Linux网络爬虫系统概要

1、爬虫是什么？

获取互联网资源的一种软件，是搜索引擎的第一个子系统，数据入口之一。

2、爬虫能做什么？

网页下载器

搜索引擎的基础应用

抓取大数据的一种手段

网店秒杀

3、关于项目

在linux下实现关于爬虫的需求、设计、编码、测试、集成。

需求：是要做什么。

设计分为业务设计和技术设计。

业务设计：

业务层的方案，对于软件工程师来说，系统要完成工作的步骤和流程。

技术设计：

关于系统的框架和处理流程。注意：要采用面向对象的思维方式。

编码：使用基本的技术细节进行系统实现。

测试：是对我们是同可靠的一个监测。

集成：将每个程序员开发的模块和子系统合成一个完整的系统。

思维方式：线性思维方式与发散思维方式。

冷更新：先停机再更新。

热更新：不停机更新（一般使用插件）。

软件开发要掌握的思想：模块思维、渐进式开发。

代码结构：

Spider：

spider.conf:配置文件，系统入口

spider：

Makefile:make执行文件

download:下载文件夹

modules：模块文件夹

src：源代码文件夹

4、项目需求：自动榨取网络资源的软件

1、资源是什么？网页、图片、音乐、视频等。

2、自动化是什么样子？一旦运行就不需要更多的干预。

生成需求说明文档。

内容：对需求不明确或不完善的说明进行解释。功能点、附加要求、性能要求等。

5、设计：

业务设计：

如何思考并完成设计：

注意：从顶层开始思考并设计，避免过早的陷入细节。

系统最粗浅的数据流

设计处理流程：

1. 得到爬取的种子（URL）
2. 根据URL下载资源（页面）
3. 解析页面，提取更多的URL
4. 对页面做持久化操作
5. 根据提取的URL再进行下载操作
6. 重复第2到5步。

系统结构图

系统设计：

注意：设计阶段必不可少。

设计可以使我们的思路更清晰，可以提高工作效率，可以提高代码质量。

详细设计：

注意：对系统设计要有输入和输出。

控制器模块



控制模块由三个模块组成：

1. 配置文件处理模块：从配置文件中读取配置项，提供配置项的提取接口。
2. URL维护模块：负责维护URL库，提供如下功能：
   1. 输入新的URL
   2. 输出一个未被抓取的URL
   3. 负责维护URL的抓取状态
3. 任务调度模块：
   1. 负责协调控制器的流程
   2. 负责调用其他系统模块完成工作
4. 维护URL列表数据结构







6、控制器设计：

配置文件解析模块：

配置文件以文件形式保存程序进行时必要时的繁琐过程。

文本类型是文本文件，内容一般以键值形式出现。

概要设计：

配置文件内容：

key=value 形式

注释规则：注释字符串前以“#”标记。

配置项设置：

|  |  |
| --- | --- |
| 项目 | 字段 |
| 并发任务数 | Job\_num |
| url种子 | Seed |
| 抓取深度 | Deeps |
| 输出日志的等级 | Log\_level |
| 模块存放路径（唯一） | Module\_path |
| 模块名称（模块文件名，可以多个） | Module\_name |
| 允许抓取的资源类型（多个，文件后缀） | File\_type |
|  |  |

模块详细设计：

1. 读取配置文件
2. 得到配置文件选项的值（键值）
3. 初始化

//类设计伪代码

class ConfigParser

{

public://共有成员函数（外部接口）

ConfigParper();

load();

对应key的操作();

操作1();

操作2();

…

private://私有成员变量

job\_num

seed

deeps

log\_level

Module\_path

Module\_name

file\_type

};

技术点：

1. 字符串操作（分割字符串）
2. 按行读取fgets()
3. 消除注释
4. 消除空格

URL维护模块：

分析：

url格式：<http://192.168.40.150/docs/index.html>

结构：域名或IP地址，路径，文件名

设计URL的数据结构

|  |  |
| --- | --- |
| 项目 | 字段名称 |
| 完整的URL | url |
| 协议类型 | protocal |
| 域名 | Sitename |
| 资源路径 | Path |
| 文件名 | Filename |
| 当前url处理状态（0-未抓取，1-抓取成功，2-抓取失败） | State |
| 当前url深度 | Deep |
| 当前资源类型 | Filetype |

http协议请求页面时的流程：

1. 输入网址
2. 向DNS发送解析请求
3. DNS返回给我们一个对应的IP地址
4. 通过IP地址向资源所在的主机发送请求
5. 如果资源存在，主机返回200状态，同时返回数据部分
6. 本地http客户端（一般来说是浏览器）接收数据
7. 得到资源

