法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

- 课程详情请咨询
  - 微信公众号：小象
  - 新浪微博：ChinaHadoop

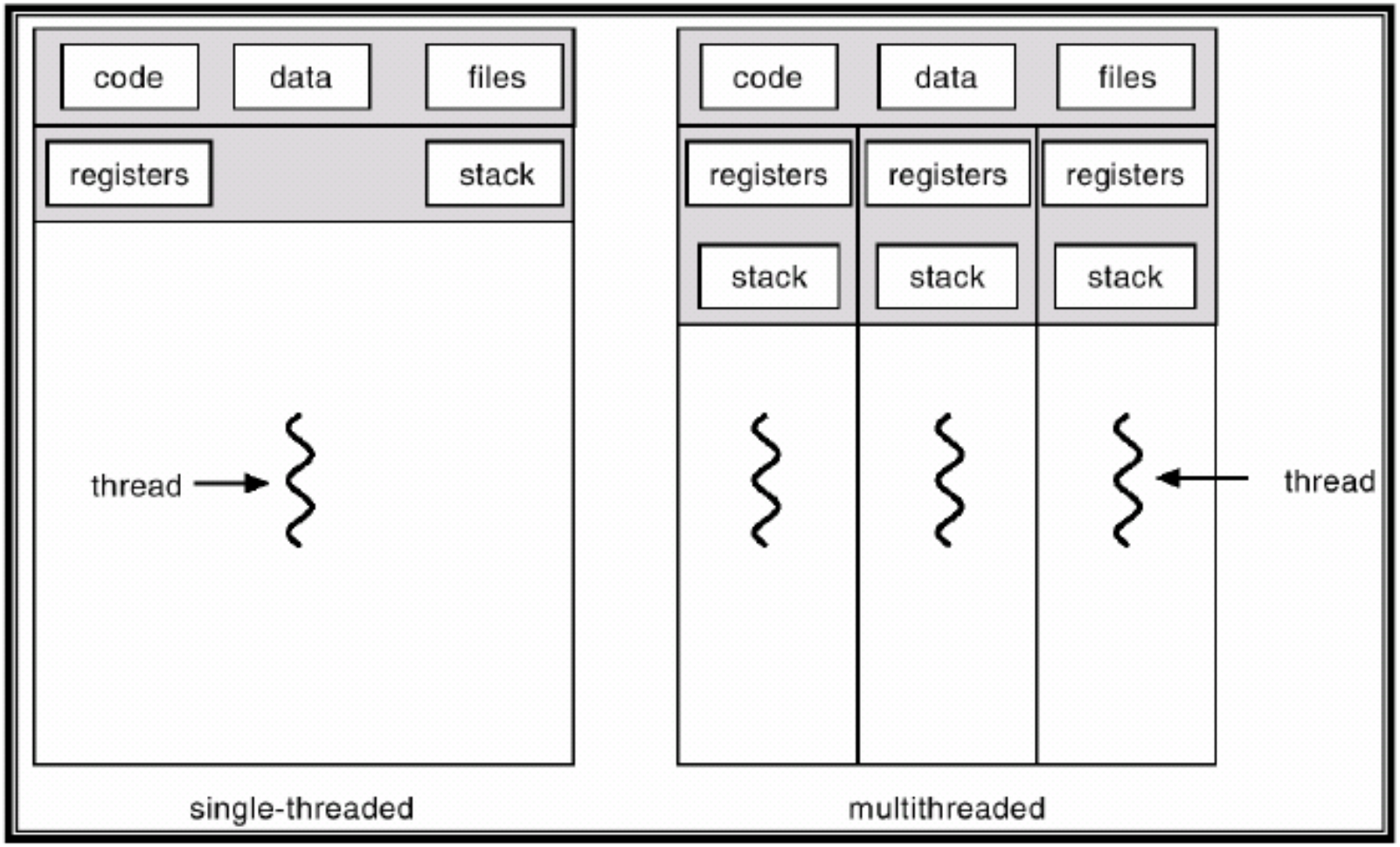
分布式爬虫

大纲

- 表单及登录
- 多线程及语言分析对比
- 多进程爬虫

多线程爬虫

多线程

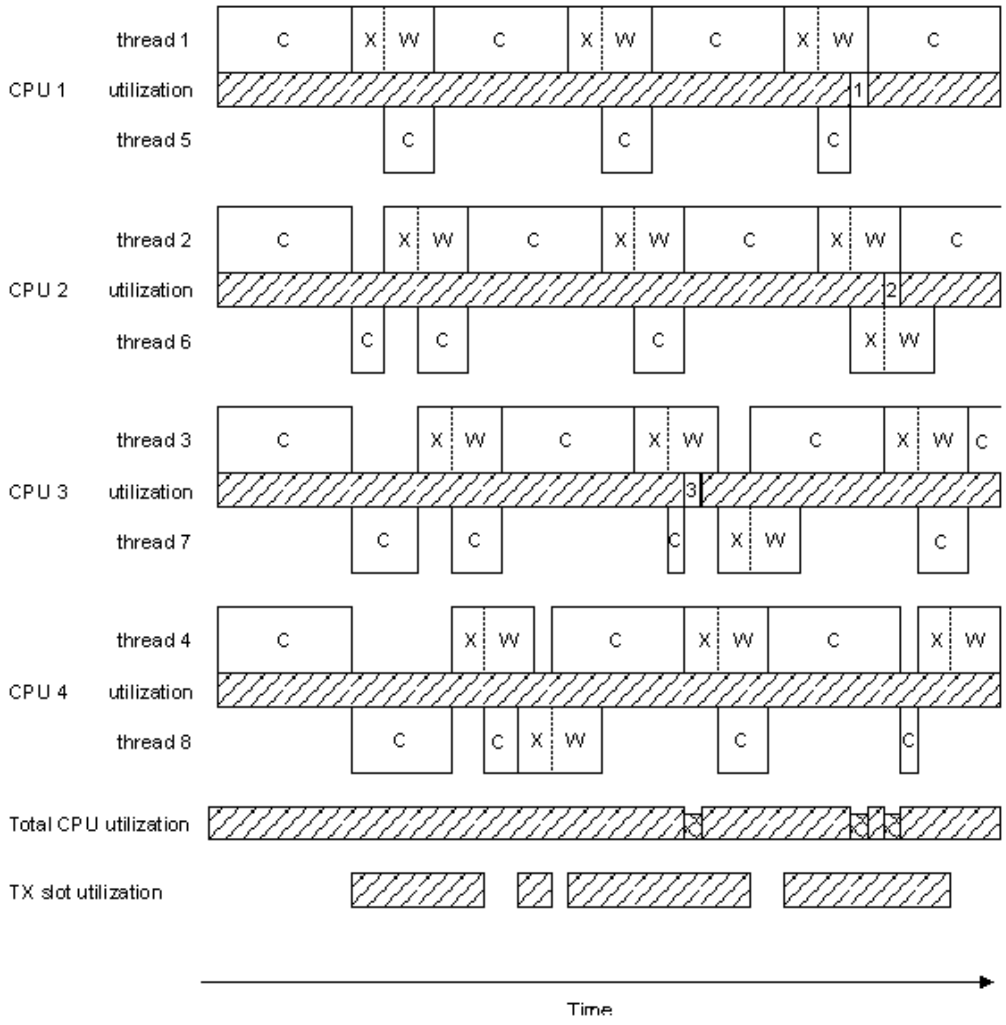

code
data
files
registers
stack
thread
single-threaded
code
data
files
registers
registers
registers
stack
stack
stack
thread
multithreaded

多线程的复杂性

- 资源、数据的安全性：锁保护
- 原子性：数据操作是天然互斥的
- 同步等待：wait() notify() notifyAll()
- 死锁：多个线程对资源互锁，造成死锁
- 容灾：任何线程出现错误，整个进程都会停止

多线程的优势

- 内存空间共享，信息数据交换效率高
- 提高CPU的使用效率
- 开发便捷
- 轻，创建、销毁的开销小



Python 线程

- 支持多线程 (Javascript PHP 不支持多线程)
- python 线程直接映射到 native 线程 (Java 1.4 的 Java 线程是JVM实现的，共同运行在一个 native thread)
- GIL (Global Interpretor Lock)：对于多核的利用能力有限

