

先装包

```
pip install scrapy-redis
```

爬虫部分的改造 (crawl爬虫)

```
import scrapy
from scrapy.linkextractors import LinkExtractor
from scrapy.spiders import CrawlSpider, Rule
from fang.settings import custom_settings_for_baidu
from scrapy_redis.spiders import RedisSpider, RedisCrawlSpider

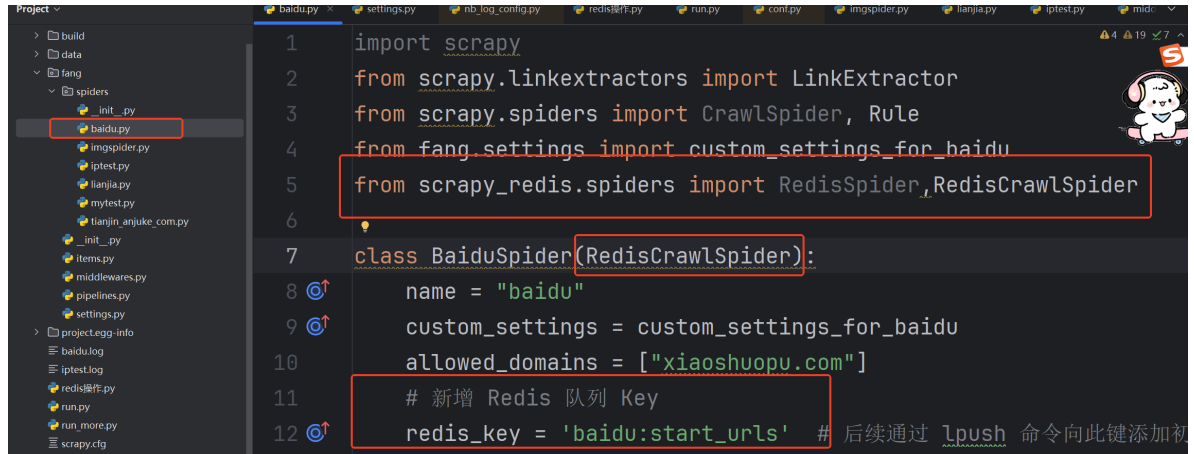
class BaiduSpider(RedisCrawlSpider):
    name = "baidu"
    custom_settings = custom_settings_for_baidu
    allowed_domains = ["xiaoshuopu.com"]
    # 新增 Redis 队列 Key
    redis_key = 'baidu:start_urls' # 后续通过 lpush 命令向此键添加初始
    # start_urls = [f"https://www.xiaoshuopu.com/class_{type}/" for type in
    range(1,4)]

    # rules = (Rule(LinkExtractor(allow=r"Items/"), callback="parse_item",
    follow=True),)
    # 爬取规则配置
    rules = (
        Rule(
            LinkExtractor(
                allow=r'https://www.xiaoshuopu.com/xiaoshuo/\d+/\d+/',
                deny=r'https://www.xiaoshuopu.com/xiaoshuo/\d+/\d+/\d+\.html',
                # restrict_xpaths='//*[@id="at"]'
            ),
            callback='parse_books',
            follow=True
        ),
        Rule(
            LinkExtractor(
                allow=r'https://www.xiaoshuopu.com/xiaoshuo/\d+/\d+/\d+\.html',
                restrict_xpaths='//*[@id="at"]'
            ),
            callback='parse_item',
            follow=False
        ),
    )

    def parse_books(self, response):
        book_name = response.xpath('//h1/text()').extract()
        print('book_name:', book_name, response.url)

    def parse_item(self, response):
        title = response.xpath('//*[ @id="amain"]/dl/dd[1]/h1/text()').extract()
        content = response.xpath('//*[ @id="htmlContent"]//text()').extract()
        print('title:', title, response.url)
        # item = {}
        # item["domain_id"] = response.xpath('//input[@id="sid"]/@value').get()
```

```
#item["name"] = response.xpath('//div[@id="name"]').get()
#item["description"] = response.xpath('//div[@id="description"]').get()
# return item
```



