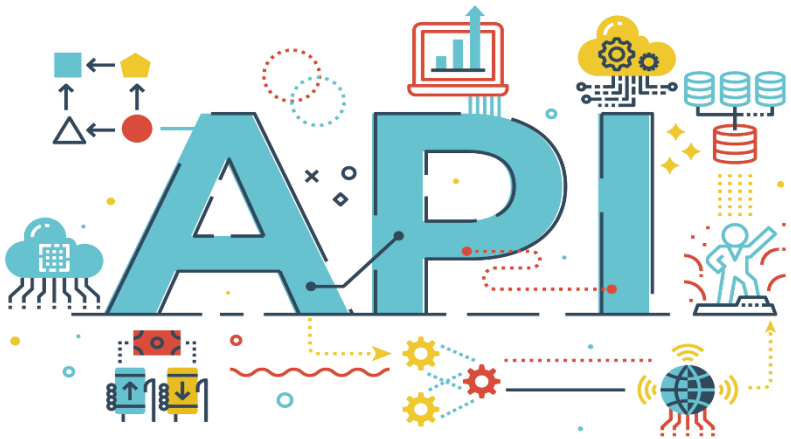
河南中医药大学信息技术学院（智能医疗行业学院）与河南方和信息科技有限公司 联合建设

河南中医药大学信息技术学院智能医疗教研室

https://aitcm.hactcm.edu.cn

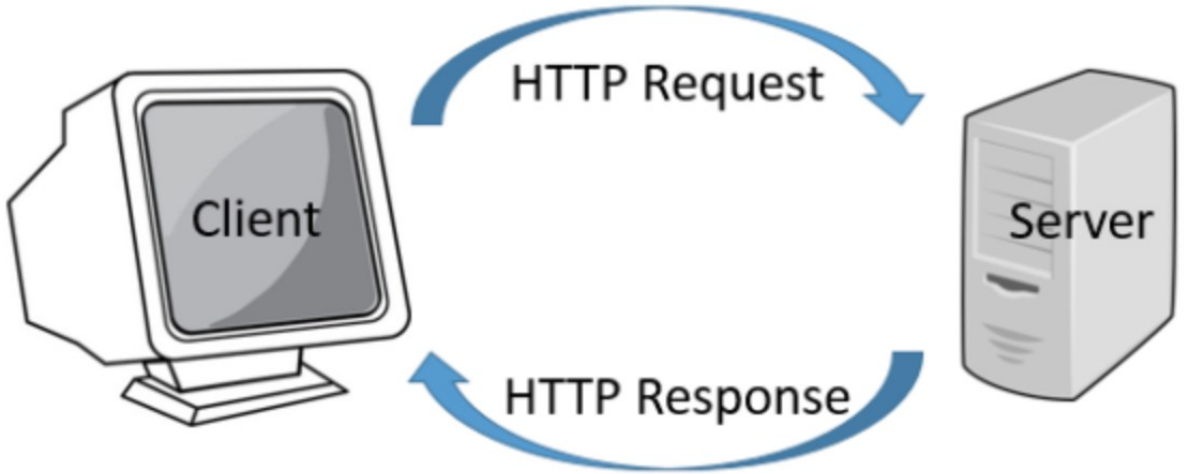
2025/2/16

本章概要



14.1 调用第三方接口

- 网络请求的定义
  - 网络请求是客户端（如浏览器、移动应用、桌面应用或服务器）与服务器之间交换数据的过程。这些请求通常通过HTTP（超文本传输协议）或HTTPS（HTTP的安全版本）等协议进行，允许客户端从服务器获取数据或向服务器发送数据。

