



Руководство по проведению тестирования производительности системы ELMA с помощью Apache Jmeter



Оглавление

Введение.....	5
Глава 1. Цель и типы тестирования производительности.....	7
1.1. Основные определения тестирования производительности ..	7
1.2. Типы тестирования производительности	9
Глава 2. Установка и запуск Jmeter.....	11
2.1. Установка окружения.....	11
2.1.1. Установка JRE	11
2.2. Установка Jmeter	13
2.2.1. Назначение файлов и каталогов	13
2.3. Запуск Jmeter.....	14
2.3.1. Запуск Jmeter в графическом режиме (GUI).....	14
2.3.2. Запуск Jmeter без графического режима (non-GUI).....	16
Глава 3. Инструменты создания текстового сценария.....	17
3.1. Отправка HTTP(s) запросов	17
3.1.1. Структура HTTP-запроса	17
3.1.2. Метод HTTP-запроса.....	18
3.1.3. Код состояния HTTP-запроса	19
3.1.4. Пример HTTP-запроса.....	20
3.2. Главное окно Jmeter	26
3.3. Логирование в JMeter.....	29
3.4. Структура тестового плана.....	31
3.4.1. Thread Group.....	31
3.4.2. Элементы конфигурации	33
3.4.3. Контроллеры	45
3.4.4. Сэмплеры	46
3.4.5. Логические контроллеры.....	51

3.4.6. Препроцессоры.....	56
3.4.7. Пост-обработчики	60
3.4.8. Таймеры	69
3.4.9. Элементы подтверждения	72
3.4.10. Листенеры.....	81
3.4.11. Порядок выполнения элементов	90
Глава 4. Сценарий тестирования производительности	92
4.1. Подготовка к записи сценария нагрузочного тестирования....	
.....	93
4.2. Запись с помощью HTTP(S) Test Script Recorder.....	100
4.2.1. Настройка HTTPS.....	103
4.2.2. Настройка прокси.....	114
4.2.3. Запись тестового сценария	121
4.3. Запись с помощью Fiddler.....	130
4.3.1. Установка	130
4.3.2. Запуск.....	131
4.3.3. Настройка HTTPS.....	135
4.3.4. Запись сценария в Fiddler	137
4.4. Настройка и отладка сценария нагрузочного тестирования....	
.....	152
4.4.1. Авторизация	154
4.4.2. Создание документа	162
4.4.3. Отправка на ознакомление	182
4.4.4. Ознакомление с документом	195
4.4.5. Логаут.....	208
Глава 5. Проведение нагрузочного тестирования	211
5.1. Первая итерация тестирования.....	211
5.2. Вторая итерация тестирования	216
5.3. Настройка DashBoard Report	217

5.4. Результаты тестирования производительности	219
5.4.1. Ключевые метрики производительности	219
Глава 6. Полезные ресурсы.....	224

Введение

Данная книга является руководством по проведению тестирования производительности веб-приложений с помощью Apache Jmeter на примере системы ELMA. Выбор данного инструмента тестирования производительности в качестве основного обусловлен его преимуществами:

- высокая популярность и бесплатная модель распространения. В сети существует широкий выбор готовых компонентов и руководств, написанных сообществом. Благодаря открытому исходному коду, инструмент позволяет создавать самописные тестовые сценарии, используя API;
- простота разворачивания и интуитивно понятный интерфейс. Являясь приложением, полностью написанным на Java, его использование возможно на разных платформах при наличии установленной Java (JRE);
- широкий спектр стандартных отчетов, которые позволяют максимально подробно визуализировать результаты тестирования;
- модульная архитектура, которая позволяет дополнить инструмент новыми функциями, начиная от добавления кастомизированных отчетов с отображением всевозможных циклических метрик, заканчивая полным или частичным изменением UI, чтобы сделать сбор и отображение результатов тестирования более гибкими в зависимости от предпочтений конкретного пользователя;
- возможность его интеграции с Bean Shell и Selenium для выполнения автоматизированного тестирования.

Основная задача данного руководства – обозначить минимальный набор знаний, необходимых для написания параметризованных нагрузочных тестов по сценарию заказчика с использованием Jmeter, и последующего анализа результатов такого тестирования на основании предъявляемых заказчиком требований.

Данная книга является справочным руководством, в котором последовательно разбираются основные настройки и возможности Jmeter, как инструмента для проведения тестирования производительности. В книге также приведены общая терминология и основная цель тестирования

производительности, равно как и способы анализа полученных результатов тестирования.

