代码分析Python requests库中文编码问题
(http://xiaorui.cc/2016/02/19/%e4%bb%a3%e7%a0%81%e5%88%86%e6%9e%90python-requests%e5%ba%93%e4%b8%ad%e6%96%87%e7%bc%96%e7%a0%81%e9%97%ae%e9%a2%98/)

2-19 4,334 views

...

Python reqeusts在作为代理爬虫节点抓取不同字符集网站时遇到的一些问题总结. 简单说就是中文乱码的问题. 如果单纯的抓取微博，微信，电商，那么字符集charset很容易就确认，你甚至可以单方面把encoding给固定住。 但作为舆情数据来说，他每天要抓取几十万个不同网站的敏感数据，所以这就需要我们更好确认字符集编码,避免中文的乱码情况.

该文章写的有些乱，欢迎来喷！另外文章后续不断更新中，请到原文地址查看更新。

http://xiaorui.cc/2016/02/19/%E4%BB%A3%E7%A0%81%E5%88%86%E6%9E%90python-requests%E5%BA%93%E4%B8%AD%E6%96%87%E7%BC%96%E7%A0%81%E9%97%AE%E9%A2%98/
(http://xiaorui.cc/2016/02/19/%E4%BB%A3%E7%A0%81%E5%88%86%E6%9E%90python-requests%E5%BA%93%E4%B8%AD%E6%96%87%E7%BC%96%E7%A0%81%E9%97%AE%E9%A2%98/)

我们首先看这个例子. 你会发现一些有意思的事情.

```Python

#blog: xiaorui.cc

In [9]: r = requests.get('http://cn.python-requests.org/en/latest/')

In [10]: r.encoding
Out[10]: 'ISO-8859-1'

In [11]: type(r.text)
Out[11]: unicode

In [12]: type(r.content)
Out[12]: str

In [13]: r.apparent_encoding
Out[13]: 'utf-8'

In [14]: chardet.detect(r.content)
Out[14]: {'confidence': 0.99, 'encoding': 'utf-8'}
```

第一个问题是，为什么会有ISO-8859-1这样的字符集编码？

iso-8859是什么？ 他又被叫做Latin-1或“西欧语言” . 对于我来说，这属于requests的一个bug，在requests库的github里可以看到不只是中国人提交了这个issue. 但官方的回复说是按照http rfc设计的。

下面通过查看requests源代码，看这问题是如何造成的！

requests会从服务器返回的响应头的 Content-Type 去获取字符集编码，如果content-type有charset字段那么requests才能正确识别编码，否则就使用默认的 ISO-8859-1. 一般那些不规范的页面往往有这样的问题.

```Python

In [52]: r.headers
Out[52]: {'content-length': '16907', 'via': 'BJ-H-NX-116(EXPIRED), http/1.1 BJ-UNI-1-JCS-116 ( [cHs f ])', 'ser': '3.81', 'content-encoding': 'gz
```

文件: requests.utils.py

```Python

#blog: xiaorui.cc
def get_encoding_from_headers(headers):
    """通过headers头部的dict中获取编码格式"""

    content_type = headers.get('content-type')

    if not content_type:
        return None

    content_type, params = cgi.parse_header(content_type)

    if 'charset' in params:
        return params['charset'].strip("'\"")

    if 'text' in content_type:
        return 'ISO-8859-1'
```

第二个问题， 那么如何获取正确的编码？

requests的返回结果对象里有个apparent_encoding函数, apparent_encoding通过调用chardet.detect()来识别文本编码. 但是需要注意的是，这有些消耗计算资源.
至于为毛，可以看看chardet的源码实现.

```python

#blog: xiaorui.cc
@property
def apparent_encoding(self):
    """使用chardet来计算编码"""
    return chardet.detect(self.content)['encoding']
```

第三个问题，requests的text() 跟 content() 有什么区别？

requests在获取网络资源后，我们可以通过两种模式查看内容。 一个是r.text，另一个是r.content，那他们之间有什么区别呢？

分析requests的源代码发现，r.text返回的是处理过的Unicode型的数据，而使用r.content返回的是bytes型的原始数据。也就是说，r.content相对于r.text来说节省了计算资源，r.content是把内容bytes返回. 而r.text是decode成Unicode. 如果headers没有charset字符集的化,text()会调用chardet来计算字符集，这又是消耗cpu的事情.

