my_WB_parser_08 20250525

'''
↓↓↓↓↓↓↓↓↓↓----------

Парсинг WB WILDBERRIES с учётом кликабельности карточки товаров
20250525

Проблема, которую пришлось решать вплоть до этой версии [вариант №8] - не полностью считывались со страницы сайта карточки товаров (например, на сайте после применения фильтра остаются 100 карточек товаров, а при работе парсера считываются не все 100, а меньше). При этом элемент пагинации считывается без ошибок.

Версии парсера:

my_WB_parser_01
my_WB_parser_02
my_WB_parser_03.py
my_WB_parser_04
my_WB_parser_05.py
my_WB_parser_06.py - исследование подходов контролированию целостности сайта
my_parser_07_[v01-v05].py - Scrapy
my_WB_parser_08.py - финал

С большой степенью вероятности, могу утверждать – причина неполноты спарсенной информации о карточках в том, что:

1. Виртуализация карточек (windowing):
Wildberries может использовать технологию виртуализации: в DOM одновременно находятся только те карточки, которые видимы пользователю, а остальные динамически удаляются и добавляются при прокрутке. Это видно по использованию классов типа j-card-item и событиям, связанным с появлением элемента в зоне видимости (elementInViewportEvent).

2. Недостаточное время ожидания:
Карточки подгружаются с задержкой, а цикл скроллинга завершается раньше, чем все карточки успеют появиться в DOM.

3. Пагинация:
На Wildberries может быть реализована как бесконечная прокрутка, так и классическая пагинация (переключение страниц).
В моём коде реализован переход по страницам, но на каждой странице всё равно может работать виртуализация.

Здесь реализован алгоритм гарантированного сбора всех карточек по мере прокрутки в процедуре

def collect_all_cards_on_page(driver, pause_time=1.5, step=500):.

Если карточки "исчезают" из DOM при прокрутке вверх/вниз (виртуализация), единственный надёжный способ — собирать карточки по мере появления и сохранять их уникальные data-nm-id в отдельное множество (set). Таким образом, даже если карточка исчезла из DOM, её данные останутся уже сохранёнными для дальнейшего использования.

↑↑↑↑↑↑↑↑↑↑----------
'''

```python
# pip install selenium

from selenium.webdriver import Chrome
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait as waits
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.common.by import By
from selenium.common.exceptions import StaleElementReferenceException
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.action_chains import ActionChains
from bs4 import BeautifulSoup
from time import sleep
from tqdm import tqdm
from pprint import pprint
import time
from datetime import timedelta
import json
import pandas as pd

###################### ОБЩЕЕ КОЛИЧЕСТВО СТРАНИЦ С ТОВАРОМ (ПАГИНАЦИЯ)

def get_max_pages(driver):
    """Функция для определения общего количества страниц с товарами"""
    try:
        pagination = waits(driver, 20).until(
            EC.presence_of_element_located((By.CSS_SELECTOR, '.pagination'))
        )
        pages = pagination.find_elements(By.CSS_SELECTOR, 'a.pagination-item')
        if pages:
            last_page = pages[-1].text
            return int(last_page) if last_page.isdigit() else 1
        return 1
    except:
        return 1

###################### СБОР ВСЕХ КАРТОЧЕК ТОВАРА ПО МЕРЕ ПРОКРУТКИ ПРИ ДИНАМИЧЕСКОЙ ВЕРСТКИ САЙТА

'''
Этот вариант — реализует гарантированный сбор всех карточек по мере прокрутки

Если карточки "исчезают" из DOM при прокрутке вверх/вниз (виртуализация), единственный надёжный способ — собирать карточки по мере появления и сохранять их уникальные data-nm-id в отдельное множество (set). Таким образом, даже если карточка исчезла из DOM, её данные останутся у меня прочитанными.

DOM (Document Object Model) — это объектная модель документа, которую браузер создаёт в памяти на основе HTML-кода, полученного с сервера. Проще говоря, это структура веб-страницы в виде дерева, где каждый узел — это HTML-элемент, текст или атрибут.

DOM позволяет программам (например, JavaScript или инструментам автоматизации вроде Selenium) обращаться к элементам страницы, изменять их, добавлять новые или удалять
```

существующие без перезагрузки страницы. Это основа для динамического взаимодействия с веб-страницами.

