第一章 HTTP 获取城市温度

1. 学习目的及目标

- 掌握 HTTP 原理和工作过程
- 掌握乐鑫 ESP32HTTP 获取服务器温度的程序设计

2. HTTP 原理

HTTP 是一套计算机网络通讯规则。下面只讲下请求格式，其他原理。

2.1. HTTP 请求格式

HTTP 请求是客户端往服务端发送请求动作，告知服务器自己的要求。其中信息由三部分组成：

1. 请求方法，URI 协议/版本：包括请求方式 Method、资源路径 URL、协议版本 Version
2. 请求头：包括一些访问的域名、用户代理、Cookie 等信息
3. 请求正文：就是 HTTP 请求的数据

备注：请求方式 Method 一般有 GET、POST、PUT、DELETE，含义分别是获取、修改、上传、删除，其中 GET 方式仅仅为获取服务器资源，方式较为简单，因此在请求方式为 GET 的 HTTP 请求数据中，请求正文部分可以省略，直接将想要获取的资源添加到 URL 中。下图所示就是 GET 的请求，没有请求正文。详细的说明在下边。

现在大多数协议版本为 http/1.1

```
1  GET/sample.jspHTTP/1.1
2  Accept:image/gif.image/jpeg,*/*
3  Accept-Language:zh-cn
4  Connection:Keep-Alive
5  Host:localhost
6  User-Agent:Mozila/4.0(compatible;MSIE5.01;Window NT5.0)
7  Accept-Encoding:gzip,deflate
8
9  username=jinqiao&password=1234
```

2.2. HTTP 应答格式

服务器收到了客户端发来的 HTTP 请求后，根据 HTTP 请求中的动作要求，服务端做出具体的动作，将结果回应给客户端，称为 HTTP 响应。数据主要由三部分组成：

1. 协议状态：包括协议版本 Version、状态码 Status Code、回应短语
2. 响应头：包括搭建服务器的软件，发送响应的时间，回应数据的格式等信息
3. 响应正文：就是响应的具体数据

```
1  HTTP/1.1 200 OK
2  Server:Apache Tomcat/5.0.12
3  Date:Mon,6Oct2003 13:23:42 GMT
4  Content-Length:112
```

```
5
6  data(返回数据)
```

备注：我们主要关心并且能够在客户端浏览器看得到的是三位数的状态码，不同的状态码代表不同的含义，其中

1xx	表示 HTTP 请求已经接受，继续处理请求
2xx	表示 HTTP 请求已经处理完成
3xx	表示把请求访问的 URL 重定向到其他目录
4xx	表示客户端出现错误
5xx	表示服务端出现错误

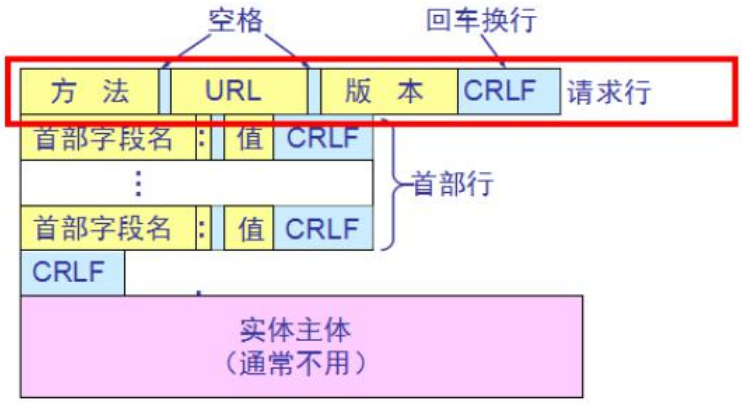
常见状态码的含义：

1. 200---OK/请求已经正常处理完毕
2. 301---/请求永久重定向
3. 302---/请求临时重定向
4. 304---/请求被重定向到客户端本地缓存
5. 400---/客户端请求存在语法错误
6. 401---/客户端请求没有经过授权
7. 403---/客户端的请求被服务器拒绝，一般为客户端没有访问权限
8. 404---/客户端请求的 URL 在服务端不存在
9. 500---/服务端永久错误
10. 503---/服务端发生临时错误