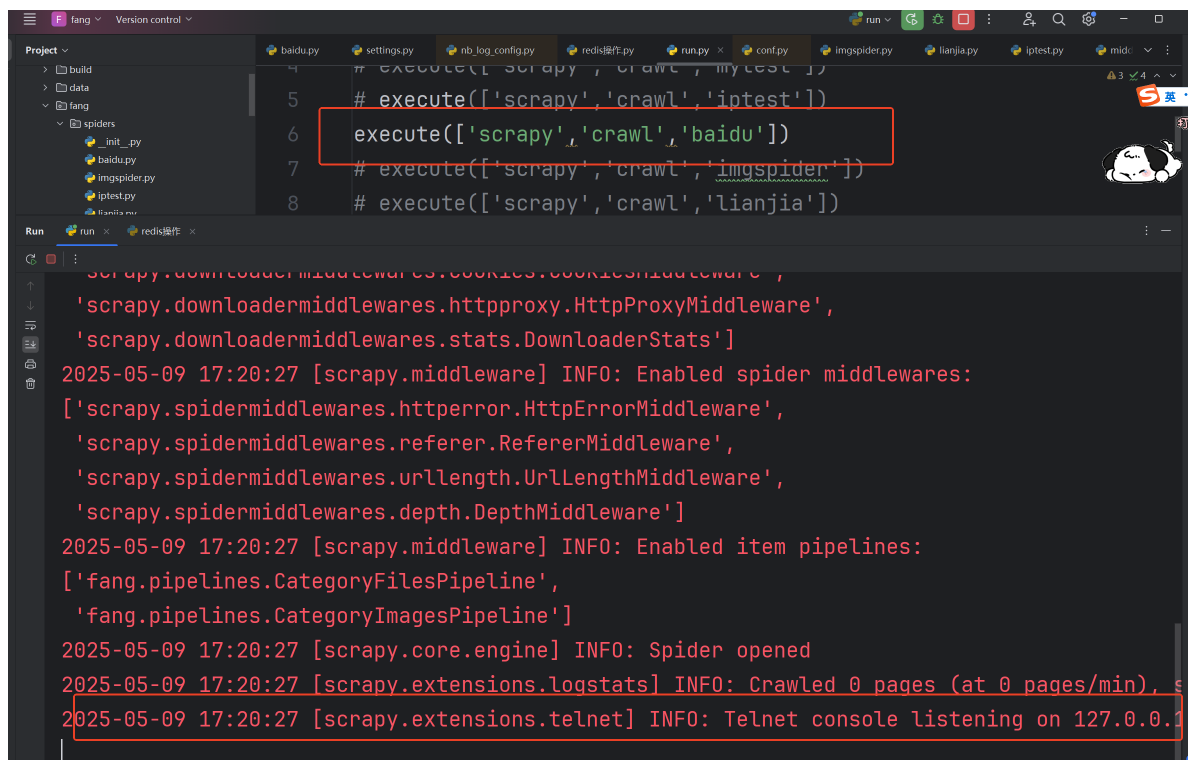
这里可以看到有个

custom_settings=custom_settings_for_baidu这块对应着

setting位置的配置

```
custom_settings_for_baidu = {
    # 启用 Redis 调度器
    'SCHEDULER': 'scrapy_redis.scheduler.Scheduler',
    # 启用 Redis 去重过滤器
    'DUPEFILTER_CLASS': 'scrapy_redis.dupefilter.RFPDupeFilter',
    # Redis 连接地址（根据实际环境修改）
    'REDIS_URL': 'redis://localhost:6379/0',
    # 持久化队列（重启爬虫不丢失任务）
    'SCHEDULER_PERSIST': True,
    # 可选：启用 Redis 存储 Item
    # 'ITEM_PIPELINES': {
    #     'scrapy_redis.pipelines.RedisPipeline': 300
    # }
}
```

