Android 中的网络

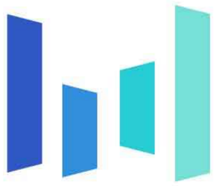
温如日-字节跳动 Android 工程师

wenruri@bytedance.com

沈立家-字节跳动 Android 工程师

shenlijia@bytedance.com

ByteDance 字节跳动

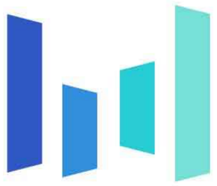


目录

- 网络基础知识
- Android网络通信基础实现

ByteDance 字节跳动

网络基础知识



问题：需要实现网络通信，需要做哪些工作？解决哪些问题？

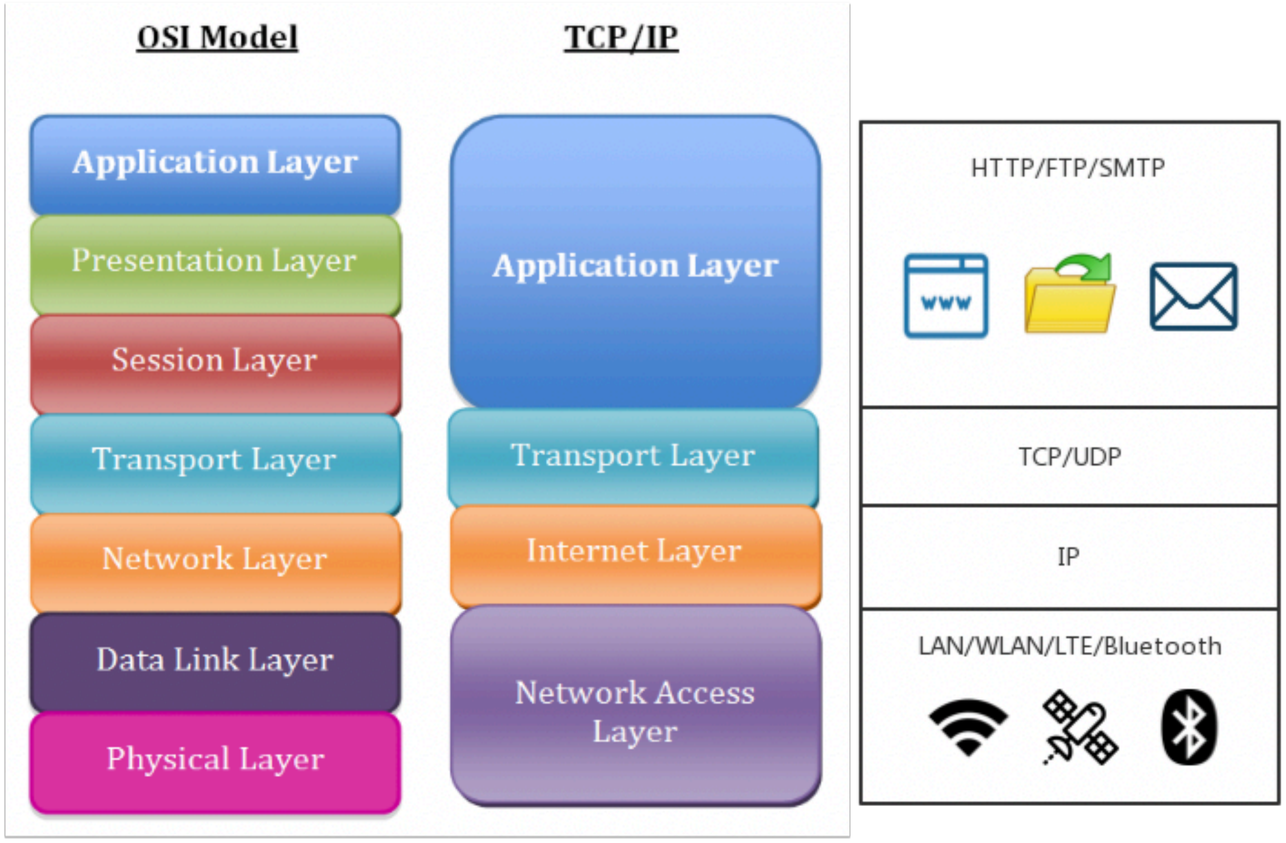
网络分层模型

互联网是一个分层架构

相同层次的对等实体通过协议进行通信

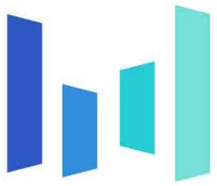
第N层给第N+1层提供服务

网络分层模型



问题：

每一层的作用是什么？



网络层：IP协议

主机在网络中如何定位：IP地址

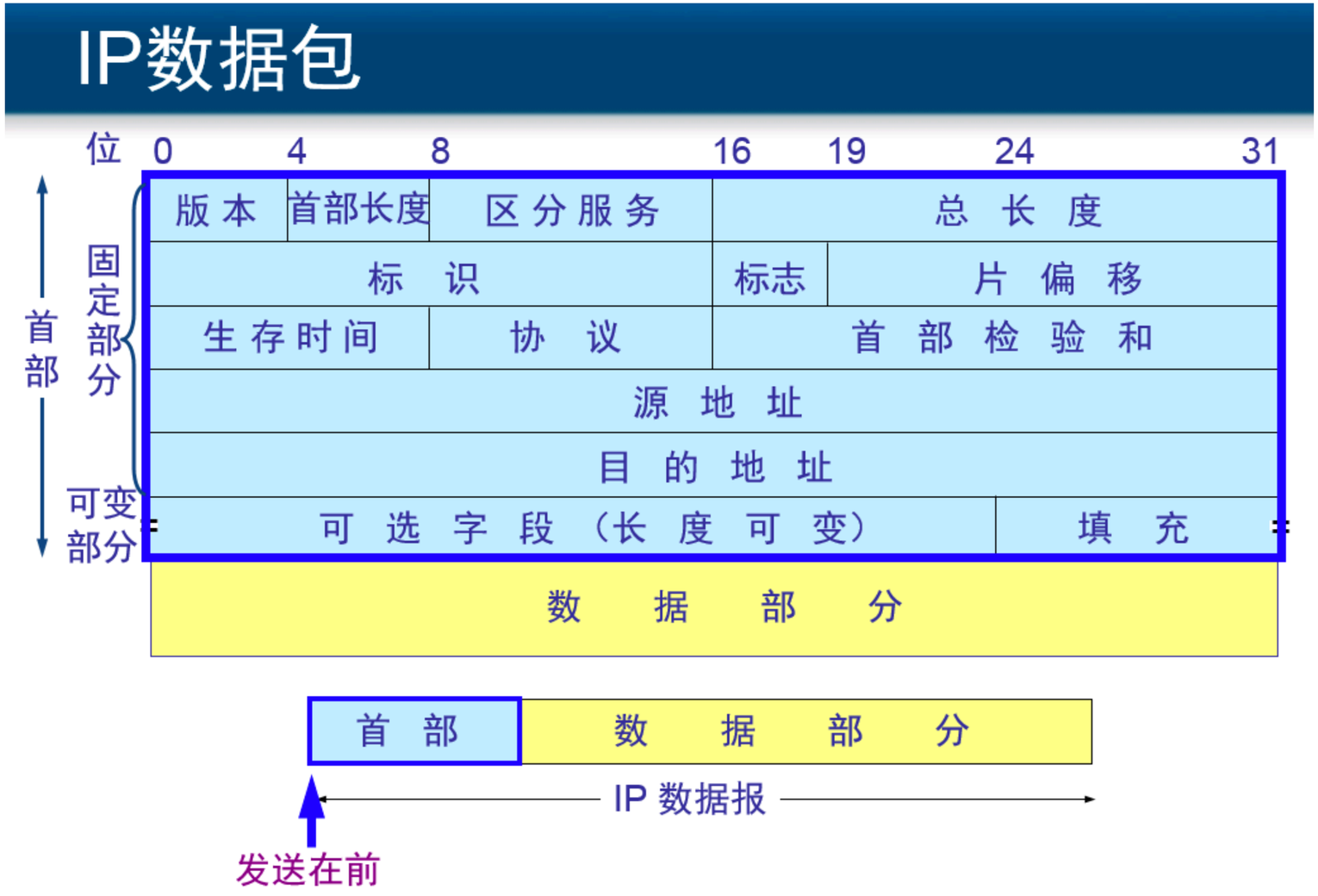
数据包如何到达物理不相邻的主机：IP分组的转发规则

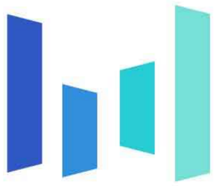
当数据包大于链路层的最大传输单元时的处理方案：IP分片

信息传输的格式：IP分组结构

ByteDance
字节跳动

IPv4 数据包格式


IP数据包
位
0
4
8
16
19
24
31
首部
固定部分
可变部分
版本
首部长度
区分服务
总长度
标识
标志
片偏移
生存时间
协议
首部检验和
源地址
目的地址
可选字段（长度可变）
填充
数据部分
首部
数据部分
IP 数据报
发送在前



