# 功能概述

1. 使用多进程对上交所技术官网所有第一层超链接资源实现无重复并行下载；
2. 若第一层超链接资源为html页面，则该html页面所包含的超链接资源也需下载；
3. 支持在远程机器部署、执行。

# 主要进程

远程爬虫功能主要由remote、local和crawler进程完成。remote进程负责爬虫在远程机器上的部署，local进程负责远程机器上多进程爬虫的执行，crawler进程负责爬取上交所技术官网所有第一层超链接资源。

## remote进程

remote进程首先将项目代码发送至远程机器，启动远程机器上的local进程，然后开始监听本地机器的系统中断信号。若收到中断信号，则通过tcp发送中断消息给远程机器，否则等待local进程执行完成，输出local进程返回的执行结果后结束。

### 函数定义

* uploadSrc

|  |  |
| --- | --- |
| 描述 | 上传项目文件至远程机器 |
| 输入 | ip string 远程机器ip  port string 远程机器ssh连接端口号  user string 远程机器登录用户  pwd string 远程机器登录密码  uploadPath string 远程机器上传路径 |
| 输出 | error |

* runSrc

|  |  |
| --- | --- |
| 描述 | 启动远程服务器上的local进程 |
| 输入 | ip string远程机器ip  port string远程机器ssh连接端口号  user string远程机器登录用户  pwd string 远程机器登录密码  procNum int 远程机器上启动crawler进程的数量  url string 远程机器上crawler需要爬取的url |
| 输出 | error |

* processInterrupt

|  |  |
| --- | --- |
| 描述 | 处理本地中断信号，并通知远程机器 |
| 输入 | ip string 远程机器ip  port string 远程机器监听远程中断消息的端口号 |
| 输出 | error |

### 流程图



图 1 remote流程图

函数接口

## local进程

首先在所在机器上启动etcd进程，然后启动n个crawler进程，并开始监听tcp中断消息和系统中断信号。若收到中断消息或中断信号，结束已经启动的n个crawler进程，输出“interrupt“后结束。否则在等待所有crawler进程结束后，对crawler进程的输出日志进行分析，检测是否存在多个进程处理了相同的html节点和下载了相同的url资源，输出检测结果后结束。

### 函数定义

* loadLogs(filePaths []string) (fileContents [][]string)

|  |  |
| --- | --- |
| 描述 | 加载日志文件 |
| 输入 | filePaths []string 日志文件路径 |
| 输出 | fileContents [][]string 加载后的日志文件内容 |

* checkLogs(fileContents [][]string) bool

|  |  |
| --- | --- |
| 描述 | 检测日志文件， 是否存在重复下载的资源 |
| 输入 | fileContents [][]string 日志文件内容 |
| 输出 | bool |

* processLocalInterrupt

|  |  |
| --- | --- |
| 描述 | 处理本地中断信号 |
| 输入 | interruptChan chan int 用于监听中断信号的channel |
| 输出 | 无 |

* processRemoteInterrupt

|  |  |
| --- | --- |
| 描述 | 处理远程中断消息 |
| 输入 | addr string 监听远程中断消息的地址  interruptChan chan int用于监听中断信号的channel |
| 输出 | 无 |

* listenRemoteInterrupt

|  |  |
| --- | --- |
| 描述 | 监听远程中断消息 |
| 输入 | addr string监听远程中断消息的地址  interruptChan chan int 用于监听中断信号的channel |
| 输出 | error |

### 流程图



图 2 local流程图

## crawler进程

crawler进程通过使用xcrawler包，在http请求的response阶段检测访问的url是否需要下载以及是否为html页面，如果为html页面则对其页面中的元素进行遍历，否则抛弃该response。同时在遍历html页面中的元素时，对元素属性中包含的url进行访问，直到达到访问的最大深度。

### 流程图



图 3 crawler流程图

## 进程交互时序图

时序图描述了本地机器上的remote进程与远程机器上的local进程、crawler进程、etcd进程间的交互过程。



图 4 进程交互时序图

# 包设计

## remotessh

remotessh封装了基于ssh提供与远程机器进行命令行交互的功能。

### 类设计

* SSHConfig

ssh连接的配置信息

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| Network:string | 通讯方式，tcp或udp |
| IP:string | 远程机器ip |
| Port:string | 远程机器端口 |
| User:string | 远程机器登录用户 |
| Password:string | 登陆密码 |
| Timeout:time.Duration | 超市时间 |

