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

表单及登录

HTML 提交数据

- form 表单

```
<form>
        <input type="text" name=”username">
</form>
```

HTML 的标签，由浏览器实现POST方法

- ajax 请求

```
$.(ajax){
}
```

基于ajax技术，异步发送http请求并获得返回数据，然后利用JavaScript对网页进行处理

HTML 表单

<form></form> tag 包围

里面包含了表单的元素：

<select> <textarea> <input> 等

<input> type 包含了多种输入类型，例如：

- <input type="text" name="name"> 文本输入框
- <input type="password" name="password"> 密码输入框
- <input type="submit" value="Submit"> 提交按钮

表单类型: form-data

POST /start.htm HTTP/1.1

Host: 192.168.1.1

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

Accept-Language: zh-CN,en-US;q=0.8,en;q=0.6

Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

Cache-Control: no-cache

------WebKitFormBoundary7MA4YWxkTrZu0gW

Content-Disposition: form-data; name="name”

chacha

------WebKitFormBoundary7MA4YWxkTrZu0gW

Content-Disposition: form-data; name="password”

chinahadoop

------WebKitFormBoundary7MA4YWxkTrZu0gW--

表单类型

- **form-data**

  http请求中的**multipart/form-data**，它会将表单的数据处理为一条消息，以标签为单元，用分隔符分开。既可以上传键值对，也可以上传文件。当上传的字段是文件时，会有Content-Type来表名文件类型。 由于有boundary隔离，所以multipart/form-data既可以上传文件，也可以上传键值对，它采用了键值对的方式，所以可以上传多个文件。

- **x-www-form-urlencoded**

  application/x-www-from-urlencoded，会将表单内的数据转换为键值对

表单类型: form-data

POST /start.htm HTTP/1.1

Host: 192.168.1.1

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

Accept-Language: zh-CN,en-US;q=0.8,en;q=0.6

Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

Cache-Control: no-cache

------WebKitFormBoundary7MA4YWxkTrZu0gW

Content-Disposition: form-data; name="name”

chacha

------WebKitFormBoundary7MA4YWxkTrZu0gW

Content-Disposition: form-data; name="password”

chinahadoop

------WebKitFormBoundary7MA4YWxkTrZu0gW--

表单类型: x-www-form-urlencoded

POST /start.htm HTTP/1.1

Host: 192.168.1.1

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

'content-type': "application/x-www-form-urlencoded",

Cache-Control: no-cache

name=chacha&password=chinahadoop

ajax 登录

一般以 json 的方式提交数据

POST /start.htm HTTP/1.1

Host: 192.168.1.1

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

Cache-Control: no-cache

{"name":"chacha", "password":"chinahadoop"}

登录

1. 根据 chrome inspector 里检查到的参数，来设置登录方式
   - 最常用的是 x-www-form-urlencoded 和 json 方式，两种方式只是body编码不同
   - 如果是 form-data，按照form-data的方式组合body
   - body 部分经常需要按照特定的方式编码，比如 base64，或者 md5
   - 对于非常复杂的登录协议，利用第5章的动态网页方式登录
2. request = urllib2.Request(url, data, headers = loginheaders)
   - url 是访问的地址
   - data 是 body 部分
   - headers = loginheaders 是HTTP请求的 HEADER 数据

表单登录

```
import urllib2

url = http://jc.lo/dev/login/login.php
headers = {
    'host': "jc.lo",
    'connection': "keep-alive",
    'cache-control': "no-cache",
    'content-type': "application/x-www-form-urlencoded",
    'upgrade-insecure-requests': "1",
    'accept-language': "zh-CN,en-US;q=0.8,en;q=0.6",
}
data = {'name':'caca', "password":'c'}
payload = urllib.urlencode(data)
request = urllib2.Request(url, payload , headers=headers)
response = urllib2.urlopen(request)
```

获取并设置Cookie

登录的核心是为了获得 Cookie

登录成功后，HEADER 会有设置cookie的相关信息

```
Set-Cookie:main[UTMPKEY]=1381959; path=/; domain=.newsmth.net
Set-Cookie:main[UTMPNUM]=14820; path=/; domain=.newsmth.net
Set-Cookie:main[UTMPUSERID]=abc; path=/; domain=.newsmth.net
```