页面抓取的处理流程：

1. 得到一个新的URL
2. URL进入抓取队列等待抓取
3. 从队列中得到一个URL，把其分配给一个下载器的实例
4. 得到下载器的处理状态（URL处理状态需要被改写，得到当前URL深度，得到当前资源类型假如下载成功）
5. 得到当前页面中存在的下一级URL列表



URL维护模块

URL维护模块操作（对外接口）：

1. 添加新URL
2. 使URL进入抓取队列
3. 从抓取队列中移除一个URL
4. 修改URL库中某一个URL的值
5. 添加新URL的列表

class Url

{

public:

//初始化

//set

//get

private:

//

url

protocal

sitename

path

filename

state

deep

filetype

};

class UrlManager

{

public://外部接口

addUrl();

addUrlList();

getUrlForQueue();

removeUrlForQueue();

//setUrlValue();

private://内部接口

findurl()

private:

list<Url>Urls;

map<string,Url\*> UrlMap;

queue<Url\*> Urlqueue;

};

任务调度模块：

程序的主框架流程：

要求：

1. 可以控制程序按照普通程序模式还是按照守护进程模式运行。
2. 可以通过参数提供帮助。
3. 提供一个运行框架，可以支持多任务管理。
4. 包含程序的主流程。

尽可能把可拆分的功能封装成独立函数进行调用。

定义主程序框架的处理流程：

1. 程序运行时先处理命令行参数，根据参数跳转到相应分支或调用对应的函数。
2. 检测是否按照收获进程模式运行（控制选项从命令行参数中得到）。
3. 初始化环境
   1. 读取配置文件，提取配置文件中的参数。
   2. 根据守护进程模式的标记将当前进程转变为守护进程。
   3. 载入程序模块的动态库
4. 开始程序的主处理流程
   1. 检测种子是否存在，把种子交给URL管理器
   2. 分析种子，得到种子的url的IP地址（DNS解析）
   3. 根据种子URL得到第一个页面
   4. 对页面进行处理（复杂流程，由其他模块实现细节）
   5. 从URL管理器中取出一个URL
   6. 启动一个处理任务（先检测是否达到最大任务数量，功能封装到一个独立函数中）
   7. 监控任务处理数量，如果任务维护池中有空闲任务，那么重复e步骤
   8. 回收资源，准备结束程序或当前任务

继续分解复杂步骤：

1. 对页面进行处理：
   1. 对页面进行解析，提取内部的下级url
   2. 生成url列表，把url列表传给URL管理器
   3. 对页面进行持久化操作
2. 处理任务
   1. 从url管理器得到一个未处理的URL
   2. 通过调用epoll框架产生一个新任务
   3. 调用页面处理流程（重复步骤1）
   4. 释放处理过程中产生的临时资源（Socket句柄，文件操作句柄，临时申请的内存等）

其他辅助功能：

1. 输出帮助信息

多路复用框架：

1. 为什么不用select而是用epoll

Epoll相关技术要点：

Epoll两种模式：



Epoll应用场景：



日志工具：

为什么需要日志工具？

方便调试

方便代码维护

日志输出信息设计：

日志信息等级+日期时间+调试信息

日志输出等级设计（5个等级）

0 调试[debug]:仅仅用于调试

1 普通信息[info]：可以让使用者了解的一些信息、

2 警告信息[warn]:意味着程序中出现了错误，但是并不重要。

3 错误信息[error]:意味着程序中发生了严重错误，根据实际情况可选择使程序继续运行或使程序终止。

4 程序崩溃[crash]:程序无法继续运行了。

需要控制日志信息输出到终端还是输出到日志文件中

日志调用接口设计：

SPIDER\_LOG（日志等级标记，日志输出信息）；

注意，配置文件中的日志输出等级和接口中的日志等级标记不是一个概念。

日志等级标记，纯粹是一个标记，体现在输出的日志字符串中。

配置文件中的日志输出等级字段用来控制那些日志被输出

接口内部的处理流程：

* + 1. 得到控制日志输出等级的标记，用来控制当前日志是否要输出
    2. 得到调用日志接口的时间
    3. 得到日志输出信息并进行日志字符串的拼接
    4. 把日志信息输出到指定的设备