Глава 1. Цель и типы тестирования производительности

У каждой системы имеются те или иные критерии производительности, которым она должна соответствовать. Проверка того, удовлетворяет ли система данным критериям, и является главной целью тестирования производительности. Благодаря собираемым результатам тестирования производительности можно определить, какой элемент нагрузки или часть системы приводит к снижению производительности. Это поможет выработать стратегию для дальнейшего конфигурирования системы и при увеличении нагрузки оперативно реагировать на возможные трудности.

Ниже приведены основные термины и их определения, необходимые для понимания процесса тестирования производительности.

1.1. Основные определения тестирования производительности

Виртуальный пользователь – программный процесс, с помощью которого будут выполняться необходимые операции.

Нагрузка – общее количество операций пользователей, выполняемых на ресурсе за определенное время.

Профиль нагрузки – это набор операций с заданными интенсивностями, полученный на основе сбора статистических данных либо определенный путем анализа требований к тестируемой системе.

Производительность – совокупность операций, которые выполняет система за определенный период времени.

Масштабируемость – пропорциональное увеличение способности системы справляться с нагрузкой (увеличение производительности) при добавлении ресурсов (обычно аппаратных).

Нагрузочная точка – рассчитанное или заданное количество виртуальных пользователей в группах, выполняющих операции с определенной интенсивностью.

Метрика производительности – числовое значение, определяющее производительность системы.

Итерация – действия пользователя и все операции, которые должны выполняться за один повтор.

Интенсивность выполнения операций – время между двумя итерациями или частота выполнения операций за единицу времени.

В зависимости от характеристик, которые необходимо оценить, тестирование производительности делится на несколько типов.

1.2. Типы тестирования производительности

1. Нагрузочное тестирование (Load Testing)

Задачами нагрузочного тестирования являются сбор показателей и определение производительности и масштабируемости системы или приложения посредством создания разного профиля нагрузки, которую она должна выдерживать. Для этого используется имитация работы пользователей, одновременно выполняющих в системе определенные операции. Основная цель нагрузочного тестирования заключается в том, чтобы, создав определенную нагрузку, наблюдать за показателями производительности системы.

Результатами данного тестирования являются:

- измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций;
- определение максимального числа пользователей, одновременно работающих в системе или приложении, с учетом критериев производительности;
- определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций).

2. Стressовое тестирование (Stress Testing)

Стрессовое тестирование позволяет проверить, насколько приложение и система в целом работоспособны в условиях стресса, а также оценить способность системы к регенерации, т.е. к возвращению к нормальному состоянию после прекращения воздействия стресса. Например, путем сравнения таких показателей, как пропускная способность и время отклика до и после его выполнения.

Стрессом в данном контексте может быть повышение интенсивности выполнения операций до пиковых значений в течение длительного времени, превышающие пределы нормального функционирования системы или приложения. Одним из возможных вариантов такого тестирования может являться создание с помощью инструментов нагрузочного тестирования виртуальных пользователей, одновременно отправляющих запредельное количество запросов на сервер, с которыми тот объективно не в состоянии справиться. Обнаружение возможных утечек памяти, уязвимости безопасности, повреждения данных, а также выявление

некорректной обработки ошибок и исключительных ситуаций, главным образом, определит, успешно ли сервер с ним справился.

3. Тестирование стабильности или надежности (Stability / Reliability Testing)

Задачей тестирования стабильности (надежности) является проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки. Время выполнения операций может играть в данном виде тестирования второстепенную роль. При этом на первое место выходит отсутствие утечек памяти, перезапусков серверов под нагрузкой и другие аспекты, влияющие именно на стабильность работы.

Важно понимать, что в рамках одной задачи тестирования производительности раскрывается только один из указанных выше аспектов. По умолчанию тестирование производительности проводится в разрезе измерения времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций.

Глава 2. Установка и запуск Jmeter

Apache Jmeter – инструмент для проведения тестирования производительности веб-приложений с моделируемыми тестовыми данными. Принцип его работы заключается в эмуляции отправки запросов на целевой сервер группой пользователей и возвращении статистической информации с результатами выполнения таких запросов. Эта информация отображается с помощью графических диаграмм.

В данной главе мы рассмотрим установку и запуск данного приложения, а также изучим его важные характеристики.

2.1. Установка окружения

2.1.1. Установка JRE

Поскольку Jmeter – приложение, которое полностью написано на языке программирования Java, для начала его использования необходимо установить **Java Runtime Environment (JRE)**. Для установки JRE необходимо выполнить следующие действия:

1. Перейдите на официальный сайт Java.
2. Нажмите на кнопку **Загрузить Java бесплатно**.