此时我们需要把服务器返回的 Cookie 信息，写入到我们后续请求的HEADER 的 Cookie 里

意外？？

当遇到网页登录后，返回302跳转的情况下，urllib2 的 Response 会丢失 Set-Cookie 的信息，导致登录不成功

更通常的情况下，我们需要一个通用的能处理 Cookie 的工具

- 自动处理 Set-Cookie 请求
- 自动处理管理过期 Cookie
- 自动在对应的域下发送特殊 Cookie

使用 urllib2 的插件功能

urllib2 可以绑定一系列的处理对象 handler，hanlder 可以对urllib2 的http 请求提供额外的功能支持。handler 是以绑定的顺序依次调用的。例如 HTTPRedirectHandler 用来处理跳转的情况，ProxyHandler 用于处理代理 。可以开发自定义的 handler （从 BaseHandler继承）

build_opener([*handler*, ...]) # 注册一系列的 handler

```
proxy_handler = urllib2.ProxyHandler({'http': 'http://www.example.com:3128/'})
proxy_auth_handler = urllib2.ProxyBasicAuthHandler()
proxy_auth_handler.add_password('realm', 'host', 'username', 'password')
opener = urllib2.build_opener(proxy_handler, proxy_auth_handler)
# This time, rather than install the OpenerDirector, we use it directly:
opener.open('http://www.example.com/login.html')
```

CookieJar

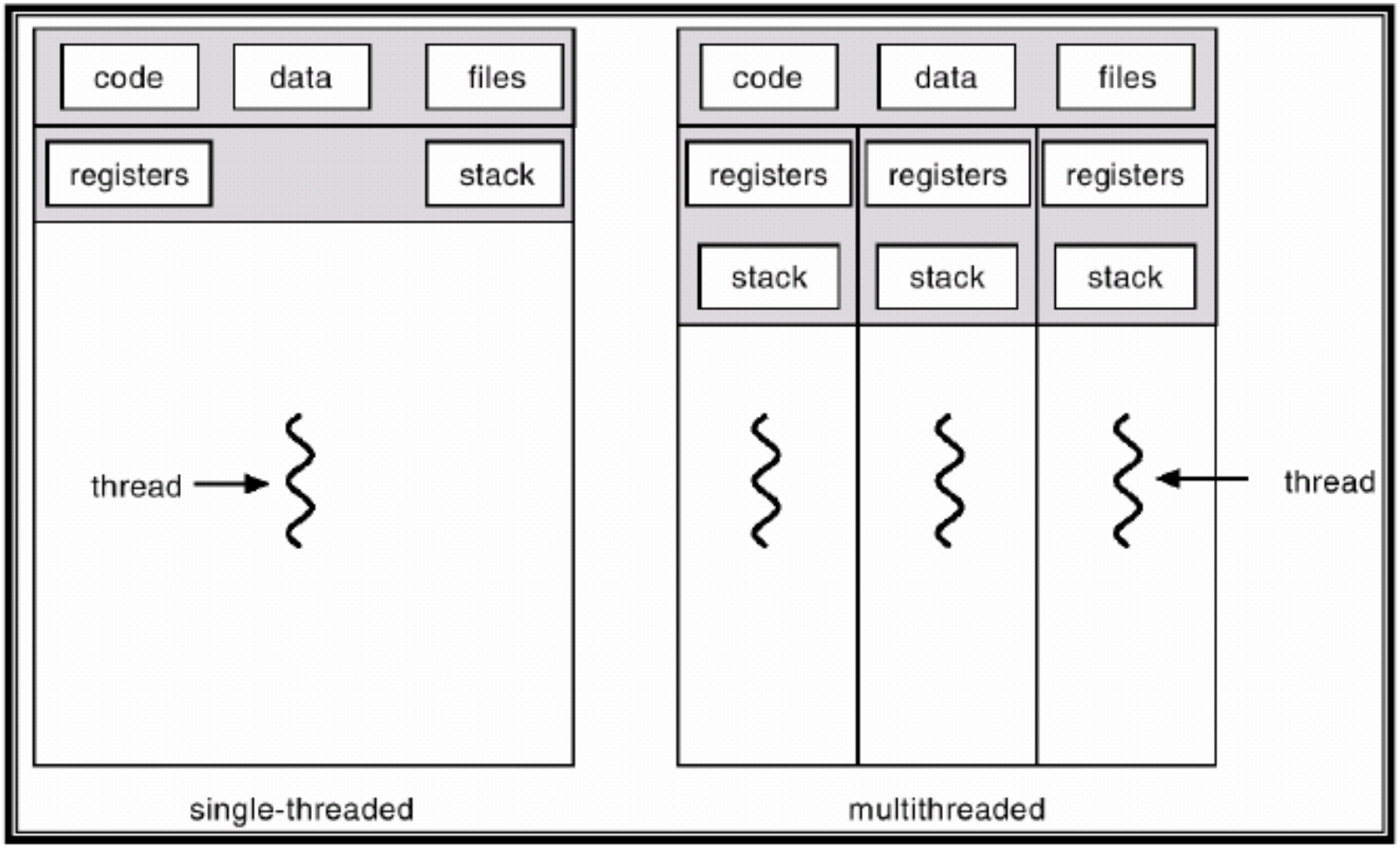
```
import cookielib

cj = cookielib.CookieJar() #创建 CookieJar 对象
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj)) #注册插件
request = urllib2.Request(url, payload, headers=headers)
response = opener.open(request)

print response.items() #打印headers信息
print response.read() #打印返回的网页
#打印 cookie 内容
for cookie in cj:
    print cookie.name, cookie.value, cookie.domain
```