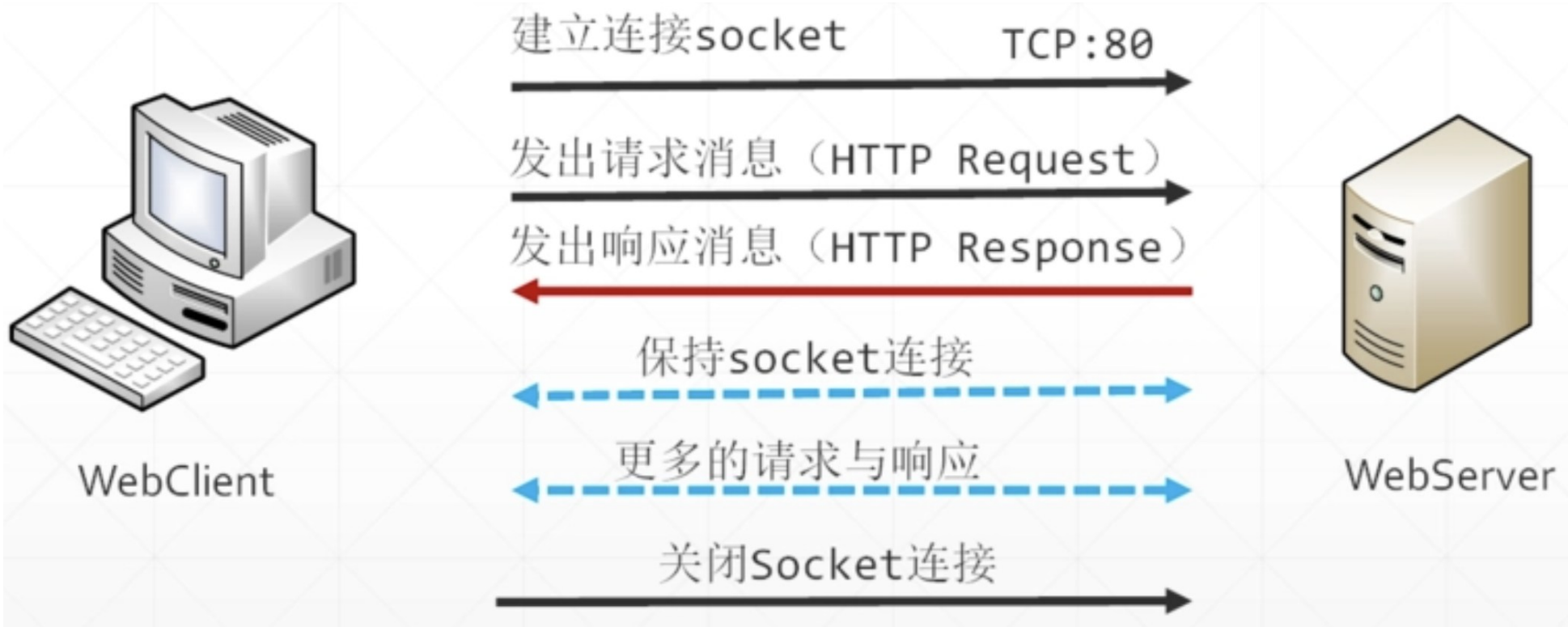


14.1 调用第三方接口

- 网络请求的基本步骤
  - **建立连接**：客户端首先与服务器建立网络连接。这通常是通过TCP/IP协议完成的，而HTTP/HTTPS则是基于TCP/IP的应用层协议。
  - **发送请求**：客户端向服务器发送一个HTTP请求。请求中包含了请求的方法（如GET、POST）、请求的URI（统一资源标识符，用于定位资源）、HTTP版本以及请求头（包含了一些额外的信息，如客户端类型、请求的内容类型等）和请求体（对于POST和PUT请求，请求体会包含要发送的数据）。
  - **处理请求**：服务器接收到请求后，会根据请求的方法、URI等信息来决定如何处理该请求。这可能涉及到读取数据库、计算数据、访问文件系统等操作。
  - **发送响应**：服务器处理完请求后，会向客户端发送一个HTTP响应。响应中包含了状态码（表示请求是否成功，以及失败的原因）、响应头和响应体（包含了请求的结果或数据）。
  - **关闭连接**：客户端接收到响应后，可以根据需要关闭与服务器的连接。然而，对于HTTP/1.1和HTTP/2等协议，连接可能会被复用，以支持多个请求/响应的交换。

