МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра информатики и программирования

**Анализ методов парсинга сайтов**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

студента 4 курса 441 группы

направления (специальности) 02.03.03 МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ И АДМИНИСТРИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Факультет Компьютерных Наук И Информационных Технологий

Турченкова Павла Александровича

Научный руководитель (руководитель)

должность, уч. степень, уч. звание подпись, дата инициалы, фамилия

Консультант

должность, уч. степень, уч. звание подпись, дата инициалы, фамилия

Зав. кафедрой

должность, уч. степень, уч. звание подпись, дата инициалы, фамилия

Саратов 2019

**Содержание**

[Введение 6](#_Toc9131402)

[Описание актуальности работы 7](#_Toc9131403)

[Алгоритм парсинга сайта LetyShops 8](#_Toc9131404)

[Алгоритм парсинга сайта MegaBonus 9](#_Toc9131405)

[Отбор библиотек для парсинга 10](#_Toc9131406)

[Реализация на языке Java 11](#_Toc9131407)

[**Jsoup** 11](#_Toc9131408)

[**Selenium** 13](#_Toc9131409)

[**Unirest** 15](#_Toc9131410)

[Реализация на языке C# 23](#_Toc9131411)

[**HtmlAgilityPack** 23](#_Toc9131412)

[**Fizzler** 24](#_Toc9131413)

[**AngleSharp** 25](#_Toc9131414)

[**CsQuery** 27](#_Toc9131415)

[**RestSharp** 28](#_Toc9131416)

[**Selenium** 31](#_Toc9131417)

[Реализация на языке Python 34](#_Toc9131418)

[**BeautifulSoup** 34](#_Toc9131419)

[**Scrapy** 36](#_Toc9131420)

[**Request** 40](#_Toc9131421)

[**Selenium** 43](#_Toc9131422)

[Сравнение скорости реализаций 46](#_Toc9131423)

[Описание визуализации результатов 47](#_Toc9131424)

[Описание выхода из специфических ситуаций при парсинге 48](#_Toc9131425)

[Заключение 49](#_Toc9131426)

[Список использованной литературы 50](#_Toc9131427)

[Приложение А. JSoup 51](#_Toc9131428)

[Приложение Б. Selenium Java 52](#_Toc9131429)

[Приложение В. Unirest 53](#_Toc9131430)

[Приложение Г. HtmlAgilityPack 54](#_Toc9131431)

[Приложение Д. Fizzler 55](#_Toc9131432)

[Приложение Е. AngleSharp 56](#_Toc9131433)

[Приложение Ж. CsQuery 57](#_Toc9131434)

[Приложение З. RestSharp 58](#_Toc9131435)

[Приложение И. Selenium C# 59](#_Toc9131436)

[Приложение К. BeutifulSoup 60](#_Toc9131437)

[Приложение Л. Scrapy 61](#_Toc9131438)

# Введение

# Описание актуальности работы

# Алгоритм парсинга сайта LetyShops

В этой главе описывается алгоритм парсинга сайта LetyShops. В результате действия алгоритма в списке будут находиться магазины с кэшбэками.

По ходу разбора алгоритма будут описаны нюансы при парсинге данного сайта.

Для разбора алгоритма будет использоваться пример на языке программирования C#, используя библиотеку Fizzler.

Первым шагом для построения алгоритма парсинга сайта является ознакомление с самим сайтом. Сначала надо перейти на страницу, которая отображает предложенные магазины. В данном случае это <https://letyshops.com/shops>. Страница сайта изображена на Рисунке 1. При просмотре данной страницы следует обратить внимание на саму структуру сайта, то есть на расположение магазинов, а также на те элементы, которые позволяют получиться следующие магазины.

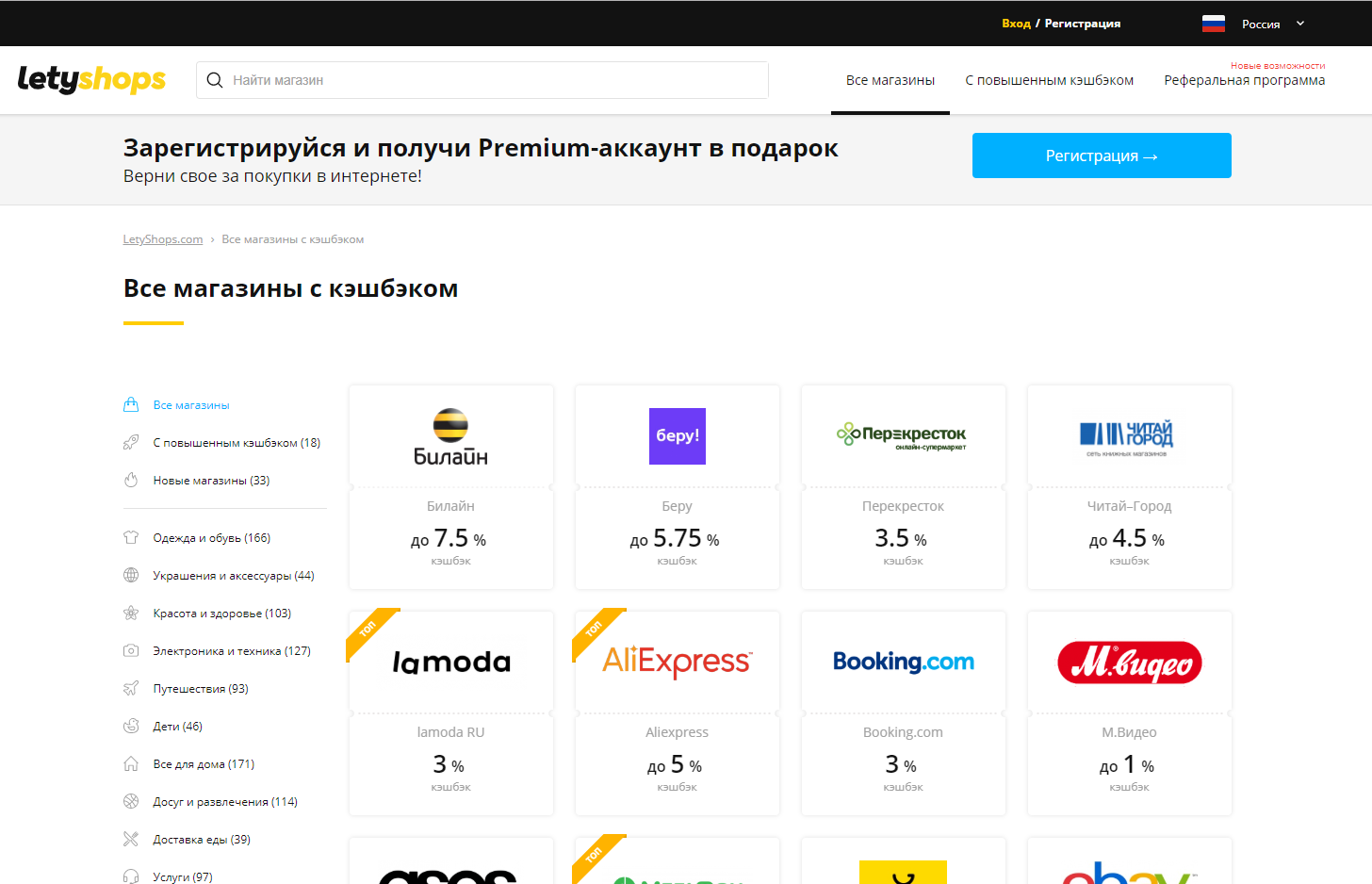


Рисунок 1 – страница с магазинами сайта LetyShops

После всех магазинов расположен элемент, который позволяет переходить по страницам магазина. Элемент изображен на Рисунке 2



Рисунок 2 – элемент страницы для перехода по страницам

После перехода на последующие страницы, URL меняется так, что в него добавляются параметры в виде номера страницы – https://letyshops.com/shops?page=2. Это облегчает создание алгоритма, так как для перехода по сайту потребуется лишь менять один параметр.

Для просмотра структуры страницы в виде html используется панель разработчика. С помощью этой панель осуществляется поиск элементов. На Рисунке 3 изображен выбор конкретного магазина.

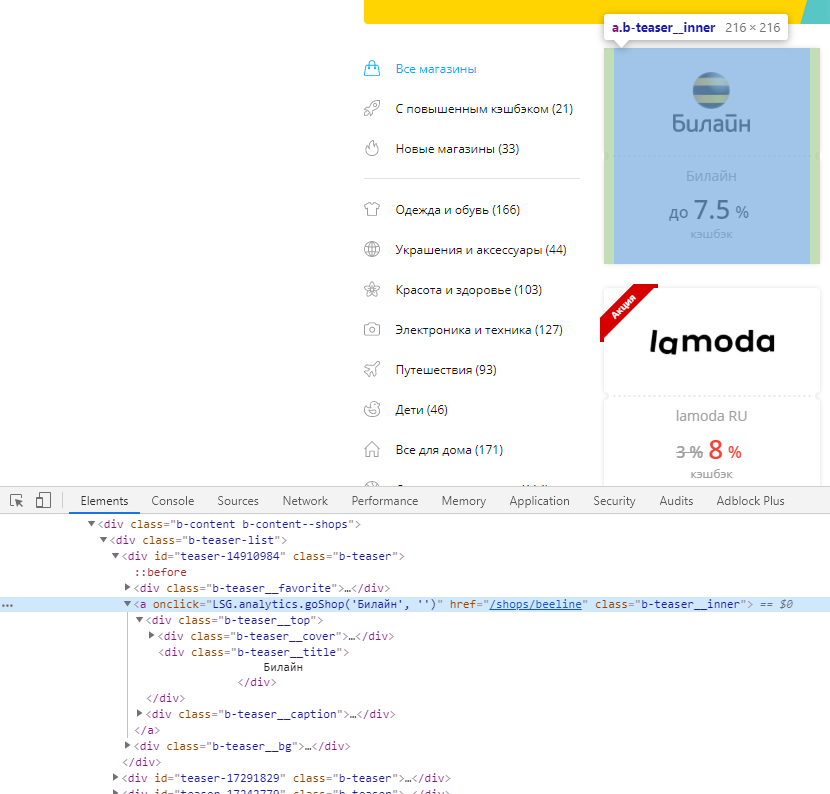


Рисунок 3 – выбор элемента с магазином

Из предыдущего рисунка видно, что элемент находится в тэге «а» с классом «b-teaser\_\_inner». Внутри этого элемента содержится вся нужная информация. То есть нужно извлечь все элементы с приведённым ранее классом. Ниже представлен код для извлечения элементов с одной страницы.

var listOfShops = html.QuerySelectorAll("div.b-teaser > a.b-teaser\_\_inner");

Используя css селектор, из html кода выбираются все элементы, которые подходят под заданный критерий и записываются в список listOfShops.