多线程爬虫

多线程

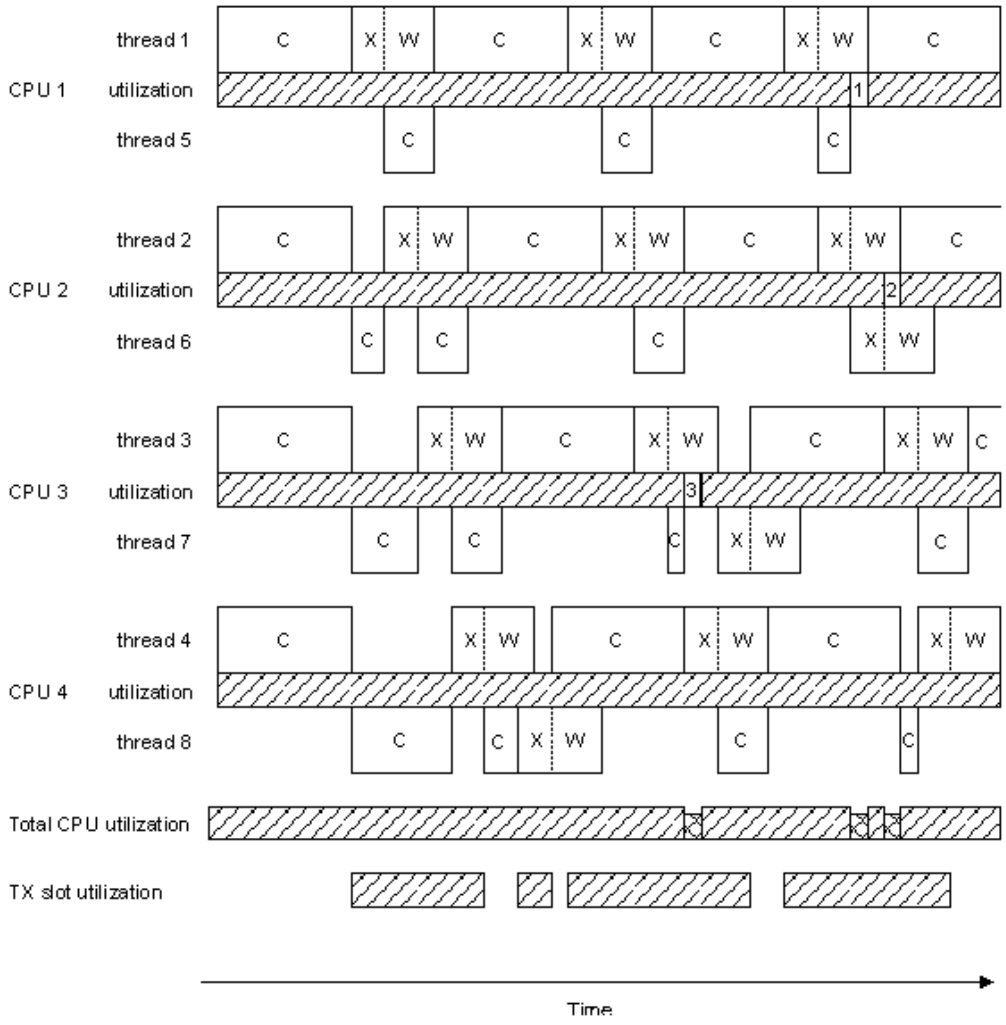

code
data
files
registers
stack
thread
single-threaded
code
data
files
registers
registers
registers
stack
stack
stack
thread