Примечание: JMeter 4.0 поддерживает как 8, так и 9 версии Java. Рекомендуется установить последний версию Java 8.x или 9.x. в целях обеспечения безопасности и производительности.

3. Появится лицензионное соглашение. После нажатия на кнопку **Согласиться и начать бесплатную загрузку** начнется загрузка самого файла.
4. После завершения загрузки запустите файл .exe для установки JRE. Следуйте указаниям установочной программы
5. После завершения установки нажмите **Закрыть**.
6. Перейдите в командную строку и введите команды "java -version".

Если все было сделано верно и JRE установлен правильно, в консоли появится версия установленной JRE.

Примечание: в некоторых случаях, например, если вы хотите дополнять исходный код приложения или создавать плагины для Jmeter, потребуется полностью совместимый JDK (Java Development Kits) версии 8

или выше. Инструкцию по установке JDK можно найти в кратком руководстве "[Создание автоматизированных текстов](#)".

2.2. Установка Jmeter

После успешной установки окружения переходим к установке самого Jmeter.

1. Перейдите на [официальный сайт Apache Jmeter](#).
2. В блоке **Binaries** скачайте .zip файл Jmeter.

Примечание: JMeter 4.0 – актуальная на момент написания данного руководства версия, которая поддерживает Java 8/9.

3. Разархивируйте скачанный архив в любое место на вашей системе.

2.2.1. Назначение файлов и каталогов

После распаковки архива структура файлов и каталогов Jmeter будет выглядеть следующим образом:

- **bin** – исполняемые файлы Apache.JMeter:
 - **ApacheJMeter.jar** – java-сборка Apache.JMeter;
 - **heapdump.cmd, heapdump.sh;**
 - **jmeter.bat, jmeter.sh, jmeter;**
 - **jmeter-n.cmd;**
- **docs** – документация;
- **extras** – дополнительные скрипты и утилиты;
- **lib** – библиотеки;
- **licenses** – лицензии;
- **printable_docs** – документация;
- **LICENSE** – лицензия Apache 2.0 на Apache.JMeter ([перевод](#), [описание](#));
- **NOTICE** – примечание;
- **README.md** – описание, требования, инструкция по установке и сборке.

2.3. Запуск Jmeter

2.3.1. Запуск Jmeter в графическом режиме (GUI)

Для запуска JMeter в графическом режиме используются скрипты, которые находятся в каталоге **jmeter/bin**:

- **jmeter.bat** – для Windows;
- **jmeter.sh** – для Linux.

Появления главного окна приложения означает, что оно готово к работе, а значит можно приступать к написанию своего тестового сценария.

ВАЖНО! Если при первом запуске Jmeter в Windows вы увидели предупреждение (Рис. 1), а затем ошибку (Рис. 2), это означает, что каталог **C:\Windows\system32** пропал из переменного окружения PATH.

