

网络爬虫

网络爬虫是一个软件应用，可以自动化批量获取网络资源，网络爬虫工作于万维网（模拟浏览器行为），万维网是由无数网页+WEB服务器构成的庞大网络资源存储仓库，万维网是网络用户获取数据的主要渠道，爬虫爬取的网络数据用途，数据分析,数据挖掘，搜索引擎(数据),数据可视化,AI训练数据集，但是大家要注意爬虫只负责获取下载资源，资源如果二次加工，如果使用与爬虫无关，它就是一个网络数据采集/下载器。

网络爬虫的拓扑与跳转

一个网页中包含大量的链接，有些链接指向站内资源(站内跳转)，某些链接指向其他网站的资源(站外跳转)，网页拥有大量（出入链接）这表示在万维网中网页网站之间有较强的关联性，爬虫可以依靠网页关联这个性质进行拓扑，跳转到其他站点或网页继续爬取，理论上爬虫可以拓扑到整个万维网中所有的网页

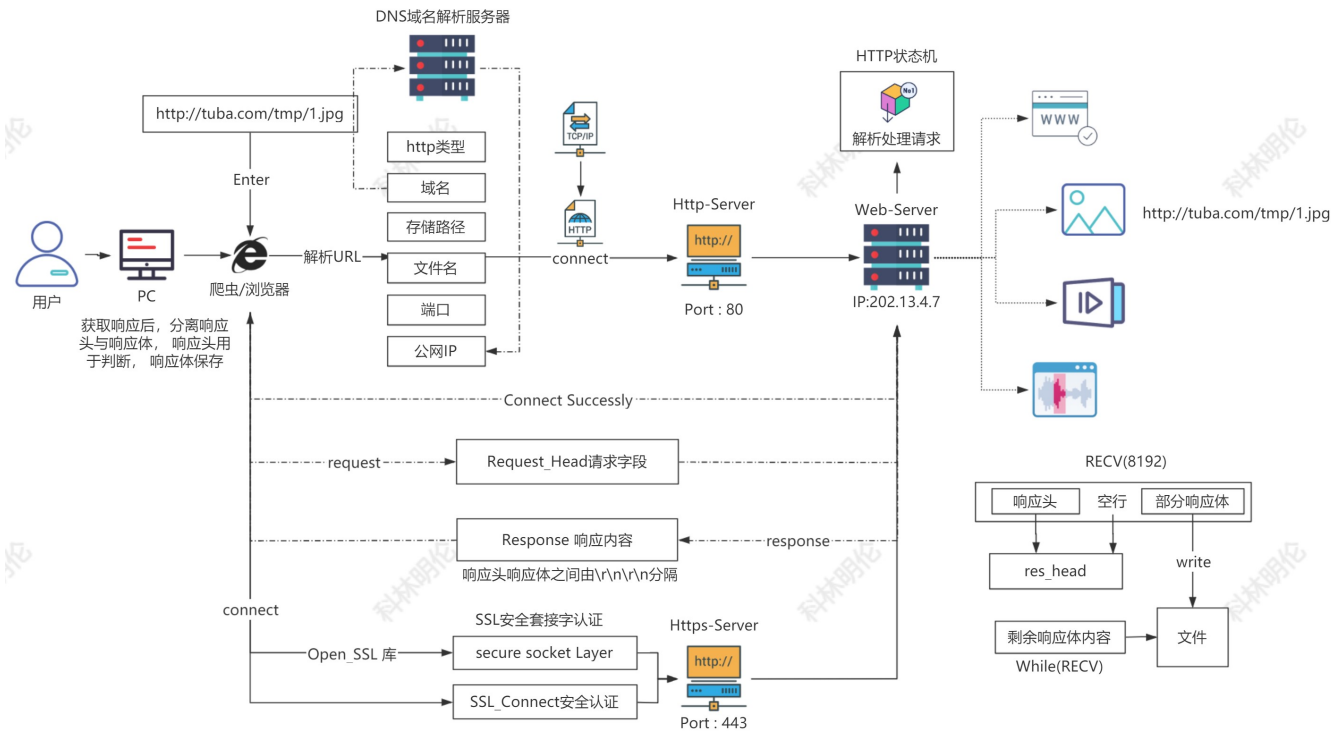
网络资源

- 文本资源
 - HTML , SHTML , XML , TXT , CSS....
- 图形图像
 - JPEG , BMP , GIF , PNG
- 视频资源
 - RMVB , FLV , AVI , MP4
- 音频
 - MP3