Пример: если в HTML есть тег Привет , в DOM он представлен как узел с тегом p и текстовым содержимым "Привет". Через DOM можно изменить этот текст или добавить новые элементы.

Таким образом, DOM — это мост между статичным HTML-кодом и динамическим содержимым, которое видит пользователь и с которым взаимодействует браузер.

Если коротко:
DOM — это внутренняя структура веб-страницы в браузере, которая позволяет программно управлять её содержимым и структурой.
'''

```python
def collect_all_cards_on_page(driver, pause_time=1.5, step=500):
    """
    Прокручивает страницу и собирает все уникальные карточки товаров по data-nm-id.
    """
    seen_ids = set()
    cards_data = []
    last_height = driver.execute_script("return document.body.scrollHeight")
    pos = 0

    while pos < last_height:
        driver.execute_script(f"window.scrollTo(0, {pos});")
        time.sleep(pause_time)
        cards = driver.find_elements(By.CSS_SELECTOR, 'article[data-nm-id]')
        for card in cards:
            nm_id = card.get_attribute("data-nm-id")
            if nm_id and nm_id not in seen_ids:
                seen_ids.add(nm_id)
                # Здесь можно собрать нужные данные с карточки
                cards_data.append(card)
        pos += step
        last_height = driver.execute_script("return document.body.scrollHeight")
    return cards_data

###################### КЛИК ПО КАРТОЧКЕ ТОВАРА

def get_country_of_production(driver, card):
    """Функция для получения страны производства через клик по карточке товара"""
    try:
        # Запоминаем текущее состояние
        main_window = driver.current_window_handle
        current_url = driver.current_url

        # Кликаем на карточку товара (открываем в новой вкладке)
        ActionChains(driver).key_down(Keys.CONTROL).click(card).key_up(Keys.CONTROL).perform()
        time.sleep(1)

        # Переключаемся на новую вкладку
        new_window = [window for window in driver.window_handles if window != main_window][0]
        driver.switch_to.window(new_window)

        # Ожидаем загрузки страницы товара
        wait = waits(driver, 10)
```

```
        wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, '.product-params.product-params--mini')))

        # Получаем страну производства
        country = None
        try:
            # Находим все строки с параметрами
            params_rows = driver.find_elements(By.CSS_SELECTOR, '.product-params__row')

            for row in params_rows:
                # Проверяем, содержит ли строка текст "Страна производства"
                if 'Страна производства' in row.text:
                    # Извлекаем значение из соседней ячейки
                    country_cell = row.find_element(By.CSS_SELECTOR, 'td.product-params__cell')
                    country = country_cell.text.strip()
                    break

        except Exception as e:
            print(f"Ошибка при поиске страны производства: {str(e)}")

        # Закрываем вкладку с товаром
        driver.close()

        # Возвращаемся на исходную вкладку
        driver.switch_to.window(main_window)

        # Возвращаемся на исходный URL (на случай редиректов)
        if driver.current_url != current_url:
            driver.get(current_url)
            wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, '.product-card__wrapper')))

        return country

    except Exception as e:
        print(f"Ошибка при получении страны производства: {str(e)}")
        # Восстанавливаем исходное состояние
        try:
            if 'new_window' in locals() and new_window in driver.window_handles:
                driver.switch_to.window(new_window)
                driver.close()
            driver.switch_to.window(main_window)
            if driver.current_url != current_url:
                driver.get(current_url)
        except:
            pass
        return None

##################### ГЛАВНАЯ ПРОЦЕДУРА

def main():
    # Начало измерения длительности исполнения кода
    start_time = time.time()

    # Указываем путь к ChromeDriver
    service = Service(ChromeDriverManager().install())

    # Настройки Chrome
```

```
    chrome_options = Options()
    chrome_options.add_argument("--headless") # Скрытый режим