实现一个多线程爬虫

1. 创建一个线程池 threads = []
2. 确认 url 队列线程安全 Queue Deque
3. 从队列取出 url，分配一个线程开始爬取 pop()/get() threading.Thread
4. 如果线程池满了，循环等待，直到有线程结束 t.is_alive()
5. 从线程池移除已经完成下载的线程 threads.remove(t)
6. 如果当前级别的url已经遍历完成，t.join() 函数等待所有现场结束，然后开始下一级别的爬取

多线程爬虫评价

优势：

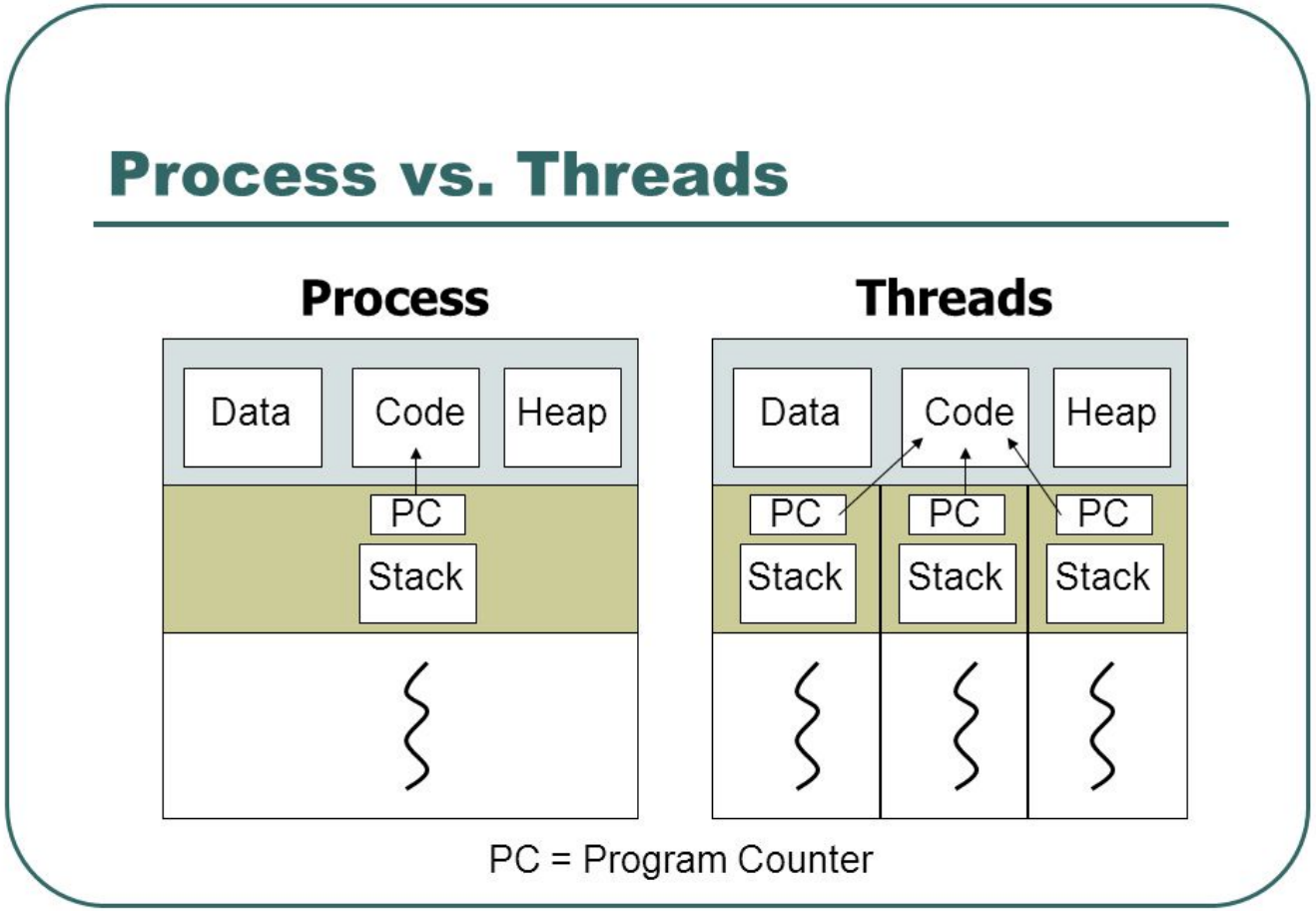
- 有效利用CPU时间
- 极大减小下载出错、阻塞对抓取速度的影响，整体上提高下载的速
- 对于没有反爬虫限制的网站，下载速度可以多倍增加