2.3. HTTP 报文格式（原文）

HTTP 报文是 HTTP 应用程序之间传输的数据块，HTTP 报文分为 HTTP 请求报文和 HTTP 响应报文，但是无论哪种报文，他的整体格式是类似的，大致都是由起始、首部、主体三部分组成，起始说明报文的动作，首部说明报文的属性，主体则是报文的数据。接下来具体说明。

HTTP 请求报文



请求报文的起始由请求行构成（有些资料称为状态行，名字不一样而已，都是指的一个东西），用来说明该请求想要做什么，由<Method>、<URL>、<Version> 三个字段组成，注

意每个字段之间都有一个空格。

其中<Method>字段有不同的值：

GET --- 访问服务器的资源

POST --- 向服务器发送要修改的数据

HEAD --- 获取服务器文档的首部

PUT --- 向服务器上传资源

DELETE--- 删除服务器的资源

<URL>字段表示服务器的资源目录定位

<Version>字段表示使用的 http 协议版本

首部部分由多个请求头（也叫首部行）构成，那些首部字段名有如下，不全：

Accept 指定客户端能够接收的内容格式类型

Accept-Language 指定客户端能够接受的语言类型

Accept-Ecoding 指定客户端能够接受的编码类型

User-Agent 用户代理，向服务器说明自己的操作系统、浏览器等信息

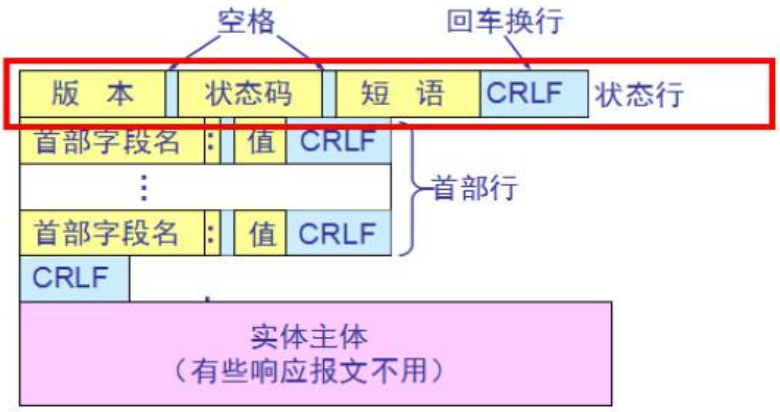
Connection 是否开启持久连接（keepalive）

Host 服务器域名

...

主体部分就是报文的具体数据。

HTTP 响应报文



响应报文的起始由状态行构成，用来说明服务器做了什么，由<Version>、<Status-Code>、<Phrase>三个字段组成，同样的每个字段之间留有空格；

<Status-Code> 上边已经说明；

首部由多个响应头(也叫首部行)组成， 首部字段名如下，不全：

Server 服务器软件名，Apache/Nginx

Date 服务器发出响应报文的时间

Last-Modified 请求资源的最后的修改时间

...