multithreaded

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：

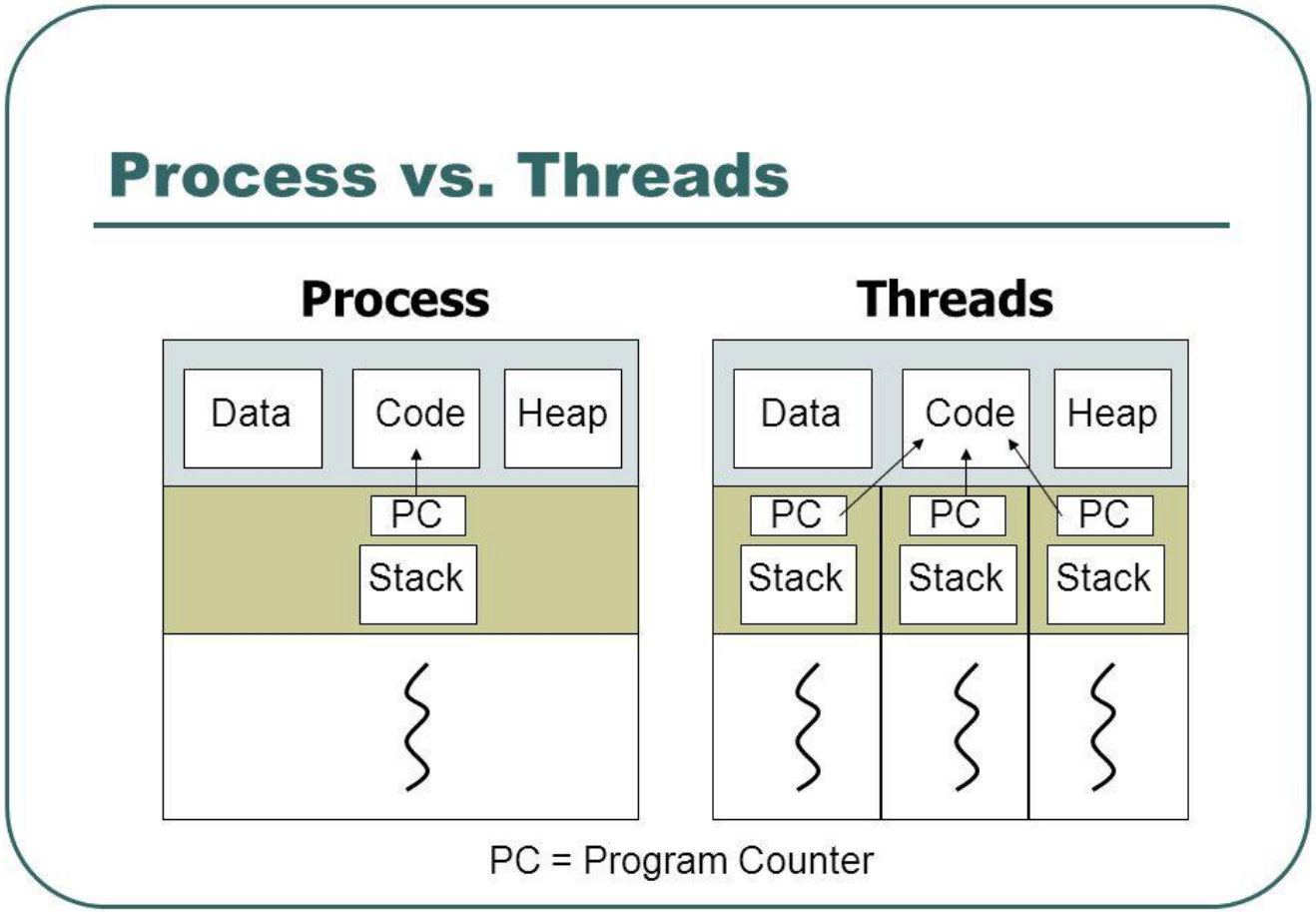
- 有效利用CPU时间
- 极大减小下载出错、阻塞对抓取速度的影响，整体上提高下载的速
- 对于没有反爬虫限制的网站，下载速度可以多倍增加