(тут я пишу более просто, чтобы потом переделать более научно)

Далее проходимся по списку, и каждый элемент передаём на вход методам в качестве аргумента. После получения данных из методов, происходит проверка, а затем добавление в список. Ниже представлен код.

1. foreach (var item in listOfShops)
2. {
3. String name = GetName(item);
4. Double discount = GetDiscount(item);
5. String label = GetLabel(item);
6. String url = GetUrl(item);
7. String image = GetImage(item);
8. if (!(String.IsNullOrEmpty(name) || Double.IsNaN(discount) || String.IsNullOrEmpty(label) || String.IsNullOrEmpty(image) || String.IsNullOrEmpty(url)))
9. {
10. Shops.Add(new Shop(name, discount, label, image, url));
11. }
12. }

Как видно из примера, для каждого элемента есть свой метод, в котором содержится логика для извлечения информации.

Метод GetName() извлекает из элемента имя, которое находится в элементе «div» с классом «b-teaser\_\_title». С помощью css селектора происходит перемещение к этому элементу, а с помощью свойства InnerText извлекается текст тэга. Метод Trim() нужен для удаления лишних пробелов. Пример кода приведён ниже.

1. private String GetName(HtmlNode html)
2. {
3. return html.QuerySelector("div.b-teaser\_\_title").InnerText.Trim();
4. }

Метод GetDiscount извлекает из элемента значение кэшбэка, которое может находиться в двух местах в зависимости от того старое значение или новое. Если значение кэшбэка не менялось и не помечалось красным, то оно находится в тэге «span» с классом «b-shop-teaser\_\_cash». Если значение кэшбэка новое и оно помечено красным, как на Рисунке 4, то путь к элементу будет уже иным. Элемент находится в тэге «span» с классом «b-shop-teaser\_\_new-cash». Далее с помощью css селектора происходит перемещение к этому элементу, а с помощью свойства InnerText извлекается текст тэга. Метод Trim() нужен для удаления лишних пробелов. После извлечения значения кэшбэка, его надо привести к типу double. Если в строке разделителем целой и дробной части является точка, то язык программирования C# выдаёт ошибку, в которой написано, что строка имеет неправильный формат. Для решения этой проблемы нужно заменить точку запятой с помощью метода replace(). Пример кода приведён ниже.

1. private Double GetDiscount(HtmlNode html)
2. {
3. string discount = "";
4. try
5. {
6. discount = html.QuerySelector("div.b-teaser\_\_caption > div.b-teaser\_\_cashback-rate > div > div > span.b-shop-teaser\_\_cash").InnerText.Trim();
7. }
8. catch (NullReferenceException e)
9. {
10. logger.Info("Вторая попытка взять discount " + e);
11. discount = html.QuerySelector("div.b-teaser\_\_caption > div.b-teaser\_\_cashback-rate > div > div > span.b-shop-teaser\_\_new-cash").InnerText.Trim(); ;
12. }
13. if (Double.TryParse(discount.Replace('.', ','), out double result))
14. {
15. return result;
16. }
17. return Double.NaN;
18. }

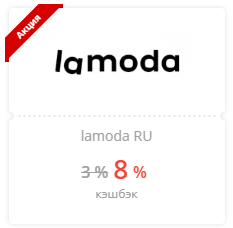


Рисунок 4 – новое значение кэшбэка

Метод GetLabel() извлекает из элемента валюту. На сайте есть три вида кэшбэка по расположению в html коде: первый содержит в себе только значение и валюту – тэг «span» с классом «b-shop-teaser\_\_label»; второй содержит в себе слово «до», который расположен в тэге «span» с классом «b-shop-teaser\_\_label», а потом значение и валюту, которая расположена в том же тэге с таким же классом; третий – кэшбэк красного цвета, как изображено на Рисунке 4, находится в тэге «span» с классом «b-shop-teaser\_\_label b-shop-teaser\_\_label--red». Для того, чтобы не писать множество проверок было решено с помощью QuerySelectorAll найти все элементы, которые подходили под следующий критерий: тэг «span» с классом «b-shop-teaser\_\_label», и из коллекции брать последний элемент. Далее с помощью css селектора происходит перемещение к этому элементу, а с помощью свойства InnerText извлекается текст тэга. Метод Trim() нужен для удаления лишних пробелов. Пример кода приведён ниже.

1. private String GetLabel(HtmlNode html)
2. {
3. var labelList = html.QuerySelectorAll("div.b-teaser\_\_caption > div.b-teaser\_\_cashback-rate > div > div > span.b-shop-teaser\_\_label");
4. return labelList.Last().InnerText.Trim();
5. }

Метод GetUrl() извлекает из элемента страницу магазина на сайте. Ссылка находится в тэге «a» с классом «b-teaser\_\_inner» в атрибуте «href». Так как передаваемый методу элемент как раз совпадают критерию, с помощью метода GetAttributeValue() извлекается значение заданного атрибута. Так как ссылка не полная, в начало добавляется основной URL сайта. Пример кода приведён ниже.

1. private String GetUrl(HtmlNode html)
2. {
3. return addressOfSite + html.GetAttributeValue("href", "");
4. }

Метод GetImage() извлекает из элемента ссылку на картинку магазина на сайте. Ссылка находится в тэге «img», который является дочерним элементом тэга «div» с классом «b-teaser\_\_cover». С помощью css селектора происходит переход к элементу. Далее с помощью метода GetAttributeValue() извлекается значение заданного атрибута. Пример кода приведён ниже.

1. private String GetImage(HtmlNode html)
2. {
3. return html.QuerySelector("div.b-teaser\_\_top > div.b-teaser\_\_cover > img").GetAttributeValue("src", "");
4. }

После получения элементов происходит проверка этих элементов с помощью условного оператора if. Если элементы не пустые и правильные, то они передаются в конструктор для создания объекта класса Shop, который в последствии добавляется в список, где содержатся все магазины.

Все вышеописанные методы обрабатывают одну страницу сайта. Чтобы обработать следующие страницы, потребуется знать точное количество страниц, а потом передавать в параметр числа. Для того, чтобы узнать максимальное количество страниц, есть метод GetMaxPage().

Метод GetMaxPage() извлекает из элемента номер последней страницы сайта. Номер находится тэге «a», который хранится в пятом дочернем элементе списка со всеми номерами страниц. Далее с помощью css селектора происходит перемещение к этому элементу, а с помощью свойства InnerText извлекается текст тэга. Метод Trim() нужен для удаления лишних пробелов. Пример кода приведён ниже.

1. private Int32 GetMaxPage()
2. {
3. var htmlWeb = new HtmlWeb
4. {
5. OverrideEncoding = Encoding.UTF8
6. };
7. var document = htmlWeb.Load("https://letyshops.com/shops?page=1");
8. var html = document.DocumentNode;
9. string maxPage = html.QuerySelector("div.b-content.b-content--shops > ul > li:nth-child(5) > a").InnerText.Trim();
10. if (Int32.TryParse(maxPage, out int result))
11. {
12. return result;
13. }
14. return 30;
15. }

После определения максимального количества страниц, программа может полностью обойти страницы с магазинами на сайте и записать нужную информацию. Пример кода приведён ниже.

1. var maxPage = GetMaxPage();
2. Parallel.For(1, maxPage + 1, ParseElements);

# Алгоритм парсинга сайта MegaBonus

# Отбор библиотек для парсинга

Для парсинга сайтов были отобраны специальные библиотеки для трёх языков программирования: Java, C# и Python.

Для языка программирования Java использованы следующие библиотеки:

1. Jsoup;
2. Unirest.

Для языка программирования C# использованы следующие библиотеки:

1. HtmlAgilityPack;
2. Fizzler;
3. AngleSharp;
4. CsQuery;
5. RestSharp.

Для языка программирования Python использованы следующие библиотеки:

1. BeatifulSoup;
2. Request;

Также для Python был использован фреймворк Scrapy.

Помимо библиотек был использован Selenium – это набор различных программных бесплатных инструментов, каждый из которых имеет свой подход к поддержке автоматизации тестирования. Весь набор инструментов обеспечивает широкий набор функций тестирования, специально предназначенных для тестирования веб-приложений всех типов. Эти операции очень гибкие, что позволяет использовать множество опций для определения местоположения элементов пользовательского интерфейса. Одной из ключевых функций Selenium является поддержка выполнения тестов на нескольких платформах браузера.

В случае с парсингом сайтов Selenium предоставляет возможность полностью имитировать поведение пользователя на сайте, то есть переход по ссылкам, использование активных объектов, заполнение некоторых полей и так далее.

# Реализация на языке Java

Во время практики были реализованы следующие способы парсинга контента сайтов на языке Java: Jsoup, Selenium и Unirest.

### **Jsoup**

Jsoup – это библиотека для анализа документа HTML. Jsoup предоставляет API для получения и манипулирования данными взятых либо с помощью URL, либо из HTML. У этой библиотеки есть методы похожие на DOM, CSS и JQuery для того, чтобы получать и обрабатывать данные. (<https://o7planning.org/ru/10399/jsoup-java-html-parser-tutorial>)

Для парсинга сайта с помощью библиотеки Jsoup были использованы следующие методы:

(<https://jsoup.org/apidocs/org/jsoup/Connection.html>) (<https://jsoup.org/>)

* connect(String url) – в качестве входного аргумента требует строку в виде URL. Создает новый объект типа Connection для заданного URL, и используется для получения и анализа HTML-страницы;
* get() – выполняет GET запрос и анализирует результат. Возвращает объект Document;
* post() – выполняет POST запрос и анализирует результат. Возвращает объект Document;
* userAgent(String userAgent) – в качестве входного аргумента требует строку в виде агента пользователя, который выглядит следующим образом: «Mozilla/5.0 (Windows NT 6.2; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1667.0 Safari/537.36». Данный метод устанавливает заголовок запроса агента пользователя;
* parse(String html) – анализирует код HTML и возвращает объект Document;
* getElementsByClass(String className) – в качестве входного аргумента требует строку в виде наименования класса. Находит элементы, которые имеют этот данный класс и возвращает их. Тип возвращаемого значения Elements;
* getElementsByTag(String tagName) – в качестве входного аргумента требует строку в виде наименования html тэга. Находит элементы, которые имеют этот данный тэг и возвращает их. Тип возвращаемого значения Elements;
* attr(String key) – в качестве входного аргумента требует строку в виде наименования атрибута тэга. Возвращает содержимое атрибута;
* text() – возвращает текст внутри основного тэга и дочерних элементов;
* select(String cssQuery) – в качестве входного аргумента требует строку в виде css селектора. Возвращает элементы, которые соответствуют селектору.
* selectFirst(String cssQuery) – в качестве входного аргумента требует строку в виде css селектора. Возвращает первое соответствие заданному селектору. Аналогичный результат будет при select(String cssQuery).first(), но этот вариант менее эффективный, так как первый метод будет искать все совпадения, что будет затратно по времени, и только потом второй метод вернёт первый элемент.