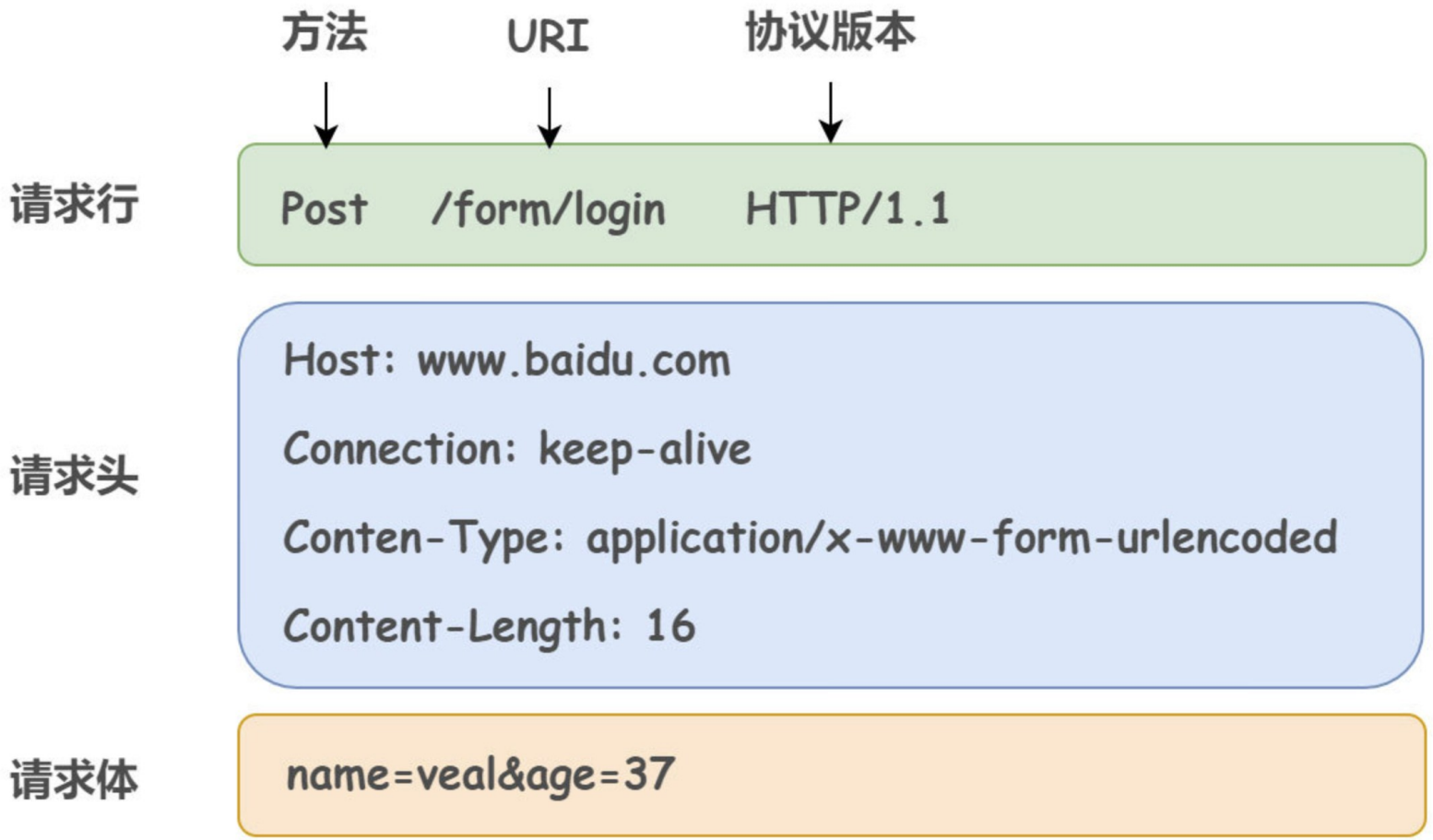
14.1 调用第三方接口

- 网络请求示例



14.1 调用第三方接口

- HTTP数据请求

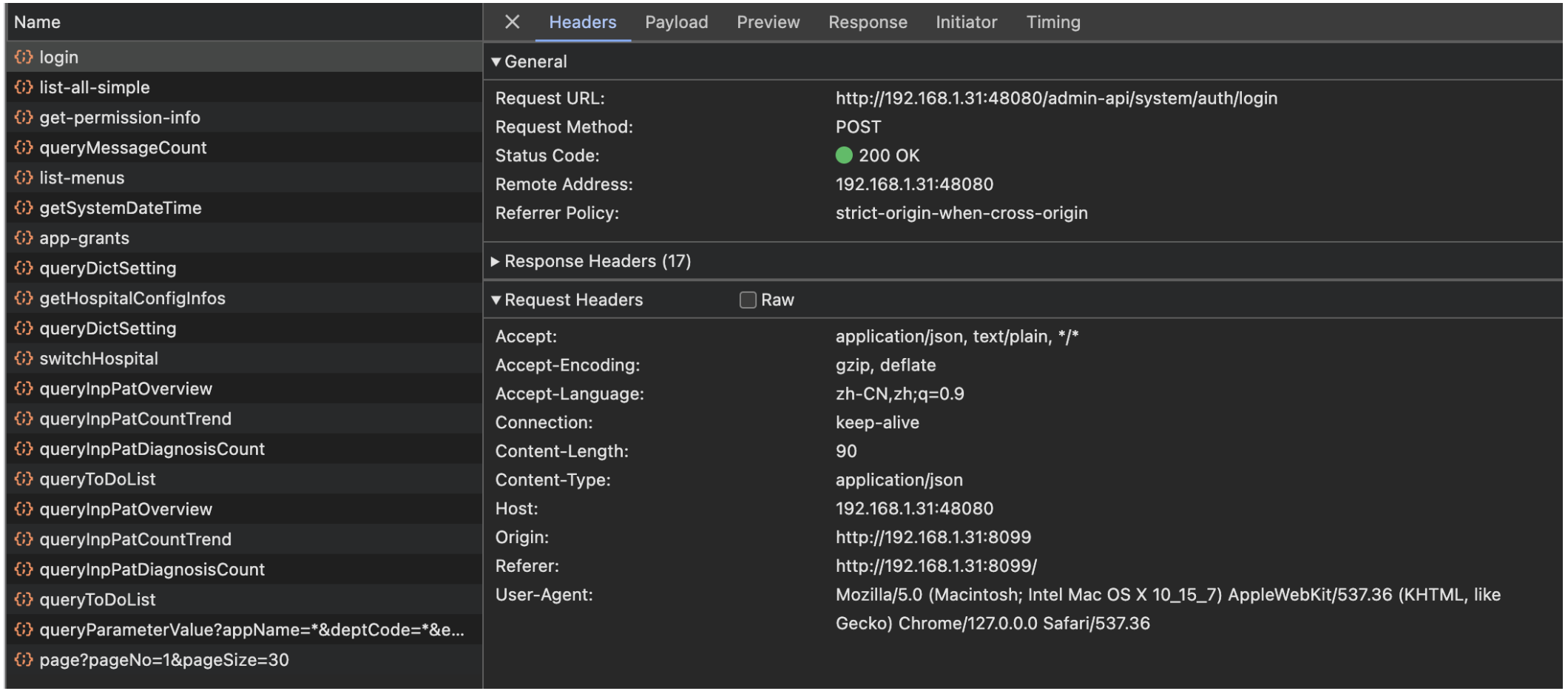


14.1 调用第三方接口

- 常用的HTTP请求方法
  - **GET**：用于请求服务器发送资源。GET请求通常用于从服务器检索数据。
  - **POST**：用于向服务器提交数据。POST请求通常用于提交表单数据、上传文件等。
  - **PUT**：用于更新服务器上的资源。PUT请求通常包含要更新的资源的完整表示。
  - **DELETE**：用于删除服务器上的资源。
  - HEAD：与GET类似，但只返回响应头，不返回响应体。
  - OPTIONS：用于获取目标资源所支持的通信选项。
  - TRACE：回显服务器收到的请求，主要用于测试或诊断。
  - CONNECT（HTTP/1.1）：用于建立到由目标资源标识的服务器的隧道。
  - PATCH：用于对资源进行部分修改。