Web相关知识

- 万维网B/S架构 B=浏览器 S=Web服务器
- 网页(html + css) index(网站首页)
 - html文件:网页源码文件，网页源码采用html超文本标记语言编写
 - css文件: 网页样式表，通过css样式脚本语言为网页设置样式
- WEB服务器
 - 与浏览器进行网络连接，提供数据支持，数据存储，业务处理等服务..
- HTTP协议
 - 超文本传输协议:浏览器与服务器交互的主要协议，用于传输网络资源(网页等等)
 - HTTP (标准传输协议)，不安全
 - HTTPS (升级版传输协议)，具有很高的安全性，可以完成双端认证并加密传数据
 - 基于TCP实现，整体的连接过程与使用方式与tcp相同
- URL(链接): 统一资源定位符
 - 网络网络资源地址
 - url具备唯一性（全网唯一）
 - <http://tuba.top.com/cs/bin/tmp/20210631.jpg>
 - http协议类型 网页域名(服务主机) 资源存储路径 资源名 参数，替换与拼接跳转
 - 获取服务端IP (DNS域名解析器获取),分析网页域名反馈真实公网IP
 - 所有Web服务端使用通用端口号:
 - http : 80
 - https : 443

Web网络资源请求与响应过程



HTTP协议的请求与响应

- HTTP协议经典请求方式
 - GET (请求方式)
 - 请求头数据为纯字符串，里面用关键字段填充，支持请求时携带参数
 - POST (请求方式)
 - 请求头数据为纯字符串，支持用户自定义请求体
- HTTP请求头内容 (由若干特殊字段构成的字符串)
 - 1.请求方式 [GET or POST]
 - 2.请求的资源URL,资源地址
 - 3.指定HTTP协议版本
 - 4.ACCEPT，客户端可接收的数据类别及优先级(服务器按优先级别响应内容)
 - 5.User_Agent,浏览器版本号及兼容信息
 - 6.Host,服务端主机名(域名)
 - 7.Connection 连接方式
 - Keep-alive(长连接)
 - 连接成功后，双方可通过该连接持续交互，直到浏览器主动断开连接，交互结束
 - close(短连接)
 - 连接成功后，双方可通过该连接交互一次数据，而后服务端主动断开，交互结束
- HTTP响应头内容(由若干特殊字段构成的字符串)
 - 1.服务器版本信息(apache , nginx)
 - 2.服务器时间戳
 - 3.服务器响应状态码 StatusCode
 - 通过响应码判断这次请求是否成功！
 - 200 OK ,服务端成功反馈用户请求资源，200表示成功
 - 301|302，服务端资源重定向
 - 404，请求失败，网络资源失效不存在
 - 501|502，服务器异常，故障
 - 4.请求资源的大小
- 响应体内容
 - 响应的资源内容(文本、二进制、视音频流)

OPEN_SEEL库

openssl安装

```
sudo apt-get install libssl-doc #openssl文档
sudo apt-get install libssl-dev #openssl库
```

使用openssl编译时需要链接库

```
gcc *.c -I../include -lssl -lcrypto -o app
```

openssl常用函数

```
#include <openssl/ssl.h>

#include <openssl/err.h>

SSL * sslsocket; //安全套接字

SSL_CTX * sslctx; //安全认证上下文

SSL_load_error_strings(); //初始化openssl错误处理函数

SSL_library_init(); //初始化openssl库

OpenSSL_add_ssl_algorithms(); //初始化散列函数

SSL_CTX * sslctx = SSL_CTX_new(版本信息);

版本信息 = SSLv23_method();

SSL * sslsocket = SSL_new(sslctx) //使用认证上下文信息 ， 创建安全套接字

SSL_set_fd(sslsocket , webfd); //用已连接成功的webfd对sslsocket进行服务端关联设置,让sslsocket可以访问服务端

SSL_connect(sslsocket) //与https服务端完成安全认证 ， 认证成功可以交互数据
//SSL提供了读写模块， 加密解密读写

SSL_read(SSL * sslsocket , cosnt char * buffer , ssize_t rsize);

SSL_write(SSL * sslsocket , char * buffer , ssize_t wsize);

RETURN VALUE:
成功返回读到的数据量， 读取完毕返回0 ， 失败返回-1
```