传输层：TCP/UDP 协议

TCP: 提供可靠的、面向连接的的端到端传输

- 数据分片
- 到达确认
- 超时重发
- 滑动窗口
- 失序和重复处理
- 数据校验

UDP: 提供不可靠的、无连接的的端到端传输

ByteDance 字节跳动

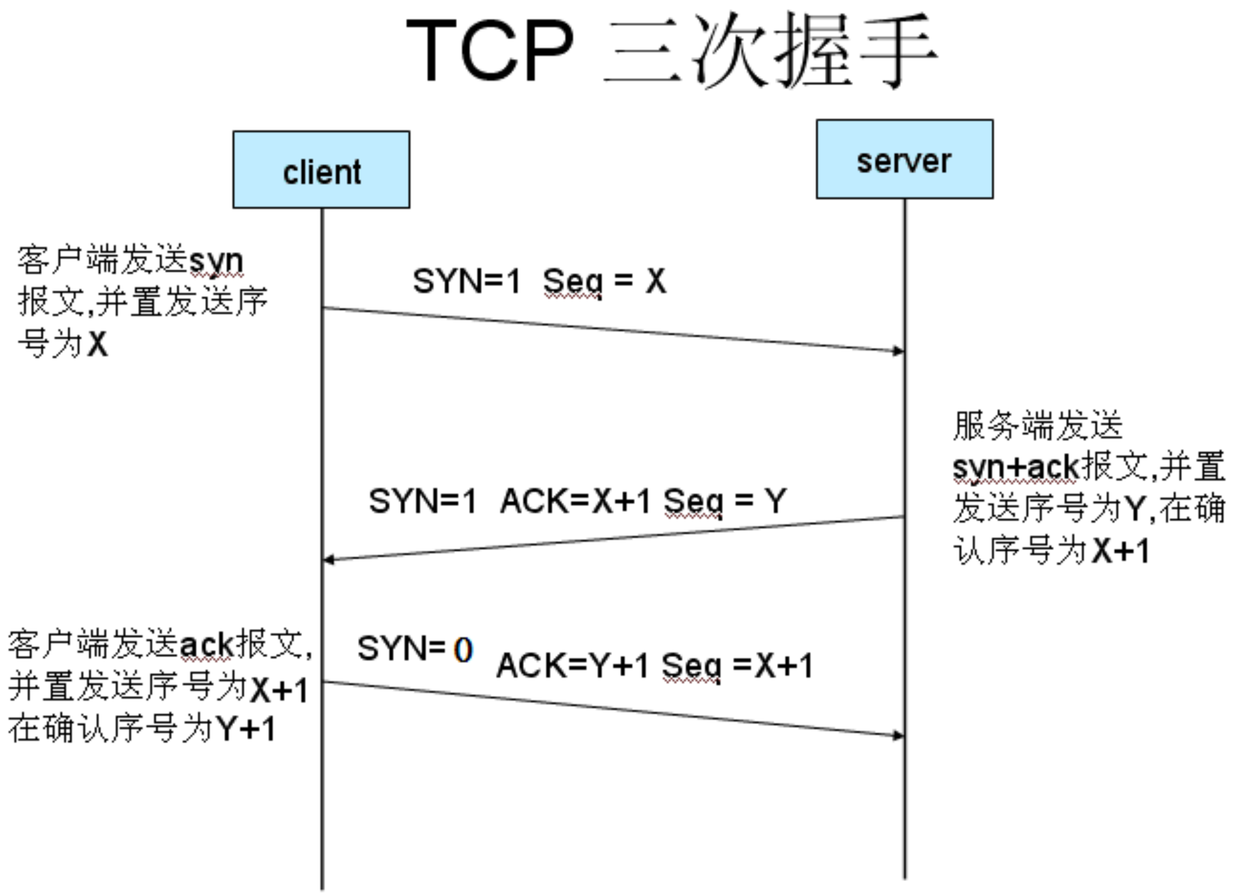
TCP 分组格式示意图

问题：TCP和UDP各有什么利弊？

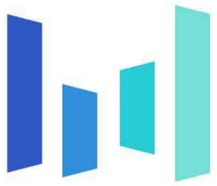
分别合适哪些场景

ByteDance 字节跳动

TCP三次握手



ByteDance 字节跳动



IPv4协议的最大缺陷——IP地址不足

补丁方案

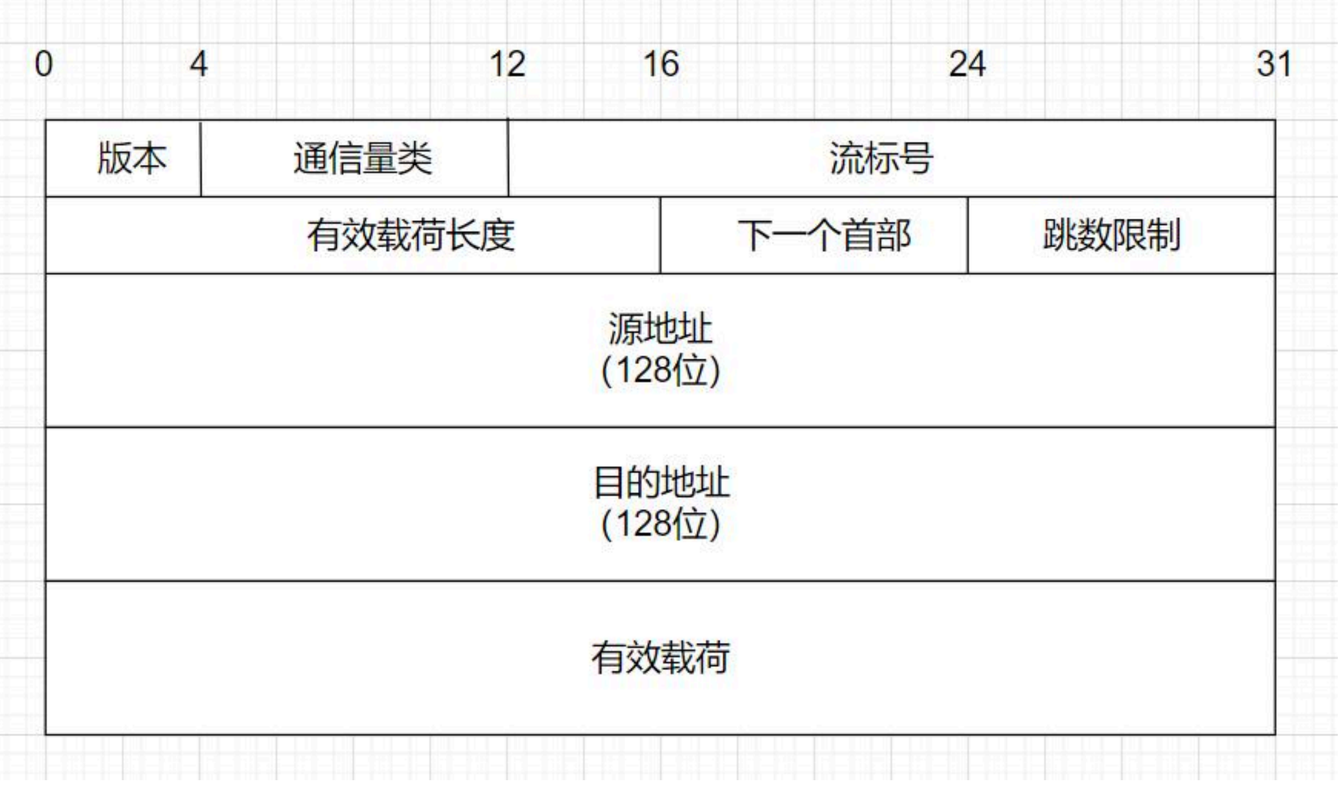
- NAT(Network Address Translation)

内网	外网
192.168.1.55:5566	219.152.168.222:9200
192.168.1.59:80	219.152.168.222:9201
192.168.1.59:4465	219.152.168.222:9202

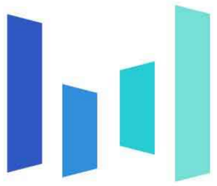
问题：NAT有什么弊端?