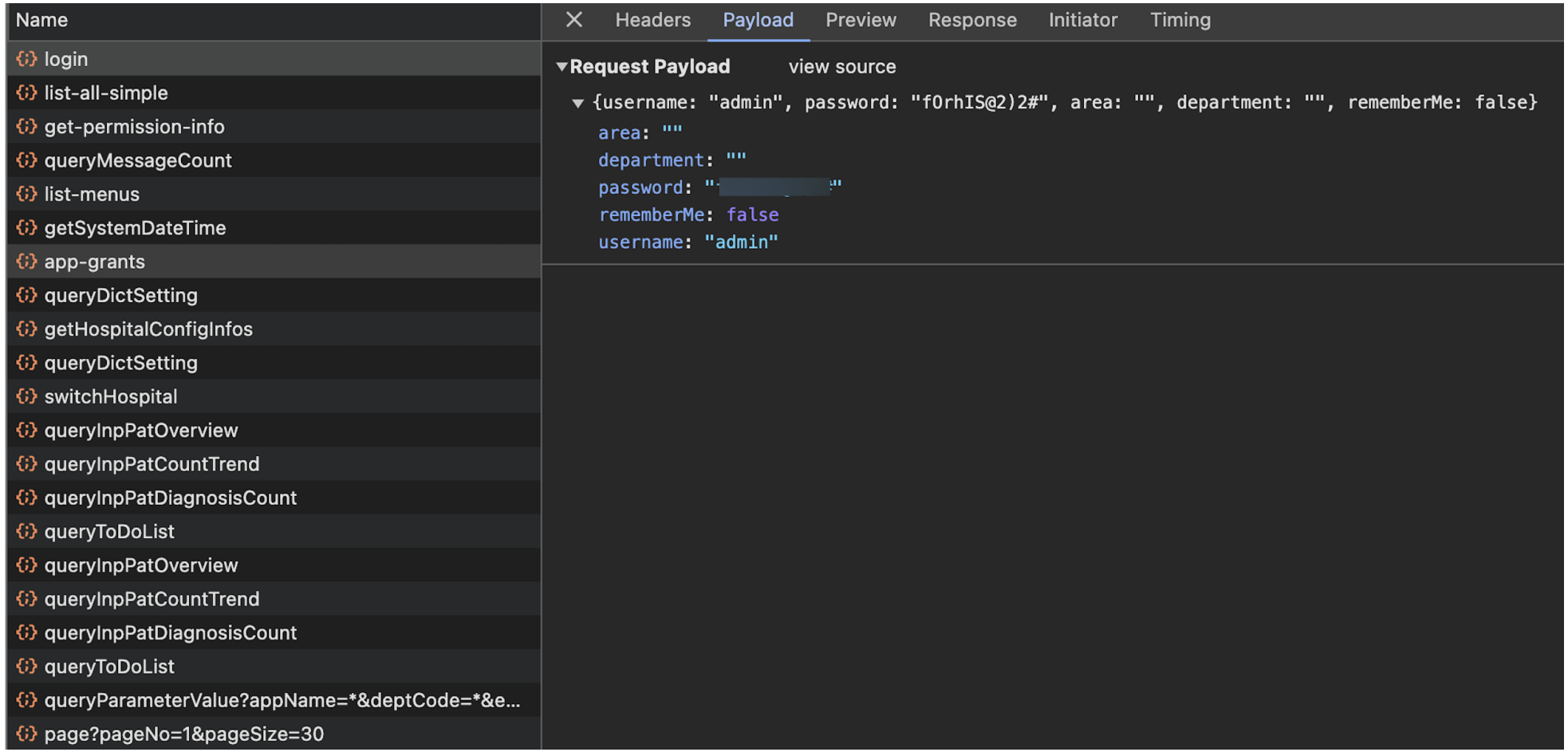
14.1 调用第三方接口

- HTTP请求示例



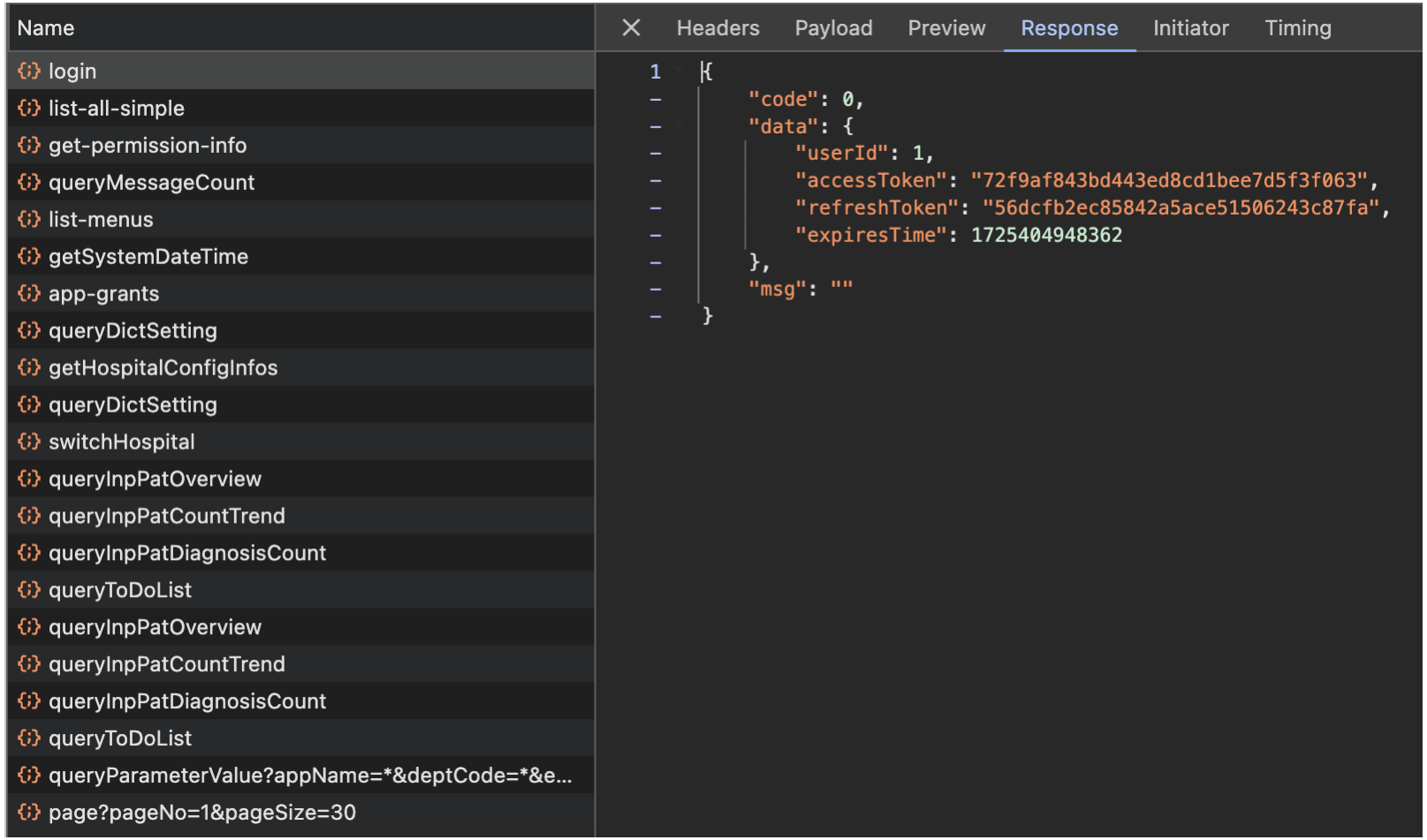
14.1 调用第三方接口

- HTTP请求示例



14.1 调用第三方接口

- HTTP请求示例



14.1 调用第三方接口

- 发起网络请求的工具和库
  - 浏览器：内置支持HTTP请求，用户通过浏览网页或表单提交来发起请求。
  - JavaScript（Ajax/Fetch/XMLHttpRequest）：用于在Web应用中异步发起HTTP请求。
  - 移动应用（如Android、iOS）：使用各自的HTTP客户端库（如Android的OkHttp，iOS的URLSession）来发起请求。
  - 桌面应用：使用各种编程语言提供的HTTP客户端库（如Java的HttpClient，Python的requests）来发起请求。
  - 命令行工具：如curl、wget等，可以直接在命令行中发起HTTP请求。
  - 编程语言内置库：许多编程语言都提供了内置的网络请求库或模块，如Python的urllib、Node.js的http模块等。