    chrome_options.add_argument("--disable-software-rasterizer") # Отключить аппаратное ускорение при запуске браузера
    chrome_options.add_argument("--disable-gpu") # Отключить аппаратное ускорение при запуске браузера
    chrome_options.add_argument('--log-level=3') # Меньше логов
    chrome_options.add_argument("--no-sandbox")
    chrome_options.add_argument("--window-size=1920,1080")
    chrome_options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36")

    # Инициализация драйвера
    driver = webdriver.Chrome(service=service, options=chrome_options)
    wait = waits(driver, 20)

    try:
        # Адрес страницы
        url = 'https://www.wildberries.ru/'
        driver.get(url)


# Функция для прокрутки страницы до конца

###################### ПРОКРУТКА ВНИЗ СТРАНИЦЫ – ЧТОБЫ ДОЖДАТЬСЯ ПОЛНОЙ ЗАГРУЗКИ СТРАНИЦЫ И СОБРАТЬ ВСЕ КАРТОЧКИ

        def scroll_to_bottom(driver, pause_time=2, max_attempts=30):
            last_height = driver.execute_script("return document.body.scrollHeight")
            attempts = 0

            while attempts < max_attempts:
                driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
                time.sleep(pause_time)

                new_height = driver.execute_script("return document.body.scrollHeight")
                if new_height == last_height:
                    # Дополнительное ожидание на случай отложенной загрузки
                    time.sleep(pause_time)
                    new_height = driver.execute_script("return document.body.scrollHeight")
                    if new_height == last_height:
                        break
                last_height = new_height
                attempts += 1

###################### ОЭИДАНИЕ ЗАГРУЗКИ ВСЕХ КАРТОЧЕК

        def wait_for_cards_to_load(driver, min_cards=100, timeout=20):
            wait = waits(driver, timeout)
            for _ in range(timeout):
                cards = driver.find_elements(By.CSS_SELECTOR, 'article[data-nm-id]')
                if len(cards) >= min_cards:
                    return cards
                time.sleep(1)
            return cards  # что есть, то есть

        # Поиск и взаимодействие с поисковой строкой
```

```
    input_tab = wait.until(
        EC.presence_of_element_located((By.ID, 'searchInput'))
    )
    print('\nПоиск элемента из поисковой строки:')
    pprint(input_tab)

    # Ввод поискового запроса и активизация нажатия кнопки
    input_tab.send_keys('свой йогурт закваска' + Keys.ENTER)

    # Ожидание результатов поиска
    wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, '.product-card__wrapper')))

    # Определение количества страниц
    max_pages = get_max_pages(driver)
    print(f"\nНайдено страниц: {max_pages}\n")

    products_data = []
    processed_pages = 0

    while processed_pages < max_pages:
        # Плавная прокрутка страницы
        last_height = driver.execute_script("return document.body.scrollHeight")
        for i in range(0, last_height, 500):
            driver.execute_script(f"window.scrollTo(0, {i});")
            time.sleep(1.5)

        time.sleep(8)

        # Получение всех карточек товаров на странице

        product_cards = collect_all_cards_on_page(driver)
        print(f'Страница {processed_pages + 1}/{max_pages}. Уникальных карточек:
{len(product_cards)}')

        product_ids = [card.get_attribute("data-nm-id") for card in product_cards]
        print('Размер словаря product_IDs на этой странице =', len(product_ids))

        for card in product_cards:
            try:
                # Прокрутка к карточке товара
                driver.execute_script("arguments[0].scrollIntoView({block: 'center'});", card)
                time.sleep(0.5)

                # Получение HTML-кода карточки
                card_html = card.get_attribute('outerHTML')
                soup = BeautifulSoup(card_html, 'html.parser')

                # Извлечение ID товара
                product_link_tag = soup.find('a', class_='product-card__link')
                product_link = product_link_tag['href'] if product_link_tag else None
                product_id = product_link.split('/')[-2] if product_link else None