* Interactor

与ssh服务器进行交互的接口

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| Close() | 关闭SSH连接 |
| Run(cmds []string) error | 执行命令行 |
| GetStdout() string | 获取SSH的stderr |
| GetStderr() string | 获取SSH的stdout |

* interactor

实现了Interactor接口的结构体

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| client:\*ssh.Client | 封装的ssh.Client对象 |
| stdoutBuf:bytes.Buffer | 用于存储stdout |
| stderrBuf:bytes.Buffer | 用于存储stderr |
| Close() | 关闭SSH连接 |
| Run(cmds []string) error | 执行命令行 |
| GetStdout() string | 获取SSH的stderr |
| GetStderr() string | 获取SSH的stdout |



图 5 remotessh结构图

### 函数定义

* NewInteractor

|  |  |
| --- | --- |
| 描述 | 获取与ssh服务器交互的Interator |
| 输入 | config SSHConfig ssh连接的配置信息 |
| 输出 | Interactor  error |

* NewClientConfig

|  |  |
| --- | --- |
| 描述 | 获取与ssh服务器交互的Interator |
| 输入 | user string ssh连接的登录用户  password string 登录密码  timeout time.Duration 超时时间  publicKey ssh.PublicKey |
| 输出 | Interactor  error |

## remotesftp

remotesftp封装了基于sftp提供与远程机器进行文件传输的功能。

### 类设计

1. SFTPConfig

sftp连接的配置信息

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| Network:string | 通讯方式，tcp或udp |
| IP:string | 远程机器ip |
| Port:string | 远程机器端口 |
| User:string | 远程机器登录用户 |
| Password:string | 登陆密码 |
| Timeout:time.Duration | 超市时间 |

1. Interactor

与sftp服务器进行交互的接口

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| Close() | 关闭SSH连接 |
| Upload(src string, des string) error | 上传文件至sftp服务器 |
| Download(src string, des string) error | 从sftp服务器下载文件 |

1. interactor

实现了Interactor接口的结构体

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| sftpClient:\*sftp.Client | 封装的sftp.Client对象 |
| sshClient:\*ssh.Client | 封装的ssh.Client对象 |
| Close() | 关闭SSH连接 |
| Upload(src string, des string) error | 上传文件至sftp服务器 |
| Download(src string, des string) error | 从sftp服务器下载文件 |
| uploadFile(src string, des string) error | 上传文件 |
| uploadDirectory(src string, des string) error | 上传目录 |



图 6 remotesftp结构图

### 函数定义

* NewInteractor

|  |  |
| --- | --- |
| 描述 | 获取与服务器进行sftp交互的Interator |
| 输入 | config SFTPConfig sftp连接的配置信息 |
| 输出 | Interactor  error |

## osutil

osutil封装与os交互的功能，启动进程、监听系统信号、文件创建与写入。

### 类设计

* ECmdState

command对象状态

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| ECmdStateError | Error状态 |
| ECmdStateReady | Ready状态 |
| ECmdStateRunning | Running状态 |
| ECmdStateFinished | Finished状态 |

* Command

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| GetCmdState() ECmdState | 获得command状态 |
| GetCmdError() error | 获取错误信息 |
| Run() | 运行command |
| RunAsyn() | 异步运行command |
| Kill() error | 结束command进程 |
| GetStdout() string | 获取command的stdout |
| GetStderr() string | 获取command的stderr |
| SetStdin(in string) error | 设置command的stdin |

* command

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| cmd:\*exec.Cmd | 封装的exec.Cmd对象 |
| state:ECmdState | command的状态 |
| err:error | 错误信息 |
| stdin:bytes.Buffer | 存储stdin |
| stdout:bytes.Buffer | 存储stdout |
| stderr:bytes.Buffer | 存储stderr |
| GetCmdState() ECmdState | 获得command状态 |
| GetCmdError() error | 获取错误信息 |
| Run() | 运行command |
| RunAsyn() | 异步运行command |
| Kill() error | 结束command进程 |
| GetStdout() string | 获取command的stdout |
| GetStderr() string | 获取command的stderr |
| SetStdin(in string) error | 设置command的stdin |