SSL安全连接过程

* 单项认证与双向认证（同学回去自行查阅资料）

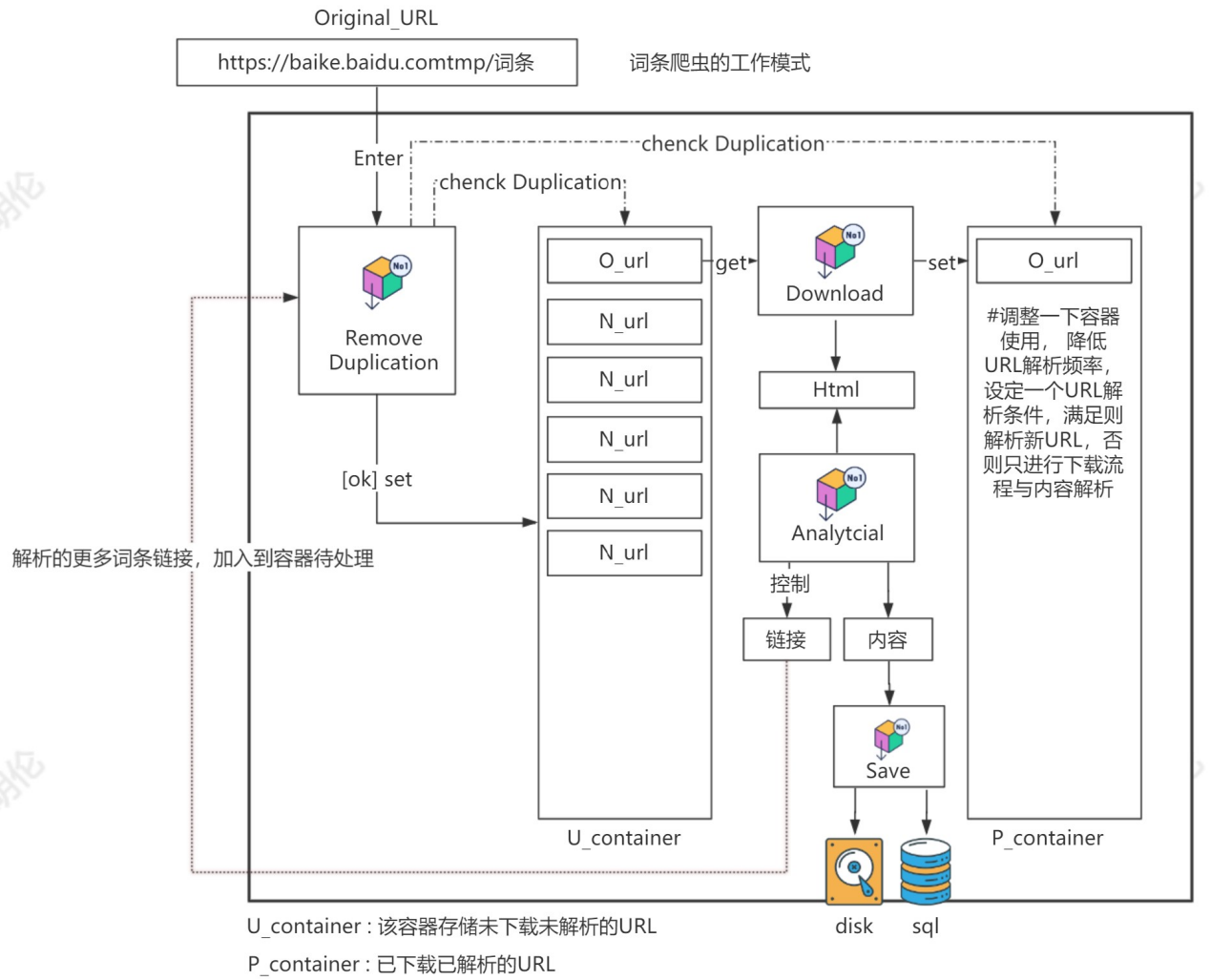
百科词条爬虫

- 爬取的关键要素，每个词条保存[词条名、词条描述、词条链接]
- 词条数据存储格式

```
<cname>词条名</cname>
<description>词条描述</description>
<url>词条链接</url>
```

