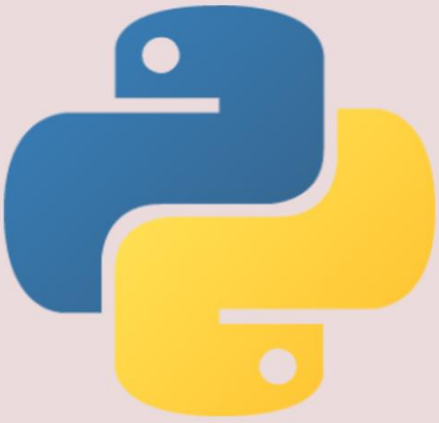
6.1.09

www.opentowork.es

Python_

Web Scrapingcon scrapy

.



#12/23 mihifidem

Python

Web Scraping con scrapy

61TE01_09_Web scraping con scrapy

¿Por qué elegir Scrapy para Web Scraping?

Hay varias bibliotecas excelentes de web scraping en Python como BeautifulSoup, Selenium, etc. Pero aquí hay algunas razones clave por las que recomiendo Scrapy como la mejor opción:

Velocidad – Scrapy es extremadamente rápido porque puede enviar solicitudes asincrónicas y extraer varias páginas al mismo tiempo. Los puntos de referencia muestran la generación de Scrapy **Más de 5000 solicitudes por minuto** en hardware modesto. Esta paralelización lo hace ideal para raspar sitios grandes.

Potentes herramientas de extracción – Los selectores Scrapy que utilizan CSS, XPath, RegEx, etc. facilitan enormemente la extracción de texto y datos. No es necesario analizar HTML desordenado, simplemente identificar patrones.

Baterias incluidas – Scrapy proporciona soporte con baterías incluidas para paginación, solicitudes de limitación, cookies, servidores proxy, etc. Esto le evita tener que reinventar la rueda.

Gran ecosistema - Existen **500+ extensiones** proporcionando integración con almacenamiento, controladores web, almacenamiento en caché, etc. Esto permite agregar funcionalidad según sea necesario.

Biblioteca	**Velocidad**	**Escalabilidad**	**Soporte Javascript**	**Curva de aprendizaje**
Scrapy	Muy rapido	Excelente	A través de extensiones	Moderado
BeautifulSoup	Lenta	Pobre	No	Fácil
Selenio	Lenta	Moderado	Excelente	Difícil

Como puede ver, Scrapy ofrece la mejor combinación de velocidad, escalabilidad y facilidad de uso. Es por eso que las principales empresas de tecnología les gusta Scrapy para el web scraping de producción a gran escala.

61TE01_09_Web scraping con scrapy

A continuación, déjame explicarte cómo empezar a usar Scrapy.

Instalando Scrapy

Scrapy está escrito exclusivamente en Python y tiene dependencias mínimas. Instalar:

```
pip install scrapy
```

¡Eso es todo! Scrapy instalará automáticamente paquetes de Python como Twisted, Parsel, etc.

Una vez instalado, puedes verificar ejecutando:

```
scrapy version
```

Esto imprimirá la versión actualmente instalada. Al momento de escribir este artículo, la última versión estable es 2.7.1.

Con Scrapy instalado, ¡estás listo para crear tu primer proyecto!

Creando un nuevo proyecto Scrapy

Scrapy organiza el código de scraping en "Proyectos" que consisten en Spiders, Pipelines, etc. Para crear un nuevo proyecto:

scrapy startproject myproject

Esto genera una myproject directorio con la siguiente estructura:

```
myproject
├── myproject/
│   ├── __init__.py
│   ├── items.py
│   ├── middlewares.py
│   ├── pipelines.py
│   ├── settings.py
│   └── spiders/
│       └── __init__.py
└── scrapy.cfg
```

61TE01_09_Web scraping con scrapy

Este diseño puede parecer complejo inicialmente. Pero ayuda a organizar grandes proyectos y proporciona separación de preocupaciones. Esto es lo que hace cada archivo/carpeta:

- **arañas/** – Dónde reside nuestro código de scraping (Spiders)
- **artículos.py** – Define tipos de datos para datos extraídos
- **oleoductos.py** – Maneja el procesamiento/almacenamiento de datos.
- **settings.py** – Proporciona configuración de tiempo de ejecución
- **middlewares.py** – Implementa middlewares de descarga personalizados
- **scrapy.cfg** – Configuración de implementación para Scrapy
  No te preocupes por todos los archivos por ahora. Principalmente trabajaremos con Spiders en la carpeta ./spiders.