主体部分是响应报文的具体数据。

3. JSON 解析

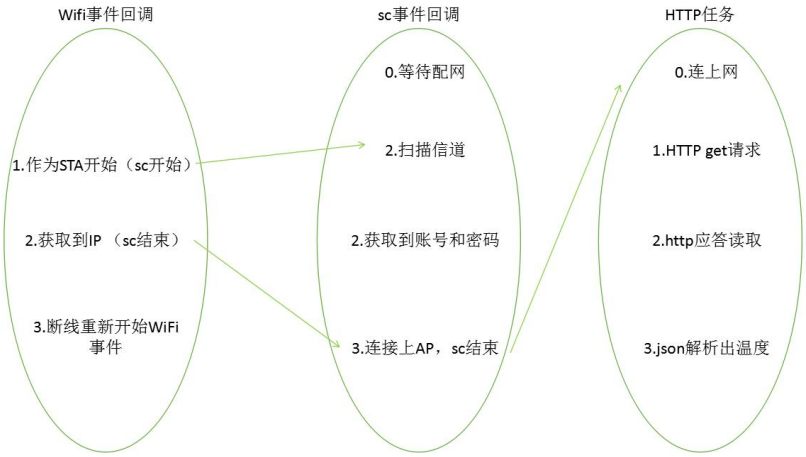
此处 HTTP 获取城市的温度是访问心知天气的服务器，心知天气返回的数据是 json 格式，那么我们就需要使用到第三方的开源库 cJSON 了。ESP32 的 SDK 已经自带这些移植好的库了，我们只需要直接使用即可。天气预报的数据格式如下所示：

```
{
  "results": [{
    "location": {
      "id": "WX4FBXXFKE4F",
      "name": "北京",
      "country": "CN",
      "path": "北京,北京,中国",
      "timezone": "Asia/Shanghai",
      "timezone_offset": "+08:00"
    },
    "daily": [{                        //返回指定 days 天数的结果
      "date": "2015-09-20",            //日期
      "text_day": "多云",              //白天天气现象文字
      "code_day": "4",                 //白天天气现象代码
      "text_night": "晴",              //晚间天气现象文字
      "code_night": "0",               //晚间天气现象代码
      "high": "26",                    //当天最高温度
      "low": "17",                     //当天最低温度
      "precip": "0",                   //降水概率，范围 0~100，单位百分比
      "wind_direction": "",            //风向文字
      "wind_direction_degree": "255",   //风向角度，范围 0~360
      "wind_speed": "9.66",            //风速，单位 km/h（当 unit=c 时）、mph（当 unit=f 时）
      "wind_scale": ""                 //风力等级
    }, {
      "date": "2015-09-21",
      "text_day": "晴",
      "code_day": "0",
      "text_night": "晴",
      "code_night": "0",
      "high": "27",
      "low": "17",
      "precip": "0",
      "wind_direction": "",
      "wind_direction_degree": "157",
      "wind_speed": "17.7",
      "wind_scale": "3"
    }, {
      ...                              //更多返回结果
    }],
    "last_update": "2015-09-20T18:00:00+08:00" //数据更新时间（该城市的本地时间）
  }]
}
```