- 种子URL (Original URL) 爬虫的工作起点,处理的第一条链接，种子URL是精心挑选包含更多链接数据的URL(指向网页的链接)
- 爬虫系统核心部件
 - 1).URL容器(URL管理器)
 - 2).下载器(请求响应流程)
 - 3).持久化器(存储方案:磁盘存储（如何存储？ 存储格式？） 数据库存储(如何建数据表， 表关系是怎么样的？ 数据架构？))
 - 4).解析器(正则技术:匹配提取关键数据)
 - 5).爬虫架构， 抓取策略(广度优先遍历爬虫 ， 深度优先遍历爬虫 ， 大站优先爬虫 ， 价值优先爬虫)
 - 6).反反爬虫机制
 - 7).IP池及UA池

词条爬虫工作模式



REGEX正则函数

```
#include <regex.h>

regex_t reg; //正则类型

//使用正则语句(字符串)生成正则类型
regcomp(regex_t * reg , const char * pstr , int cflags);
argv1=传出生成的正则类型
argv2=字符串正则语句
argv3=默认选项0

//通过正则类型从数据源中匹配抽取数据，每次返回一条结果,需要循环使用
regexexec(regex_t * reg , const char * data , size_t nmatch , regmatch_t * match , int eflags);
argv1=正则类型(用于模式匹配)
argv2=数据源(数据地址，使用正则语句从数据源中匹配内容)
argv3=正则数量(父表达式 +子表达式数量)
argv4=匹配成功，为了方便使用者获取匹配数据，传出数据的位置信息(偏移量)
argv5=默认选项,0

/*正则数量*/
匹配测试数据:  <a href="https://baike.baidu.com/item/词条">超链接标题</a>
匹配表达式:    "<a[^>]+?href=\"[^\"]+?\"[^\"]+?>[^\"]+?</a>"  num:1
                "<a[^>]+?href=\"([^\"]+?)\"[^\"]+?>([^\"]+?)</a>"  num:3
/* 传出位置信息 */
struct regmatch_t
{
    rm_so; //起始位置
    rm_eo; //末尾位置
}

regematch_t match[3] , 传出位置数组长度一般与表达式数量一致
匹配结果:<a href="https://baike.baidu.com/item/词条">超链接标题</a>
offset:  <=0      h=30      条=45      超=80      题=87      >=124
match[0].rm_so = 0 match[0].rm_eo = 124
match[1].rm_so =30 match[1].rm_eo=45;
//为了便于使用者抽取数据， 传出数据位置， 方便提取

regfree(regex_t *); //释放正则类型

regreor(); //正则函数错误处理，细节查看man手册
```