通过看requests代码来分析text() content()的区别.

```python

文件: requests.models.py
@property
def apparent_encoding(self):
    """The apparent encoding, provided by the chardet library"""
    return chardet.detect(self.content)['encoding']

@property
def content(self):
    """Content of the response, in bytes."""

    if self._content is False:
        # Read the contents.
        try:
            if self._content_consumed:
                raise RuntimeError(
                    'The content for this response was already consumed')

            if self.status_code == 0:
                self._content = None
            else:
                self._content = bytes().join(self.iter_content(CONTENT_CHUNK_SIZE)) or bytes()

        except AttributeError:
            self._content = None

    self._content_consumed = True
    # don't need to release the connection; that's been handled by urllib3
    # since we exhausted the data.
    return self._content

@property
def text(self):
    """Content of the response, in unicode.
    If Response.encoding is None, encoding will be guessed using
    ``chardet``.
    The encoding of the response content is determined based solely on HTTP
    headers, following RFC 2616 to the letter. If you can take advantage of
    non-HTTP knowledge to make a better guess at the encoding, you should
    set ``r.encoding`` appropriately before accessing this property.
    """

    # Try charset from content-type
    content = None
    encoding = self.encoding

    if not self.content:
        return str('')

    # 当为空的时候会使用chardet来猜测编码.
    if self.encoding is None:
        encoding = self.apparent_encoding

    # Decode unicode from given encoding.
    try:
        content = str(self.content, encoding, errors='replace')
    except (LookupError, TypeError):
        # A LookupError is raised if the encoding was not found which could
        # indicate a misspelling or similar mistake.
        #
        # A TypeError can be raised if encoding is None
        #
        # So we try blindly encoding.
        content = str(self.content, errors='replace')
```

对于requests中文乱码解决方法有这么几种.

方法一:

由于content是HTTP相应的原始字节串，可以根据headers头部的charset把content decode为unicode，前提别是ISO-8859-1编码.

```python

```

```
In [96]: r.encoding
Out[96]: 'gbk'

In [98]: print r.content.decode(r.encoding)[200:300]
="keywords" content="Python数据分析与挖掘实战,,机械工业出版社,9787111521235,,在线购买,折扣,打折"/>
```

另外有一种特别粗暴方式，就是直接根据chardet的结果来encode成utf-8格式.

```Python
#http://xiaorui.cc

In [22]: r  = requests.get('http://item.jd.com/1012551875.html')

In [23]: print r.content
KeyboardInterrupt

In [23]: r.apparent_encoding
Out[23]: 'GB2312'

In [24]: r.encoding
Out[24]: 'gbk'

In [25]: r.content.decode(r.encoding).encode('utf-8')
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)
<ipython-input-25-918324cdc053> in <module>()
----> 1 r.content.decode(r.apparent_encoding).encode('utf-8')

UnicodeDecodeError: 'gb2312' codec can't decode bytes in position 49882-49883: illegal multibyte sequence

In [27]: r.content.decode(r.apparent_encoding,'replace').encode('utf-8')
```

如果在确定使用text，并已经得知该站的字符集编码时，可以使用 r.encoding = ‘xxx’ 模式， 当你指定编码后，requests在text时会根据你设定的字符集编码进行转换.

```Python
>>> import requests
>>> r = requests.get('https://up.xiaorui.cc')
>>> r.text
>>> r.encoding
'gbk'
>>> r.encoding = 'utf-8'
```

方法二:

根据我抓几十万的网站的经验，大多数网站还是很规范的，如果headers头部没有charset，那么就从html的meta中抽取.

```Python
In [78]: s
Out[78]: '    <meta http-equiv="Content-Type" content="text/html; charset=gbk"'

In [79]: b = re.compile("<meta.*content=.*charset=(?P<charset>[^;\s]+)", flags=re.I)

In [80]: b.search(s).group(1)
Out[80]: 'gbk"'
```

python requests的utils.py里已经有个完善的从html中获取meta charset的函数. 说白了还是一对的正则表达式.

```Python
In [32]: requests.utils.get_encodings_from_content(r.content)
Out[32]: ['gbk']
```