局限性：

- 对于有反爬的网站，速度提升有限
- 提高了复杂度，对编码要求更高
- 线程越多，每个线程获得的时间就越少，同时线程切换更频繁也带来额外开销
- 线程之间资源竞争更激烈

多进程爬虫

线程与进程


Process vs. Threads
Process
Threads
Data
Code
Heap
PC
Stack
Data
Code
Heap
PC
PC
PC
Stack
Stack
Stack
PC = Program Counter

线程与进程

Process	Thread
Process is considered heavy weight	Thread is considered light weight
Unit of Resource Allocation and of protection	Unit of CPU utilization
Process creation is very costly in terms of resources	Thread creation is very economical
Program executing as process are relatively slow	Programs executing using thread are comparatively faster
Process cannot access the memory area belonging to another process	Thread can access the memory area belonging to another thread within the same process
Process switching is time consuming	Thread switching is faster
One Process can contain several threads	One thread can belong to exactly one process

多进程爬虫评估

目的：

- 控制线程数量
- 对线程进行隔离，减少资源竞争
- 某些环境下，在单机上利用多个IP来伪装

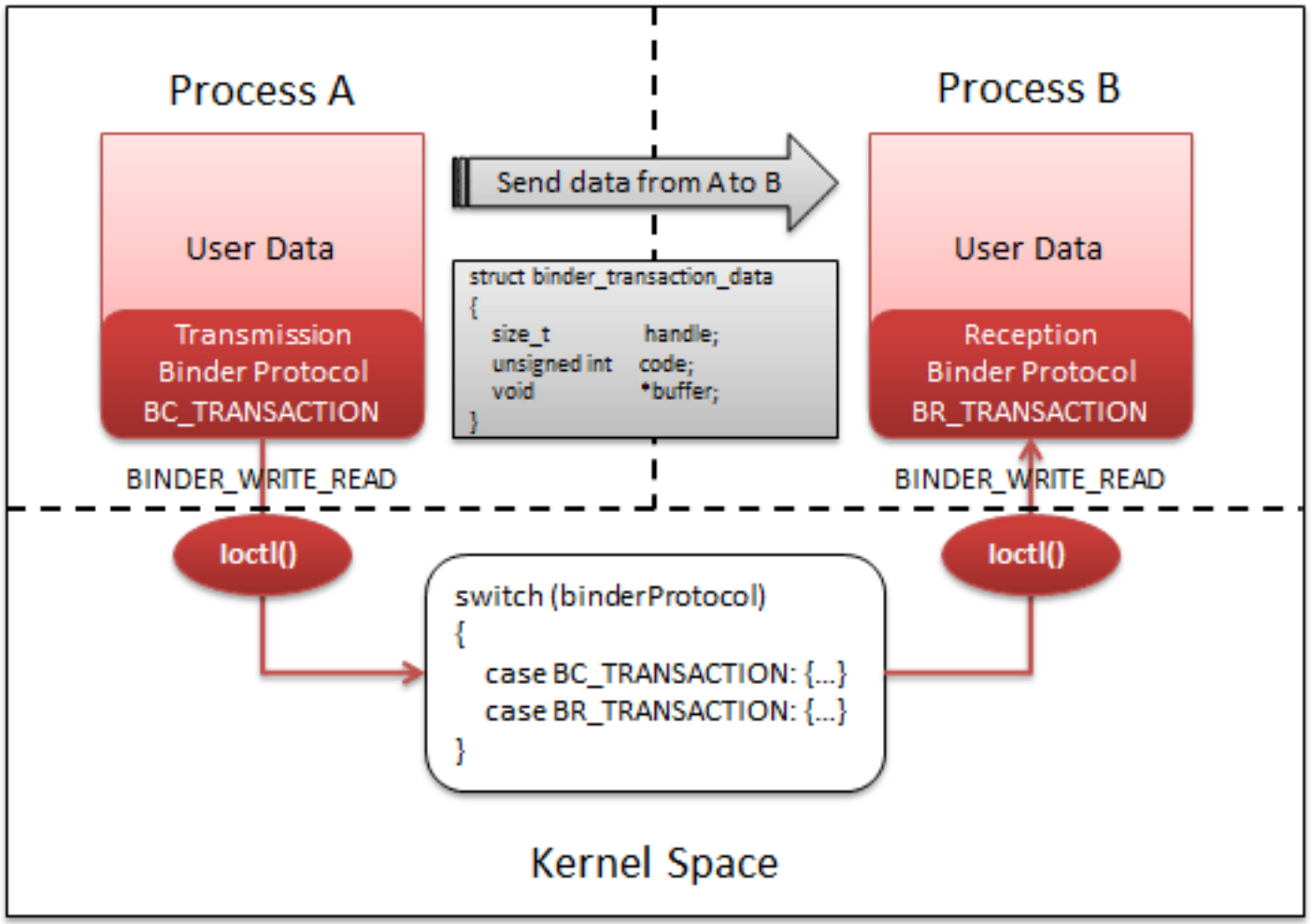
局限性：

- 不能突破网络瓶颈
- 单机单IP的情况下，变得没有意义
- 数据交换的代价更大

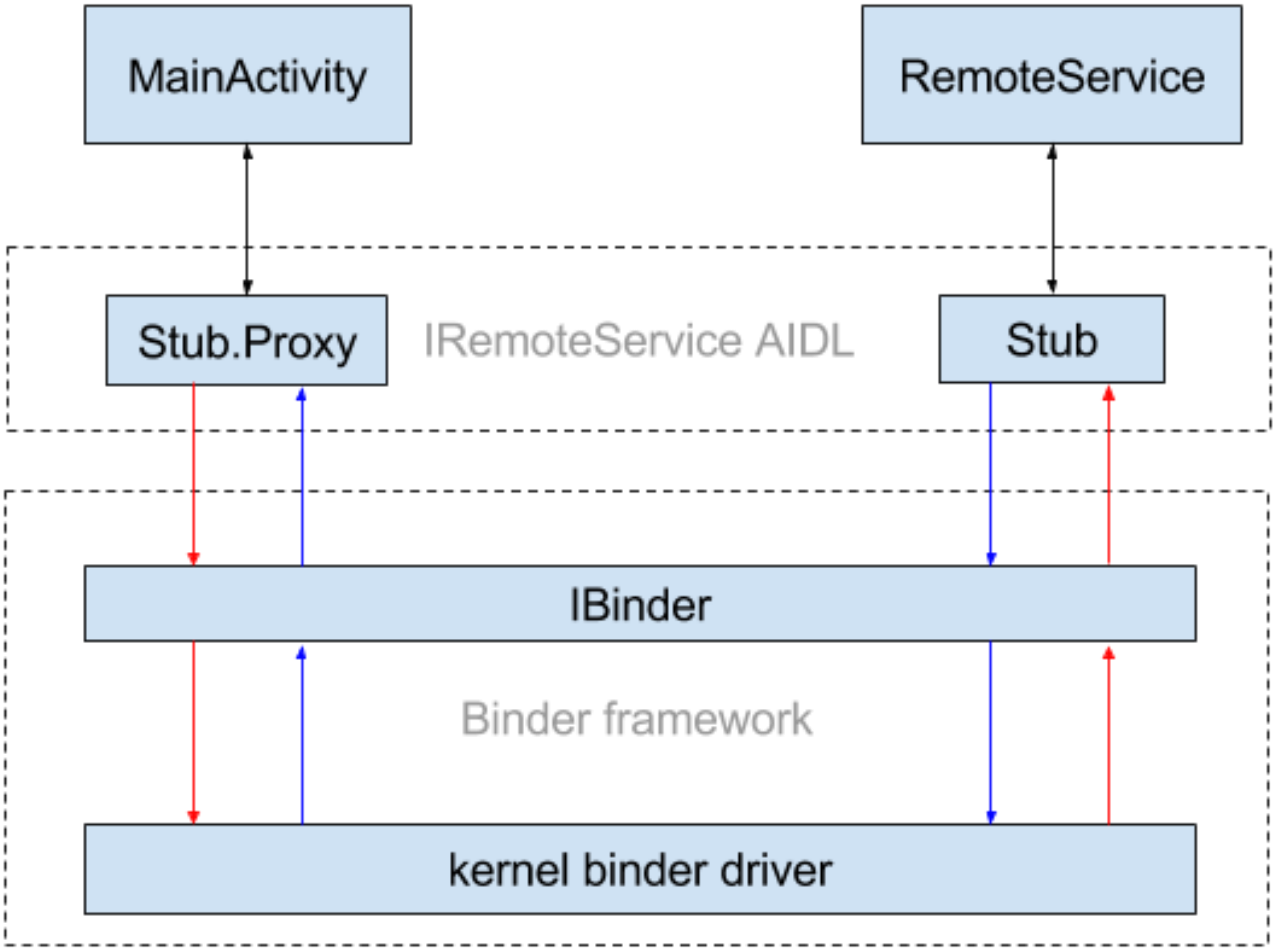
进程间通信

- 管道（PIPE）
- 信号（Signal）：复杂
- 消息队列：Posix 及 system V
- 共享内存：速度最快，需要结合信号量达到进程间同步及互斥
- 信号量：用于数据同步
- Socket：可以标准化，可以用于多机

Android 进程间通信 Binder



Android 进程间通信 AIDL



创建多进程爬虫

Solution A – C/S模式

1. 一个服务进程，入队及出队 URL，入队需检查是否已经下载
2. 监控目前的爬取状态、进度
3. 多个爬取进程，从服务进程获取URL，并将新的URL返回给服务进程
4. 使用Socket来做 IPC

Solution B – 数据库模式

1. 使用数据库来读写爬取列表
2. 多个爬取进程，URL 的获取与增加都通过数据库操

C/S v.s. 数据库

CS 优势：

- 运行速度快，添加、修改、查询都是内存的BIT位操作
- 扩展方便，例如动态URL队列重拍

数据库：

- 开发便捷，数据库天生具备读写保护及支持IPC
- 只需要写一个爬虫程序

创建 MySQL 数据库表

Key	Type	Description
index	int(11)	PRIMARY KEY AUTOINCREMENT
url	varchar(512)	UNIQUE
status	varchar(11)	download status: new, downloading, done
md5	varchar(16)	UNIQUE md5 value of url
depth	int(11)	page depth
queue_time	timestamp	CURRENT_TIMESTAMP time url enqueue
done_time	timestamp	time page downloaded