图 7 osutil结构图

### 函数定义

* NewCommand

|  |  |
| --- | --- |
| 描述 | 创建Command对象 |
| 输入 | name string  args ...string |
| 输出 | Command |

* ListenSystemSignalsWithCtx

|  |  |
| --- | --- |
| 描述 | 利用上下文监听系统信号 |
| 输入 | ctx context.Context  cancel context.CancelFunc  signalChan chan os.Signal |
| 输出 | 无 |

* WriteToFile

|  |  |
| --- | --- |
| 描述 | 将二进制内容写入指定文件 |
| 输入 | filePath string  content []byte |
| 输出 | error |

* CreateFile

|  |  |
| --- | --- |
| 描述 | 创建指定文件 |
| 输入 | filePath string |
| 输出 | \*os.File  error |

### 状态图

command状态图描述了command对象存在的几种状态和转移的条件。



图 8 command状态图

## taskutil

taskutil是任务调度执行工具。每个任务会有前置任务，所有前置任务均已完成，当前任务才能被执行。任务池根据任务状态进行任务的执行。

### 类设计

* ETaskState

tast对象状态

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| ETaskStateError | Error状态 |
| ETaskStateWaiting | Waiting状态 |
| ETaskStateReady | Ready状态 |
| ETaskStateRunning | Running状态 |
| ETaskStateFinished | Finished状态 |

* Task

任务接口

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| Equal(other container.Element) bool | 判断两个任务是否相等 |
| GetState() ETaskState | 获取任务当前状态 |
| Run() | 执行任务 |
| GetError() error | 获取任务错误信息 |

* task

实现Task接口的结构体

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| name:string | 任务名称 |
| do:func() error | 任务内容 |
| err:error | 错误信息 |
| state:ETaskState | 任务状态 |
| preTaskList:[]Task | 前置任务 |
| subTaskList:[]Task | 后置任务 |
| Equal(other container.Element) bool | 判断两个任务是否相等 |
| GetState() ETaskState | 获取任务当前状态 |
| Run() | 执行任务 |
| GetError() error | 获取任务错误信息 |
| checkIsReady() | 检测任务是否Ready |

* EtaskPoolState

taskPool对象状态

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| ETaskPoolStateError | Error状态 |
| ETaskPoolStateStop | Stop状态 |
| ETaskPoolStateRunning | Running状态 |
| ETaskPoolStateClosing | Closing状态 |
| ETaskPoolStateClosed | Closed状态 |

* TaskPool

任务池接口

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| GetTaskPoolState() ETaskPoolState | 获取任务池状态 |
| GetAllTaskNum() int | 获取任务总数 |
| GetFinishedTaskNum() int | 获取Finished的任务数 |
| GetErrorTaskNum() int | 获取Error的任务数 |
| GetWaitingTaskNum() int | 获取Waiting的任务数 |
| GetReadyTaskNum() int | 获取Ready的任务数 |
| GetRunningTaskNum() int | 获取Running的任务数 |
| Run() | 运行任务池 |
| Close() | 关闭任务池 |
| AddTask(task Task) | 添加任务 |

* taskPool

实现TaskPool的结构体

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| allTaskList:container.Vector | 所有任务列表 |
| waitingTaskList:container.Vector | Waiting任务列表 |
| errorTaskList:container.Vector | Error任务列表 |
| readyTaskQueue:container.Queue | Ready任务列表 |
| runningTaskList:container.Vector | Running任务列表 |
| finishedTaskList:container.Vector | Finished任务列表 |
| state:ETaskPoolState | 任务池状态 |
| GetTaskPoolState() ETaskPoolState | 获取任务池状态 |
| GetAllTaskNum() int | 获取任务总数 |
| GetFinishedTaskNum() int | 获取Finished的任务数 |
| GetErrorTaskNum() int | 获取Error的任务数 |
| GetWaitingTaskNum() int | 获取Waiting的任务数 |
| GetReadyTaskNum() int | 获取Ready的任务数 |
| GetRunningTaskNum() int | 获取Running的任务数 |
| Run() | 运行任务池 |
| Close() | 关闭任务池 |
| AddTask(task Task) | 添加任务 |
| checkWaitingTask() | 更新Waiting状态的任务状态 |