                product_name_long = product_link_tag['aria-label'] if product_link_tag else None

                # Извлечение основных данных о товаре
                brand_tag = soup.find('span', class_='product-card__brand')
                brand = brand_tag.text.strip() if brand_tag else None
```

```
            name_separator_tag = soup.find('span', class_='product-card__name')
            product_name = name_separator_tag.get_text(strip=True)[1:] if name_separator_tag else
None

            price_tag = soup.find('ins', class_='price__lower-price')
            price = price_tag.text.strip() if price_tag else None

            discount_tag = soup.find('span', class_='percentage-sale')
            discount = discount_tag.text.strip() if discount_tag else None

            # Извлечение рейтинга с обработкой ошибок
            rating_span = soup.find('span', class_='address-rate-mini address-rate-mini--sm')
            try:
                rating = float(rating_span.text.replace(',', '.')) if rating_span else 0
            except (ValueError, AttributeError):
                rating = 0

            # Извлечение количества оценок
            grade_count_span = soup.find('span', class_='product-card__count')
            grade_count = grade_count_span.get_text(strip=True) if grade_count_span else '0'

            # Получаем страну производства через клик по карточке
            country = get_country_of_production(driver, card)

            # Формирование данных о товаре
            product_data = {
                'ID': product_id,
                'Бренд': brand,
                'Наименование продукта': product_name,
                'Полное наименование': product_name_long,
                'Ссылка на продукт': product_link,
                'Цена': price,
                'Скидка%': discount,
                'Рейтинг': rating,
                'Количество оценок': grade_count,
                'Страна производства': country  # Добавлен новый параметр
            }

            products_data.append(product_data)

        except Exception as e:
            print(f"Ошибка при парсинге карточки: {str(e)}")
            continue

    processed_pages += 1
    if processed_pages < max_pages:
        try:
            # Переход на следующую страницу - пагинация
            next_btn = wait.until(EC.element_to_be_clickable((By.CSS_SELECTOR, '.pagination-next')))
            driver.execute_script("arguments[0].click();", next_btn)
            wait.until(EC.staleness_of(product_cards[0]))
            time.sleep(2)
        except Exception as e:
            print(f"Не удалось перейти на следующую страницу: {str(e)}")
            break

except Exception as e:
```

```
        print(f"Произошла критическая ошибка: {str(e)}")
    finally:
        if driver:
            driver.quit()

        # Обработка и сохранение данных
        df = pd.DataFrame(products_data)
        if not df.empty:
            # Очистка и преобразование данных
            df['Цена'] = df['Цена'].str.replace(r'[^\d]', '', regex=True).astype(float)
            df['Скидка%'] = df['Скидка%'].str.extract(r'(\d+)')[0].astype(float)
            df['Количество оценок'] = (
                df['Количество оценок']
                .apply(lambda x: ''.join(filter(str.isdigit, x)) if any(char.isdigit() for char in x) else '0')
                .astype(int)
            )

            # Сохранение в CSV
            df.to_csv('wildberries_products.csv', index=False, encoding='utf-8-sig')
            print(f"\nДанные сохранены в wildberries_products.csv")
            print(f"Итого собрано: {len(df)} товаров")
            print(df)
            print('\n')
            df.info()
            print('\n', df.describe())
            print('\n', df.describe(include='all'))

        # Вывод времени выполнения
        end_time = time.time()
        elapsed_time = end_time - start_time
        time_delta = timedelta(seconds=elapsed_time)
        print(f'\nВремя работы парсера: {time_delta.days} дней - '
              f'{time_delta.seconds // 3600} часов - '
              f'{(time_delta.seconds // 60) % 60} минут - '
              f'{time_delta.seconds % 60} секунд\n')

if __name__ == "__main__":
    main()
```