IPv4协议的最大缺陷——IP地址不足

终极方案——IPv6



ByteDance 字节跳动



Http协议

HTTP是一个应用最广泛的应用层协议，定义浏览器/客户端如何从Web服务器请求数据，以及Web服务器如何把数据传送给客户端

HTTP基于TCP

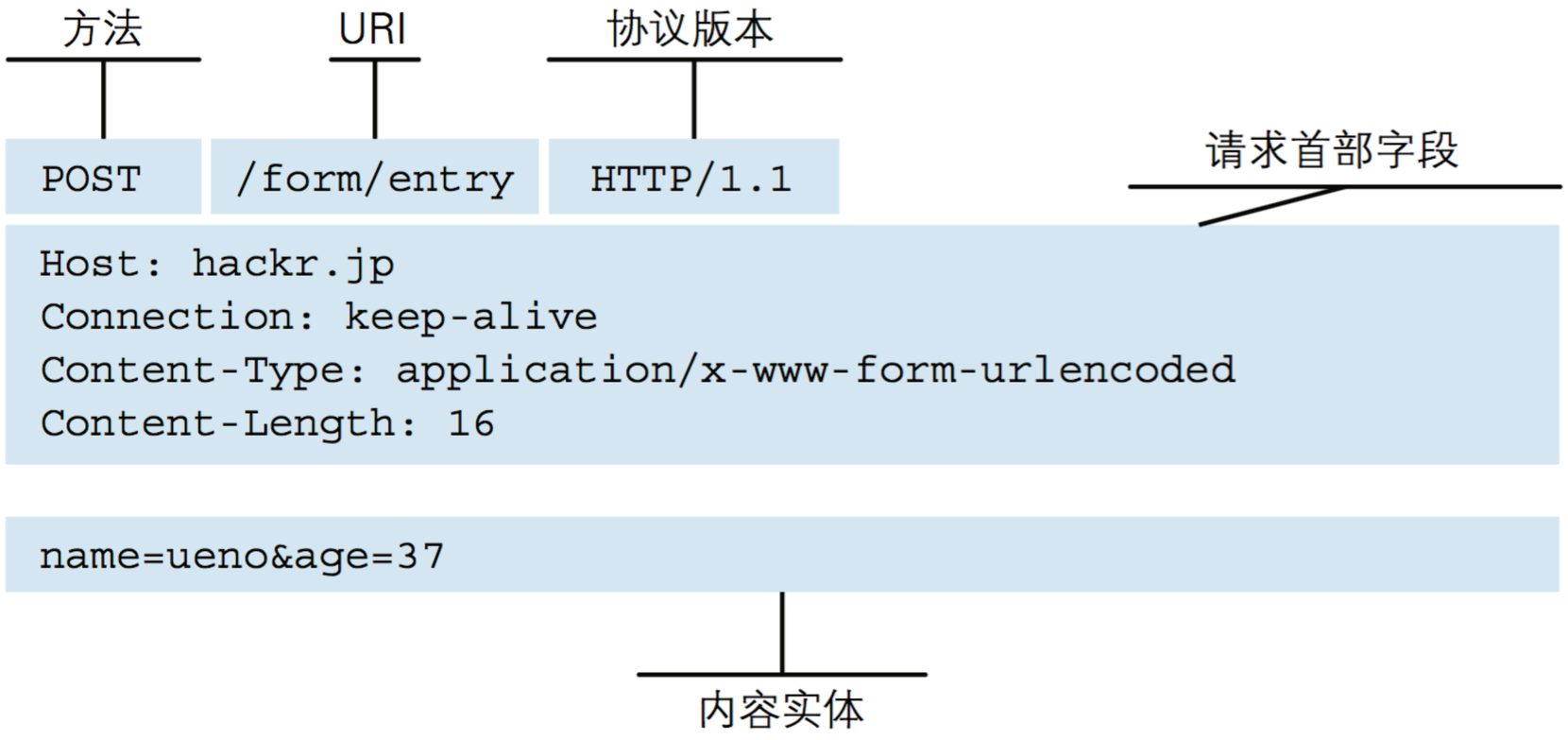
HTTP是请求-应答模式，客户端发送一个请求，服务器给一个应答

HTTP协议是无状态协议

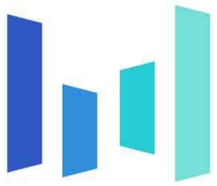
HTTP协议的默认端口号是80，HTTPS端口443

Http协议

Http Request格式



图：请求报文的构成



Http协议

Http Request 格式（访问https://www.zju.edu.cn/）

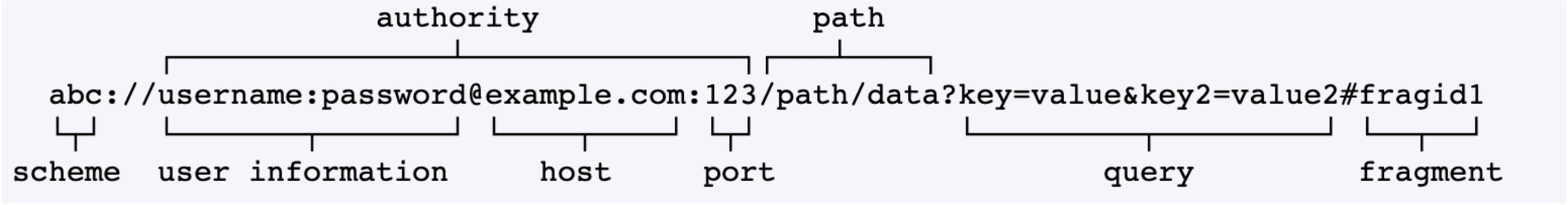
:method	GET
:authority	www.zju.edu.cn
:scheme	https
:path	/
cache-control	max-age=0
sec-ch-ua	" Not;A Brand";v="99", "Google Chrome";v="91", "Chromium";v="91"
sec-ch-ua-mobile	?0
upgrade-insecure-requests	1
user-agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
sec-fetch-site	none
sec-fetch-mode	navigate
sec-fetch-user	?1
sec-fetch-dest	document
accept-encoding	gzip, deflate, br
accept-language	zh-CN,zh;q=0.9
cookie	Hm_lvt_fe30bbc1ee45421ec1679d1b8d8f8453=1624870819,1624873038,1625814665
cookie	Hm_lpvt_fe30bbc1ee45421ec1679d1b8d8f8453=1626342633
cookie	JSESSIONID=DC807C41B243C7172695E9FFB1ED547D

ByteDance 字节跳动

Http 1.1定义的8种Method

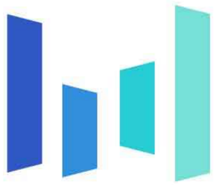
GET \ PUT \ HEAD \ POST \ DELETE \ TRACE \ OPTIONS \ CONNECT

Url的格式：



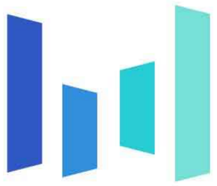
典型的http Url示例

https://api.github.com/users/JakeWharton/repos?page=0&per_page=10



常见标准的header

Accept	可接受的响应内容类型（Content-Types）。	`Accept: text/plain`
Accept-Charset	可接受的字符集	`Accept-Charset: utf-8`
Accept-Encoding	可接受的响应内容的编码方式。	`Accept-Encoding: gzip, deflate`
Accept-Language	可接受的响应内容语言列表。	`Accept-Language: en-US`
Authorization	用于表示HTTP协议中需要认证资源的认证信息	Authorization: Basic OSdjJGRpbjpvcGVuIANlc2SdDE==
Cache-Control	用来指定当前的请求/回复中的，是否使用缓存机制。	`Cache-Control: no-cache`
Connection	客户端（浏览器）想要优先使用的连接类型	`Connection: keep-alive` `Connection: Upgrade`
User-Agent	浏览器/客户端的身份标识字符串	`User-Agent: Mozilla/……`
Content-Length	以8进制表示的请求体的长度	`Content-Length: 348`
Content-Type	Body中内容的内容类型和编码	`text/html; charset=utf-8`



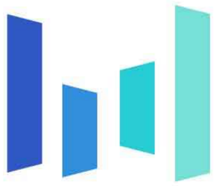
Content-Type

Http协议规定了header部分必须是ascii字符，但是并没有规定body部分的格式和编码

无论是request还是response，只要有body就需要在header中的Content-Type字段中标明内容类型（MIME）和其他一些用于解析的信息

常见类型

- application/x-www-form-urlencoded
- multipart/form-data
- application/json
- text/xml



Http协议

客户端向服务端传送自定义信息的方式

- url中的query
- header
- body

ByteDance 字节跳动

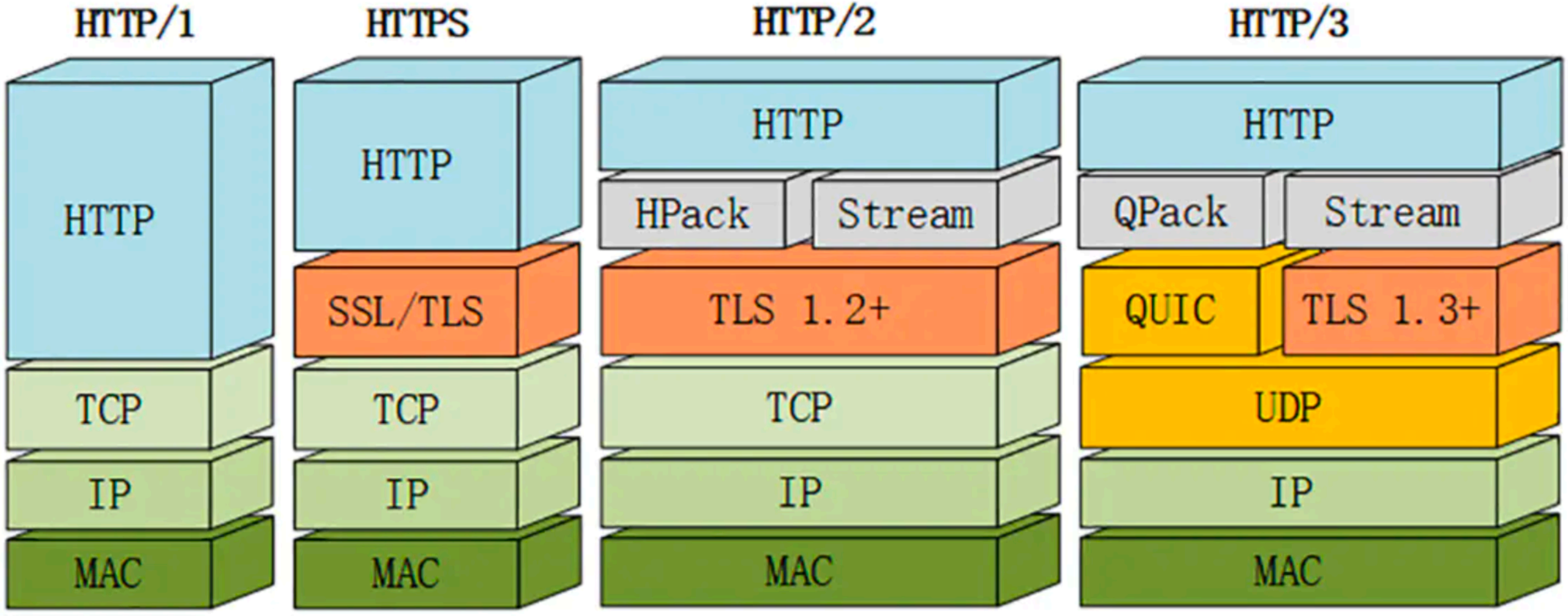
Http协议

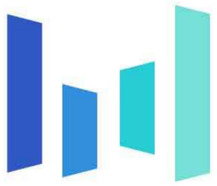
Http Response 格式

Http 状态码

	类别	原因短语
1XX	Informational（信息性状态码）	接收的请求正在处理
2XX	Success（成功状态码）	请求正常处理完毕
3XX	Redirection（重定向状态码）	需要进行附加操作以完成请求
4XX	Client Error（客户端错误状态码）	服务器无法处理请求
5XX	Server Error（服务器错误状态码）	服务器处理请求出错

Http协议的发展





Http协议的发展

Http1.0： 每一个请求都要新建一个TCP连接，请求完就断开

Http 1.1

- 默认长链接，允许TCP连接复用，节省了反复建连的过程
- 引入Range头域，允许范围请求功能
- 引入Host头域，允许虚拟主机功能
- 引入了更全面的Cache机制