图 9 taskutil结构图

### 函数定义

* NewTask

|  |  |
| --- | --- |
| 描述 | 创建新的Task |
| 输入 | name string  do func() error  preTasks ...Task |
| 输出 | Task |

* NewTaskPool

|  |  |
| --- | --- |
| 描述 | 创建新的TaskPool |
| 输入 | 无 |
| 输出 | TaskPool |

### 状态图

Task状态图与TaskPool状态图描述了Task对象与TaskPool对象存在的几种状态和转移的条件。

1. Task状态



图 10 task状态图

1. TaskPool状态



图 11 taskpool状态图

### 流程图

TaskPool在运行时，不断地将WaitingList中Ready的Task调入ReadyQueue，同时不断地执行ReadyQueue中的Task，并根据Task的运行结果，将Task放入ErrorList或FinishedList。



图 12 taskpool流程图

## etcd

etcd封装了与嵌入式etcd和etcd进程的交互功能。

### 类设计

* embedetcd

封装嵌入式etcd

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| tempdir:string | Etcd临时文件夹目录 |
| etcd:\*embed.Etcd | 封装的embed.Etcd对象 |
| close() | 关闭etcd |

* Interactor

与etcd进行交互的接口

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| Get(key string) (string, error) | Get操作 |
| Put(key string, value string) error | Put操作 |
| Del(key string) error | Del操作 |
| Lock() (context.CancelFunc, error) | Lock操作 |
| Unlock() (context.CancelFunc, error) | Unlock操作 |
| TxnSync(key string) bool | 利用txn与etcd交互，对key进行同步 |
| Close() | 关闭etcd |

* interactor

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| e:\*embedetcd | 封装的embedetcd对象 |
| c:\*clientv3.Client | 封装的clientv3.Client对象 |
| s:\*concurrency.Session | 封装的concurrency.Session对象 |
| m:\*concurrency.Mutex | 封装的concurrency.Mutex对象 |
| Get(key string) (string, error) | Get操作 |
| Put(key string, value string) error | Put操作 |
| Del(key string) error | Del操作 |
| Lock() (context.CancelFunc, error) | Lock操作 |
| Unlock() (context.CancelFunc, error) | Unlock操作 |
| TxnSync(key string) bool | 利用txn与etcd交互，对key进行同步 |
| Close() | 关闭etcd |



图 13 etcd结构图

### 函数设计

* GetClientConfig

|  |  |
| --- | --- |
| 描述 | 读取json格式的etcd配置文件 |
| 输入 | filePath string |
| 输出 | clientv3.Config |

* NewInteractorWithEmbed

|  |  |
| --- | --- |
| 描述 | 获取与嵌入etcd交互的Interactor |
| 输入 | 无 |
| 输出 | Interactor  error |

* NewInteractor

|  |  |
| --- | --- |
| 描述 | 获取与etcd交互的Interactor |
| 输入 | 无 |
| 输出 | Interactor  error |

## xcrawler

xcrawler封装了爬虫的入口，通过自定义的handler在http请求的各个阶段进行分析处理。

### 类设计

* Request

封装http.Request的接口

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| IsValid() bool | 检测Request是否有效 |
| GetDepth() int | 获取当前Request深度 |
| GetRawReq() \*http.Request | 获取封装的http.Request对象 |
| Visit(URL string) | 访问url |

* request

实现Request接口的结构体，对http.Request进行封装

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| rawReq:\*http.Request | 封装的http.Request对象 |
| depth:int | Request的深度 |
| c:Crawler | 封装的Crawler对象 |
| IsValid() bool | 检测Request是否有效 |
| GetDepth() int | 获取当前Request深度 |
| GetRawReq() \*http.Request | 获取封装的原始http.Request对象 |
| Visit(URL string) | 访问url |

* Response

封装http.Response的接口

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| GetRawResponse() \*http.Response | 获取封装的\*http.Response |
| GetStatusCode() int | 获取Response状态码 |
| GetStatus() string | 获取Response状态 |
| GetRequest() Request | 获取Response对应的Request |
| GetBody() []byte | 检测Response的body内容 |
| Abandon() | 遗弃Response |