(<https://jsoup.org/apidocs/org/jsoup/nodes/Element.html#select-java.lang.String->)

При построении логики парсинга сайта использовалась панель разработчика в браузере Google Chrome, чтобы не писать css селекторы или XPath самостоятельно. Использование возможностей панели разработчика в браузере Google Chrome изображено на Рисунке 5.

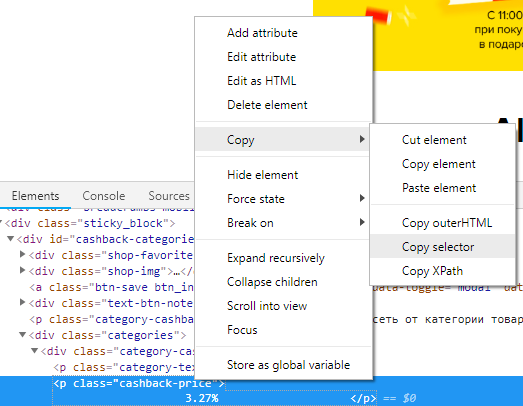


Рисунок 5 – использование возможностей панели разработчика в Google Chrome

Плюсы библиотеки Jsoup:

* является интуитивно понятной библиотекой, которой легко пользоваться;
* содержит множество методов, которые облегчают парсинг сайтов;
* присутствует имитация пользователя через userAgent;
* имеется возможность взаимодействия с cookies.

Минусы библиотеки Jsoup:

* с помощью библиотеки Jsoup можно парсить только статические страницы, то есть нельзя работать с динамически подключаемым контентом;
* Jsoup не поддерживает XPath. Но css селекторы являются альтернативой;
* среди методов, которые возвращают элементы по классу, тэгам и так далее нет методов для возвращения только первого элемента. Всегда происходит полный обход html. Приходится пользоваться методов first().

### **Selenium**

Для работы с Selenium нужно скачать вебдрайвер и подключить его через setProperty(String key, String value), где key – имя драйвера, то есть «webdriver.chrome.driver», а value – путь к самому драйверу. В данном проекте использовался chromedriver.exe. Также нужно подключить библиотеку selenium для взаимодействия с браузером.

Для парсинга сайта с помощью Selenium были использованы следующие методы:

* get(String URL) – в качестве входного аргумента требует строку в виде URL. С помощью этого метода происходит навигация по URL;
* navigate().to(String URL) - в качестве входного аргумента требует строку в виде URL. С помощью этого метода происходит навигация по URL. Разница этого метода и метода get(String URL) небольшая, но она есть. Используя navigate(), можно использовать такие методы, как back(), forward(), refresh();

(<https://www.oodlestechnologies.com/blogs/How-To-Navigate-A-URL-Using-Selenium-Web-Driver/>)

* findElement(By by) – в качестве параметра принимает объект типа By, который задает критерий поиска. Возвращает элемент типа WebElement, который соответствует заданному критерию;
* findElements(By by) - в качестве параметра принимает объект типа By, который задает критерий поиска. Возвращает список всех элементов типа WebElement, которые соответствуют заданному критерию;
* By – представляет из себя локатор, который уникально идентифицирует элемент страницы. Selenium предоставляет несколько способов использования локаторов для поиска элементов:

1. By.id – в качестве локатора используется атрибут id элемента страницы;
2. By.name – в качестве локатора используется атрибут name элемента страницы;
3. By.xpath – используется для поиска элемента по XPath выражению;
4. By.tagName – поиск по имени HTML тега;
5. By.className – поиск по классу элемента;
6. By.cssSelector – используется для поиска элемента по css селектору;
7. By.linkText – поиск ссылки с указанным текстом. Текст ссылки должен быть точным совпадением;
8. By.partialLinkText – поиск ссылки по части с указанным текстом.

* getAttribute(String s) – в качестве параметра принимает строку в виде наименования атрибута. Возвращает текстовое представление найденного атрибута.

Помимо методов для поиска элементов используются такие методы, как click(), который позволяет взаимодействовать с активными объектами такими как кнопки и ссылки и метод isDisplayed (), который проверяет отображение элемента.

(<https://www.guru99.com/find-element-selenium.html>)

(<http://internetka.in.ua/selenium-webdriver-findelement-by/>)

При построении логики парсинга сайта использовалась панель разработчика в браузере Google Chrome, чтобы не писать css селекторы или XPath самостоятельно, как в примере с Jsoup. Использование возможностей панели разработчика в браузере Google Chrome изображено на Рисунке 1.

Плюсы Selenium:

* имитация пользователя;
* методы для поиска как одного элемента, так и множества;
* поиск может осуществляться множествами локаторами.

Минусы Selenium:

* использует реальный браузер для работы, из-за чего скорость падает;
* необходимо скачивать дополнительные файлы.

### **Unirest**

Unirest - это набор облегченных HTTP-библиотек, доступных на нескольких языках, созданный и поддерживаемый Mashape, который также поддерживает API-интерфейс с открытым исходным кодом Gateway Kong.

(<http://unirest.io/java.html>)

Заголовки и параметры передаются с помощью API header() и fields(). Запрос выполняется при вызове метода asJson(). Кроме этого метода есть другие варианты, например asBinary(), asString() и asObject().

Чтобы передать несколько заголовков или полей, можно либо создать Map и передать их в .headers (<String, Object>) и .fields (<String, String>) соответственно, либо постепенно добавлять через .header(<String, Object>) и .field(<String, Object>) поле за полем.

Следующий код показывает, как с помощью запросов программа получает json с сайта [Kopikot.ru](https://www.kopikot.ru/), который в последствии обрабатывается.

Пример с использование Map:

1. HttpResponse<String> response = null;
2. Map<String, String> header = new HashMap<>();
3. header.put("User-Agent", "PostmanRuntime/7.11.0");
4. header.put("Accept", "\*/\*");
5. header.put("Cache-Control", "no-cache");
6. header.put("Postman-Token", "ac6a192d-72f5-4238-9321-55c1308e9846,09ec761f-7c53-4696-bf5a-dc44918e73c5");
7. header.put("Host", "d289b99uqa0t82.cloudfront.net");
8. header.put("accept-encoding", "gzip, deflate");
9. header.put("Connection", "keep-alive");
10. header.put("cache-control", "no-cache");
11. for (int i = 0; i <= 2000; i += 100) {
12. String url = "https://d289b99uqa0t82.cloudfront.net/sites/5/campaigns\_limit\_100\_offset\_" + i + "\_order\_popularity.json";
13. try {
14. response = Unirest.get(url)
15. .headers(header)
16. .asString();
17. } catch (UnirestException e) {
18. log.error(e + " : " + i);
19. break;
20. }

Пример без использования Map:

1. HttpResponse<String> response = null;
2. for (int i = 0; i <= 2000; i += 100) {
3. String url = "https://d289b99uqa0t82.cloudfront.net/sites/5/campaigns\_limit\_100\_offset\_" + i + "\_order\_popularity.json";
4. try {
5. response = Unirest.get(url)
6. .header("User-Agent", "PostmanRuntime/7.11.0")
7. .header("Accept", "\*/\*")
8. .header("Cache-Control", "no-cache")
9. .header("Postman-Token", "ac6a192d-72f5-4238-9321-55c1308e9846,09ec761f-7c53-4696-bf5a-dc44918e73c5")
10. .header("Host", "d289b99uqa0t82.cloudfront.net")
11. .header("accept-encoding", "gzip, deflate")
12. .header("Connection", "keep-alive")
13. .header("cache-control", "no-cache")
14. .asString();
15. } catch (UnirestException e) {
16. log.error(e + " : " + i);
17. break;
18. }
19. HttpResponse<String> finalResponse = response;
20. if (finalResponse != null) {
21. futures.add(pool.submit(() -> parseElements(finalResponse.getBody())));
22. }
23. }

(<https://www.codeflow.site/ru/article/unirest>)

Для упрощения работы с запросами использовалось бесплатное приложение Postman. Используя это приложение, можно отправлять запросы на сайты с нужными заголовками и полями. В примере выше можно заметить такие заголовки, как User-Agent, Cache-Control и прочие. Некоторые заголовки Postman ставит автоматически.

Как можно заметить из примера сверху, URL, который передаётся в запрос отличается от URL самого сайта. Такие различия объясняются тем, у запрос должен передаваться не по адресу самого сайта, а другому, который будет указан в панели разработчика. При отправке запроса на адрес самого магазина в ответе будет не html код страницы, а javaScript страницы.

Чтобы узнать URL, по которому передаются данные, нужно воспользоваться панелью разработчика и найти XHR – XMLHttpRequest, который отвечает за добавление данных на страницу. В случае с сайтом [Kopikot.ru](https://www.kopikot.ru/), запрос будет отправлен при нажатии кнопки «Больше». После этого в панели разработчика в разделе Network будет нужный XHR. Результат изображён на Рисунке 2.

