

Screen Scraping discussions

(Interrupt me!)

September 24, 2011

Screen scraping, you say?

- Getting some data, doing something useful with it.
 - Data not meant to be read by a computer
 - Data not really meant to be used for this purpose
 - Normally done on the web

Screen scraping, you say?

- Getting some data, doing something useful with it.
 - Data not meant to be read by a computer
 - Data not really meant to be used for this purpose
 - Normally done on the web

Screen scraping, you say?

- Getting some data, doing something useful with it.
 - Data not meant to be read by a computer
 - Data not really meant to be used for this purpose
 - Normally done on the web

Uses of screen scraping

- Web indexing
- Price comparison
- Mash-ups
- Visualization
- (Ab)use of APIs

Scraping Ethics

Being sued is bad:

- Terms of use
- Copyright
- Databases are protected by law for 15 years
- Ryanair thinks arbitrage is illegal
- Some German courts agree

Frustrations

- Impossible to parse / ambiguous data
- Crazy broken web servers
- Non-repeatability of bugs
- Things breaking at random
- Change! (Houses on quicksand...)

Yet screen scrapers can be written!

Guess the prices of something using the power of the Internet.

Getting some data

```
search, = ' '.join(sys.argv[1:])
with contextlib.closing(urlopen(
    'http://www.argos.co.uk/webapp/'
    'wcs/stores/servlet/Search?%s' %
    urllib.urlencode(dict(
        storeId=10001,
        catalogId=1500002901,
        langId=-1,
        searchTerms=search)))) as stream:
    d = stream.read()
```

Use some logging to debug the inevitable mistakes

```
urlopen = (urllib2.build_opener(  
    urllib2.HTTPHandler(  
        debuglevel=logging.DEBUG))  
    .open)  
logging.basicConfig(  
    stream=sys.stderr,  
    level=logging.DEBUG)
```

Parse some html

```
tree = lxml.etree.HTML(d)
import pdb; pdb.set_trace()
```

Extract the relevant parts of the page

(Works well to do this interactively at a shell, with the help of firebug)

```
price_strings = tree.xpath(
    '//li[contains(@class,"price_")]/text()')
print numpy.mean([float(x.strip())[1:])
    for x in price_strings)
```

Lots of choices about how to do things...

Fetching data

- Raw sockets! (bad idea)
- httplib
 - Low level, parses http
 - Does not handle cookies, gzip, or deflate
 - Handle keep alive
 - Connect to one server
 - Most of this can be rectified by writing more code
- urllib2
 - Builds on httplib
 - Does not support http keep-alives (boo!)
 - Can handle lots of features, proxies, basic HTTP auth,, cookies

Fetching data...

- PycURL
 - Thin wrapper around libcurl
 - Slightly crazy callback-based interface
 - Supports keep-alives, cookies, pipelining
 - I've never used it...

Parsing html

- HTMLParser - bad idea.
- BeautifulSoup
 - Nice pythonic interface
 - Recent regressions in ability to parse broken html (FUD)
 - Pure python, hence slow
 - No xpath - less expressive power.
- lxml.etree.HTML
 - Fast (pure c)
 - Xpath support
 - Stupid namespaces when parsing XML
- html5lib
 - Normalizes DOM tree produced (adds TBODYs etc)
 - Order of magnitude slower than lxml (my timing)
 - Can produce lxml.etree tree structures (and others)

Mechanize

- Built on BeautifulSoup (thus slow)
- Adds ability to parse and interact with html:
 - Interacting with forms
 - Clicking links by name

Debugging

A lot harder than writing code!

- Firebug (with traffic dump exporter)
- Tamper Data (Buggy - but uniquely useful)
- tcpflow (http only)
- wireshark, tshark, dumpcap, tcpdump (http only)
- Magical HTTPS-intercepting proxies
 - Fiddler(windows only)
 - Burp (proprietary)
 - Charles (proprietary)
 - My half-finished tools
 - Anyone know of something open source on Linux

Debugging...

- Stealing cookies (useful for server-side state)
- I hate GUI tools!

Other problems

- Javascript (Google is fond of this...)
- Flash
- Making things fast

I'm (mostly) done talking!

Things we might like to talk about:

- Maintaining scrapers
- Interesting examples
- Dealing with Javascript sites
- Making things fast
- Scraping using headless selenium (I know a bit about this)
- Scraping using headless webkit?
- The holy grail of scraper debugging
- CAPTCHAs

Making things fast

- Network time or processing time
- Latency versus throughput
- TCP window size tuning
 - Via keep alives
 - Connection pools
 - Cheating! (Does anyone know about this)
- TCP keep-alives ineffective
- Multiple connections

Scraping using selenium :Pros

Pros:

- Selenium allows you to record things
- Selenium does precisely what a browser does

Scraping using selenium: Cons

Cons:

- Selenium is slow
- Selenium has a lot of dependencies
- Probably best for non-performance critical tasks

Scraping using selenium: Practicalities

Practicalities:

- Using Xvfb you can stop selenium from popping up windows
- I've got some code for this

Holy grail of debugging

- Http diff tools
- Capture data from browsers and scrapers in same format
- Works for HTTPS
- Generate http requests from a browser which can be tuned
- Compare with selenium test cases for regression

CAPTCHAs

The world is filled with bad CAPTCHA...

- Code in img tags... or CAPTCHA is text!
- Gocr (or tesseract)
- Gocr and imagemagick
- Gocr, imagemagick, brute-force-optimizer and 8 hours
- "Matching pursuit"