这个时候运行启动文件，该爬虫不会爬取，而是在等待任务链接写入



再搞个python文件像redis数据库里写入起始的任务链接，因为现在的爬虫是从redis数据库里读取完成任务的

具体代码如下

```
import redis
from typing import Set, Optional

class RedisManager:
    """Redis 分布式爬虫操作封装工具类"""

    def __init__(self,
                 host: str = 'localhost',
                 port: int = 6379,
                 db: int = 0,
                 password: Optional[str] = None,
                 decode_responses: bool = True):
        """
        初始化Redis连接
        :param host: Redis服务器地址
        :param port: Redis端口
        :param db: 数据库编号
        :param password: 访问密码
        :param decode_responses: 是否自动解码响应
        """
        self.pool = redis.ConnectionPool(
            host=host,
            port=port,
            db=db,
            password=password,
            decode_responses=decode_responses
        )
        self.conn = redis.StrictRedis(connection_pool=self.pool)

    def push_start_url(self, url: str, spider_name: str = "baidu") -> int:
```

```

        """
        向起始URL队列添加新URL
        :param url: 要添加的URL
        :param spider_name: 爬虫名称（默认为baidu）
        :return: 添加后队列长度
        """

        key = f"{spider_name}:start_urls"
        return self.conn.lpush(key, url)

def get_pending_count(self, spider_name: str = "baidu") -> int:
    """
    获取待抓取请求数量
    :param spider_name: 爬虫名称
    :return: 队列当前长度
    """

    key = f"{spider_name}:requests"
    return self.conn.llen(key)

def get_fingerprints(self, spider_name: str = "baidu") -> Set[str]:
    """
    获取去重指纹集合
    :param spider_name: 爬虫名称
    :return: 去重指纹集合
    """

    key = f"{spider_name}:dupefilter"
    return self.conn.smembers(key)

def close(self):
    """关闭连接池"""
    self.pool.disconnect()

# 使用示例
if __name__ == "__main__":
    # 初始化连接
    redis_mgr = RedisManager(host='localhost',
                              port=6379,
                              db=0,
                              password='')

    try:
        start_urls = [f"https://www.xiaoshuopu.com/class_{type}/" for type in
range(1,4)]

        # 添加初始URL
        for url in start_urls:
            count = redis_mgr.push_start_url(url)
            print(f"添加成功, 当前队列长度: {count}")

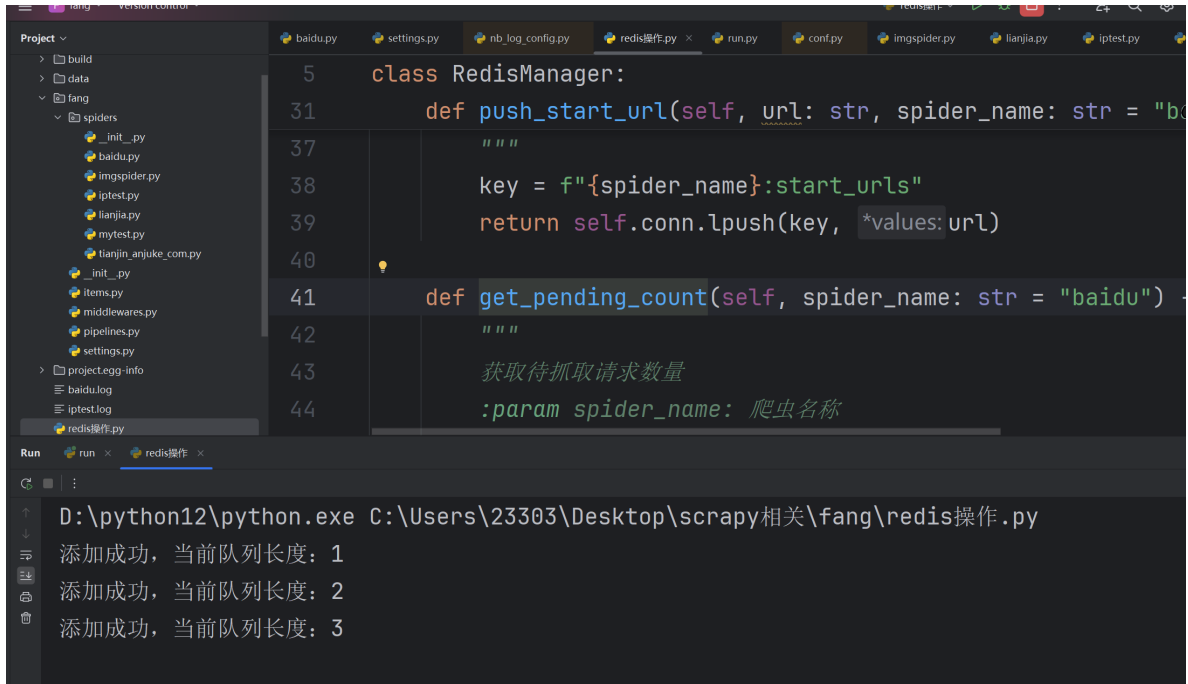
        # # 查看待抓取数量
        # pending = redis_mgr.get_pending_count()
        # print(f"待处理请求数: {pending}")
        #
        # # 获取去重指纹
        # fingerprints = redis_mgr.get_fingerprints()
        # print(f"当前去重指纹数量: {len(fingerprints)}")

    finally:

```

```
redis_mgr.close()
```

我们目前仅仅调用添加初始任务的功能

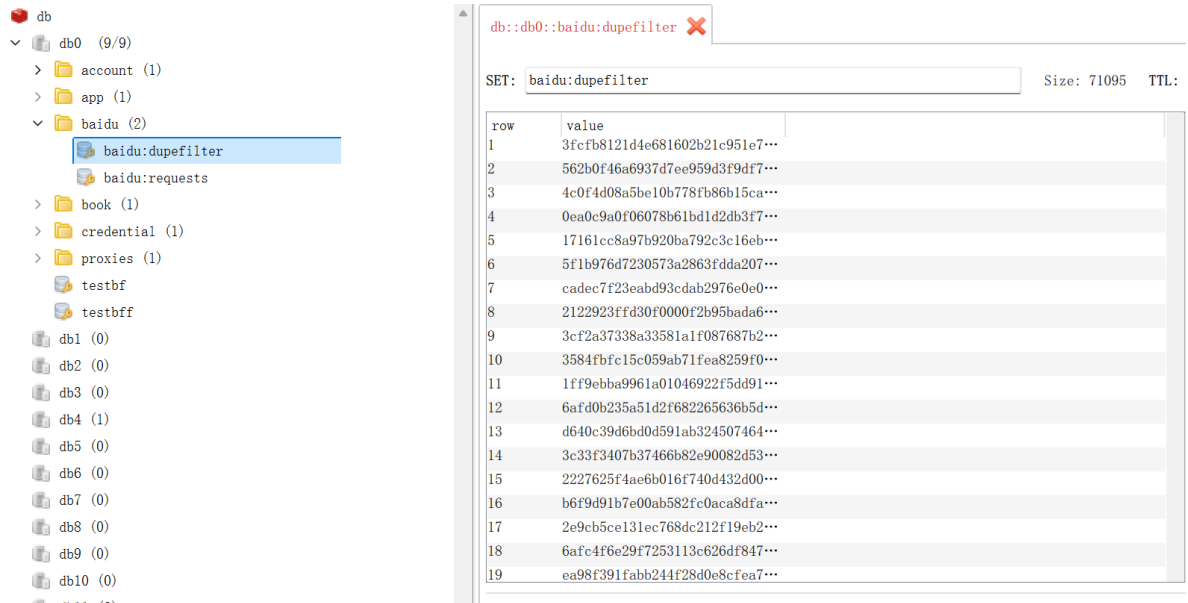


```
5 class RedisManager:
31     def push_start_url(self, url: str, spider_name: str = "baidu"):
37         """
38         key = f"{spider_name}:start_urls"
39         return self.conn.lpush(key, *values: url)
40
41     def get_pending_count(self, spider_name: str = "baidu"):
42         """
43         获取待抓取请求数量
44         :param spider_name: 爬虫名称
```