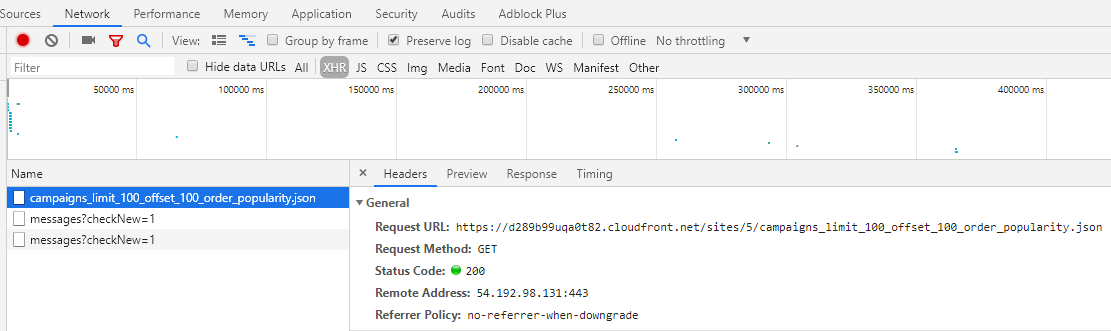


Рисунок 6 – XHR для 100 магазинов с сайта [Kopikot.ru](https://www.kopikot.ru/)

Как видно на Рисунке 2, в разделе Headers указан Request URL. Так как при отображении магазинов было выбрано показывать по 100 магазинов, в URL видно это число: «offset\_100». При нажатии кнопки ещё раз в Network появляется ещё один XHR, но уже с числом 200. Кроме самого URL важно какой метод использует запрос: GET или POST. Посмотреть это можно в «Request Method» Результат виден на Рисунке 3.

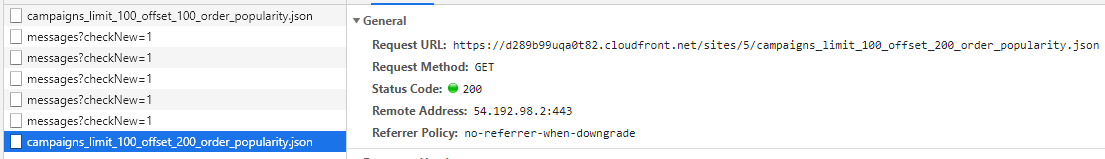


Рисунок 7 – XHR для 200 магазинов с сайта Kopikot.ru

Практически было выяснено, что таких XHR всего 14 штук: от 0 до 13.

Используя программу Postman, через этот URL отправляется запрос. В качестве ответа в разделе Body появляется JSON файл с нужной информацией. Результат запроса на Рисунок 4.

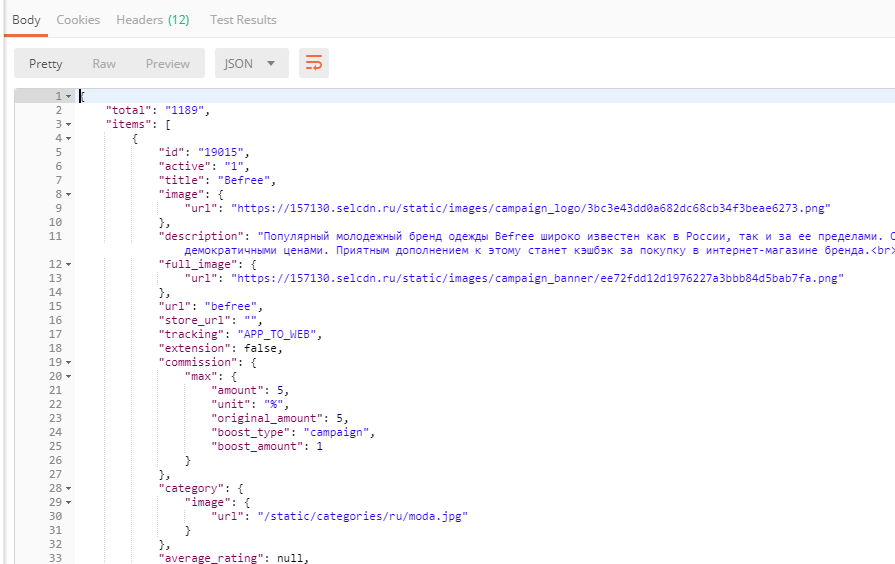


Рисунок 8 – результат запроса к URL через Postman

При просмотре какие запросы происходят на сервере важно обращать внимание на ту информацию, которая показана в Headers. Пример изображён на Рисунок 5.

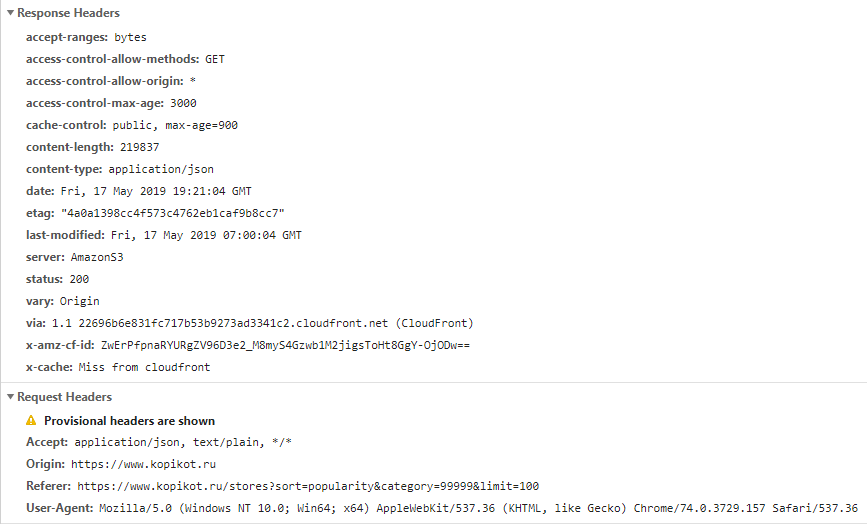


Рисунок 9 – Headers сайта Kopikot.ru

На этом рисунке можно увидеть, например, в каком формате будет ответ на запрос – поле content-type.

Также имеются случаи, когда при запросе передаются некоторые параметры. Для примера приведён Headers сайта [Cash4Brands.ru](https://cash4brands.ru/) на Рисунок 6 и Рисунок 7.

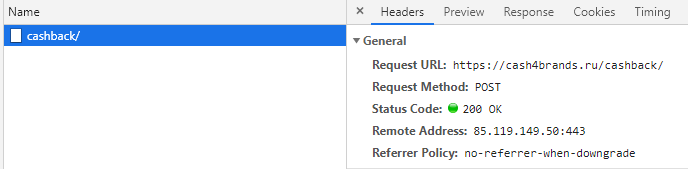


Рисунок 10 – Headers сайта Cash4Brands.ru часть 1

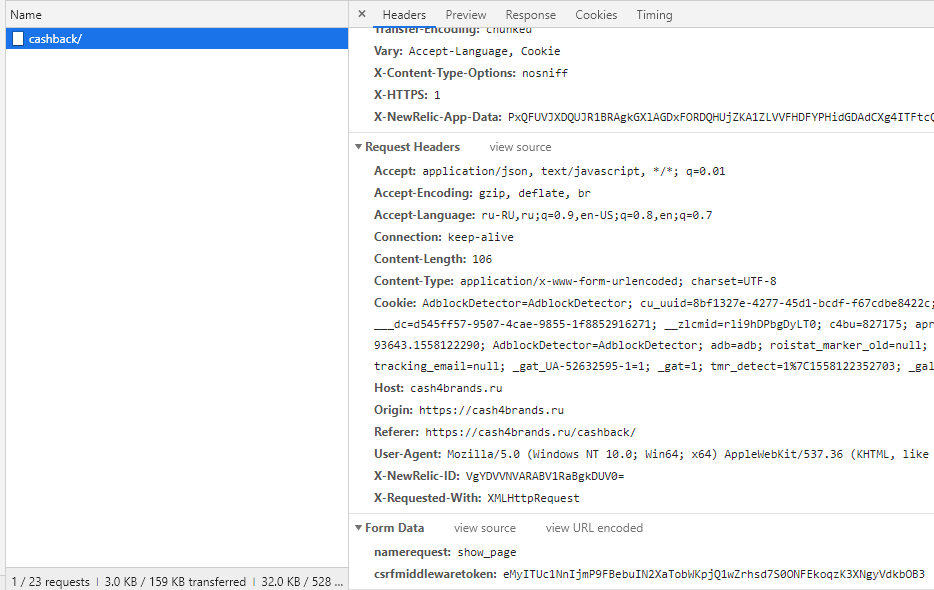


Рисунок 11 – Headers сайта Cash4Brands.ru часть 2

На Рисунке 7 в «Form Data» можно заметить 2 параметра, которые передаются: namerequest и csrfmiddlewaretoken. В данном случае нужен лишь первый параметр, так как второй отвечает за cookies. У поля с именем csrfmiddlewaretoken значением является хэш для идентификатора сессии и секретный ключ. Поле namerequest и его значение show\_page надо добавить в Body. Пример приведён на Рисунок 8.( <https://djbook.ru/ch14s05.html>)

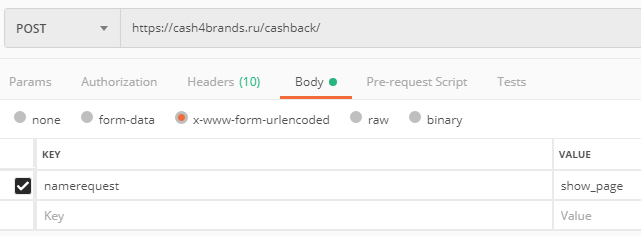


Рисунок 12 – параметры для запроса на Cash4Brands.ru

Результатом данного запроса является JSON, который в дальнейшем обрабатывается.

Программа Postman предоставляет возможность сгенерировать код для запроса. Для этого нужно нажать кнопку Code. В окне можно выбрать язык программирования и какие библиотеки использовать. Пример изображен на Рисунок 9.

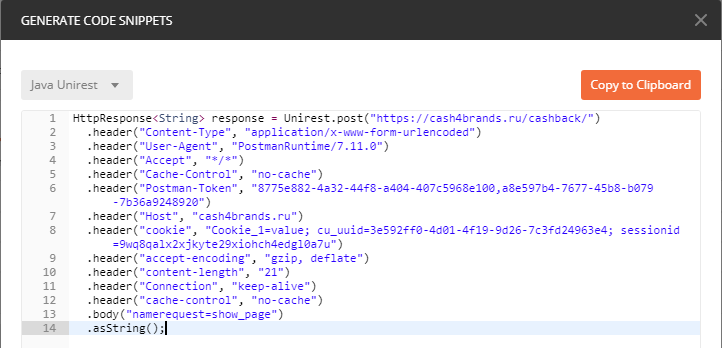


Рисунок 13 – генерация Unirest кода в Postman

После получения JSON с запроса, он обрабатывается с помощью парсера для JSON. Разбор JSON осуществляется с помощью библиотеки JsonPath. Для примера взят код для обработки данных с Kopikot.ru.