Https

- 引入SSL/TSL协议保证安全

Http协议的发展

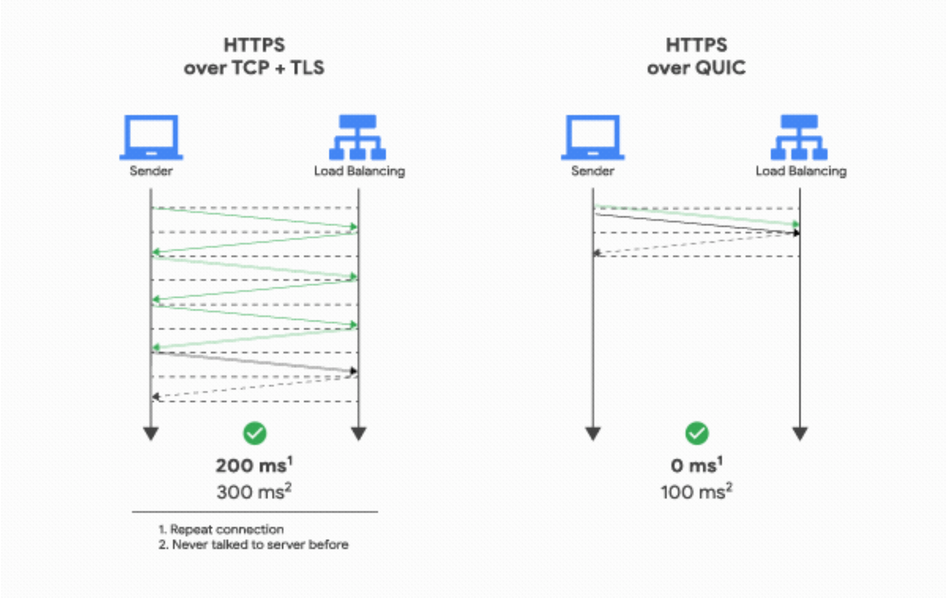
HTTP/2 (HTTP-Over-QUIC)

- 二进制分帧
- 多路复用机制
- 服务端推送
- 头部压缩

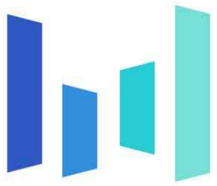
Http协议的发展

HTTP/3 (HTTP-Over-QUIC)

- 解决TCP队头阻塞问题
- 连接迁移(64位connectionId)
- 0RTT——通信双方在发起连接时，发起方第一个数据包里面便可以携带有效的业务数据



ByteDance 字节跳动



RESTful API

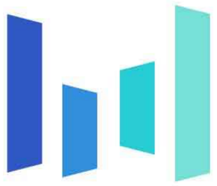
问题：

服务器的服务怎么提供?

怎么设置服务的地址

用什么方法？Post还是Get?

参数放哪？path/query/body?



RESTful API

（Resources）REpresentational State Transfer

（资源）表现层状态转换

资源：一段文本、一张图片、一首歌曲

表现形式：json、xml

状态变化：通过 HTTP 动词实现

API 是面向资源的，资源表达的形式可以是 json 或者 xml，
它的 url 中不包含动词，而是通过 HTTP 动词表达想要的动作

目的：
看 URL 就知道要什么
看 HTTP method 就知道干什么

RESTful API

SOAP Web API

GET http://127.0.0.1/user/select/1 根据用户id查询用户数据
POST http://127.0.0.1/user/save 新增用户
GET/POST http://127.0.0.1/user/delete 删除用户信息

不在 url 里面做具体动作的描述，而是通过 HTTP 请求方式表达意图

RESTful Web API

GET http://127.0.0.1/user/1 据用户id查询用户数据
POST http://127.0.0.1/user 新增用户
DELETE http://127.0.0.1/user 删除用户信息

GET 用来获取资源
POST 用来新建资源（也可以用于更新资源）
PUT 用来更新资源
DELETE 用来删除资源

RESTful API

RESTful 只是一种规范，并不是标准

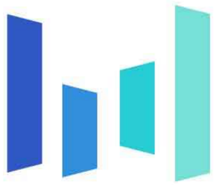
如何正确理解前后端分离？

为什么会出现 RESTful?

RESTful API 论文：RESTful API 诞生地，出自 Roy Thomas Fielding（HTTP 协议创建人之一）

Richardson 成熟度模型： Richardson 提出的 REST 四层模型

GrpahQL： 另一种 API 规范



数据传输格式

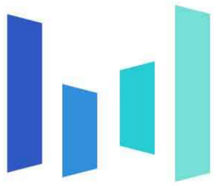
数据传输格式需要解决的问题

- 通用（可以表示任何复杂数据）
- 能够进行网络传输（二进制流或字节流）
- 跨语言使用

常用解决方法

- Json
- Protobuf
- Xml

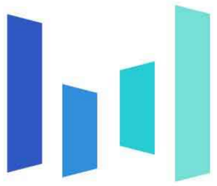
ByteDance 字节跳动



JSON (JavaScript Object Notation)

JSON

```json
{
  "name": "安卓应用开发实训",
  "location": "紫金港东1B-205",
  "lesson_count": 9,
  "students": [
    {
      "name": "令狐冲",
      "male": true
    },
    {
      "name": "张无忌",
      "male": true
    },
    {
      "name": "赵敏",
      "male": false
    }
  ]
}
```

Json基于两种结构基础元素

对象（Object）： <名称-值> 对的集合

数组（array）：值的有序列表

值（value）

- 字符串（string）：由双引号包围的任意数量Unicode字符
- 数字 (number)
- true / false
- null
- 对象
- 数组

数据传输格式

Json值类型	java 类型	备注
字符串	String	
数字	int/double/long/short/ float/Number/String	基本类型与装箱类型都可以
true / false	boolean/String	基本类型与装箱类型都可以
数组	List<String>/ Set<String>/数组String[]	注意 Set 是去重的
对象	Map<String,?> 自定义的java对象	

数据传输格式

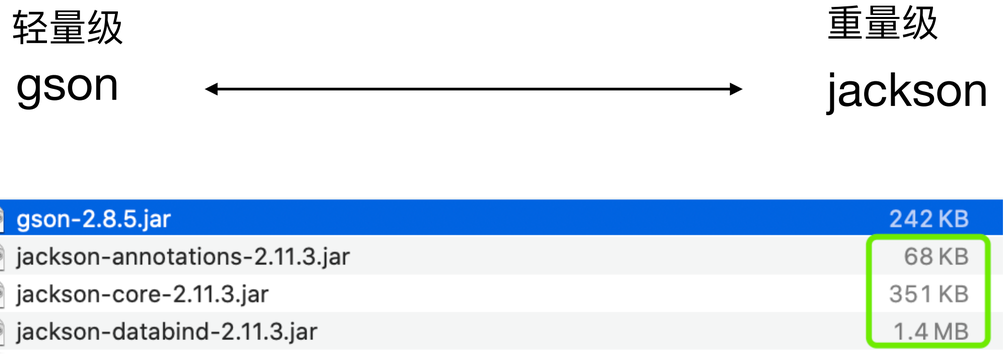
JSON 解析器

序列化：将Java对象转成Json

反序列化：将Json字符串转成Java对象

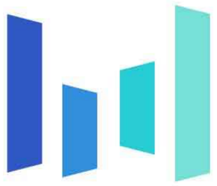
常用的JSON解析器

- JsonLib
- Gson
- Jackson
- Fastjson



（7.6倍体积之差）

ByteDance 字节跳动



数据传输格式

Gson 常用Api

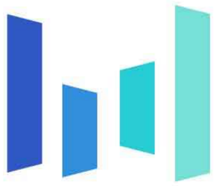
public String toJson(Object src)

public <T> T fromJson(String json, Class<T> classOfT) throws JsonIOException, JsonSyntaxException

public <T> T fromJson(String json, Type typeOfT) throws JsonIOException, JsonSyntaxException

```java
public class Course {
    @SerializedName("name")
    public String name;
    @SerializedName("location")
    public String location;
    @SerializedName("lesson_count")
    public int lessonCount;
    @SerializedName("students")
    public List<Student> students;

    public static class Student {
        @SerializedName("name")
        public String name;
        @SerializedName("male")
        public boolean male;
    }
}
```

```java
Course course = new Gson().fromJson(courseJsonStr, Course.class);
String str = new Gson().toJson(course);
```

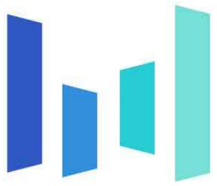


Protocol Buffers (ProtoBuf)

- 先定义proto
- 用proto编译生成代码
- 每一个field都有一个数字标识，传输的时候不传name，只传name对应的数字
- 通信的两端都保存着proto文件，用于解析消息

ProtoBuf

```
message COURSE {
  required string name = 0
  required string location = 1
  required double lesson_count = 2

  message STUDENTS {
    required string name = 0
    required bool male = 1
  }
  repeated STUDENTS students = 3
}
```



问题：这三种格式各有什么优势和劣势?

