Linux实训项目网络爬虫系统概要

1. 什么是爬虫

是一个软件机器人，是可控的，可以从互联网上抓取我们所需的资源。爬虫是搜索引擎后台的第一个子系统，数据入口之一。

1. 爬虫能做什么

搜索引擎的基础应用

抓取大数据的一种手段

网页下载器

网店秒杀

1. 关于项目

时间安排：4天

关于需求、设计、编码、测试、集成

需求：

设计：分为业务设计和技术设计。

业务设计是业务层的方案。

对我们软件设计师来讲，业务就是我们的系统想要完成一个工作，要经过哪些步骤或流程。

技术设计：关于框架和处理流程，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：是对我们系统可靠性的一个监测

集成：将每个程序员开发的模块或子系统合成为一个完成的系统

思维方式：线性思维方式与发散思维方式。

软件开发要掌握的思想：

模块思维

渐进式开发

1. 需求：

自动抓取网络资源的软件。

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：对需求中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

1. 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、得到爬取种子（URL）

2、根据爬取种子下载资源（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步到第5步

系统设计：

系统结构图

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：对系统或模块设计要有输入和输出。

控制器模块



控制器由三个模块组成：

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

1. 控制器设计

**配置文件解析模块**

配置文件是以文件形式保存程序运行时必要的参数，减少输入时的繁琐过程。

文件类型是文本文件，内容一般以键值对形式出现。

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：

操作：

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）

ConfigParser();

load();

对应key的操作();

操作1()

操作2()

….

ptivate: //私有成员变量

job\_num

seed

deeps

log\_level

Module\_path

Module\_name

file\_type

//单体：

};

技术点：

1. 按行读取fgets
2. 分割字符串
3. 消除注释
4. 消除空格

实现：（略）

**URL维护模块**

分析：

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

[www.baidu.com](http://www.baidu.com) => ip

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； -1 – 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

struct stru\_url

{

url

protocal

Sitename

Path

Filename

state

Deep

Filetype

}

class Url

{

public:

//初始化

//set…

//get…

getUrlData();

private:

//

struct stru\_url m\_url;

}

class UrlManager

{

public: //外部接口

addUrl();

addUrlList();

getUrlForQuque();

removeUrlForQuque();

//setUrlValue();

private: //内部接口

findUrl();

private:

list< Url> Urls;

map<string, Url\*> UrlMap;

quque<Url\*> Urlquque;

}

**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检测是否按照守护进程模式运行（控制选项从命令行参数中得到）

3初始化环境

a读取配置文件，提取配置文件中的参数

b根据守护进程模式的标记将当前进程转变为守护进程

c载入程序模块的动态库

4开始程序的主处理流程

a检测种子是否存在，把种子交给URL管理器

b分析种子，得到种子url的IP地址（DNS解析）

c根据种子URL得到第一个页面

d对页面进行处理（复杂流程，由其他模块实现细节）

e从URL管理器中取出一个URL

f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）

g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

h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：

a对页面进行解析，提取内部的下级url

b生成url列表，把url列表传给URL管理器

c对页面进行持久化操作

2处理任务

a从url管理器得到一个未处理的URL

b通过调用epoll框架产生一个新任务

c调用页面处理过程（复杂步骤1）

d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：

1方便调试

2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，但是并不严重

4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

5 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登等级段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

日志调用结构设计成宏：

1) 如果日志调用接口使用较多时，如果设计成函数，当频繁调用时，会增加CPU运行时间。

**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(\*init)(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

Int Load (char\* path, char\*name);

Module\* getModule(char\* name);

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打开动态库（.so）

3使用动态库

4关闭动态库文件



1. 下载器设计

下载器模块分为socket功能封装与Http功能模块

http协议对资源请求的操作get指令

HTTP头描述：

http头以文本形式体现在http协议包中

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处理事件中。（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等各式），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

* GET指令
* 请求头的结构
* 反馈头的结构

2http协议传输文件的模式



http协议描述

8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。

页面解析的处理流程：

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）

9持久化器设计

仍然以模块形式生成

分为网页保存模块和图片保存模块

注意，如果涉及到网页编码需要进行转码操作（本系统不对此进行操作）

处理流程

1得到页面的数据流或在内存缓冲区的数据

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. Soket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。