* response

实现Response接口的结构体，对http.Respons进行封装

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| rawResp:\*http.Response | 封装的http.Response对象 |
| request:Request | 该Response对应的Request |
| body:\*[]byte | Response的body |
| abandoned:bool | Response是否被遗弃 |
| GetRawResponse() \*http.Response | 获取封装的\*http.Response |
| GetStatusCode() int | 获取Response状态码 |
| GetStatus() string | 获取Response状态 |
| GetRequest() Request | 获取Response对应的Request |
| GetBody() []byte | 检测Response的body内容 |
| Abandon() | 遗弃Response |

* HTMLElement

封装html.Node的接口

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| Equal(HTMLElement) bool | 判断两个HTMLElement是否相等 |
| GetRequest() Request | 获取所属的Request |
| GetParent() HTMLElement | 获取父节点 |
| GetFirstChild() HTMLElement | 获取第一个孩子节点 |
| GetLastChild() HTMLElement | 获取最后一个孩子节点 |
| GetPrevSibling() HTMLElement | 获取同级的上一个节点 |
| GetNextSibling() HTMLElement | 获取同级的下一个节点 |
| GetType() html.NodeType | 获取节点类型 |
| GetDataAtom() atom.Atom | 获取节点的DataAtom |
| GetData() string | 获取节点的Data |
| GetNamespace() string | 获取节点的namespace |
| GetAttr(string) string | 获取节点的某个属性 |

* htmlElement

实现HTMLElement接口的结构体，对html.Node进行封装

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| node:\*html.Node | 封装的html.Node对象 |
| req:Request | 所属的Request |
| Equal(HTMLElement) bool | 判断两个HTMLElement是否相等 |
| GetRequest() Request | 获取该element所属的Request |
| GetParent() HTMLElement | 获取父节点 |
| GetFirstChild() HTMLElement | 获取第一个孩子节点 |
| GetLastChild() HTMLElement | 获取最后一个孩子节点 |
| GetPrevSibling() HTMLElement | 获取同级的上一个节点 |
| GetNextSibling() HTMLElement | 获取同级的下一个节点 |
| GetType() html.NodeType | 获取节点类型 |
| GetDataAtom() atom.Atom | 获取节点的DataAtom |
| GetData() string | 获取节点的Data |
| GetNamespace() string | 获取节点的namespace |
| GetAttr(string) string | 获取节点的某个属性 |

* Crawler

爬虫接口

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| Visit(URL string) | 访问url |
| AddHTMLHandler(handler HTMLHandler, filters ...HTMLFilter) | 添加HTMLHandler |
| AddRequestHandler(handler RequestHandler, filters ...RequestFilter) | 添加RequestHandler |
| AddResponseHandler(handler ResponseHandler, filters ...ResponseFilter) | 添加ResponseHandler |
| visit(u \*url.URL, depth int) | 访问url |

* crawler

实现Crawler接口的结构体

|  |  |
| --- | --- |
| 方法/参数 | 说明 |
| maxDepth:int | 爬虫的最大深度 |
| htmlHandlers:[]HTMLHandler | HTMLHandler |
| requestHandlers:[]RequestHandler | RequestHandler |
| responseHandlers:[]ResponseHandler | ResponseHandler |
| Visit(URL string) | 访问url |
| AddHTMLHandler(handler HTMLHandler, filters ...HTMLFilter) | 添加HTMLHandler |
| AddRequestHandler(handler RequestHandler, filters ...RequestFilter) | 添加RequestHandler |
| AddResponseHandler(handler ResponseHandler, filters ...ResponseFilter) | 添加ResponseHandler |
| visit(u \*url.URL, depth int) | 访问url |
| traversingAllElement(rootElement HTMLElement) | 遍历html节点 |



图 14 xcrawler结构图

### 类型定义

|  |  |
| --- | --- |
| 类型名 | 类型定义 |
| HTMLHandler | func(HTMLElement) |
| HTMLFilter | func(HTMLElement) bool |
| RequestHandler | func(Request) |
| RequestFilter | func(Request) bool |
| ResponseHandler | func(Response) |
| ResponseFilter | func(Response) bool |