14.1 调用第三方接口

- 网络请求注意事项
  - 安全性：确保网络请求使用HTTPS协议来保护数据传输的安全性。
  - 错误处理：客户端代码需要能够处理网络错误、服务器错误和超时等情况。
  - 异步处理：在Web和移动应用中，网络请求通常是异步的，以避免阻塞用户界面。
  - 缓存：合理利用HTTP缓存机制可以减少不必要的网络请求，提高应用性能。
  - 请求限制：注意遵守API的使用限制，如请求频率、请求大小等。

14.2 Java调用第三方接口的常见方法

- HttpClient介绍
  - 在Java中，HttpClient 是一个用于发送HTTP请求的客户端类。
  - 不过，需要注意的是，Java标准库（Java SE）在不同版本中对于HTTP客户端的支持有所不同。在Java 11之前，Java没有内置的HttpClient类，但你可以使用第三方库如Apache HttpClient或OkHttp来发送HTTP请求。
  - 从Java 11开始，Java引入了java.net.http.HttpClient作为标准库的一部分，提供了异步和同步的HTTP客户端功能。

14.2 Java调用第三方接口的常见方法

- HttpClient请求示例(JDK 11开始)

```java
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.net.http.HttpResponse.BodyHandlers;
import java.util.concurrent.CompletableFuture;

public class AsyncHttpClientExample {
    public static void main(String[] args) {
        HttpClient client = HttpClient.newHttpClient();
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create("http://example.com"))
                .header("Accept", "text/html")
                .GET()
                .build();

        CompletableFuture<HttpResponse<String>> future = client.sendAsync(request, BodyHandlers.ofString());

        future.thenApply(HttpResponse::body)
            .whenComplete((body, throwable) -> {
                if (throwable != null) {
                    throwable.printStackTrace();
                } else {
                    System.out.println(body);
                }
            });

        // 注意：异步操作是非阻塞的，因此主线程会立即继续执行。
        // 在实际应用中，你可能需要等待异步操作完成，或者继续执行其他任务。
    }
}
```

14.2 Java调用第三方接口的常见方法

- Spring框架介绍
  - Spring Framework，这是一个开源的Java/Java EE全功能栈（full-stack）的应用程序框架。Spring框架以控制反转（IoC）和面向切面编程（AOP）为内核，提供了展现层Spring MVC和持久层Spring JDBC以及业务层事务管理等众多的企业级应用技术，还能整合开源世界众多著名的第三方框架和类库，逐渐成为使用最多的企业应用开源框架。
  - Spring的主要特点包括：
  - 轻量级：Spring框架的初衷就是解决企业级应用开发的复杂性，它旨在简化开发，因此设计得十分轻量。
  - 控制反转（IoC）：IoC容器是Spring框架的核心，它将对象的创建和依赖关系的维护交给容器来管理，降低了组件之间的耦合度。
  - 面向切面编程（AOP）：Spring支持面向切面编程，可以在不修改源代码的情况下，增强代码功能，如日志、事务等。
  - 一站式解决方案：Spring框架提供了从表现层到持久层的全栈解决方案，可以[illegible]成。

14.2 Java调用第三方接口的常见方法

- Spring Project

Spring Framework

Provides core support for dependency injection, transaction management, web apps, data access, messaging, and more.

Spring Boot

Takes an opinionated view of building Spring applications and gets you up and running as quickly as possible.

Spring Cloud

Provides a set of tools for common patterns in distributed systems. Useful for building and deploying microservices.

Spring Security

Protects your application with comprehensive and extensible authentication and authorization support.

14.2 Java调用第三方接口的常见方法

- Spring框架的RestTemplate介绍
  - RestTemplate 是 Spring 框架中用于同步客户端HTTP请求的模板工具类。它简化了与HTTP服务的通信，并且以统一的方式封装了HTTP请求的发送和接收过程。通过 RestTemplate，开发者可以很方便地调用RESTful服务，而无需直接处理底层的HTTP连接和消息解析。
  - 主要特点
    - 简化HTTP调用：RestTemplate 提供了一系列的方法（如 getForObject、postForObject、exchange 等），用于发送HTTP请求并接收响应。
    - 自动转换响应体：可以自动将响应体转换为指定的Java对象类型，减少了手动解析JSON或XML的麻烦。
    - 支持多种HTTP方法：支持GET、POST、PUT、DELETE等HTTP方法。
    - 自定义请求头：可以方便地添加、修改HTTP请求头。
    - 错误处理：提供了错误处理机制，可以捕获并处理HTTP请求过程中发生的异常。