List<Object> items = JsonPath.read(response, "$..items[\*]");

В этой строчке в список items записываются все элементы, которые содержатся в тэге «items». Далее элементы этого списка передаются методам, которые извлекают из этих элементов нужную информацию.

String label = JsonPath.read(item, "$.commission.max.unit");

В этой строчке кода из метода, который извлекает валюту кэшбэка. В данному случае это процент. Через точку происходит обращение к последующим элементам для извлечения информации.

Плюсы Unirest:

* быстрая скорость выполнения.

Минусы Unirest:

* поиск нужных запросов.

# Реализация на языке C#

Во время практики были реализованы следующие способы парсинга контента сайтов на языке C#: HtmlAgilityPack, Fizzler, AngleSharp, CsQuery, RestSharp, Selenium.

### **HtmlAgilityPack**

HtmlAgilityPack – это HTML-анализатор, написанный на C# для чтения и записи DOM, и поддерживает обычный XPATH или XSLT – eXtensible Stylesheet Language Transformations.

(<https://habr.com/ru/post/273807/> , <https://habr.com/ru/post/112325/> , <https://html-agility-pack.net/documentation> )

Для парсинга сайта с помощью парсера HtmlAgilityPack были использованы следующие методы:

* SelectNodes(string xpath) – в качестве входного аргумента требует строку в виде XPath выражения. Возвращает HtmlAgilityPack.HtmlNodeCollection, которая содержит коллекцию узлов, соответствующих запросу XPath выражения, или ноль, если ни один узел не соответствует XPath выражению. Метод выбирает список узлов, соответствующих XPath выражению;
* SelectSingleNode(string xpath) – в качестве входного аргумента требует строку в виде XPath выражения. Возвращает первый HtmlAgilityPack.HtmlNode, соответствующий запросу XPath или нулевой ссылке, если не найдено ни одного соответствующего узла. Метод выбирает первый HtmlNode, соответствующий XPath выражению;
* Load(string url) – в качестве входного аргумента требует строку в виде URL. Возвращает объект класса HtmlDocument, который в дальнейшем обрабатывается. Метод получает HTML-документ из интернет-ресурса;
* GetAttributeValue(string name, string def) – в качестве первого входного аргумента требует строку в виде имени атрибута, второго аргумента – строку в виде значения, которое будет возвращено если не будет найден атрибут. Возвращает строку в виде значения атрибута. Метод ищет совпадение по названию атрибута в старшем элементе. Если совпадений нет, то вернётся второй аргумент.

Помимо методов используется свойство InnerText для извлечения текста между начальным и конечным тегами объекта.

Плюсы HtmlAgilityPack:

* средняя скорость выполнения.

Минусы:

* работает только с XPath выражениями.

### **Fizzler**

Fizzler – Библиотека .NET для выбора элементов из дерева узлов на основе css селектора CSS. Реализация по умолчанию основана на HTMLAgilityPack и выбирает из документов HTML. (<https://code.google.com/archive/p/fizzler/>).

Для парсинга сайта с помощью библиотеки Fizzler были использованы следующие методы:

* Load(string url) – в качестве входного аргумента требует строку в виде URL. Возвращает объект класса HtmlDocument, который в дальнейшем обрабатывается. Метод получает HTML-документ из интернет-ресурса;
* QuerySelector(string selector) – в качестве входного аргумента требует строку в виде css селектора. Возвращает элемент типа HtmlNode. Метод осуществляет поиск элемента по селектору.
* QuerySelectorAll(string selector) – в качестве входного аргумента требует строку в виде css селектора. Возвращает список элементов типа HtmlNode. Метод осуществляет поиск элементов по селектору.
* GetAttributeValue(string name, string def) – в качестве первого входного аргумента требует строку в виде имени атрибута, второго аргумента – строку в виде значения, которое будет возвращено если не будет найден атрибут. Возвращает строку в виде значения атрибута. Метод ищет совпадение по названию атрибута в старшем элементе. Если совпадений нет, то вернётся второй аргумент.

Помимо методов используется свойство InnerText для извлечения текста между начальным и конечным тегами объекта.

Плюсы Fizzler:

* быстрая скорость выполнения;
* имеет все те же методы, что и HtmlAgilitypack;
* работает с css селекторами.

Минусы Fizzler:

* работает только с XPath выражениями и css селекторами.

### **AngleSharp**

AngleSharp – это библиотека .NET, которая позволяет анализировать гипертексты такие как HTML, SVG и MathML. XML также поддерживается библиотекой. Важным аспектом AngleSharp является то, что CSS также может быть проанализирован. Включенный анализатор построен на основе официальной спецификации W3C – World Wide Web Consortium. Также присутствуют стандартные функции DOM, такие как querySelector или querySelectorAll.

(<https://github.com/AngleSharp/AngleSharp>)

Для парсинга сайта с помощью библиотеки AngleSharp были использованы следующие методы:

* ParseDocument(string source) – в качестве входного аргумента требует строку в виде html. Метод парсит переданный аргумент и возвращает результат.
* GetElementsByClassName(string classNames) – в качестве входного аргумента требует строку в виде имени класса. Метод возвращает список уникальных элементов, которые имели класс с именем, которое было передано в виде строки.
* GetElementsByTagName(string tagName) – в качестве входного аргумента требует строку в виде имени тэга. Метод возвращает список уникальных элементов, которые имели тэг с именем, которое было передано в виде строки.
* GetAttribute(string name) – в качестве входного аргумента требует строку в виде имени атрибута. Метод возвращает значение искомого атрибута, либо null, если атрибут не найден.
* QuerySelectorAll(string selectors) – в качестве входного аргумента требует строку в виде css селектора. Возвращает элемент типа IElement. Метод осуществляет поиск элемента по селектору.
* QuerySelectorAll(string selector) – в качестве входного аргумента требует строку в виде css селектора. Возвращает список элементов типа IElement. Метод осуществляет поиск элементов по селектору.

Помимо методов используется свойство InnerText для извлечения текста между начальным и конечным тегами объекта и свойство InnerHtml для извлечения html кода элемента.

Для получения HTML используется класс WebClient, с помощью которого происходит подключение к страницам сайта.

1. WebClient webClient = new WebClient
2. {
3. Encoding = Encoding.UTF8
4. };
5. string page = "https://letyshops.com/shops?page=" + i;
6. string html = webClient.DownloadString(page);
7. HtmlParser parser = new HtmlParser();
8. var result = parser.ParseDocument(html).GetElementsByClassName("b-teaser");

Плюсы AngleSharp:

* быстрая скорость выполнения;
* при отсутствии элемента возвращает null, а не выбрасывает ошибку.
* работает с css селекторами

Минусы AngleShrap:

* выбор по локаторам возвращает только коллекцию

### **CsQuery**

CsQuery – это порт jQuery для .NET 4. Он реализует все css селекторы CSS2 и CSS3, все методы манипулирования DOM в jQuery и некоторые служебные методы. (<https://github.com/jamietre/CsQuery>)

Для парсинга сайта с помощью CsQuery были использованы следующие методы:

* CreateFromUrl(string url) – в качестве входного аргумента требует строку в виде URL. Возвращает HTML. Метод создаёт новое DOM дерево из файла html.
* GetAttribute(string name) – в качестве входного аргумента требует строку в виде имени атрибута. Метод ищет атрибут с заданным именем и возвращает его значение.
* CreateDocument(string html) – в качестве входного аргумента требует строку в виде html. Метод создаёт новое DOM дерево из переданного html.

Помимо методов используется свойство Attributes для создания коллекции из атрибутов текущего элемента и индексатор CQ this[string selector] { get; } для доступа к элементам через css селектор.

Плюсы CsQuery:

* хорошая скорость выполнения;
* есть возможность парсинга json;
* работает с css селекторами;
* подключение к сайту осуществляется без дополнительных библиотек.

Минусы CsQuery:

* работает только с css селекторами;
* для получения значения атрибута требуется создать список всех атрибутов.

### **RestSharp**

RestSharp – простой REST и HTTP API клиент для .NET. С помощью него осуществляется построение запросов на сервер.

RestSharp является аналогичной библиотекой для создания запросов, как Unirest в языке программирования Java и Request в языке программирования Python.

Сначала надо создать объект класса RestClient – client, передав в качестве параметра строку в виде URL. Далее создаётся запрос как объект класса RestRequest – request, передав в качестве параметра метод запроса, в данном случае POST. Заголовки и параметры передаются с помощью методов AddHeader() и AddParameter(). Запрос выполняется после того, как вызовется метод класса RestClient Execute(), в который передаётся request. Метод вернёт ответ с сервера в виде переменной типа IRestResponse, из которой в дальнейшем можно извлечь информацию.

Чтобы передать несколько заголовков можно создать Dictionary из значений, либо передавать их последовательно

Следующий код показывает, как с помощью запросов программа получает json с сайта [Cash4Brands.ru](https://www.kopikot.ru/), который в последствии обрабатывается.