数据库创建

```
"CREATE TABLE `urls` ("
"  `index` int(11) NOT NULL AUTO_INCREMENT,"
"  `url` varchar(512) NOT NULL,"
"  `md5` varchar(16) NOT NULL,"
"  `status` varchar(11) NOT NULL DEFAULT 'new',"
"  `depth` int(11) NOT NULL,"
"  `queue_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,"
"  `done_time` timestamp NOT NULL DEFAULT 0 ON UPDATE CURRENT_TIMESTAMP,"
"  PRIMARY KEY (`index`),"
"  UNIQUE KEY `md5` (`md5`)"
") ENGINE=InnoDB"
```

Python Mysql Connector

- 使用MySQLConnectionPool 来管理多线程下的mysql数据库连接

  mysql.connector.pooling.MySQLConnectionPool

  self.cnxpool.get_connection()
- __init__ 类实例的时候自动检查和创建数据库及表
- Cursor 类：cursor = con.cursor(dictionary=True)
- SELECT ... FOR UPDATE 加读锁，避免多个进程取出同一个 url
- cursor.commit() 支持事物，默认关闭了 autocommit 因此需要提交

Official Docs: https://dev.mysql.com/doc/connector-python/en/

疑问

- 问题答疑： http://www.xxwenda.com/
  - 可邀请老师或者其他人回答问题

联系我们

小象学院：互联网新技术在线教育领航者

- 微信公众号：小象
- 新浪微博：ChinaHadoop