xml

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<name>中国</name>
<provinces>
    <name>黑龙江</name>
    <cities>
        <city>哈尔滨</city>
        <city>大庆</city>
    </cities>
</provinces>
<provinces>
    <name>广东</name>
    <cities>
        <city>广州</city>
        <city>深圳</city>
        <city>珠海</city>
    </cities>
</provinces>
```

Json

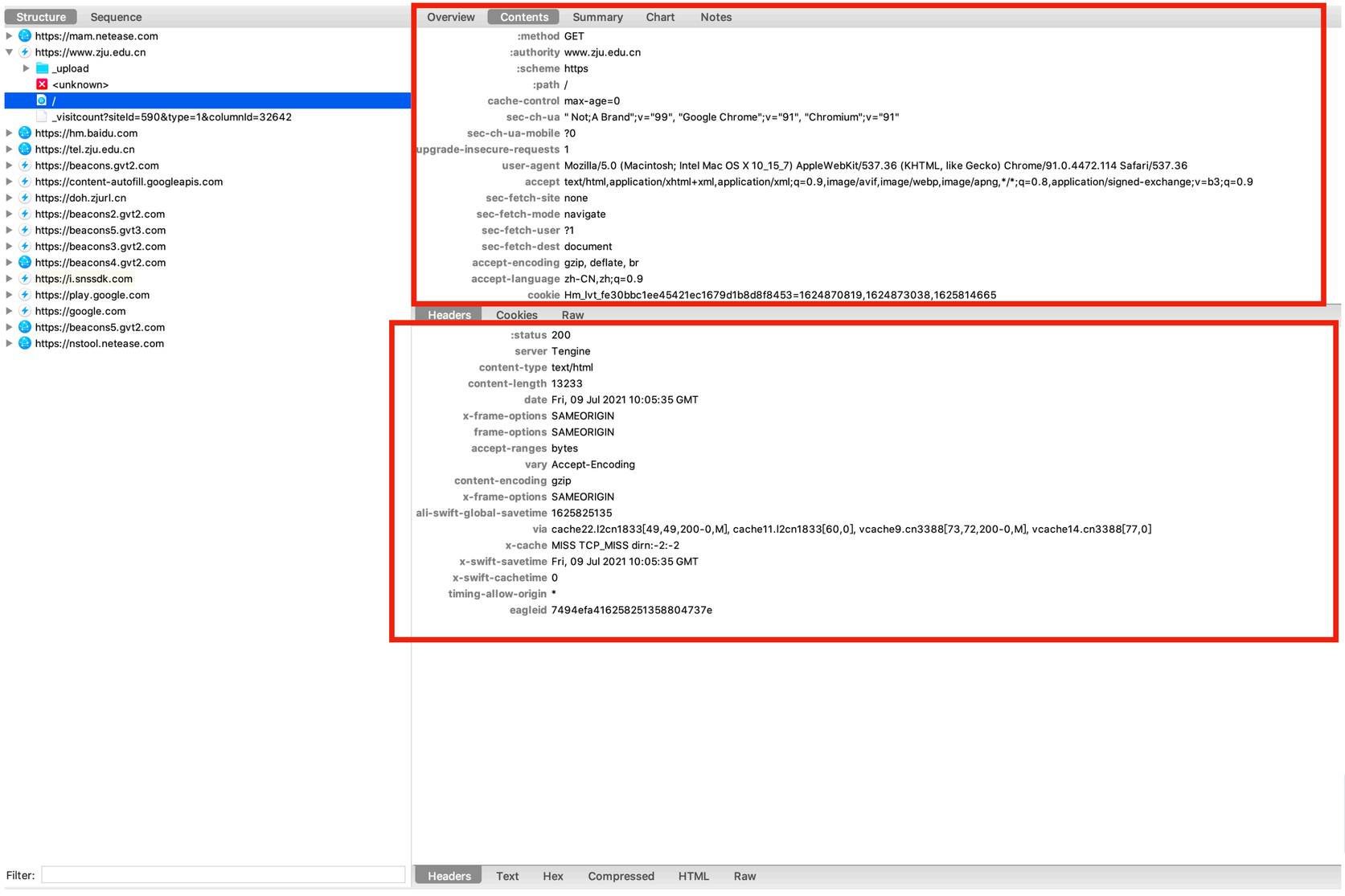
```json
{
    "name": "中国",
    "provinces": [{
        "name": "黑龙江",
        "cities": {
            "city": ["哈尔滨", "大庆"]
        }
    }, {
        "name": "广东",
        "cities": {
            "city": ["广州", "深圳", "珠海"]
        }
    }]
}
```

protobuf

```
1: "\344\270\255\345\233\275"
2 {
  1: "\351\273\221\351\276\231\346\261\237"
  2 {
    1: "\345\223\210\345\260\224\346\273\250"
    1: "\345\244\247\345\272\206"
  }
}
2 {
  1: "\345\271\277\344\270\234"
  2 {
    1: "\345\271\277\345\267\236"
    1: "\346\267\261\345\234\263"
    1: "\347\217\240\346\265\267"
  }
}
```

实用工具

抓包工具——Charles



Request

Response

ByteDance 字节跳动

实用工具

抓包工具——Charles

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta name="baidu-site-verification" content="kQwZbr6krk" />
<title>浙江大学</title>

<link type="text/css" href="/_css/_system/system.css" rel="stylesheet"/>
<link type="text/css" href="/_upload/site/1/style/1/1.css" rel="stylesheet"/>
<link type="text/css" href="/_upload/site/02/4e/590/style/526/526.css" rel="stylesheet"/>
<link type="text/css" href="/_js/_portletPlugs/simpleNews/css/simplenews.css" rel="stylesheet" />
<link type="text/css" href="/_js/_portletPlugs/datepicker/css/datepicker.css" rel="stylesheet" />
<link type="text/css" href="/_js/_portletPlugs/sudyNavi/css/sudyNav.css" rel="stylesheet" />

<script language="javascript" src="/_js/sudy-jquery-autoload.js" jquery-src="/_js/jquery-1.9.1.min.js" sudy-wp-context="" sudy-wp-siteId="590"></script>
<script language="javascript" src="/_js/jquery-migrate.min.js"></script>
<script language="javascript" src="/_js/jquery.sudy.wp.visitcount.js"></script>
<script type="text/javascript" src="/_js/_portletPlugs/datepicker/js/jquery.datepicker.js"></script>
<script type="text/javascript" src="/_js/_portletPlugs/datepicker/js/datepicker_lang_HK.js"></script>
<script type="text/javascript" src="/_js/_portletPlugs/sudyNavi/jquery.sudyNav.js"></script>
<link rel="shortcut icon" href="/_upload/tpl/05/e5/1509/template1509/images/favicon.ico" mce_href="/_upload/tpl/05/e5/1509/template1509/images/favicon.ico" type="image/x-icon">
<link rel="icon" href="/_upload/tpl/05/e5/1509/template1509/images/favicon.ico" mce_href="/_upload/tpl/05/e5/1509/template1509/images/favicon.ico" type="image/x-icon">
<meta name="viewport" content="width=1000">
<script>
var _hmt = _hmt || [];
(function() {
  var hm = document.createElement("script");
  hm.src = "https://hm.baidu.com/hm.js?fe30bbc1ee45421ec1679d1b8d8f8453";
  var s = document.getElementsByTagName("script")[0];
  s.parentNode.insertBefore(hm, s);
})();
</script>
<link type="text/css" rel="stylesheet" href="/_upload/tpl/05/e5/1509/template1509/style/common.css" />
<link type="text/css" rel="stylesheet" href="/_upload/tpl/05/e5/1509/template1509/style/default.css" />
<link type="text/css" rel="stylesheet" href="/_upload/tpl/05/e5/1509/template1509/style/menu.css" />
<script type="text/javascript" src="/_upload/tpl/05/e5/1509/template1509/js/uaredirect.js" ></script>
<script type="text/javascript">uaredirect("/mobile/");</script>

<script type="text/javascript" src="/_upload/tpl/05/e5/1509/template1509/js/ImageSwitch.js"></script>
```