Пример с использование Dictionary:

1. var client = new RestClient("https://cash4brands.ru/cashback/");
2. var request = new RestRequest(Method.POST);
3. Dictionary<String, String> headers = new Dictionary<string, string>()
4. {
5. { "cache-control", "no-cache"},
6. { "Connection", "keep-alive"},
7. { "content-length", "21"},
8. { "accept-encoding", "gzip, deflate"},
9. { "cookie", "Cookie\_1=value; cu\_uuid=3e592ff0-4d01-4f19-9d26-7c3fd24963e4; sessionid=9wq8qalx2xjkyte29xiohch4edgl0a7u"},
10. { "Host", "cash4brands.ru"},
11. { "Postman-Token", "91751f4f-c0e5-48c2-9137-cbcbc5e11a0f,1e684343-5161-4efc-8482-495ef99d971b"},
12. { "Cache-Control", "no-cache" },
13. { "Accept", "\*/\*"},
14. { "User-Agent", "PostmanRuntime/7.13.0"},
15. { "Content-Type", "application/x-www-form-urlencoded"}
16. };
17. foreach (var item in headers)
18. {
19. request.AddHeader(item.Key, item.Value);
20. }
21. request.AddParameter("undefined", "namerequest=show\_page", ParameterType.RequestBody);
22. IRestResponse response = client.Execute(request);

Пример без использования Map:

1. var client = new RestClient("https://cash4brands.ru/cashback/");
2. var request = new RestRequest(Method.POST);
3. request.AddHeader("cache-control", "no-cache");
4. request.AddHeader("Connection", "keep-alive");
5. request.AddHeader("content-length", "21");
6. request.AddHeader("accept-encoding", "gzip, deflate");
7. request.AddHeader("cookie", "Cookie\_1=value; cu\_uuid=3e592ff0-4d01-4f19-9d26-7c3fd24963e4; sessionid=9wq8qalx2xjkyte29xiohch4edgl0a7u");
8. request.AddHeader("Host", "cash4brands.ru");
9. request.AddHeader("Postman-Token", "91751f4f-c0e5-48c2-9137-cbcbc5e11a0f,1e684343-5161-4efc-8482-495ef99d971b");
10. request.AddHeader("Cache-Control", "no-cache");
11. request.AddHeader("Accept", "\*/\*");
12. request.AddHeader("User-Agent", "PostmanRuntime/7.13.0");
13. request.AddHeader("Content-Type", "application/x-www-form-urlencoded");
14. request.AddParameter("undefined", "namerequest=show\_page", ParameterType.RequestBody);
15. IRestResponse response = client.Execute(request);

Также, как и с Unirest, для упрощения построения запроса используется бесплатная программа Postman. Использовались аналогичные параметры и заголовки. Для языка C# в Postman имеется только RestSharp. Пример кода из Postman изображён на Рисунке 10.

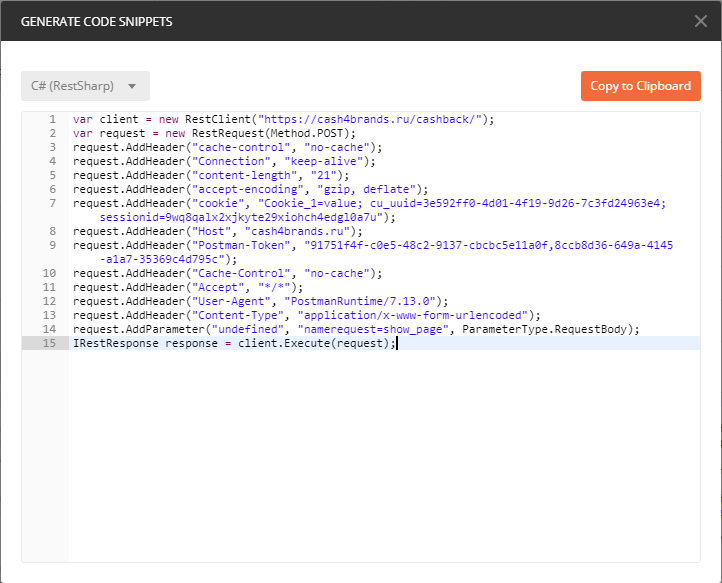


Рисунок 14 – генерация RestSharp кода в Postman

После получения JSON с запроса, он обрабатывается с помощью парсера для JSON. Разбор JSON осуществляется с помощью библиотеки Json.NET. Для примера взят код для обработки данных с Kopikot.ru.

(<https://github.com/JamesNK/Newtonsoft.Json> , <https://www.newtonsoft.com/json>)

Для начала работы с парсером JSON нужно создать переменную тпиа JObject, используя статический метод класса JObject Parse(string json), который принимает строку в виде json. Ниже приведён пример извлечения из ответа с сервера json.

JObject jsonParse = JObject.Parse(response.Content);

Переход по элементам осуществляется с помощью индексатора JToken this[object key]. В качестве переменной выступает строка, в которой содержится имя тэга. Ниже приведён примеры взятия всех дочерних элементов тэга и взятия отдельного элемента в виде валюты кэшбэка.

Пример 1:

var listOfItems = jsonParse["items"];

Пример 2:

token["commission"]["max"]["unit"].ToString();

Плюсы RestSharp:

* быстрая скорость выполнения.

Минусы RestSharp:

* поиск нужных запросов.

### **Selenium**

Для работы с Selenium на платформе .Net не нужно скачивать вебдрайвер, как в случаях с Java и Python. Но нужно подключить библиотеку selenium для взаимодействия с браузером.

Для парсинга сайта с помощью Selenium были использованы следующие методы:

* Navigate().GoToUrl(String URL) - в качестве входного аргумента требует строку в виде URL. С помощью этого метода происходит навигация по URL. Используя Navigate(), можно использовать такие дополнительно такие методы, как Back(), Forward(), Refresh();

(<https://www.oodlestechnologies.com/blogs/How-To-Navigate-A-URL-Using-Selenium-Web-Driver/>)

* FindElement(By by) – в качестве параметра принимает объект типа By, который задает критерий поиска. Возвращает элемент типа IWebElement, который соответствует заданному критерию;
* FindElements(By by) - в качестве параметра принимает объект типа By, который задает критерий поиска. Возвращает список всех элементов типа IWebElement, которые соответствуют заданному критерию;
* By – представляет из себя локатор, который уникально идентифицирует элемент страницы. Selenium предоставляет несколько способов использования локаторов для поиска элементов:

1. By.Id – в качестве локатора используется атрибут id элемента страницы;
2. By.Name – в качестве локатора используется атрибут name элемента страницы;
3. By.Xpath – используется для поиска элемента по XPath выражению;
4. By.TagName – поиск по имени HTML тега;
5. By.ClassName – поиск по классу элемента;
6. By.CssSelector – используется для поиска элемента по css селектору;
7. By.LinkText – поиск ссылки с указанным текстом. Текст ссылки должен быть точным совпадением;
8. By. PartialLinkText – поиск ссылки по части с указанным текстом.

* GetAttribute(String s) –в качестве параметра принимает строку в виде наименования атрибута. Возвращает текстовое представление найденного атрибута.

Помимо методов для поиска элементов используются такой метод, как Click(), который позволяет взаимодействовать с активными объектами такими как кнопки и ссылки и свойство Displayed, которое возвращает true если элемент на экране.

(<https://www.guru99.com/find-element-selenium.html>)

(<http://internetka.in.ua/selenium-webdriver-findelement-by/>)

(<https://www.seleniumhq.org/docs/01_introducing_selenium.jsp>)

При построении логики парсинга сайта использовалась панель разработчика в браузере Google Chrome, чтобы не писать css селекторы или XPath самостоятельно, как в примере с Jsoup. Использование возможностей панели разработчика в браузере Google Chrome изображено на Рисунке 1.

Плюсы Selenium:

* имитация пользователя;
* методы для поиска как одного элемента, так и множества;
* поиск может осуществляться множествами локаторами;
* не нужно скачивать дополнительные файлы.

Минусы Selenium:

* использует реальный браузер для работы, из-за чего скорость падает.

# Реализация на языке Python

Во время практики были реализованы следующие способы парсинга контента сайтов на языке Python: BeautifulSoup, Request, Scrapy.

### **BeautifulSoup**

Библиотека BeautifulSoup является самой популярной библиотекой для веб-скрапинга. Кроме того, эта библиотека является самой доступной для понимания и для работы.

Своё имя библиотека BeautifulSoup получила в честь одноимённого стихотворения Черепахи Квази из «Приключения Алисы в Стране чудес».

Данная библиотека не является библиотекой языка Python по умолчанию, поэтому её нужно установить. Если использовать среду разработки PyCharm от компании JetBrains, то установку можно выполнить через настройки, найдя библиотеку bs4. Также можно использовать «pip install», прописав в консоли pip install beautifulsoup4. Но если «pip install» не установлен, то его тоже надо установить, также прописав в консоли следующую строку: «easy\_install pip». Также во время установки следует учитывать, что если у вас стоят и Python2, и Python3, то следует указать под какую версию следует установить то, что потребуется. Это осуществляется с помощью добавления цифры 3 в pip: «pip3 install». После установки BeautifulSoup можно импортировать в код, как обычную библиотеку.

Так как BeutifulSoup не может самостоятельно подключаться к сайтам и считывать html, для работы с веб-скрапингом нужно из библиотеки urllib.request импортировать метод urlopen. Этот метод создает файлоподобный объект, который читается методом read(). Объект в виде html передаётся в конструктор BeautifulSoup для создания объекта класса, с которым можно будет в дальнейшем работать.

BeautifulSoup может использовать различные HTML-парсеры , каждый из которых имеет свои преимущества и недостатки. В данном примере будет использован парсер lxml в качестве анализатора.

(<https://pythonworld.ru/osnovy/pip.html>,<https://www.crummy.com/software/BeautifulSoup/bs4/doc/> , <https://lxml.de/elementsoup.html>)

Для парсинга сайта с помощью BeautifulSoup были использованы следующие методы:

1. findAll(name=None, attrs={}, recursive=True, text=None, limit=None,\*\*kwargs) – метод для поиска элементов. Он извлекает список объектов Tag, соответствующих заданному критерию.

Аргумент name является наиболее важным. Этот аргумент ограничивает набор имён тегов. Аргумент kwargs налагает ограничения на атрибуты тега. Также, как и с атрибутом name, можно передать различные значения. Аргумент text позволяет находить объекты Tag. Его значением может быть строка или регулярное выражение, или список, или словарь, или специальные значения: True и None, или вызываемый объект. Логический аргумент recursive, который по умолчанию равен True, сообщает BeautifulSoup о том, нужно ли обходить всё поддерево или искать лишь среди непосредственных потомков объекта Tag или объекта парсера.

Ниже приведён пример использования метода find\_all().

shops = soup.find\_all('div', class\_='b-teaser')

В результате в переменной shops будут элементы, которые соответствуют заданным критериям, то есть тэг «div» с классом «b-teaser».

1. find(name, attrs, recursive, text, \*\*kwargs) – метод для поиска элемента. Он возвращает только первый дочерний элемент этого тега, соответствующий заданному критерию.