¡Aprendamos cómo funcionan las arañas Scrapy!

Anatomía de una araña raspadora

Las arañas son clases que definen la lógica de raspado de un sitio (o grupo de sitios). Estos son los componentes clave de una araña:

- **nombre** – Identificador único de la araña.
- **dominios_permitidos** – Lista de dominios permitidos para raspar
- **URL_de_inicio** – Lista de URL desde las que comenzar a rastrear
  La lógica principal de scraping está escrita dentro de dos métodos:

- **analizar gramaticalmente()** – Extrae datos de las respuestas.
- **solicitud()** – Genera solicitudes para seguir enlaces.
  Cuando ejecutamos una araña, Scrapy llama al parse() método en la respuesta descargada de cada URL que solicitamos.

El parse() El método analiza la respuesta utilizando selectores y produce:

- Los datos extraídos
- URL adicionales para extraer generando objetos de solicitud
  Esto permite rastrear páginas de forma recursiva siguiendo enlaces. ¡Ahora veamos esto en acción!

61TE01_09_Web scraping con scrapy

Nuestra primera araña

Escribamos una araña simple para extraer listados de productos del sitio books.toscrape.com.

Primero, generaremos una plantilla de araña llamada books_spider.py mediante:

```
scrapy genspider books_spider books.toscrape.com
```

Esto crea una araña con las URL ya completadas:

```
import scrapy

class BooksSpider(scrapy.Spider):
    name = 'books_spider'
    allowed_domains = ['books.toscrape.com']
    start_urls = ['http://books.toscrape.com/']

    def parse(self, response):
        pass
```

Ahora completemos el parse() Método para extraer el título y el precio del producto:

```
    def parse(self, response):
        for product in response.css('article.product_pod'):

            title = product.css('h3 > a::attr(title)').get()
            price = product.css('.price_color::text').get()

            yield {
                'title': title,
                'price': price
            }
```

Aquí nosotros:

- Recorre el product_pod elementos
- Extraiga el título y el precio usando selectores CSS
- Generar un dictado de Python con los datos extraídos
  ¡Y eso es! Nuestro primer pequeño raspador web está listo. Ejecutémoslo:

```
scrapy crawl books_spider
```

Esto rastreará las URL que comienzan en books.toscrape.com y extraerá títulos y precios en un archivo JSON.

61TE01_09_Web scraping con scrapy

Ahora que dominamos los conceptos básicos, aprendamos a desplazarnos por páginas paginadas.

Manejo de la paginación en Scrapy

La mayoría de los sitios web dividen el contenido en varias páginas. Para eliminarlos todos, debemos seguir los enlaces de paginación de forma recursiva.

Así es como nuestra araña de libros puede manejar la paginación:

```
class BooksSpider(scrapy.Spider):

 # ...

 def parse(self, response):

   # Scraping logic..

   next_page = response.css('li.next a::attr(href)').get()

   if next_page is not None:
      yield response.follow(next_page, callback=self.parse)
```

Primero extraemos la URL de la página siguiente usando un selector de CSS. Entonces nosotros yield otra solicitud a Scrapy pasando:

- next_page Enlance
- devolución de llamada como self.parse de nuevo
  Esto hace que Scrapy llame recursivamente parse() en cada respuesta de página hasta que no se encuentren más páginas siguientes. ¡Con buena pinta!

Recuperar contenido paginado es muy sencillo con Scrapy. A continuación veamos cómo almacenar nuestros datos extraídos.