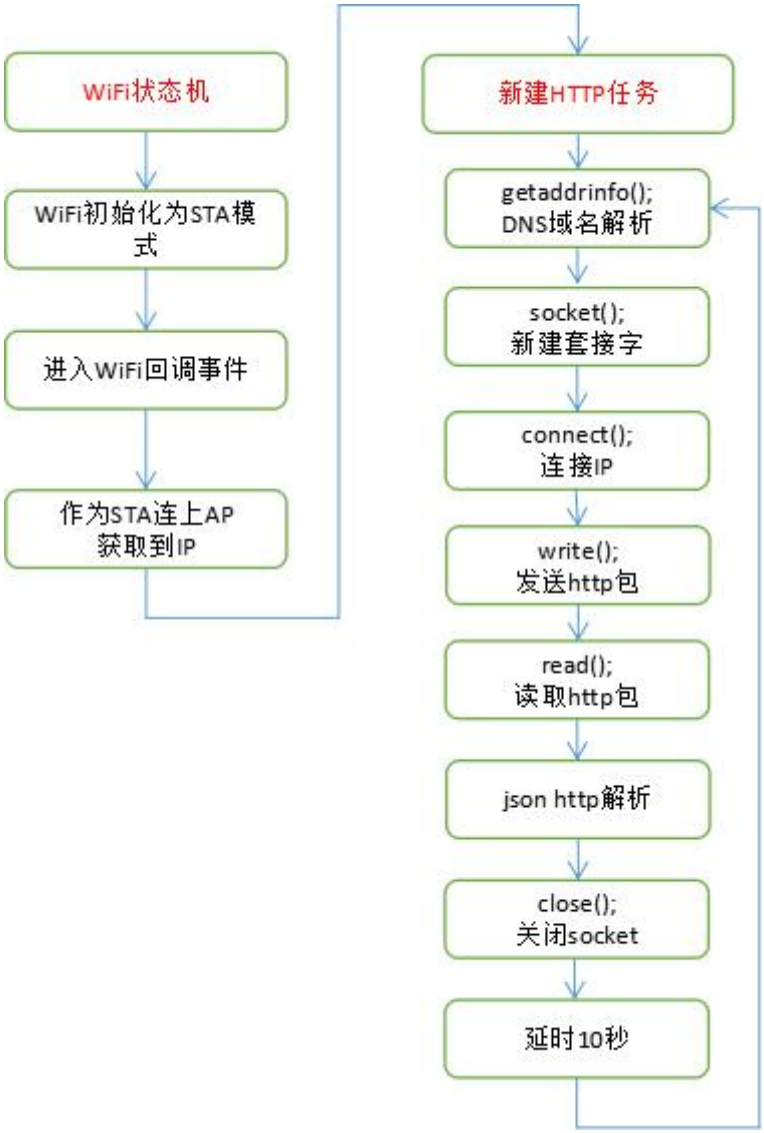
关于 cjson 的使用，可参考。

4. 软件设计

4.1. HTTP 获取城市温度的主逻辑



4.2. ESP32 的 HTTP 详细过程逻辑



4.3. ESP32 的 HTTP 接口介绍

同 TCP 接口，因为此处是使用 TCP 数据包模拟 HTTP 包，完成发送和读取。

更多更详细接口请参考官方指南。

4.4. ESP32 的 HTTP 总结

初始化 wifi 配置后，程序会根据 WIFI 的实时状态，在回调函数中给出状态返回，所以只需要在回调中进行相关操作，STA 开始事件触发 SC 开始进行一键配置，在 SC 的回调中处理 SC 配置过程的事件，sc 完成后,WIFI 连上网后，就开始了 HTTP 的工作过程，很简单。

4.5. HTTP 任务编写

```
void http_get_task(void *pvParameters)
{
    const struct addrinfo hints = {
        .ai_family = AF_INET,
        .ai_socktype = SOCK_STREAM,
    };
    struct addrinfo *res;
    struct in_addr *addr;
    int s, r;
    char recv_buf[1024];
    char mid_buf[1024];
    int index;
    while(1) {

        //DNS 域名解析
        int err = getaddrinfo(WEB_SERVER, WEB_PORT, &hints, &res);
        if(err != 0 || res == NULL) {
            ESP_LOGE(HTTP_TAG, "DNS lookup failed err=%d res=%p\r\n", err, res);
            vTaskDelay(1000 / portTICK_PERIOD_MS);
            continue;
        }

        //打印获取的 IP
        addr = &((struct sockaddr_in *)res->ai_addr)->sin_addr;
        //ESP_LOGI(HTTP_TAG, "DNS lookup succeeded. IP=%s\r\n", inet_ntoa(*addr));

        //新建 socket
        s = socket(res->ai_family, res->ai_socktype, 0);
        if(s < 0) {
            ESP_LOGE(HTTP_TAG, "... Failed to allocate socket.\r\n");
            close(s);
            freeaddrinfo(res);
            vTaskDelay(1000 / portTICK_PERIOD_MS);
            continue;
```

```
        }
        //连接 ip
        if(connect(s, res->ai_addr, res->ai_addrlen) != 0) {
            ESP_LOGE(HTTP_TAG, "... socket connect failed errno=%d\r\n", errno);
            close(s);
            freeaddrinfo(res);
            vTaskDelay(4000 / portTICK_PERIOD_MS);
            continue;
        }
        freeaddrinfo(res);
        //发送 http 包
        if (write(s, REQUEST, strlen(REQUEST)) < 0) {
            ESP_LOGE(HTTP_TAG, "... socket send failed\r\n");
            close(s);
            vTaskDelay(4000 / portTICK_PERIOD_MS);
            continue;
        }
        //清缓存
        memset(mid_buf,0,sizeof(mid_buf));
        //获取 http 应答包
        do {
            bzero(recv_buf, sizeof(recv_buf));
            r = read(s, recv_buf, sizeof(recv_buf)-1);
            strcat(mid_buf,recv_buf);
        } while(r > 0);
        //json 解析
        cjson_to_struct_info(mid_buf);
        //关闭 socket，http 是短连接
        close(s);

        //延时一会
        for(int countdown = 10; countdown >= 0; countdown--) {
            vTaskDelay(1000 / portTICK_PERIOD_MS);
        }
    }
}
```