Ниже приведён пример использования метода find().

label = shop.find('span', class\_='b-shop-teaser\_\_label ')

В результате в переменной label будет элемент, которые соответствует заданному критерию, то есть тэг «span» с классом «b-shop-teaser\_\_label ».

В качестве именованного аргумента нельзя использовать зарезервированные слова, такие как name, for и так далее. Чтобы решить эту проблему, можно использовать attrs, если необходимо наложить ограничения на атрибуты, имена которых совпадают с зарезервированными словами, либо добавлять нижнее подчёркивание, как приведено в примерах.

get(name=None) – в качестве входного аргумента требует строку в виде имени аргумента. Возвращает значение атрибута. Если такого атрибута нет, то выбрасывает ошибка: AttributeError: 'NoneType' object has no attribute 'get'. Это метод принадлежит библиотеке lxml.

Ниже приведён пример использования метода get().

1. image = shop.find('div', class\_='b-teaser\_\_cover').find('img').get('src')

В результате в переменной image будет ссылка на картинку магазина.

Плюсы BeautifulSoup:

* может использовать другие парсеры;
* быстрая скорость выполнения.

Минусы BeautifulSoup:

* не работает с XPath выражениями и css селекторами, но эта проблема решается добавлением lxml.

### **Scrapy**

Один из видов программ, которые написаны специально для скрапинга, называются Пауками. Для них существует отдельный фреймворк Scrapy.

Scrapy – один из наиболее популярных и производительных фреймворков Python для получения данных с веб-страниц, которая включает в себя большинство общих функциональных возможностей. Это значит, что многие функции уже заложены в Scrapy. (<https://www.8host.com/blog/web-scraping-s-pomoshhyu-scrapy-i-python-3/>)

Фреймворк Scrapy можно установить в PyCharm через настройки.

Для создания проекта в консоли нужно прописать «scrapy startproject project\_name», где вместо project\_name нужно написать имя проекта. После этого в директории создастся проект с определённой файловой структурой следующего типа:

orphanage/

scrapy.cfg

orphanage/

\_\_init\_\_.py

items.py

pipelines.py

settings.py

spiders/

\_\_init\_\_.py

...

В данной файловой структуре имеются следующие элементы:

scrapy.cfg – настройки проекта;

orphanage/ – Python модуль проекта;

orphanage/items.py – классы, описывающие модель собираемых данных;

orphanage/pipelines.py – используется в основном для описания пользовательских форматов сохранения результатов парсинга;

orphanage/settings.py – пользовательские настройки паука;

orphanage/spiders/ – директория, в которой хранятся файлы с классами пауков. Каждого паука принято писать в отдельном файле.

В файл items.py описываются атрибуты, которые будут изыматься с сайтов. Например,

class Shop(scrapy.Item):

# define the fields for your item here like:

name = scrapy.Field()

url = scrapy.Field()

discount = scrapy.Field()

label = scrapy.Field()

image = scrapy.Field()

В этом примере описываются аргументы, которые будут соответствовать объектам, находящиеся на сайте.

В папке spyders хранится класс с самим пауком. Для примера, ниже приводится класс паука

class ArcadySpider(scrapy.Spider):

name = "arcady"

address = "https://letyshops.com/shops?page="

clear\_address = 'https://letyshops.com'

allowed\_domains = ['https://letyshops.com']

start\_urls = []

max\_page = get\_max\_page()

for i in range(1, max\_page + 1):

start\_urls.append(address + i.\_\_str\_\_())

rules = (

Rule(LinkExtractor(allow=('')), callback="parse", follow=False)

)

def parse(self, response):

shops = response.xpath('//div[@class="b-teaser"]')

for i, shop in enumerate(shops):

item = {

'name': self.get\_name(shop, i),

'discount': self.get\_discount(shop, i),

'label': self.get\_label(shop, i),

'image': self.get\_image(shop, i),

'url': self.get\_url(shop, i)

}

items.append(item)

return items

Полный код приведён в Приложении Л.

Основным методом является parse(). Он с отвечает за обращение к нужным url и вызов определённых методов. Также parse() является методом по умолчанию.

В переменную start\_urls заносятся URL’ы, которые будут обходиться пауком.

Переменная rules – список правил обхода ресурса. Правило передаёт ссылки из start\_urls методу parse. Аргумент callback отвечает за метод, которому будут переданы данные. Аргумент follow отвечает за извлечение ссылок.

(<http://pythonr.blogspot.com/2015/01/scrapy-linkextractor-rules.html>)

Запуск осуществляется с помощью консольной команды «scrapy crawl [имя паука], либо с помощью создания объекта класса CrawlerProcess, с последующим вызовом метода crawl(), в который нужно передать имя класса, в котором реализован паук. После этого запустить методом start(). Ниже представлен пример кода.

process = CrawlerProcess()

process.crawl(ArcadySpider)

process.start()

Для того, чтобы данные записывались в нужный файл и с нужным форматом надо дописать специальные команды: «scrapy crawl [имя паука] -o scraped\_data.csv -t csv». В этом примере данные будут сохраняться в файле с именем «scraped\_data.csv» в формате csv.

В Scrapy используется собственный механизм извлечения данных, который основан так же, как и lxml на libxml2, из HTML-документов – селекторы или selectors. Фактически, селекторы – это отдельные классы, при создании экземпляров которых, на вход передаётся объект класса Response, представляющий собой ответ сервера.

(<http://gis-lab.info/qa/scrapy.html> , <https://doc.scrapy.org/en/latest/intro/tutorial.html> , <http://setmefirst.ru/blog/web_scraping_chto_eto_takoe_i_s_chem_ego_edyat> ,

<http://pi-code.blogspot.ru/2008/12/1.html> , <https://tyvik.ru/posts/parsing-scrapy/>)

Плюсы Scrapy:

* быстрая скорость выполнения;
* имеет множество настроек, например, переход по ссылкам и ограничение домена;
* работает с XPath выражениями и css селекторами;
* вывод результатов в файлы.

Минусы Scrapy:

* создание приложение из консоли;
* не может искать файлы по локаторам.

### **Request**

Библиотека Request – библиотека для выполнения запросов к серверу и обработки ответов на языке программирования Python. Данная библиотека является основной для веб-скрапинга страниц сайтов. Пользуясь данной библиотекой, можно получить содержимое страницы в виде html-кода для дальнейшего веб-скрапинга.

(<http://docs.python-requests.org/en/master/user/quickstart/>

<http://lecturesnet.readthedocs.io/net/requests/python.html#python>)

Request является аналогичной библиотекой для создания запросов, как Unirest в языке программирования Java и RestSharp в языке программирования C#

Ниже приведён пример запроса с помощью Request на Python.

def get\_json(self, i):

url= "https://d289b99uqa0t82.cloudfront.net/sites/5/campaigns\_limit\_100\_offset\_" + str(

i) + "\_order\_popularity.json"

payload = ""

headers = {

'User-Agent': "PostmanRuntime/7.11.0",

'Accept': "\*/\*",

'Cache-Control': "no-cache",

'Postman-Token': "b6eeb7b4-63dd-454e-b213-6d2d62b74946,1e851911-db3b-4406-88de-5ffc9ecbaa5d",

'Host': "d289b99uqa0t82.cloudfront.net",

'accept-encoding': "gzip, deflate",

'Connection': "keep-alive",

'cache-control': "no-cache"

}

response = requests.request("GET", url, data=payload, headers=headers)

data = json.loads(response.text)

items = data["items"]

return items

Заголовки в HTTP-запрос добавляются по типу словаря в headers. В payload хранятся параметры, которые перечисляются через знак амперсанд «&», либо, как и headers, то есть в виде словаря. Пример с параметрами через амперсанд изображён на Рисунке 11.

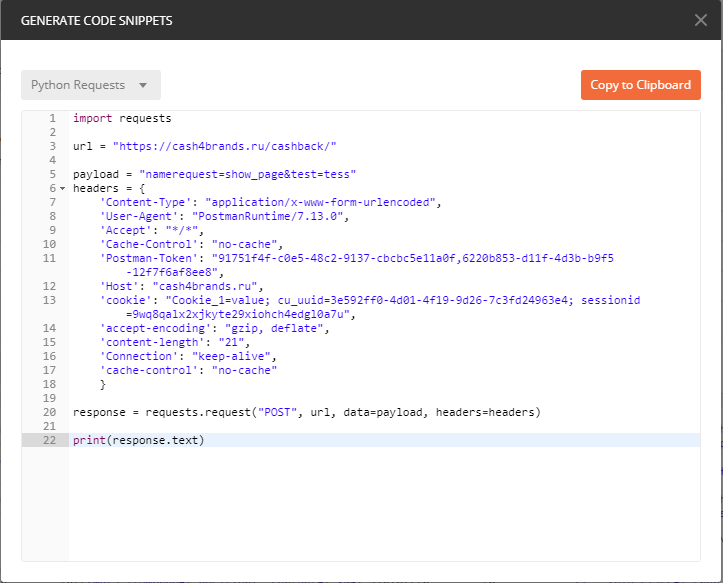


Рисунок 15 – генерация Request кода в Postman

Для отправки запроса нужно вызвать метод request(), в который передаются имя запроса – «GET», адрес сайта – url, параметры – payload и заголовки – headers.

После получения запроса, он обрабатывается с помощью встроенной библиотеки json. Ниже приведён пример работы библиотеки для парсинга json.

data = json.loads(response.text)

items = data["items"]

В примере видно, что json загружается с помощью метода loads, в который передаётся результат запроса в виде текста. Чтобы получить данные, нужно обратиться к тэгу, как к ключу. Аналогичные действия надо сделать, используя библиотеку Json NET.

Плюсы Request:

* быстрая скорость выполнения;
* удобное добавление заголовков и параметров.

Минус Request:

* поиск нужных запросов.

(<https://rtfm.co.ua/python-modul-requests/>)

### **Selenium**

Для работы с Selenium нужно скачать вебдрайвер. При создании объекта для работы с Selenium, в качестве аргумента конструктора нужно указать путь к вебдрайверу. Ниже приведён пример.

\_\_file\_path\_to\_Chrome = "E:\Документы\PyCharmProject\Parsing-on-python\chromedriver\_win32\chromedriver.exe"

driver = webdriver.Chrome(self.\_\_file\_path\_to\_Chrome)

В данном проекте использовался chromedriver.exe. Также нужно подключить библиотеку selenium для взаимодействия с браузером.