14.2 Java调用第三方接口的常见方法

- RestTemplate示例

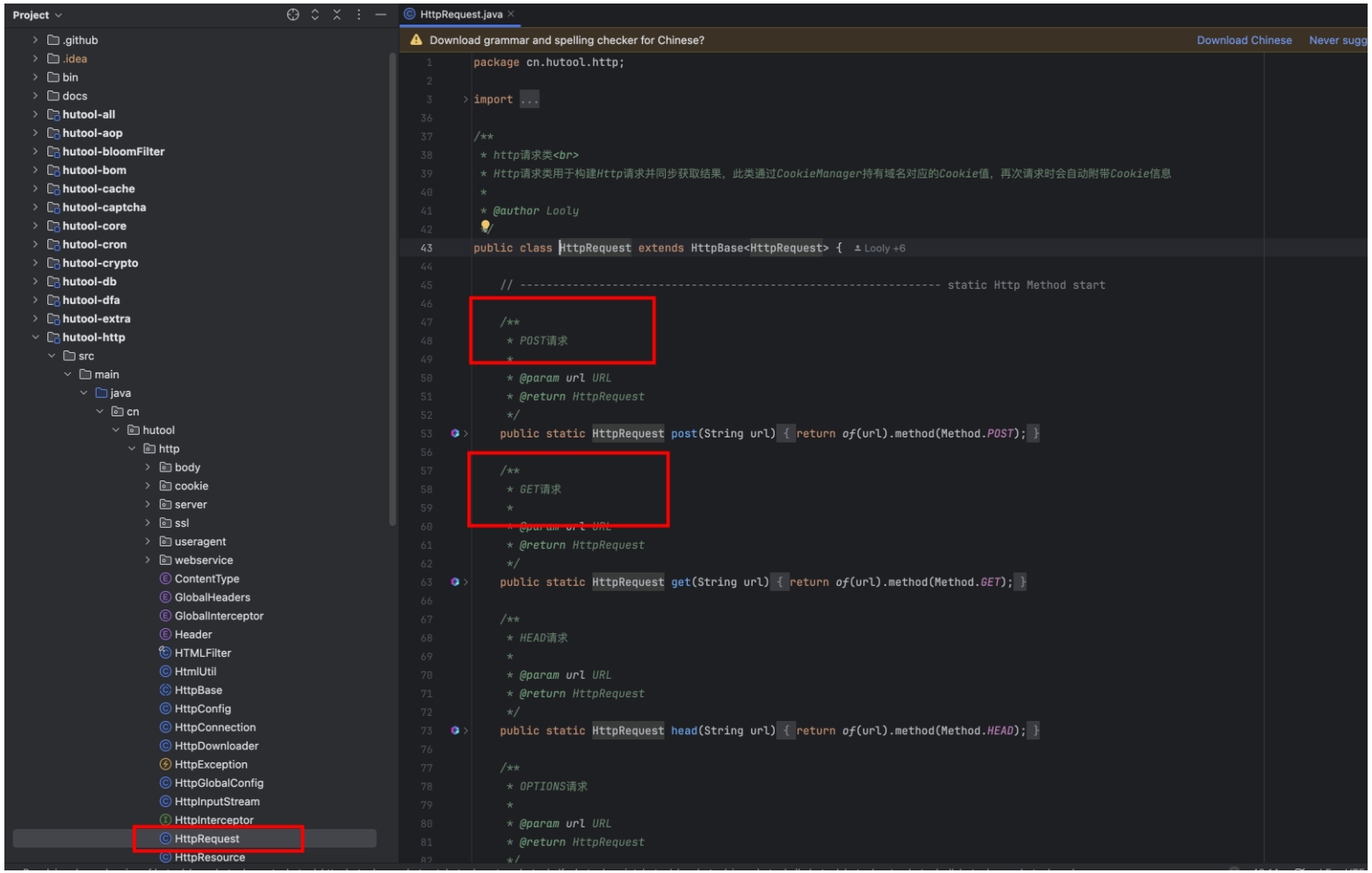
```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

@Service
public class MyService {

    @Autowired
    private RestTemplate restTemplate;

    public String getGreeting() {
        // 调用http://example.com/greeting服务，并期望返回String类型的响应体
        return restTemplate.getForObject("http://example.com/greeting", String.class);
    }
}

// 配置RestTemplate Bean
@Configuration
public class AppConfig {

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

14.2 Java调用第三方接口的常见方法

- 第三方API框架
  - Hutool
  - 网站：https://www.hutool.cn/
  - Hutool是一个功能丰富且易用的Java工具库，通过诸多实用工具类的使用，旨在帮助开发者快速、便捷地完成各类开发任务。 这些封装的工具涵盖了字符串、数字、集合、编码、日期、文件、IO、加密、数据库JDBC、JSON、HTTP客户端等一系列操作， 可以满足各种不同的开发需求。

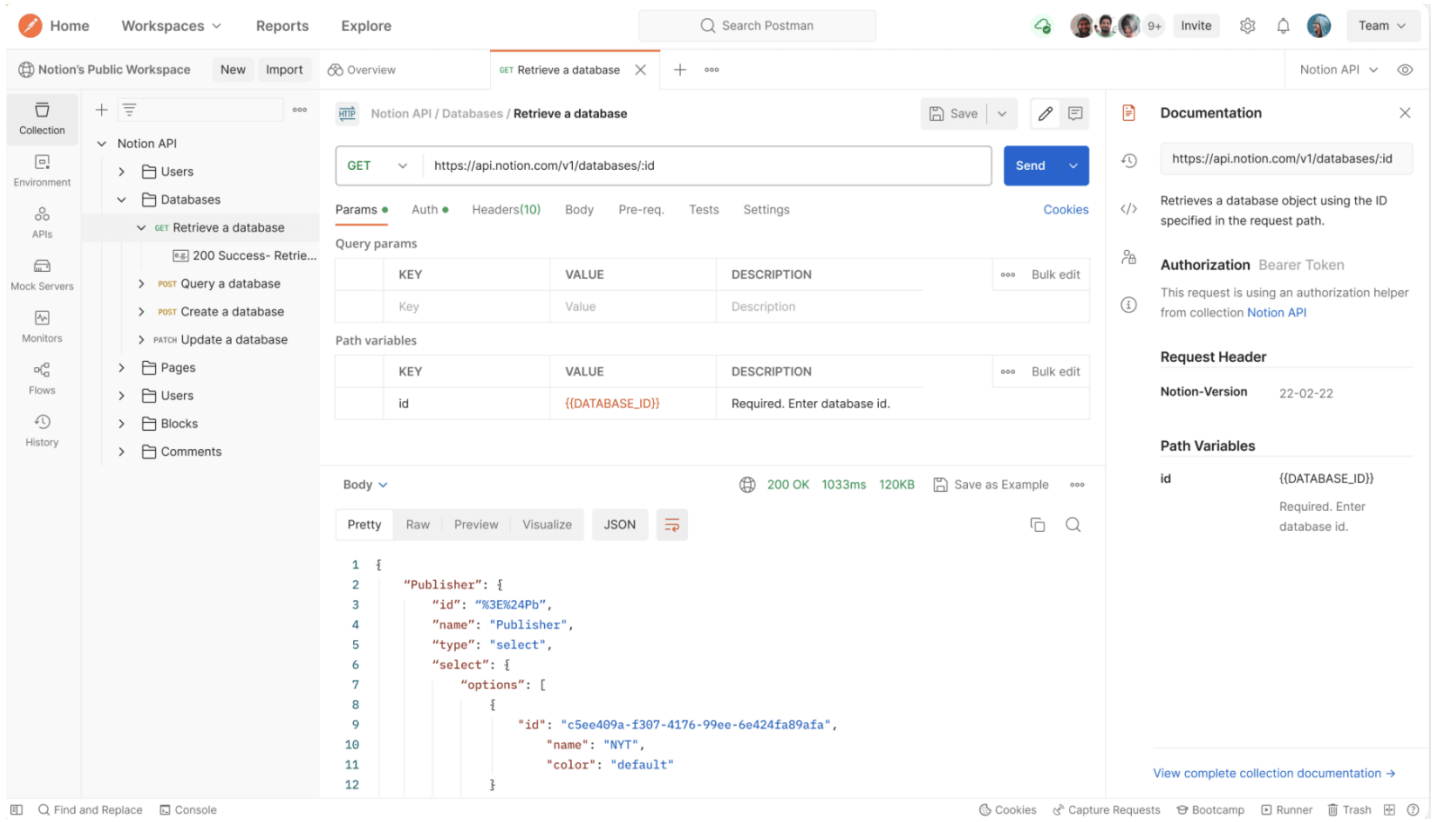
14.2 Java调用第三方接口的常见方法

- Hutool示例



14.2 Java调用第三方接口的常见方法

- 其他第三方接口调试工具
  - Postman https://www.postman.com/
  - Apifox https://apifox.com/



14.3 调用第三方天气API

- 网址： https://www.juhe.cn/docs/api/id/73
- 功能介绍：

功能介绍

温度查询

获取指定城市的当前温度信息，帮助用户了解实时气温。

湿度查询

提供指定城市的当前湿度情况，让用户知晓空气湿度状况。

空气质量查询

获取指定城市的空气质量指数（AQI），让用户知晓空气污染程度。

天气状况查询

提供指定城市当前的天气状况，如晴天、多云、阴天等，让用户了解当下天气。

14.3 调用第三方天气API

- 接口地址：http://apis.juhe.cn/simpleWeather/query
- 请求方式：http get/post
- 返回类型：json
- 请求参数说明：

名称	必填	类型	说明
city	是	string	要查询的城市名称/id，城市名称如：温州、上海、北京，需要utf8 urlencode
key	是	string	在个人中心->我的数据,接口名称上方查看

14.3 调用第三方天气API

14.3.1 确定API端点和参数

- 返回参数说明：

名称	类型	说明
error_code	int	返回码，0为查询成功
reason	string	返回说明
result	object	返回结果集
-	-	-
realtime	object	天气实况
info	string	天气情况，如：晴、多云
wid	string	天气标识id，可参考小接口2
temperature	string	温度，可能为空
humidity	string	湿度，可能为空
direct	string	风向，可能为空
power	string	风力，可能为空
aqi	string	空气质量指数，可能为空
-	-	-
future	array	近5天天气情况
date	string	日期
temperature	string	温度，最低温/最高温
weather	string	天气情况
direct	string	风向

14.3 调用第三方天气API

- 请求示例：