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
Data
Code
Heap
PC
Stack
Threads
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

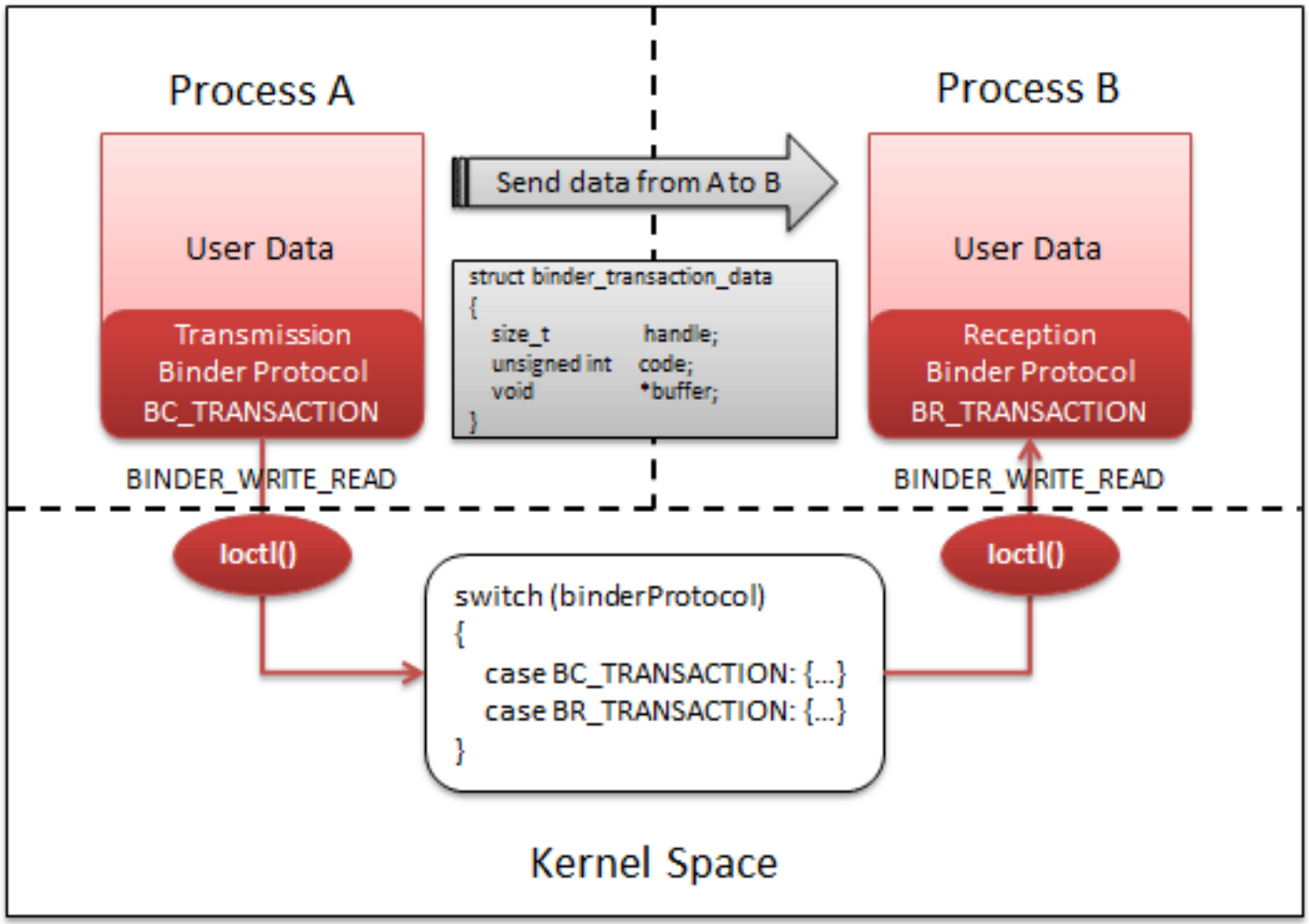