爬虫数据分析与模块分析

```
#include <stdio.h>

#include <stdlib.h>
```

```
#include <unistd.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <fcntl.h>

#include <openssl/ssl.h>

#include <openssl/err.h>

typedef struct{
    char alpha_url[4096]; //原始url地址
    char domain[1024]; //域名
    char save_path[1024]; //存储路径
    char domain_ip[16]; //服务端公网ip
    char save_file[1024]; //资源文件名
    int http_type; //0表示http ,1表示https
    int port; //服务端端口
}url_t;

typedef struct {
    SSL * sslsocket;
    SSL_CTX * sslctx;
}ssl_t;

typedef struct{
    url_t * node_queue;
    int front;
    int rear;
    int max;
    int cur;
}container_t;

模块分析：

//参数为url结构体，为传入传出参数，传入原始地址，传出详细信息
int spider_analytical_url(url_t * );

//网络初始化模块，创建socket并返回
int spider_net_init(void);

//与服务端进行TCP连接 ，参数为服务端网络信息
int spider_connect_webserver(int , url_t * );

//创建构造http请求头 ，传入地址，传出请求头， node为服务端url信息
int spider_create_request_head(char * head , url_t * node);

//参数为响应头，查找并返回响应头中的响应状态码(int)

int spider_get_statuscode(const char *);

//下载模块，发送请求，接收并处理响应 ，如果ssl为NULL表示http交互方式，否则https交互方式
int spider_response_download(int , char * , url_t * , ssl_t *);

//openssl，安全认证过程
ssl_t * spider_openssl_create(int webfd);

//容器创建初始化
container_t * spider_container_create(int csize);

// URL去重校验并添加到容器中 ，成功添加返回0，重复则失败返回-1
int spider_remove_duplication(container_t * , container_t * ,const char * );

//容器添加节点
int spider_container_setnode(container_t * , url_t);

//从容器中获取
int spider_container_getnode(container_t * , url_t * );

//链接解析及内容解析
int spider_analytical_html(url_t * , container_t * ,container_t *);

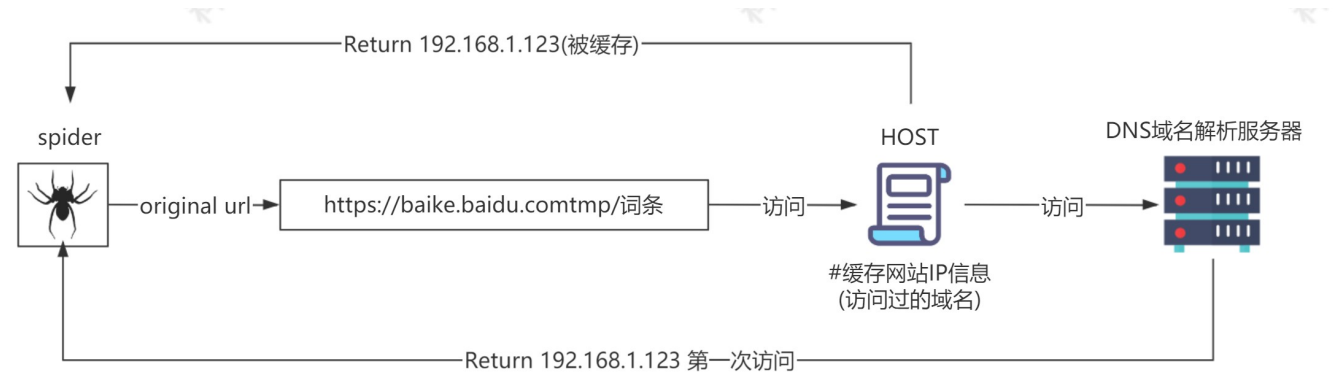
//持久化模块,将解析出的关键要素，以固定格式存储在文件中
int spider_save_data(const char * name , const char * desc , const char * url);
//爬虫控制器，抓取过程
int spider_url_controler(const char * alpha_url);
```


爬虫代码实现

课上演示并完成。


爬虫的优化选项

- URL去重优化，布隆过滤器(Hash)
- 并发爬虫（线程池并发），提高爬虫抓取效率，提高可用性
- DNS优化, 爬虫内部自建IP缓存表(内部保存曾访问过的主机IP地址)，降低访问HOST或DNS服务器的频率



 爬虫内部自建DNS缓存表(内存)，缓存访问过的网络地址信息（便于下次使用，极快获取）

 Host文件获取地址：操作磁盘IO文件读写并查找对应地址信息(小效率:中)

 DNS域名解析服务获取地址：网络请求与响应查找对应地址信息(小效率:低)

- 了解抓取策略(广度优先遍历爬虫，深度优先遍历爬虫，大站优先爬虫，价值优先爬虫)
- UserAgent池:
 - 获取大量的浏览器兼容信息，随机更换使用
- 代理IP
 - 准备大量的可用代理IP(抓取ip网站并测试连通)，更换使用，防封
- 爬虫模拟登录
 - 爬虫获取内部UI获取登录窗口，开发者自行登录(输入用户名密码)
 - 使用者提供明文用户名密码，爬虫内部完全模拟网站登录过程(掌握抓包工具使用)
- 通过反爬验证(验证码)，防反爬虫手段
 - 提高爬虫开发成本，将一大部分爬虫阻断