Headers Text Hex Compressed HTML Raw

ByteDance 字节跳动

实用工具

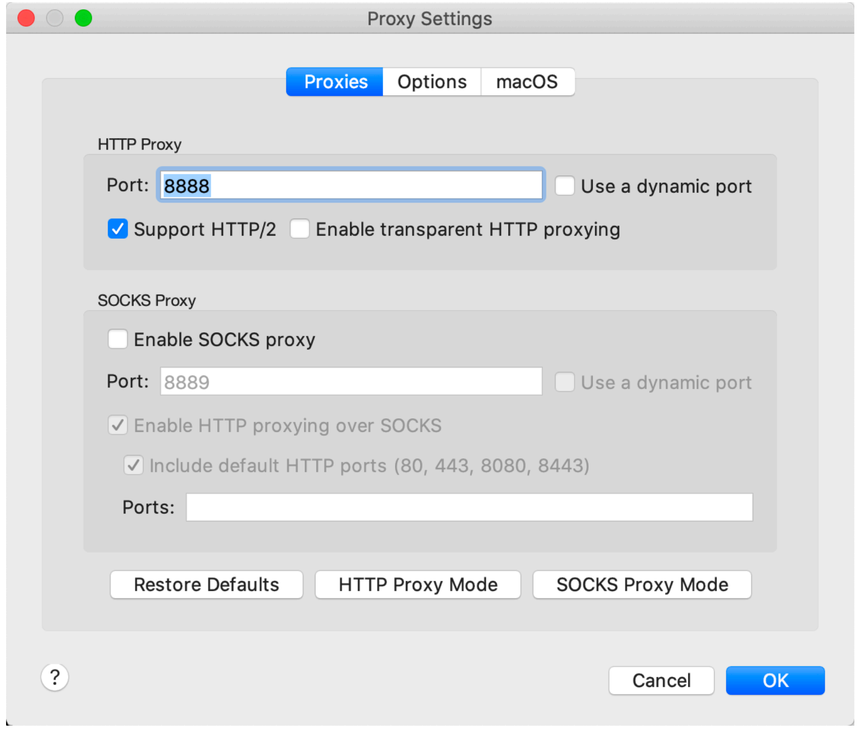
抓包工具——Charles

1、将手机和电脑连到同一个局域网

2、设置Charles代理端口

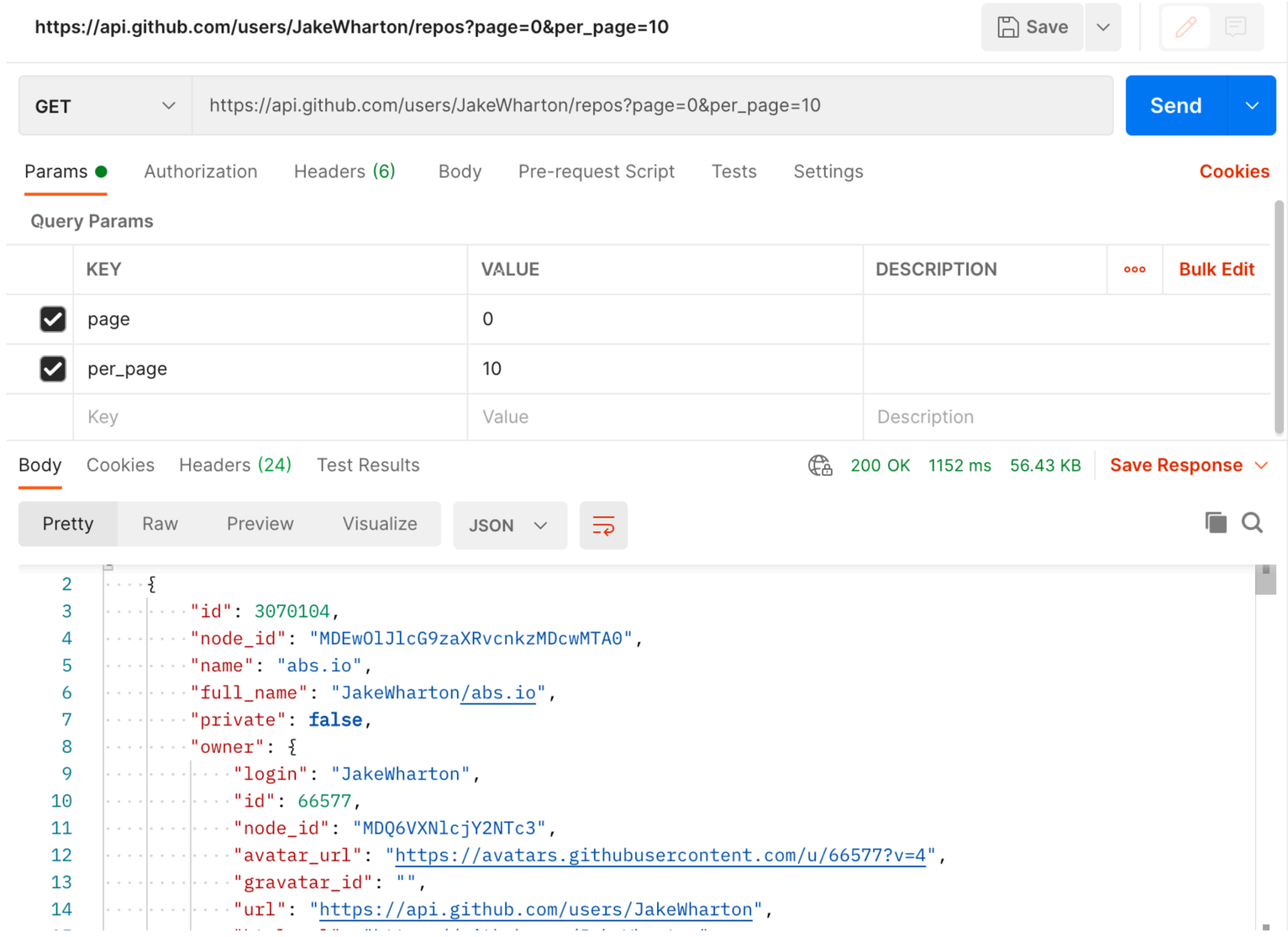
3、将手机wifi的代理服务器设到电脑上

4、如果要抓取https包，手机要安装证书



实用工具

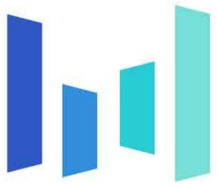
Postman——轻松创建请求



ByteDance 字节跳动

Android网络通信基础实现

ByteDance字节跳动

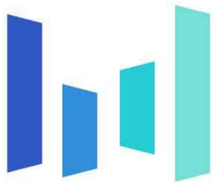


直接在传输层进行网络通信：Socket

Socket(套接字)，是应用层 与 TCP/IP 协议族通信的中间软件抽象层

一个Socket由一个IP地址和一个端口号唯一确定

两种套接字：Socket(TCP) ，DatagramSocket(UDP)



Socket的基本工作流程

客户端流程

（1） 创建Socket，连接到服务器；

（2） 打开连接到Socket的输入/出流；

（3） 按照一定的协议对Socket进行读/写操作；

（4） 关闭Socket

服务端流程

（1） 监听特定的端口

（2） 当有连接到来时accept(), 得到Socket

（2） 打开连接到Socket的输入/出流；

（3） 按照一定的协议对Socket进行读/写操作；

（4） 关闭Socket

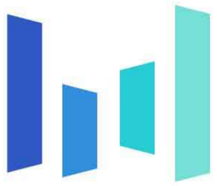
ByteDance 字节跳动

```
try {
    Socket socket = new Socket( host: "localhost",  port: 30000); //服务器IP及端口
    BufferedOutputStream os =new BufferedOutputStream(socket.getOutputStream());
    BufferedInputStream is = new BufferedInputStream(socket.getInputStream());
    // 读文件
    double n = 1;
    byte[] data = new byte[1024*5];//每次读取的字节数
    int len=-1;
    while (!stopFlag && socket.isConnected()) {
        if (!message.isEmpty()){
            Log.d( tag: "socket", msg: "客户端发送 "+ message);
            os.write(message.getBytes());
            os.flush();
            clearMsg();
            int reciveLen = is.read(data);
            String receive = new String(data, offset: 0,reciveLen);
            Log.d( tag: "socket", msg: "客户端收到 "+ receive);
            activity.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    Toast.makeText(activity,receive , Toast.LENGTH_SHORT).show();
                }
            });
        }
        sleep( millis: 300);
    }
    Log.d( tag: "socket", msg: "客户端断开 ");
    os.flush();
    os.close();
    socket.close(); //关闭socket

} catch (Exception e) {
    e.printStackTrace();
}
```

创建socket

发送数据

接收数据，read()方法会一直阻塞直到有数据



Datagram Socket工作流程

发送端

（1） 创建Datagram Socket

（2） 创建Packet包装内容；

（3） 发送send()

（4） 关闭

接收端

（1） 创建Datagram Socket

（2） 创建一个Packet

（3） 接收receive()

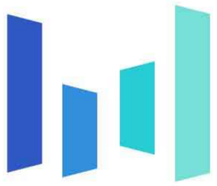
（4） 关闭

```
try {
    DatagramSocket datagramSocket = new DatagramSocket( port: 30002); //注意，这本地端口

    // 读文件
    double n = 1;
    byte[] data = new byte[1024 * 5];
    int len = -1;
    while (!stopFlag) {
        if (!message.isEmpty()) {
            Log.d( tag: "socket", msg: "UDP客户端发送 " + message);
            byte[] sendData = message.getBytes();
            /**注意用于发送的的Paceket需要制定地址**/
            datagramSocket.send(new DatagramPacket(sendData,sendData.length,InetAddress.getByName("localhost"), port: 30001));
            clearMsg();
            DatagramPacket receivePacket =  new DatagramPacket(data,data.length);
            datagramSocket.receive(receivePacket);
            String result = new String(receivePacket.getData(), receivePacket.getOffset(), receivePacket.getLength());
            Log.d( tag: "socket", msg: "UDP客户端收到 " + result);
        }
        sleep( millis: 300);
    }
    Log.d( tag: "socket", msg: "客户端断开 ");
    datagramSocket.close();

} catch (Exception e) {
    e.printStackTrace();
}
```

创建socket

发送数据

接收数据，receive()



Android中的Http相关库

HttpURLConnection

Java提供的Http客户端接口，底层基于OkHttp

Retrofit

Square公司开源的Http请求接口，基于OkHttp

OkHttp

实现了Http协议的客户端，功能强大，Square公司开源；从4.4版本开始Android的默认Http 实现

ByteDance 字节跳动

示例

通过 github API 获取 Android 超级大牛 Jake Wharton 仓库列表，分页获取，每页10条数据，

显示每个仓库的名称、fork数量和star数量

List repositories for a user

Lists public repositories for the specified user.

`GET` `/users/{username}/repos`

Parameters

Name	Type	In	Description
`accept`	string	header	Setting to `application/vnd.github.v3+json` is recommended. See preview notice
`username`	string	path	
`type`	string	query	Can be one of `all`, `owner`, `member`.
`sort`	string	query	Can be one of `created`, `updated`, `pushed`, `full_name`.
`direction`	string	query	Can be one of `asc` or `desc`. Default: `asc` when using `full_name`, otherwise `desc`
`per_page`	integer	query	Results per page (max 100).
`page`	integer	query	Page number of the results to fetch.

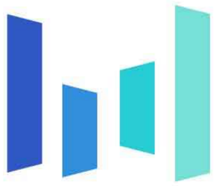
例如

https://api.github.com/users/JakeWharton/repos?page=0&per_page=10

1 添加网络权限

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bytedance.network">

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Network"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