Run D:\python12\python.exe C:\Users\23303\Desktop\scrapy相关\fang\redis操作.py

添加成功, 当前队列长度: 1
添加成功, 当前队列长度: 2
添加成功, 当前队列长度: 3

刚才等代的爬虫会自己启动



db

- db0 (9/9)
 - account (1)
 - app (1)
 - baidu (2)
 - baidu:dupefilter
 - baidu:requests
 - book (1)
 - credential (1)
 - proxies (1)
 - testbf
 - testbff
- db1 (0)
- db2 (0)
- db3 (0)
- db4 (1)
- db5 (0)
- db6 (0)
- db7 (0)
- db8 (0)
- db9 (0)
- db10 (0)

db::db0::baidu:dupefilter

SET: baidu:dupefilter Size: 71095 TTL:

row	value
1	3fcfb8121d4e681602b21c951e7...
2	562b0f46a6937d7ee959d3f9df7...
3	4c0f4d08a5be10b778fb86b15ca...
4	0ea0c9a0f06078b61bd1d2db3f7...
5	17161cc8a97b920ba792c3c16eb...
6	5f1b976d7230573a2863fdda207...
7	cadec7f23eabd93cdab2976e0e0...
8	2122923ffd30f0000f2b95bada6...
9	3cf2a37338a33581a1f087687b2...
10	3584fbfc15c059ab71fea8259f0...
11	1ff9ebba9961a01046922f5dd91...
12	6afd0b235a51d2f682265636b5d...
13	d640c39d6bd0d591ab324507464...
14	3c33f3407b37466b82e90082d53...
15	2227625f4ae6b016f740d432d00...
16	b6f9d91b7e00ab582fc0aca8dfa...
17	2e9cb5ce131ec768dc212f19eb2...
18	6afc4f6e29f7253113c626df847...
19	ea98f391fab244f28d0e8cfea7...

The screenshot shows a Redis Explorer interface. On the left, a tree view displays the database structure: db0 (9/9) contains folders for account (1), app (1), baidu (2), book (1), credential (1), proxies (1), testbf, and testbff. The 'baidu' folder is expanded, showing 'baidu:dupefilter' and 'baidu:requests'. The 'baidu:requests' folder is selected. On the right, a ZSET list is displayed for the key 'baidu:requests'. The list contains 18 items, each with a row number, a value (a long hexadecimal string), and a score of 0. The ZSET is named 'baidu:requests' and has a size of 92419 and a TTL of -.

查看redis可以看到去重队列和任务队列

如果是基础模版，那爬虫部分这么改

```
import scrapy
from fang.items import BookItem
from scrapy_redis.spiders import RedisSpider, RedisCrawlSpider
from fang.settings import custom_settings_for_mytest

class MytestSpider(RedisSpider):
    name = "mytest"
    allowed_domains = ["xiaoshuopu.com"]
    custom_settings = custom_settings_for_mytest
    redis_key = 'mytest:start_urls'
    # start_urls = ["https://www.xiaoshuopu.com/xiaoshuo/69/69434/"]

    def parse(self, response):
        chapter_url = response.xpath('//td[@class="L"]/a/@href').extract()
        for url in chapter_url:
            # 'https://www.xiaoshuopu.com/xiaoshuo/69/69434/32780156.html'
            yield
            scrapy.Request(f'https://www.xiaoshuopu.com{url}', self.get_content)

    def get_content(self, response):
        item = BookItem()
        item['title'] = response.xpath('//h1/text()').extract_first()
        item['contents'] =
        ''.join(response.xpath('//div[@id="htmlContent"]//text()').extract())
        print(item)
        return item
```

其他的没什么区别