61TE01_09_Web scraping con scrapy

Almacenamiento de datos extraídos

De forma predeterminada, Scrapy devuelve datos extraídos mediante generadores. Podemos almacenarlos en archivos/bases de datos mediante:

1. Exportaciones de piensos

Por ejemplo, para exportar datos extraídos como un archivo JSON:

```
from scrapy.exporters import JsonItemExporter

class BooksSpider(scrapy.Spider):

 # ..spider code

 def close(self, reason):
    jsonFile = open('books.json', 'wb')
    exporter = JsonItemExporter(jsonFile)
    exporter.start_exporting()

    for item in self.scraped_items:
       exporter.export_item(item)

    exporter.finish_exporting()
    jsonFile.close()
```

Scrapy proporciona exportadores integrados para formatos JSON, CSV y XML.

2. **Oleoductos**

Para almacenamiento avanzado, puede escribir Item Pipelines. Algunos ejemplos de tuberías:

- Validar datos extraídos
- Deduplicar elementos duplicados
- Almacenar datos en bases de datos.
- Subir imágenes al almacenamiento en la nube
  Las canalizaciones le permiten posprocesar elementos a medida que se eliminan. Son extremadamente útiles para la limpieza y el almacenamiento de datos.

Ahora que conocemos los conceptos básicos de las arañas y las páginas de raspado, profundicemos un poco más.

61TE01_09_Web scraping con scrapy

Configurar los ajustes de Scrapy

Scrapy utiliza un archivo de configuración para controlar su comportamiento y funcionalidad en tiempo de ejecución. Algunas configuraciones clave incluyen:

AGENTE DE USUARIO – La cadena User-Agent del navegador para enviar con las solicitudes. Aleatorizar esto ayuda a evitar detecciones.

ROBOTSTXT_OBEY – Si se deben obedecer las reglas de robots.txt. Establezca en False para ignorar.

COOKIES_ENABLED – Si se debe habilitar el manejo de cookies. Desactivar para raspado sin cabeza.

SOLICITUDES_CONCURRENTES – El número máximo de solicitudes simultáneas para enviar. Reduzca esto para evitar estrangulamiento.

DESCARGAR_DELAY – El retraso en segundos a esperar antes de enviar cada solicitud. Aumente para acelerar la tasa de raspado.

AGREGAS – Un dictado que contiene complementos y extensiones habilitados para el proyecto.

Hay muchas más configuraciones para modificar la funcionalidad de Scrapy. Recomiendo dominar el configuración principal para personalizar el comportamiento de raspado.

Ahora veamos algunas herramientas útiles para depurar arañas Scrapy.

61TE01_09_Web scraping con scrapy

Depuración de arañas con Scrapy Shell

Scrapy proporciona una herramienta increíblemente útil llamada Scrapy Shell para probar respuestas y selectores.

Para probarlo, ejecute:

```
scrapy shell "http://books.toscrape.com"
```

Esto lo lleva a un shell de Python con la respuesta cargada y con alcance:

```
In [1]: response.css('title')
Out[1]: [<Selector xpath='descendant-or-self::title' data='<title>All products</title>'>]

In [2]: response.xpath('//h1/text()')
Out[2]: [<Selector xpath='//h1/text()' data='All products'>]
```

El shell le permite probar interactivamente los selectores CSS/XPath y ver cómo Scrapy analiza las respuestas.

Lo uso ampliamente en casi todos mis proyectos de scraping para afinar la extracción sin ejecutar las arañas repetidamente. ¡A menudo detecta el 90% de los problemas del selector incluso antes de escribir el código de análisis!

Aprender a aprovechar el caparazón aumentará enormemente su productividad. Scrapy acaba de hacer que la depuración sea divertida

Uso de proxies y agentes de usuario

Mientras realiza el scraping, es común que los sitios lo bloqueen si usted:

- Abrumar al servidor con demasiadas solicitudes
- No utilice encabezados de solicitud similares a los de un navegador
  Aquí hay dos ajustes simples para evitar bloqueos:

61TE01_09_Web scraping con scrapy

1. Rotar agentes de usuario

Seleccione las USER_AGENT configuración para rotar agentes de usuario aleatorios:

```
USER_AGENT = 'RandomUserAgentMiddleware' #Rotating user-agent middleware
```

Esto varía el User-Agent encabezado para que su tráfico parezca más humano.