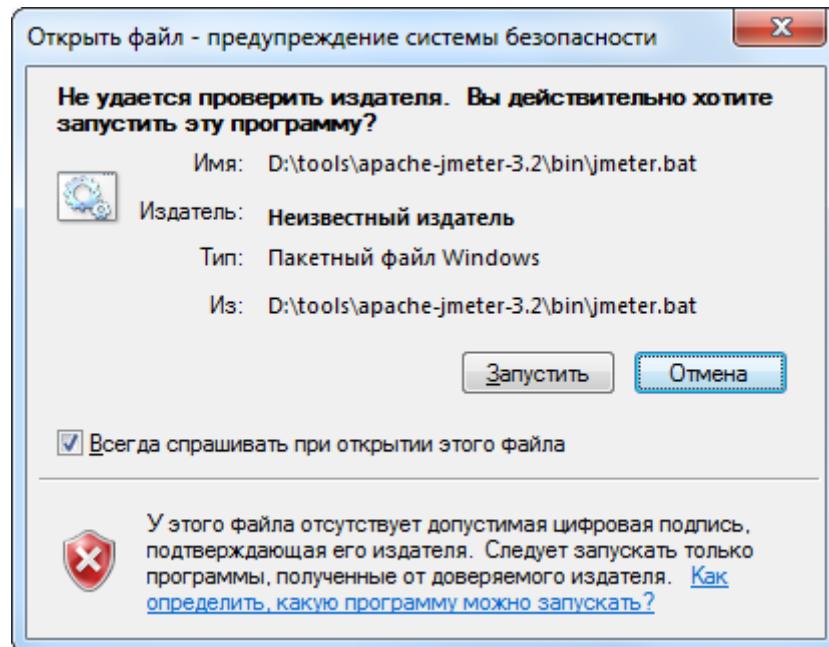


Рис. 1. Предупреждение системы безопасности о невозможности проверки издателя

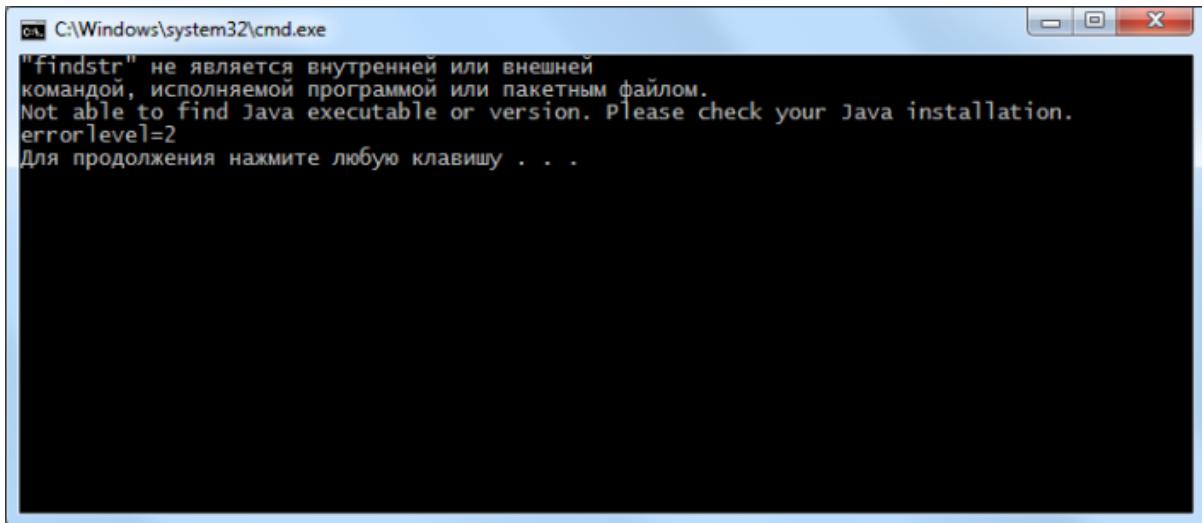


Рис. 2. Ошибка при первом запуске Jmeter в Windows

Для того чтобы **C:\Windows\System32** вернуть в PATH, необходимо модифицировать файл **jmeter.bat**, добавив в него строку (Рис. 3, Рис. 4).

```
set PATH=C:\Windows\System32;%PATH%
```

Рис. 3. Стока для модификации файла jmeter.bat

```

24 rem JM_START - set this to "start" to launch JMeter
25 rem           this is used by the jmeterw.cmd scrip
26 rem
27 rem =====
28
29 setlocal
30
31 set PATH=C:\Windows\System32;%PATH%
32 |
33 rem Minimal version to run JMeter
34 set MINIMAL_VERSION=1.8.0
35
36 for /f "tokens=3" %%g in ('java -version 2^>^&1 ^| f
37 rem @echo Debug Output: %%g
38 set JAVAVER=%%g
39 )

```

Рис. 4. Модификация файла jmeter.bat

Кроме того, можно отредактировать значение переменной PATH для всей операционной системы, что позволит сделать доступной видимость Java и для других приложений. Для этого выполните следующее:

1. Перейдите в **Панель управления – Система**.
2. Выберите **Дополнительные параметры системы**.
3. На вкладке **Дополнительно** нажмите **Переменные среды**.
4. Добавьте в переменную среды Path-путь к папке **bin** в папке, куда был установлен JRE.
5. Нажмите **Ок**.

2.3.2. Запуск Jmeter без графического режима (non-GUI)

При проведении нагрузочного тестирования рекомендуется не запускать Jmeter в графическом режиме, а использовать его только во время создания и отладки тестового сценария. Это обусловлено тем, что графический режим потребляет системные ресурсы, которые гораздо эффективнее направлять на генерацию самого теста и его данных.

Для запуска Jmeter в non-GUI режиме необходимо сделать следующее:

1. Удалить из тестового сценария все графические листенеры, так как в NON-GUI режиме они не будут работать.
2. В командной строке ввести следующую команду:

jmeter -n -t D:\TestScripts\script.jmx -l D:\TestScripts\scriptresults.jtl, где:

- **-n** – параметр, при котором запускается NON-GUI режим;
- **-t** – путь до вашего тест плана в формате .JMX;
- **-l** – место, куда запишется файл формата .JTL с результатами прохождения тестового сценария.