ByteDance 字节跳动

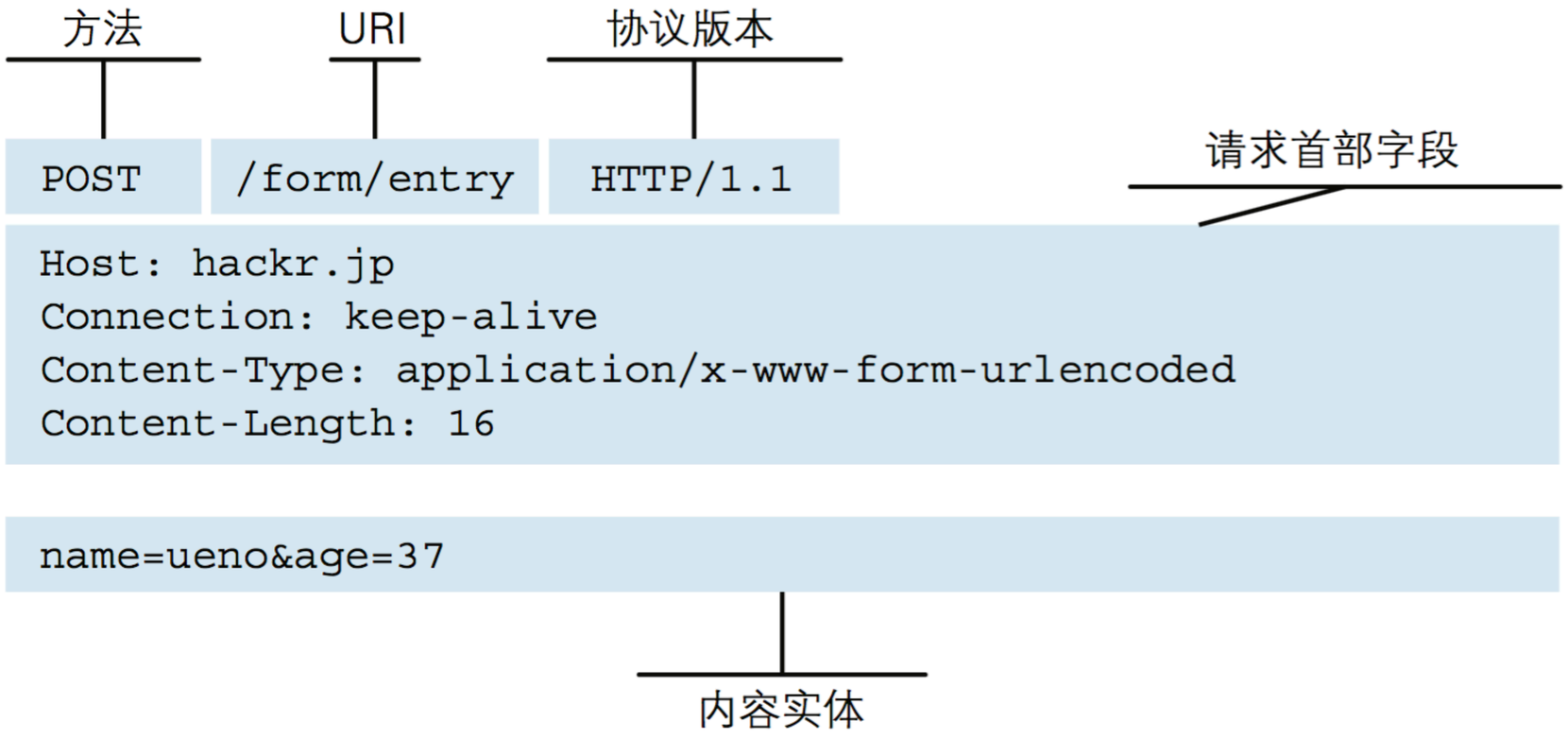


2 创建一个新的线程，在这个线程中执行网络任务

```
private void requestBase(final String userName){
    new Thread(new Runnable() {
        @Override
        public void run() {
            List<Repo> repos = baseGetReposFromRemote(userName);
            if (repos != null && !repos.isEmpty()){
                page++;
                new Handler(getMainLooper()).post(new Runnable() {
                    @Override
                    public void run() {
                        showRepos(repos);
                    }
                });

            }
        }
    }).start();
}
```

问：如果在主线程进行网络请求会有什么后果?

3 用HttpURLConnection建立连接获取数据



图：请求报文的构成

```java
public List<Repo> baseGetReposFromRemote(String userName, int page, int perPage, String accept) {
    String urlStr =
        String.format("https://api.github.com/users/%s/repos?page=%d&per_page=%d", userName, page, perPage);
    List<Repo> result = null;
    try {
        URL url = new URL(urlStr);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setConnectTimeout(6000);

        conn.setRequestMethod("GET");

        conn.setRequestProperty("accept", accept);

        if (conn.getResponseCode() == 200) {

            InputStream in = conn.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(in, StandardCharsets.UTF_8));

            result = new Gson().fromJson(reader, new TypeToken<List<Repo>>() {
            }.getType());

            reader.close();
            in.close();

        } else {
            // 错误处理
        }
        conn.disconnect();

    } catch (Exception e) {
        e.printStackTrace();
        Toast.makeText( context: this,  text: "网络异常" + e.toString(), Toast.LENGTH_SHORT).show();
    }
    return result;
}
```

拼接url

由url获得连接对象

设置http method

设置header

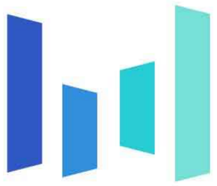
判断返回的code

从InputStream读取数据

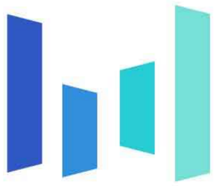
Gson解析

断开连接

ByteDance 字节跳动



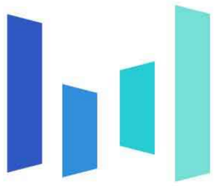
- 完整的url字符串
- 创建URL对象
- 通过URL#openConnection() 强制转换得到HttpURLConnection 连接对象
- 设置header、超时、cookies等属性
- 通过HttpURLConnection#getOutputStream() 输出流写入request body
- 判断返回的code
- 从HttpURLConnection#getInputStream() 中读取返回内容
- 解析内容并使用



4 在主线程更新界面

```
private void requestBase(final String userName){
    new Thread(new Runnable() {
        @Override
        public void run() {
            List<Repo> repos = baseGetReposFromRemote(userName);
            if (repos != null && !repos.isEmpty()){
                page++;
                new Handler(getMainLooper()).post(new Runnable() {
                    @Override
                    public void run() {
                        showRepos(repos);
                    }
                });

            }
        }
    }).start();
}
```

进阶实现——Retrofit



retrofit

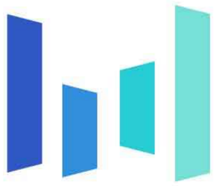
Java注解

- Java 注解用于为 Java 代码提供元数据。
- 注解不直接影响你的代码执行
- 使用 @XXX
- 定义 @interface XXX

```
@Override
protected void onCreate(Bundle savedInstanceState) {
```

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.SOURCE)
public @interface Override {
}
```

ByteDance 字节跳动



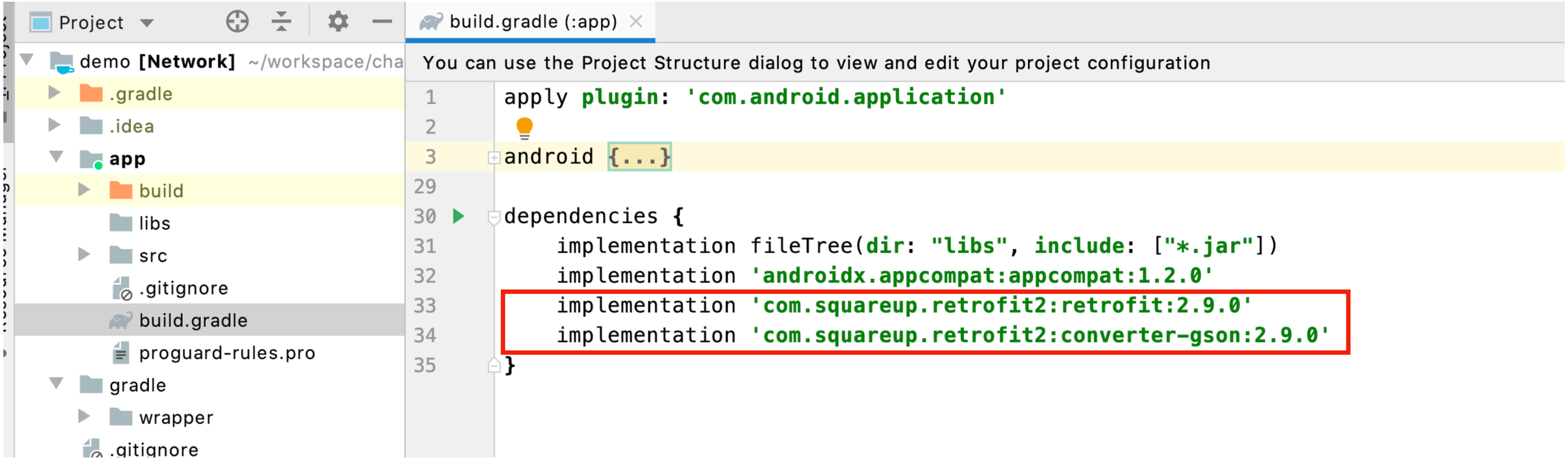
元注解（meta-annotation）：作用于注解的注解

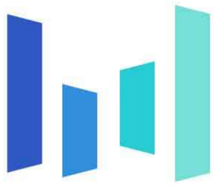
- @Target 注解作用的类型
  - FIELD, METHOD, PARAMETER, CONSTRUCTOR, LOCAL_VARIABLE, ANNOTATION_TYPE, PACKAGE, TYPE_PARAMETER, TYPE_USE
- @Retention 注解的保留时间范围

  SOURCE CLASS RUNTIME
- @Documented javadoc生成文档时是否保留注解信息
- @Inherited 使注解具有继承性
- @Repeatable （java1.8加入）

retrofit

1、添加 retrofit 依赖与retrofit-gson 依赖

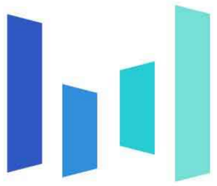




retrofit

2、创建 java 类，加上Gson注解

```java
public class Repo {
    // 仓库名
    @SerializedName("name")
    private String name;

    // 仓库描述
    @SerializedName("description")
    private String description;

    // 仓库 fork 数量
    @SerializedName("forks_count")
    private long forksCount;

    // 仓库 star 数量
    @SerializedName("stargazers_count")
    private long starsCount;

    public String getName() { return name; }

    public String getDescription() { return description; }

    public long getForksCount() { return forksCount; }

    public long getStarsCount() { return starsCount; }
}
```



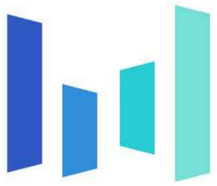
retrofit

3 、根据 api 信息（请求方法、URL、响应体结构等）创建 retrofit 所需要的接口方法

```
public interface GitHubService {
    /**
     * 假设我们现在传入的 userName 参数为 JakeWharton，那么经过在程序运行时，经过 @Path 的匹配
     * 替换掉 {username} 之后就会变成 users/JakeWharton/repos
     */
    @GET("users/{username}/repos")
    Call<List<Repo>> getRepos(@Path("username") String userName,
                              @Query("page") int page,
                              @Query("per_page") int perPage,
                              @Header("accept") String accept);
}
```

@Path: 替换url中的path；@Query 在url中加入query；@Header在http头部中加入键值对

ByteDance 字节跳动



Retrofit的注解

标识Http方法

@GET、@POST、@HEAD、@PUT、@DELETE、@OPTIONS、@PATCH…

标识参数类型方法

@Query、@QueryMap、@Body、@Header、@Field、@Tag、@Path…

标识参数类型方法

@Multipart、@Streaming、FormUrlEncoded

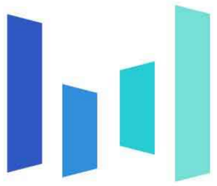
ByteDance 字节跳动

retrofit

4、创建 retrofit 对象