Для парсинга сайта с помощью Selenium были использованы следующие методы:

* get(url) – в качестве входного аргумента требует строку в виде URL. С помощью этого метода происходит навигация по URL;
* find\_element\_by\_class\_name(className) – в качестве параметра требует строку в виде имени класса. Возвращает список всех элементов, которые соответствуют заданному критерию;
* get\_attribute(attr) – в качестве параметра требует строку в виде имени аргумента. Возвращает значение аргумента, если он найден, либо выбрасывает ошибку.

Помимо использованных методов, есть все те методы, которые поддерживает selenium, то есть поиск элементов по локаторам. При этом можно выбрать метод с использованием By, либо без него.

При построении логики парсинга сайта использовалась панель разработчика в браузере Google Chrome, чтобы не писать css селекторы или XPath самостоятельно, как в примере с Jsoup. Использование возможностей панели разработчика в браузере Google Chrome изображено на Рисунке 1.

Так как Selenium использует реальный браузер, удобнее было бы распараллелить программу для более быстрого парсинга. Для этого на Python есть библиотека multiprocessing. Но возникает проблема, связанная с типом данных найденных элементов. Они являются несереализуемыми из-за чего библиотека multiprocessing не может с ними работать. Для решения данной проблемы пришлось до распараллеливания найти нужные элементы с помощью методов Selenium, но дальше передать их библиотеке BeautifulSoup. При попытке распараллеливания полученных и переконвентированных в другой формат элементов снова возникает ошибка – превышена максимальная глубина рекурсии. Эта ошибка появляется из-за того, что в данных обнаружились некоторые ссылки, которые использовала библиотека multiprocessing. Из-за этой ошибки дальнейшее распараллеливание было не возможным. Есть 2 способа решить данную проблему.

Первый способ заключается в том, что надо повысить максимальную глубину рекурсии с помощью метода sys.setrecursionlimit, который принимает на вход число. Это число будет максимально допустимой глубиной рекурсии. Но при больших данных этот способ не решает проблему.

Второй способ заключается в том, что надо привести все элементы к строке, а затем передавать их методу, где они снова будут приведены к начальному формату.

Плюсы Selenium:

* имитация пользователя;
* методы для поиска как одного элемента, так и множества;
* поиск может осуществляться множествами локаторами;
* не нужно скачивать дополнительные файлы.

Минусы Selenium:

* использует реальный браузер для работы, из-за чего скорость падает;
* типы объектов являются несереализуемыми.

# Сравнение скорости реализаций

В главе описывается сравнение скорости выполнения приведённых выше библиотек, фреймворков и инструментов. Сравниваться будут распараллеленные версии, так как некоторые нераспараллеленные версии работают очень долго.

Сравниваться методы парсинга будут по двум сайтам: MegaBonus и LetyShops.

Для сайта MegaBonus сравниваются реализации Selenium на трёх языках. Так как с Selenium на Python возникла проблема, сравниваться будут реализации на Selenium, совмещенные с ещё одной библиотекой. Для версии на Java и C# будут сравнения с использованием только Selenium.

Для сайта LetyShops сравниваются реализации других приведённых библиотек и фреймворков на трёх языках.

### **MegaBonus**

На Рисунке 12 изображены замеры времени выполнения работы парсеров на Java и C#, используя только Selenium.

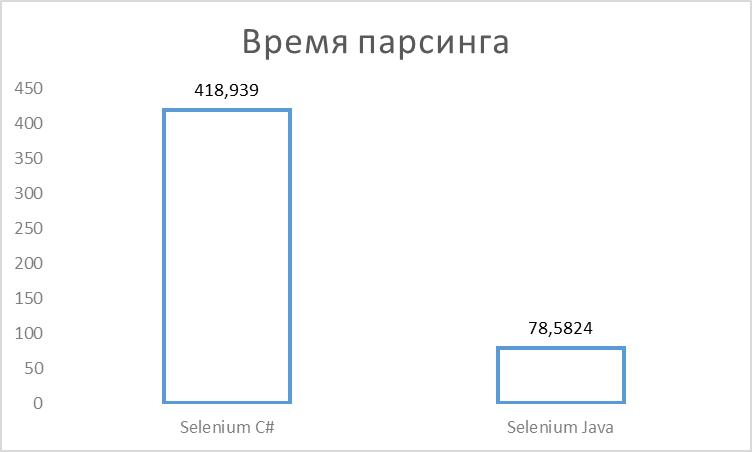


Рисунок 16 – время выполнения Selenium в секундах

На Рисунке 13 изображены замеры времени выполнения работы парсеров на Java, C# и Python, используя Selenium и стороннюю библиотеку. Так как на C# много вариантов из чего выбирать, среди них также было сделано сравнение по скорости выполнения. Результаты скорости выполнения парсеров среди библиотек C# на Рисунке 14.

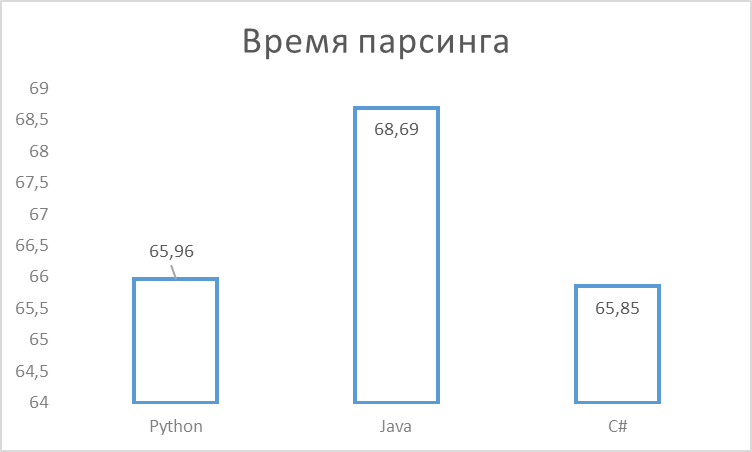


Рисунок 17 – время выполнения Selenium и другой библиотеки в секундах

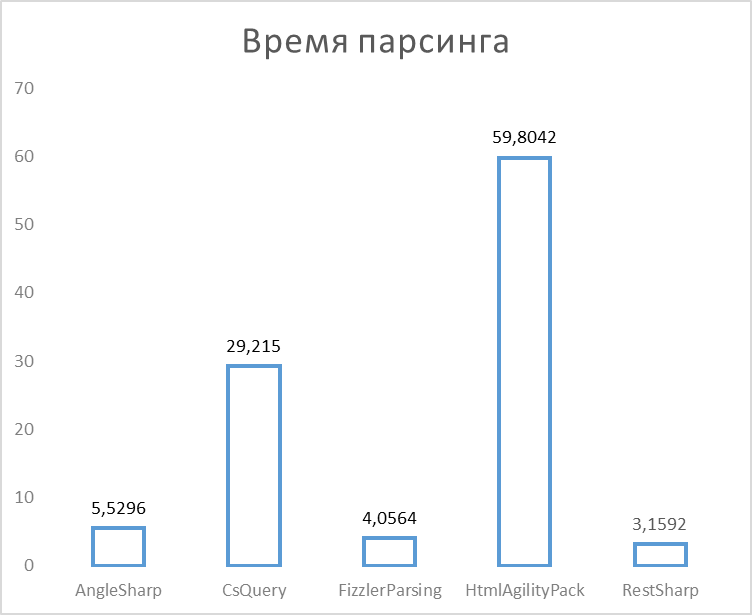


Рисунок 18 – время выполнения парсеров на C# в секундах

### **LetyShops**

На Рисунке 15 изображены замеры времени выполнения работы парсеров на Python.

Рисунок 19 – время выполнения парсеров на Python в секундах

На Рисунке 16 изображены замеры времени выполнения работы парсеров на Java.

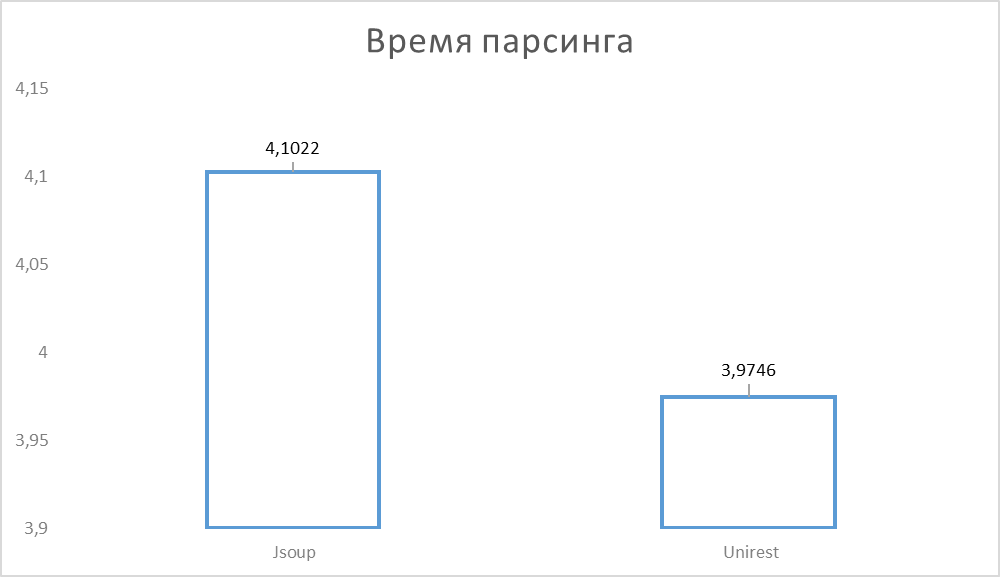


Рисунок 20 – время выполнения парсеров на Java в секундах

На Рисунке 14 изображены замеры времени выполнения работы парсеров на C#.

### **Вывод**

Все библиотеки важны, все библиотеки нужны (перефразировать более научно).

# Описание визуализации результатов

# Описание выхода из специфических ситуаций при парсинге

# Заключение

# Список использованной литературы

# Приложение А. JSoup

# Приложение Б. Selenium Java

# Приложение В. Unirest

# Приложение Г. HtmlAgilityPack

# Приложение Д. Fizzler

# Приложение Е. AngleSharp

# Приложение Ж. CsQuery

# Приложение З. RestSharp

# Приложение И. Selenium C#

# Приложение К. BeutifulSoup

# Приложение Л. Scrapy