插件框架设计：

为什么要有插件模式？

1. 升级和扩充功能
2. 维护方便
3. 动态修改

设计：

1. 动态载入.so文件
2. 维护.so文件中的接口函数

设计原则：

1. 一个功能一个模块
2. 可以自我维护
   1. 维护版本号
   2. 知道自己的名称
   3. 维护本模块内部的接口
   4. 可以对模块进行初始化

设计模块的描述结构：

|  |  |
| --- | --- |
| 字段 | 名称 |
| 主版本号 | version |
| 次版本号 | subver |
| 模块名称 | name |
| 入口函数指针 | int(\*handle)(void\*); |
| 初始化函数指针 | int(\*int)(Module\*); |

设计入口函数指针原型：

int(\*handle)(void\*);

设计初始化函数指针原型：

int(\*init)(Module\*);

模块管理器设计：

1. 载入模块的操作
   1. int load(char\* path,char\* name);
   2. Module\* getModule(char\* name);
2. 载入模块操作的处理流程：
   1. 通过路径找到模块文件(.so)
   2. 调用dlopen打开动态(.so)
   3. 使用动态库
   4. 关闭动态库文件



1. 下载器设计
   1. 下载器模块分为Socket功能 和http功能模块
   2. http协议对资源请求的操作get指令
   3. http头描述：
      1. http头以文本形式体现在http协议包中
      2. http包的内容由http头进行描述，描述的格式：一条数据一行，每条描述以描述字段名称开头

Socket功能框架设计

1. 初始化socket
2. 向服务器发送请求
3. 接受服务器反馈的数据
4. 设置socket模式

把socket句柄注册到epoll处理时间中（在主流程中）

http协议处理模块

1. 解析头
2. 组装头

提取http数据内容（单独一个模块）

设计两个模块：文本处理模块（html格式），二进制处理模块（image，jpg，png等格式），还可以继续添加其他文件处理模块（pdf，流媒体等）

注意：

需要了解的相关知识：

1. http协议（可以通过查询rfc协议文档了解更多）RFC文档是用来解释网络协议的一种文档集合
   1. GET指令
   2. 请求头的结构
   3. 反馈头的结构
2. http协议传输文件的模式



http协议描述

1. 页面解析器设计

Html文档，是一种标记性语言

对页面解析的目的：

得到页面中存在的下级URL

URL保存在<A>标签的href属性中

可以通过正则表达式提取页面中的URL

注意提取到的URL深度

页面解析器处理流程：

1. 得到下载的页面
2. 得到页面对应的URL结构体（用于得到当前页面的深度）（注意判断是否已经达到最大深度）
3. 使用正则表达式得到页面中所有的URL列表
4. 处理URL中的相对路径
5. 一个细节：把当前页面深度加一，生成并填充URL结构体
6. 把得到的URL列表回写到URL管理器中（生成列表数据，以返回值形式回传给上层代码）
7. 持久化器设计：
   1. 仍然以模块形式生成
   2. 分为网页保存模块和图片保存模块

注意：如果涉及到页面编码需要进行转换操作（本系统不对此进行操作）

处理流程：

1. 得到页面的数据流或内存缓冲区的数据
2. 得到当前页面的URL描述结构体
3. 生成保存目录（目录已存在和未存在的处理情况）
4. 把文件按照指定模式写入磁盘系统
5. 向主处理流程发送一个反馈，表示当前页面处理的进度

系统的核心代码：

1. 系统主处理框架
2. Epoll框架的调用
3. 插件框架的完整实现
4. Socket功能封装
5. http协议头解析
6. html解析并提取url列表
7. url管理器实现

后续工作：如何进行单元测试 cunit

把软件做成系统服务，需要shell脚本。

继承测试。