文件: utils.py

```Python
def get_encodings_from_content(content):
    charset_re = re.compile(r'<meta.*?charset=["\']*(.+?)["\'>]', flags=re.I)
    pragma_re = re.compile(r'<meta.*?content=["\']*;?charset=(.+?)["\'>]', flags=re.I)
    xml_re = re.compile(r'^<\?xml.*?encoding=["\']*(.+?)["\'>]')

    return (charset_re.findall(content) +
            pragma_re.findall(content) +
            xml_re.findall(content))
```

最后，针对requests中文乱码的问题总结:

统一编码，要不都成utf-8, 要不就用unicode做中间码！

国内的站点一般是utf-8、gbk、gb2312 ，当requests的encoding是这些字符集编码后，是可以直接decode成unicode.

但当你判断出encoding是 ISO-8859-1 时，可以结合re正则和chardet判断出他的真实编码. 可以把这逻辑封装补丁引入进来.

```Python
```

```
1
2  import requests
3  def monkey_patch():
4      prop = requests.models.Response.content
5      def content(self):
6          _content = prop.fget(self)
7          if self.encoding == 'ISO-8859-1':
8              encodings = requests.utils.get_encodings_from_content(_content)
9              if encodings:
10                 self.encoding = encodings[0]
11             else:
12                 self.encoding = self.apparent_encoding
13             _content = _content.decode(self.encoding, 'replace').encode('utf8', 'replace')
14             self._content = _content
15         return _content
16     requests.models.Response.content = property(content)
17 monkey_patch()
```

Python3.x解决了这编码问题，如果你还是python2.6 2.7，那么还需要用上面的方法解决中文乱码的问题.

END.

对Python及运维开发感兴趣的朋友可以加QQ群 ： 478476595 !!!

另外如果大家觉得文章对你有些作用！ 帮忙点击广告. 一来能刺激我写博客的欲望，二来好维护云主机的费用.
如果想赏钱，可以用微信扫描下面的二维码. 另外再次标注博客原地址 xiaorui.cc (http://xiaorui.cc) …… 感谢!

上一篇 (http://xiaorui.cc/2016/02/18/%e5%85%b3%e4%ba%8epython-peewee%e7%9a%84%e4%ba%8b%e5%8a%a1%e5%9b%9e%e6%bb%9a%e7%94%a8%e6%b3%95/)

下一篇 (http://xiaorui.cc/2016/03/01/%e4%ba%8c%e5%88%86%e6%9f%a5%e6%89%be%e7%ae%97%e6%b3%95%e5%ae%9e%e7%8e%b0%e7%9a%84python-bisect%e6%9c%89%e5%ba%8f%e9%98%9f%e5%88%97/)

公告

有事可以发邮件，rfyiamcool@163.com

我的另一个博客，偏向运维方面的 ~ ops.xiaorui.cc (http://ops.xiaorui.cc)

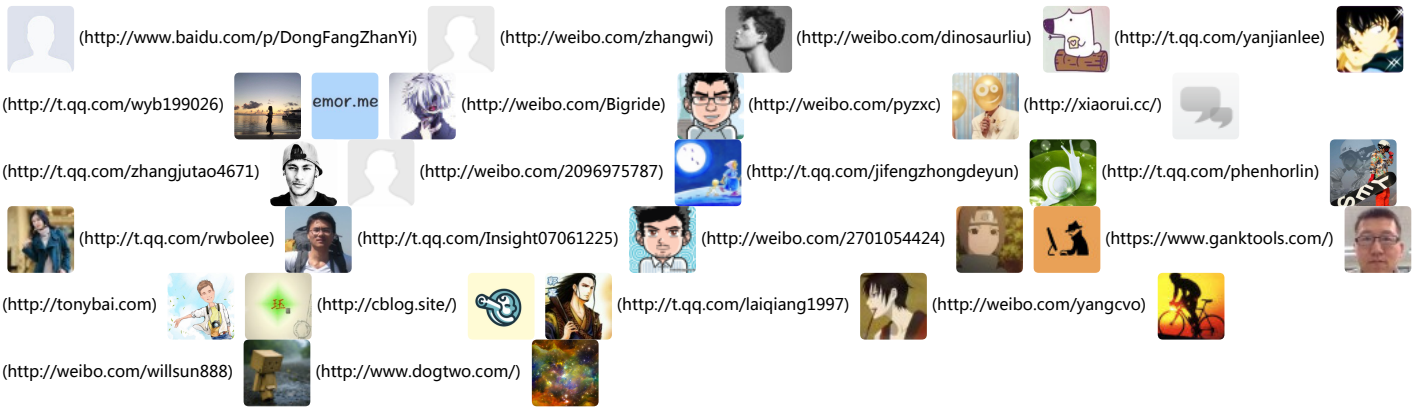
我个人的github地址是，github.com/rfyiamcool (http://github.com/rfyiamcool)