```
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.io.IOException;
import java.net.URLEncoder;

public class WeatherApiClient {

    private static final String API_KEY = "你的API密钥";
    private static final String BASE_URL = "https://api.example.com/data/2.5/weather";

    public static void main(String[] args) {
        String cityName = "北京";
        String encodedCityName = URLEncoder.encode(cityName, "UTF-8");

        HttpClient client = HttpClient.newHttpClient();
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(BASE_URL + "?q=" + encodedCityName + "&appid=" + API_KEY + "&units=metric"))
                .header("Accept", "application/json")
                .GET()
                .build();

        try {
            HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());

            if (response.statusCode() == 200) {
                System.out.println("Weather data: " + response.body());
            } else {
                System.out.println("Failed to retrieve weather data: " + response.statusCode());
            }
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

14.3 调用第三方天气API

- 处理响应和错误（错误代码）

服务级错误码参照(error_code)：

错误码	说明
207301	错误的查询城市名
207302	查询不到该城市的相关信息
207303	网络错误，请重试

- 请求超时
  - 在使用 Apache HttpClient 发送 HTTP 请求时，处理超时是一个常见的需求，以确保当网络延迟或服务器无响应时，客户端不会无限期地等待。HttpClient 提供了灵活的方式来设置不同类型的超时时间，包括连接超时、请求超时（也称作套接字超时）以及响应超时（在某些版本的 HttpClient 中可能需要通过不同的方式来实现）。

14.4 接口调用过程中可能遇到的问题和解决方案

- 请求超时示例（使用 HttpClientBuilder 设置超时）

```java
import org.apache.http.client.config.RequestConfig;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;

public class HttpClientExample {
    public static CloseableHttpClient createHttpClient() {
        RequestConfig requestConfig = RequestConfig.custom()
                .setSocketTimeout(5000) // 设置等待数据超时时间，毫秒为单位
                .setConnectTimeout(5000) // 设置连接超时时间，毫秒为单位
                .setConnectionRequestTimeout(5000) // 设置从连接池获取连接实例的超时时间，毫秒为单位
                .build();

        // 注意：PoolingHttpClientConnectionManager 是可选的，用于管理连接池
        PoolingHttpClientConnectionManager cm = new PoolingHttpClientConnectionManager();
        // cm.setMaxTotal(200); // 最大连接数
        // cm.setDefaultMaxPerRoute(20); // 每个路由的最大连接数

        return HttpClients.custom()
                .setDefaultRequestConfig(requestConfig)
                .setConnectionManager(cm) // 如果需要连接池，则设置
                .build();
    }

    // ... 使用创建的 httpClient 发送请求
}
```

- API限制
  - API限制通常包括以下几个方面：
  - 请求频率限制：
    - 固定窗口算法：在固定时间窗口内限制请求数量。
    - 滑动窗口算法：使用滑动窗口实现更平滑的限流，允许在窗口内的一定时间范围内突发请求，但总体请求速率仍受限制。
    - 令牌桶算法：允许一定程度的突发流量，但限制平均速率。Guava的RateLimiter是一个基于令牌桶算法的限流工具。
    - 漏桶算法：控制数据流入速率，平滑突发流量，但不允许突发请求。
  - 请求大小限制：对单个请求的数据大小进行限制，以防止过大的请求导致服务器过载。
  - 并发连接数限制：限制同时建立的连接数，以防止过多的并发连接耗尽服务器资源。