### 函数定义

* NewCrawler

|  |  |
| --- | --- |
| 描述 | 创建新的Crawler |
| 输入 | maxDepth int |
| 输出 | Crawler |

* NewHTMLElement

|  |  |
| --- | --- |
| 描述 | 创建新的HTMLElement |
| 输入 | node \*html.Node  req Request |
| 输出 | HTMLElement |

* NewRequest

|  |  |
| --- | --- |
| 描述 | 创建新的Request |
| 输入 | rawReq \*http.Request  depth int  c Crawler |
| 输出 | Request |

* NewResponse

|  |  |
| --- | --- |
| 描述 | 创建新的Response |
| 输入 | rawResp \*http.Response  req Request  body \*[]byte |
| 输出 | Response |

* FilterHTML

|  |  |
| --- | --- |
| 描述 | 根据HTMLFilter对HTMLElement进行过滤 |
| 输入 | element HTMLElement  filters ...HTMLFilter |
| 输出 | bool |

* FilterRequest

|  |  |
| --- | --- |
| 描述 | 根据RequestFilter对Request进行过滤 |
| 输入 | req Request  filters ...RequestFilter |
| 输出 | bool |

* FilterResponse

|  |  |
| --- | --- |
| 描述 | 根据ResponseFilter对Response进行过滤 |
| 输入 | resp Response  filters ...ResponseFilter |
| 输出 | bool |

### 流程图

xcrawler对网页visit过程中，广度遍历html节点时，通过自定义的handler与filter对节点进行处理。



图 15 xcrawler visit流程图

## pathlib

pathlib包含了路径或url的处理方法。

### 函数定义

* GetModulePath

|  |  |
| --- | --- |
| 描述 | 获取当前模块的绝对路径，工作路径需要在该模块中 |
| 输入 | moduleName string |
| 输出 | string |

* GetParentPath

|  |  |
| --- | --- |
| 描述 | 获取路径的上级路径 |
| 输入 | p string |
| 输出 | string |

## crawlerutil

crawlerutil包含了对href进行过滤和处理的结构和方法。

### 类型定义

|  |  |
| --- | --- |
| 类型名 | 类型定义 |
| URLFilter | func(u \*url.URL) bool |
| URLHandler | func(u \*url.URL) bool |

### 函数定义

FilterElementWithURL

|  |  |
| --- | --- |
| 描述 | 过滤html的元素， href与src属性不为空的html元素返回true |
| 输入 | element xcrawler.HTMLElement |
| 输出 | bool |

HandleElementWithURL()

|  |  |
| --- | --- |
| 描述 | 处理具有url的html元素 |
| 输入 | element xcrawler.HTMLElement  etcdInteractor etcd.Interactor |
| 输出 | 无 |

* FilterURL(u \*url.URL, filters ...URLFilter) bool

|  |  |
| --- | --- |
| 描述 | 使用filters对url进行过滤，符合所有filer则返回true，否则返回false |
| 输入 | u \*url.URL  filters ...URLFilter |
| 输出 | bool |

* filterURLWithHTTP

|  |  |
| --- | --- |
| 描述 | 过滤url，schema为http或https的url返回true |
| 输入 | u \*url.URL |
| 输出 | bool |

* filterURLWithJS

|  |  |
| --- | --- |
| 描述 | 过滤url， schema为javascript的url返回true |
| 输入 | u \*url.URL |
| 输出 | bool |

* NewURLHandlerWithFilters

|  |  |
| --- | --- |
| 描述 | 根据handler和filter生成新的handler |
| 输入 | handler URLHandler  filters ...URLFilter |
| 输出 | URLHandler |

* handleURLWithHTTP

|  |  |
| --- | --- |
| 描述 | 处理schema为http或https的url |
| 输入 | u \*url.URL |
| 输出 | bool |

* generateKeyForCrawler

|  |  |
| --- | --- |
| 描述 | 根据输入的keys，生成用于etcd同步的key |
| 输入 | keys ...string |
| 输出 | string |

* getURLString

|  |  |
| --- | --- |
| 描述 | 从html元素的属性中获取url，若无url则返回“” |
| 输入 | element xcrawler.HTMLElement |
| 输出 | string |