我给pypi提交过的项目，[话说曾经被删过一些] pypi地址 (https://pypi.python.org/pypi?%3Aaction=search&term=fengyun)

博客使用阿里云主机服务，大家可以使用我的优惠推荐码购买主机... 九折优惠哦 pmcp6m

对Python及运维开发感兴趣的朋友可以加QQ群 ： 478476595

最近访客



标签

ansible (http://xiaorui.cc/tag/ansible/) ansible api (http://xiaorui.cc/tag/ansible-api/) docker (http://xiaorui.cc/tag/docker/)
docker api (http://xiaorui.cc/tag/docker-api/) elasticsearch (http://xiaorui.cc/tag/elasticsearch/) elk (http://xiaorui.cc/tag/elk/)
epoll (http://xiaorui.cc/tag/epoll/) etcd (http://xiaorui.cc/tag/etcd/) gevent (http://xiaorui.cc/tag/gevent/)
golang (http://xiaorui.cc/tag/golang-2/) grafana (http://xiaorui.cc/tag/grafana/) gunicorn (http://xiaorui.cc/tag/gunicorn/)
happybase (http://xiaorui.cc/tag/happybase/) Influxdb (http://xiaorui.cc/tag/influxdb/)
InfluxDB 性能 (http://xiaorui.cc/tag/influxdb-%e6%80%a7%e8%83%bd/) kibana (http://xiaorui.cc/tag/kibana/)
kibana4 (http://xiaorui.cc/tag/kibana4/) logstash (http://xiaorui.cc/tag/logstash/) mysql (http://xiaorui.cc/tag/mysql/)
peewee (http://xiaorui.cc/tag/peewee/) python (http://xiaorui.cc/tag/python/) python elasticsearch (http://xiaorui.cc/tag/python-elasticsearch/)
python epoll (http://xiaorui.cc/tag/python-epoll/) python etcd (http://xiaorui.cc/tag/python-etcd/)
python gevent (http://xiaorui.cc/tag/python-gevent/) python happybase (http://xiaorui.cc/tag/python-happybase/)
python hbase (http://xiaorui.cc/tag/python-hbase/) python influxdb (http://xiaorui.cc/tag/python-influxdb/)
python mongodb (http://xiaorui.cc/tag/python-mongodb/) python multiprocessing (http://xiaorui.cc/tag/python-multiprocessing/)

python peewee (http://xiaorui.cc/tag/python-peewee/) python redis (http://xiaorui.cc/tag/python-redis/)

python requests (http://xiaorui.cc/tag/python-requests/) python rq (http://xiaorui.cc/tag/python-rq/)

python thrift (http://xiaorui.cc/tag/python-thrift/) python zabbix (http://xiaorui.cc/tag/python-zabbix/) redis (http://xiaorui.cc/tag/redis/)

redis cluster (http://xiaorui.cc/tag/redis-cluster/) redis 源码 (http://xiaorui.cc/tag/redis-%e6%ba%90%e7%a0%81/)

saltstack (http://xiaorui.cc/tag/saltstack/) saltstack api (http://xiaorui.cc/tag/saltstack-api/) tokudb (http://xiaorui.cc/tag/tokudb/)

zabbix (http://xiaorui.cc/tag/zabbix/) zabbix二次开发 (http://xiaorui.cc/tag/zabbix%e4%ba%8c%e6%ac%a1%e5%bc%80%e5%8f%91/)

zabbix开发 (http://xiaorui.cc/tag/zabbix%e5%bc%80%e5%8f%91/)

最热文章