局限性：

- 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

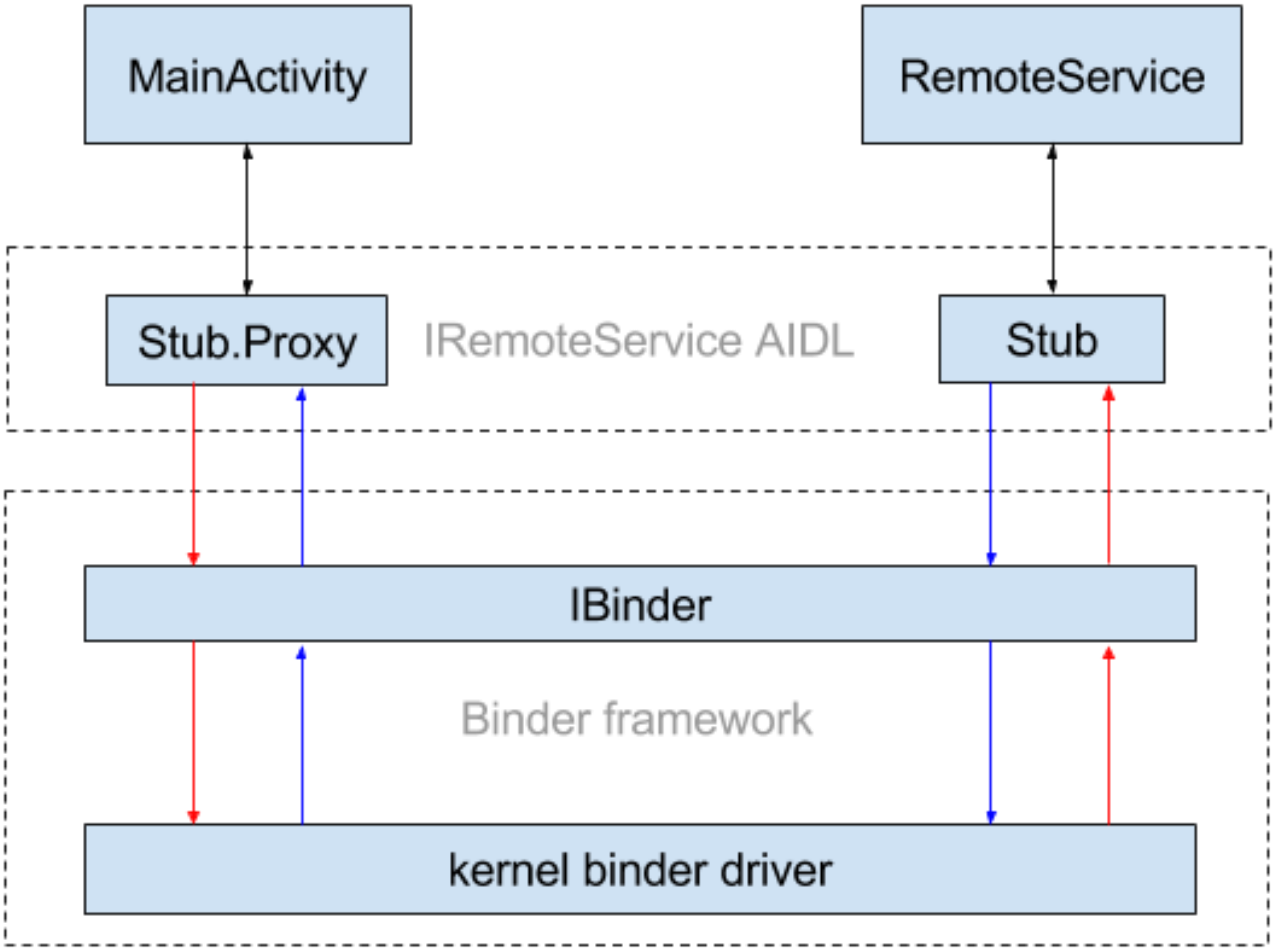
进程间通信

- 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