- API配额
  - API配额是分配给每个用户或应用在一定时间段内可以使用的API请求数量的上限。配额管理通常与用户或组织相关联，用于限制一段时间内的API使用量。
  - 配额类型：
    - 每日配额：如Google Photos Library API为每个项目每天设置10,000个请求配额（不包括访问媒体字节的请求），以及75,000个访问媒体字节的请求配额。
    - 每小时/每分钟配额：某些API可能还设置了更细粒度的配额限制，如每秒或每分钟可以发送的请求数。
  - 配额调整：
    - 如果用户或应用需要更高的配额，通常需要向API提供商提交申请，并提供相关证明和理由。API提供商会根据实际情况审核申请，并可能要求用户或应用优化现有使用策略以减少不必要的请求。
  - 配额监控和报告：
    - 用户或应用应实时监控API的使用情况，以便及时发现并处理潜在的配额超限问题。同时，API提供商通常会提供配额使用情况的报告和监控工具，帮助用户或应用更好地管理配额。

- 处理配额超限和限制
  - 当API请求达到限制或配额时，通常会返回相应的错误代码（如HTTP 429 Too Many Requests）或错误消息。此时，用户或应用可以采取以下措施：
  - 减少请求频率：通过增加请求之间的时间间隔来降低请求频率。
  - 使用缓存：缓存频繁访问的数据以减少对API的依赖。
  - 合并请求：将多个小请求合并为一个大请求以减少请求次数。
  - 重试机制：实现智能重试逻辑以处理临时的网络问题或API不可用的情况。
  - 升级付费计划：如果API提供商提供付费计划以获取更高的配额或更高级的功能，用户或应用可以考虑升级到适当的付费计划。

- API限制和配额总结
  - API限制和配额是保障API服务稳定性和可用性的重要手段。用户或应用应了解并遵守这些限制和配额要求，并采取相应的措施来优化API使用策略以减少不必要的请求和配额超限问题。

- API变更和版本控制
  - API（Application Programming Interface，应用程序编程接口）变更是指对现有API的修改、扩展或废弃，这些变更可能是由于多种原因，如改进性能、增加新功能、修复安全漏洞、提高代码的可维护性等。而版本控制则是对这些变更进行管理的一种手段，以确保不同版本的API可以并存，同时尽可能减少变更对现有系统的影响。

- API变更
  - 变更类型
    - 修改：对现有API的功能或行为进行更改，但保持其基本结构不变。
    - 扩展：为现有API添加新的功能或参数，以支持更多的使用场景。
    - 废弃：将不再使用的API标记为废弃，并鼓励开发者使用新的替代API。
  - 变更影响
    - API变更对开发者和应用程序有着深远的影响。开发者需要密切关注自己使用的库和框架的更新日志，及时了解和适应API的变更。如果API变更导致现有代码无法正常工作，开发者需要进行相应的修改和测试，以确保系统的稳定性和可靠性。

- 版本控制
  - 目的
    - 保持向后兼容性：确保旧版本的客户端能够继续与新版本的服务器通信。
    - 有序管理变更：通过版本控制，可以清晰地记录API的变更历史，方便开发者追溯和理解。
    - 支持并行开发：在大型项目中，不同团队可能同时在开发不同版本的API，版本控制可以确保这些工作不会相互干扰。
  - 方法
    - URL中添加版本号：在API的URL中直接添加版本号，如/api/v1/users表示第一版的用户管理接口。这种方法简单直观，易于实现和识别。
    - HTTP头部添加版本信息：在HTTP请求的头部添加版本信息，如X-Api-Version: v1。这种方法不会改变URL结构，但需要在请求中额外指定版本信息。
    - 自定义参数传递版本信息：在API的参数中添加版本信息，如/api/users?version=v1。这种方法适用于需要动态选择版本的场景。
    - 使用不同的域名或端口：为不同版本的API分配不同的域名或端口，如api.example.com表示第一版的API接口，api-v2.example.com表示第二版的API接口。这种方法可以清晰地隔离不同版本的API。

- 版本控制
  - 兼容性规范
    - 定义API版本之间的兼容性规范，以便确保不同版本的API可以相互协作。这包括明确哪些功能在哪些版本中可用，以及如何在新旧版本之间进行迁移。
    - 提供兼容性测试工具，以确保新版本的API与旧版本的API之间的兼容性。这有助于减少因API变更导致的系统不稳定和故障。
  - 挑战与解决方案
    - 兼容性问题：不同版本的API之间可能存在兼容性问题。解决方案是进行充分的测试，确保新版本的API能够兼容旧版本的接口。
    - 文档更新问题：随着API的升级，相关的文档也需要及时更新。解决方案是建立完善的文档更新机制，确保文档能够及时跟随API的升级而更新。
    - 升级策略问题：如何制定合理的API升级策略以避免频繁升级导致的系统不稳定是一个重要问题。解决方案是根据项目的实际需求和团队的开发能力制定合理的升级计划和策略。

  - API变更和版本控制是软件开发中不可或缺的一部分。通过合理的版本控制策略和清晰的兼容性规范可以确保API的稳定性和可靠性，降低变更对现有系统的影响。同时开发者也需要密切关注API的变更情况并及时进行适应和更新。

本章重点

- 网络请求的定义
- 网络请求的基本步骤
- 常用的HTTP请求方法（掌握 GET，POST，PUT，DELETE）
- 掌握通过HttpClient发送HTTP请求

本章作业

- 通过Java代码实现访问天气API
  - 创建一个URL连接对象，并配置连接地址
  - 设置请求方式（GET/POST）
  - 设置请求头以及请求请求体（请求参数）
  - 创建连接并发送请求到目标服务器
  - 获取服务端响应代码
  - 将服务端响应参数打印输出到控制台

信创智能医疗系统研发课程体系

河南中医药大学信息技术学院（智能医疗行业学院）

河南中医药大学信息技术学院（智能医疗行业学院）智能医疗教研室

河南中医药大学医疗健康信息工程技术研究所