4.6. 温度数据 JSON 解析

```
void cjson_to_struct_info(char *text)
{
    cJSON *root,*psub;
    cJSON *arrayItem;
    //截取有效 json
    char *index=strchr(text,'{');
    strcpy(text,index);

```

```
    root = cJSON_Parse(text);

    if(root!=NULL)
    {
        psub = cJSON_GetObjectItem(root, "results");
        arrayItem = cJSON_GetArrayItem(psub,0);

        cJSON *locat = cJSON_GetObjectItem(arrayItem, "location");
        cJSON *now = cJSON_GetObjectItem(arrayItem, "now");
        if((locat!=NULL)&&(now!=NULL))
        {
            psub=cJSON_GetObjectItem(locat,"name");
            sprintf(weathe.cit,"%s",psub->valuestring);
            ESP_LOGI(HTTP_TAG,"city:%s",weathe.cit);

            psub=cJSON_GetObjectItem(now,"text");
            sprintf(weathe.weather_text,"%s",psub->valuestring);
            ESP_LOGI(HTTP_TAG,"weather:%s",weathe.weather_text);

            psub=cJSON_GetObjectItem(now,"code");
            sprintf(weathe.weather_code,"%s",psub->valuestring);
            //ESP_LOGI(HTTP_TAG,"%s",weathe.weather_code);

            psub=cJSON_GetObjectItem(now,"temperature");
            sprintf(weathe.temperatur,"%s",psub->valuestring);
            ESP_LOGI(HTTP_TAG,"temperatur:%s",weathe.temperatur);


//ESP_LOGI(HTTP_TAG,"--->city %s,weather %s,temperature %s<---\r\n",weathe.cit,weathe.weather_text,
weathe.temperatur);
        }
    }
    //ESP_LOGI(HTTP_TAG,"%s 222",__func__);
    cJSON_Delete(root);
}
```