```
private final Retrofit retrofit = new Retrofit.Builder()
        .baseUrl("https://api.github.com/")
        .addConverterFactory(GsonConverterFactory.create())
        .build();
```

GsonConverterFactory 是 retrofit 用来将 json 转换成 java 类的工具，
如果你想用 jackson 的，有 JacksonConverterFactory



retrofit

5、用retrofit对象和api的class动态创建接口服务对象

```
GitHubService service = retrofit.create(GitHubService.class);
```

这里用了动态代理技术，可以想象这句代码干了下面的事

```
GitHubService service = new GitHubService() {
    @Override
    public Call<List<Repo>> getRepos(String userName, int page, int perPage, String accept) {
        //这里用了动态代理的技术,

        Call<Response> call = new Call() {
            @Override
            public Response execute() throws IOException {
                // 这个方法将传入的参数配置好，进行http请求，并将结果用GsonConvert转成List<Repo>对象
                return response;
            }
        };
        return call;
    }
};
```

retrofit

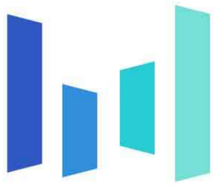
6、调用接口的方法，向远端服务器发送请求

```
private void requestRetrofitSync(final String userName){
    new Thread(new Runnable() {
        @Override
        public void run() {
            GitHubService service = retrofit.create(GitHubService.class);
            Call<List<Repo>> call = service.getRepos(userName, page, perPage: 10, accept: "application/vnd.github.v3+json");
            try {
                Response<List<Repo>> response = call.execute();
                if (response.isSuccessful() && !response.body().isEmpty()){
                    page++;
                    new Handler(getMainLooper()).post(new Runnable() {
                        @Override
                        public void run() {
                            showRepos(response.body());
                        }
                    });

                }
            } catch (IOException e) {
                e.printStackTrace();
            }

        }
    }).start();
}
```

首先调用service的方法，得到一个Call对象

一个Call对象表示对远端服务器的一次请求

调用call.execute执行这次请求



retrofit

7、根据Response判断成功失败，并切回主线程更新UI

```
private void requestRetrofitSync(final String userName){
    new Thread(new Runnable() {
        @Override
        public void run() {
            GitHubService service = retrofit.create(GitHubService.class);
            Call<List<Repo>> call = service.getRepos(userName, page,  perPage: 10, accept: "application/vnd.github.v3+json");
            try {
                Response<List<Repo>> response = call.execute();
                if (response.isSuccessful() && !response.body().isEmpty()){
                    page++;
                    new Handler(getMainLooper()).post(new Runnable() {
                        @Override
                        public void run() {
                            showRepos(response.body());
                        }
                    });

                }
            } catch (IOException e) {
                e.printStackTrace();
            }

        }
    }).start();
}
```

ByteDance 字节跳动

URLConnection vs Retrofit

```
private void requestBase(final String userName) {
    new Thread(new Runnable() {
        @Override
        public void run() {
            List<Repo> repos = baseGetReposFromRemote(
                userName, page,  perPage: 10,  accept: "application/vnd.github.v3+json")
            if (repos != null && !repos.isEmpty()) {
                page++;
                new Handler(getMainLooper()).post(new Runnable() {
                    @Override
                    public void run() {
                        showRepos(repos);
                    }
                });

            }
        }
    }).start();
}
```

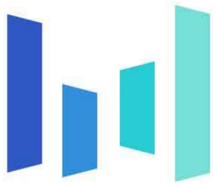
这里面复杂的工作Retrofit做掉了

```
private void requestRetrofitSync(final String userName){
    new Thread(new Runnable() {
        @Override
        public void run() {
            GitHubService service = retrofit.create(GitHubService.class);
            Call<List<Repo>> call = service.getRepos(userName, page,  perPage: 10, accept: "applic
            try {
                Response<List<Repo>> response = call.execute();
                if (response.isSuccessful() && !response.body().isEmpty()){
                    page++;
                    new Handler(getMainLooper()).post(new Runnable() {
                        @Override
                        public void run() {
                            showRepos(response.body());
                        }
                    });

                }
            } catch (IOException e) {
                e.printStackTrace();
            }

        }
    }).start();
}
```

ByteDance 字节跳动

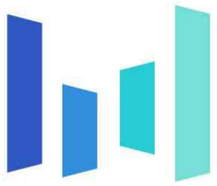
Retrofit提供了同步和异步两种调用方式，其异步方式处理了线程切换

```
/**
 * Synchronously send the request and return its response.
 *
 * @throws IOException if a problem occurred talking to the server.
 * @throws RuntimeException (and subclasses) if an unexpected error occurs creating the request or
 *     decoding the response.
 */
Response<T> execute() throws IOException;

/**
 * Asynchronously send the request and notify {@code callback} of its response or if an error
 * occurred talking to the server, creating the request, or processing the response.
 */
void enqueue(Callback<T> callback);

/**
 * Returns true if this call has been either {@linkplain #execute() executed} or {@linkplain
 * #enqueue(Callback) enqueued}. It is an error to execute or enqueue a call more than once.
 */
```



```java
private void requestRetrofitAsync(String userName) {

    GitHubService service = retrofit.create(GitHubService.class);

    Call<List<Repo>> repos = service.getRepos(userName, page, perPage: 10, accept: "application/vnd.github.v3+json");
    repos.enqueue(new Callback<List<Repo>>() {
        @Override
        public void onResponse(final Call<List<Repo>> call, final Response<List<Repo>> response) {
            if (!response.isSuccessful()) {
                return;
            }
            final List<Repo> repoList = response.body();
            if (repoList == null || repoList.isEmpty()) {
                return;
            }
            page++;
            showRepos(repoList);
        }

        @Override
        public void onFailure(final Call<List<Repo>> call, final Throwable t) {
            t.printStackTrace();
        }
    });
}
```

作业



实现一个留言板

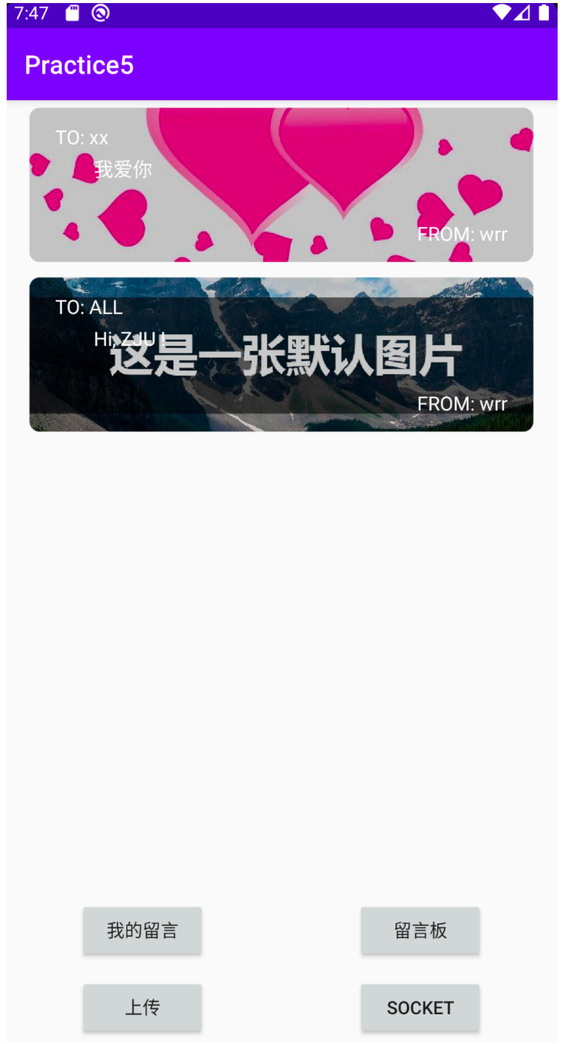
（1）通过HttpUrlConnection 的方式拉取留言列表

（2）通过Retrofit 的异步接口方式实现提交留言

（3）选做: 通过HttpUrlConnection方式实现留言提交

用Socket实现简单的Head请求

用Socket的方式实现简单Head请求


7:47
Practice5
TO: xx
我爱你
FROM: wrr
TO: ALL
Hi,ZJU!
这是一张默认图片
FROM: wrr
我的留言
留言板
上传
SOCKET


11:40
Practice5
选择封面
TA的名字
想对TA说的话
提交

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bytedance.practice5">
    <!--   TODO 0 补上网络权限-->
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

```java
public class Constants {
    public static final String BASE_URL = "https://api-sjtu-camp-2021.bytedance.com/homework/invoke/";
    //TODO 1这里写上自己的学号和名字
    public static final String STUDENT_ID = "1234";
    public static final String USER_NAME = "我的名字";
```

```java
//TODO 2
// 用HttpUrlConnection实现获取留言列表数据，用Gson解析数据，更新UI（调用adapter.setData()方法）
// 注意网络请求和UI更新分别应该放在哪个线程中
private void getData(String studentId){

}
```

ByteDance 字节跳动

```
private void initNetwork() {
    //TODO 3
    // 创建Retrofit实例
    // 生成api对象
}
```

```
public interface IApi {

    //TODO 4
    // 补全所有注解
    Call<UploadResponse> submitMessage(String studentId,
                                       String extraValue,
                                       MultipartBody.Part from,
                                       MultipartBody.Part to,
                                       MultipartBody.Part content,
                                       MultipartBody.Part image);
}
```

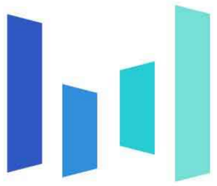
```
//TODO 5
// 使用api.submitMessage()方法提交留言
// 如果提交成功则关闭activity，否则弹出toast
```

ByteDance 字节跳动

```java
@Override
public void run() {
    // TODO 6 用socket实现简单的HEAD请求（发送content）
    //  将返回结果用callback.onresponse(result)进行展示
}
```

```java
// TODO 7 选做 用URLConnection的方式实现提交
private void submitMessageWithURLConnection(){

}
```

ByteDance 字节跳动



- 提交到你的 Repository 中 Homework-Chapter-5 目录。
- 截止日期：7月16日 24:00
- 将你的github地址用邮件发送到wenruri@bytedance.com; 邮件标题 浙大+姓名+学号+第五讲

Q&A

ByteDance 字节跳动

THANKS