2. Utilice la rotación de proxy

Para evitar bloqueos basados en IP, puede enrutar solicitudes a través de servidores proxy:

```
class ProxyMiddleware:

 def process_request(self, request, spider):
  request.meta['proxy'] = random.choice(PROXIES)
```

Aquí asignamos aleatoriamente un proxy de una lista a cada solicitud. Esto distribuye cargas entre IP para evitar prohibiciones.

¡Y eso es apenas una muestra de lo que es posible mezclar y combinar componentes de Scrapy!

Ahora abordemos un desafío común: eliminar sitios web dinámicos con JavaScript.

Raspado de sitios web de JavaScript

Una limitación importante de Scrapy es que solo ve contenido HTML estático devuelto inicialmente por los sitios web.

Los sitios modernos dependen en gran medida de JavaScript para representar contenido. Para eliminarlos, necesitamos **analizar JavaScript** y espere a que el sitio se cargue dinámicamente.

Scrapy proporciona integración con herramientas como **chapoteo** y **Dramaturgo** para representar páginas de JavaScript.

Así es como puedes configurar Scrapy para usar Splash:

```
SPLASH_URL = 'http://localhost:8050' #Running Splash instance

DOWNLOADER_MIDDLEWARES = {
   'scrapy_splash.SplashCookiesMiddleware': 723,
   'scrapy_splash.SplashMiddleware': 725,
   'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware': 810,
}

SPIDER_MIDDLEWARES = {
   'scrapy_splash.SplashDeduplicateArgsMiddleware': 100,
}

DUPEFILTER_CLASS = 'scrapy_splash.SplashAwareDupeFilter'
```

Agregamos las clases de middleware necesarias para la integración de Splash.

Para extraer una página, ahora puedes solicitarla a través de Splash:

```
yield SplashRequest(url, args={'wait': 3})
```

Esto renderizará la página con una espera de 3 segundos antes de extraer HTML. ¡Mucho más poderoso que el selenio!

Mejores prácticas de rastreo para evitar prohibiciones

A lo largo de años de web scraping, he aprendido algunas de las mejores prácticas clave de la manera más difícil. Aquí hay algunos consejos:

- Respeto robots.txt y verifique las políticas de un sitio antes de realizar scraping
- Establecer un precio razonable DOWNLOAD_DELAY de 2+ segundos
- Deshabilite las cookies a menos que sea absolutamente necesario
- Utilice proxies y rote los agentes de usuario con cada solicitud
- Evite raspar demasiado rápido; limite a **<100 solicitudes/min**
- Analice datos extraídos y capture regresiones rápidamente
- Vuelva a intentarlo en casos de fallas comunes, como errores 500
  Seguir los estándares y rastrear con educación ayuda a evitar el bloqueo dirigido. ¡Dominar Scrapy teniendo en cuenta estos principios te llevará lejos!

¡Con eso, hemos llegado al final de nuestro viaje de raspado web Scrapy! Concluyamos con algunas conclusiones clave.

61TE01_09_Web scraping con scrapy

Puntos clave

- Scrapy proporciona un marco muy rápido y potente para el web scraping a escala.
- Simplifica las tareas comunes de raspado con sus bibliotecas y herramientas que incluyen baterías.
- Las arañas abstraen la complejidad de rastrear páginas de forma recursiva.
- Los datos extraídos se pueden exportar y almacenar en formatos amigables para los humanos como CSV/JSON.
- El módulo de configuración y el Shell ayudan a personalizar el comportamiento y depurar los raspadores.
- El uso de complementos como Splash permite manejar sitios con mucho JavaScript.
- Seguir los estándares y las mejores prácticas ayuda a evitar fallas en los raspadores.