集成测试。

同学提问：

1. 关于项目的实施，项目有多少人来做

根据项目的大小决定

1. 进入项目编码阶段：

仍然是先进性更小的设计，把当前模块的处理流程想清楚

先完成程序框架

1. 如果在编码中遇到了从来没有用到过的技术该怎么处理？

应该先写一个独立的验证demo，否则一旦发现bug，对bug的定位和解决非常困难。

1. 关于工作的分配是项目管理的流程问题，作为程序员只需要保证自己负责的模块按时提交即可
2. 遇到自己难以解决的问题一定要向项目负责人寻求帮助，并说明原因。

编写源代码时需要遵循的原则：

1. 代码风格
2. 注释
3. 源码文件的组织

如果软件比较复杂，那么源码文件按照模块存放入下级目录。

================================================================

项目中用到的一个开源库

Libevent

为什么要采用开源库适当的采用一些开源库，对开发效率非常有帮助。

怎样采用开源库

1. 开源库一定要成熟。
   1. 怎样判断开源库的成熟度？

口碑，被广泛使用的一般是成熟的

Release版成熟的，稳定的，一般要采用

Beta版为测试版，一般不采纳

Rc版稳定性比beta版高，一般也不建议采纳

最后一个提交版本，一般来说最不稳定

1. 我们使用Libevent的DNS解析功能

系统中可以被开源库替换的部分

下载器：

Wget命令，最简单的linux下的命令行客户端，使用方法：先fork一个子进程，然后用exec系列系统同调用吧wget拉起

Curl命令，使用方法和wget相同

Curl开发库 提供了头文件和静态库，实现多任务可以使用epoll模型

对代码进行单元测试：

1. 代码产生bug的原因
   1. 低级错误，编译器可以帮助我们检查大部分
   2. 一些编译器检测不到的低级错误，例如

if（var == 1）

{

….

}

可能会写成这样：

If( var = 1 )

{

…

}

对已这一类错误要求我们对语法熟悉，养成遇到类似情况提起重视的习惯

* 1. 逻辑上的错误（很难完全消除，但是可以尽可能的减少。编码之前先进行设计，对设计进行评估）
  2. 某些该进行容错的地方没有容错

例如打开一个文件的流程

FILE \*pf = fopne(…);

If(pf==NULL) //检查打开文件是否失败

{

//容错逻辑

}

///////////////////////////////////////

int fd = open(…)

if(fd < 0 )

{

//容错逻辑

}

Ret = Recv(sockhandle,。。。)

If(…)

{

Errno();

…..

}

不做容错代码的后果：

一旦前面步骤出错，那么后续步骤必定受到影响，从而导致结果是错误的。

1. 进行测试

进行单元测试是程序员开发最后一个步骤。

进行测试前先要有测试用例。

测试用例是我们做好的一个测试方案，包括对模块的输入和模块运行之后预期的输出。

输入的设计要有合乎规则的数据和非法数据

有了测试用例需要针对用例编写测试程序。

1. 手段
   1. 使用断言（assert）

含义是在使用断言的位置，条件必定成立，如果不成立，那么程序终止运行，同时assert会给出断言失败的代码文件名和行号

**断言只是一个调试工具。意味着断言条件表达式不能包含程序处理流程。**

* 1. 使用单元测试工具例如CUNIT

使用CUNIT之前要先有测试用例

根据测试用例编写测试代码

执行测试用例代码

单元测试之后是整体测试

==================================================

如何保护我们完成的代码：

保护代码的安全性。

1. 备份代码
   1. 把源代码备份到其他存储设备上保留
   2. 使用源代码管理工具（代码配置管理工具）
   3. 常用的配置管理工具： CVS（开源）、SVN（开源）、VSS（微软）、GIT（开源）

源码分析（C风格代码）：

分析代码的方法：

1. 先找到程序入口

一般和程序名称相同名字的源码文件为入口，或main.c，或通过查阅说明文件找到入口。

1. 分析代码先找到主函数，搞清楚主处理流程。注意在这个过程中不要过早的陷入细节。

代码说明：

Spider.cpp 程序入口

qstring.cpp 字符串处理工具函数（仔细研究，可能面试题中会出现）

url.cpp 与URL处理相关的函数，包括URL字符串解析，DNS解析，对URL队列的维护

socket.cpp 与网络通信相关的函数。对socket的初始化，对资源的请求，对socket模式的设定