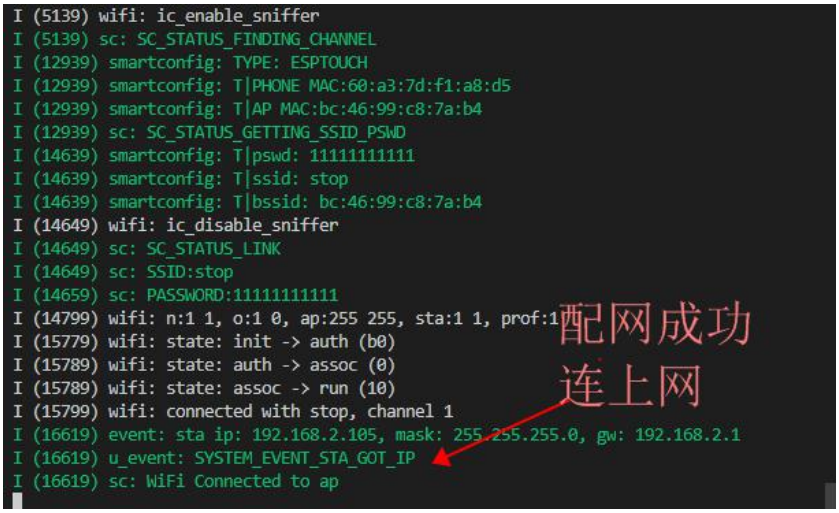
5. 效果展示

5.1. 测试流程

- SmartConfig 快配账号密码
- 自动连服务器获取温度
- 串口打开即可看获取的温度

5.2. 效果展示

- SmartConfig 配置


I (5139) wifi: ic_enable_sniffer
I (5139) sc: SC_STATUS_FINDING_CHANNEL
I (12939) smartconfig: TYPE: ESPTOUCH
I (12939) smartconfig: T|PHONE MAC:60:a3:7d:f1:a8:d5
I (12939) smartconfig: T|AP MAC:bc:46:99:c8:7a:b4
I (12939) sc: SC_STATUS_GETTING_SSID_PSWD
I (14639) smartconfig: T|pswd: 11111111111
I (14639) smartconfig: T|ssid: stop
I (14639) smartconfig: T|bssid: bc:46:99:c8:7a:b4
I (14649) wifi: ic_disable_sniffer
I (14649) sc: SC_STATUS_LINK
I (14649) sc: SSID:stop
I (14659) sc: PASSWORD:11111111111
I (14799) wifi: n:1 1, o:1 0, ap:255 255, sta:1 1, prof:1
I (15779) wifi: state: init -> auth (b0)
I (15789) wifi: state: auth -> assoc (0)
I (15789) wifi: state: assoc -> run (10)
I (15799) wifi: connected with stop, channel 1
I (16619) event: sta ip: 192.168.2.105, mask: 255.255.255.0, gw: 192.168.2.1
I (16619) u_event: SYSTEM_EVENT_STA_GOT_IP
I (16619) sc: WiFi Connected to ap
配网成功
连上网

➢ 获取温度


I (19679) sc: smartconfig over
I (19759) http_task: city:Suzhou
I (19769) http_task: weather:Clear
I (19769) http_task: temperatur:0
I (30839) http_task: city:Suzhou
I (30839) http_task: weather:Clear
I (30839) http_task: temperatur:0
I (41909) http_task: city:Suzhou
I (41909) http_task: weather:Clear
I (41909) http_task: temperatur:0
I (52989) http_task: city:Suzhou
I (52989) http_task: weather:Clear
I (52989) http_task: temperatur:0
I (69669) http_task: city:Suzhou
I (69669) http_task: weather:Clear
I (69669) http_task: temperatur:0
I (83459) http_task: city:Suzhou
I (83459) http_task: weather:Clear
I (83459) http_task: temperatur:0
温度


苏州温度
01月01日 周二 农历十一月廿六
2 ℃
晴(实时)
0 ~ 6℃
阴转多云
北风3-4级
70 良
周三
01月02日
0 ~ 8℃
多云
东北风微风
良
周四
01月03日
4 ~ 10℃
多云转小雨
东风微风
良
周五
01月04日
7 ~ 11℃
小雨
东风微风
良
周六
01月05日
6 ~ 9℃
小雨
无持续风向微风
优
15
10
5
0
-5
2℃
0℃
0℃
1℃
6℃
7℃
7℃
5℃
4℃
现在
02点
05点
08点
11点
14点
17点
20点
23点
24小时温度
风力风向
降水量
相关指数

6. HTTP 总结

- 此处 HTTP 是使用 TCP 模拟的，所以过程和 TCP 章类似。
- HTTP 部分参考官方的源码，sc 参考官方源码。
- Sc 没有保存密码。
- 源码地址：https://github.com/xiaolongba/wireless-tech