Помимо этих параметров, Jmeter имеет еще ряд других, которые можно применять в non-GUI режиме:

- **-R** – список удаленных серверов. Запускает тест на указанных удаленных серверах;
- **-H** – имя хоста прокси-сервера или IP-адрес;
- **-P** – порт прокси-сервера.

Данные параметры используются для удаленного выполнения тестов JMeter и для использования JMeter через прокси-сервер.

Глава 3. Инструменты создания текстового сценария

3.1. Отправка HTTP(s) запросов

HTTP (англ. HyperText Transfer Protocol – протокол передачи гипертекста) – широко распространенный протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов (то есть документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам).

Протокол HTTP предполагает использование клиент-серверной структуры передачи данных. Клиентское приложение формирует запрос и отправляет его на сервер, после чего серверное программное обеспечение обрабатывает данный запрос, формирует ответ и передает его обратно клиенту. После этого клиентское приложение может продолжить отправлять другие запросы, которые будут обработаны аналогичным образом.

Основная задача протокола HTTP – обмен данными между пользовательским приложением, осуществляющим доступ к веб-ресурсам (обычно это веб-браузер) и веб-сервером. Также HTTP часто используется как протокол передачи информации для других протоколов прикладного уровня, таких как SOAP, XML-RPC и WebDAV. API многих программных продуктов также подразумевает использование HTTP для передачи данных. Сами данные при этом могут иметь любой формат, например, XML или JSON.

В Jmeter HTTP-запрос передается с помощью элемента HTTP-Request в категории Sampler, который будет рассмотрен ниже.

3.1.1. Структура HTTP-запроса

HTTP-запрос состоит из трех основных частей:

1. **Строка запроса** – указывает метод передачи, URL-адрес, к которому нужно обратиться и версию протокола HTTP.
2. **Заголовки** – это набор пар ключ-значение, разделенных двоеточием, которые описывают тело сообщений. В заголовках передается различная служебная информация: кодировка сообщения, название и версия браузера, адрес, с которого пришел клиент (Referrer) и так далее.

3. **Тело сообщения** – это сами данные, которые передаются в запросе. В ответе передаваемыми данными, как правило, является разметка страницы или ее часть, которую запросил браузер. А в запросе, например, в теле сообщения передается содержимое файлов, загружаемых на сервер. Тело сообщения – это необязательный параметр и может отсутствовать (Рис. 5).

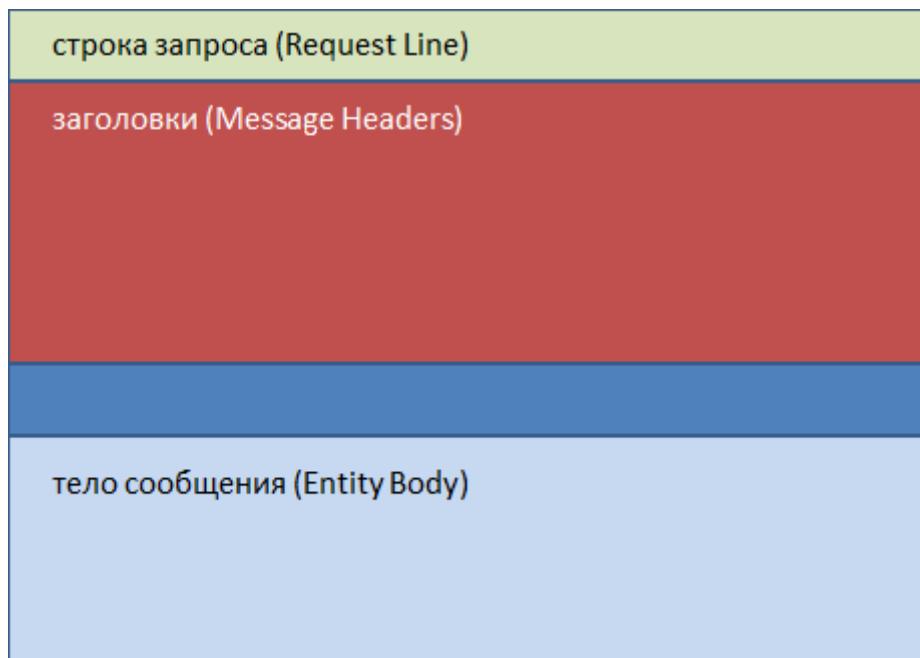


Рис. 5. Структура HTTP-запроса

На рисунке изображен порядок расположения частей запроса, который остается неизменным. Между заголовками и телом сообщения находится пустая строка, чтобы отделить служебную информацию от тела сообщения. Она представляет собой символ перевода строки.

URL (англ. Uniform Resource Locator) – унифицированный указатель ресурса), указатель размещения сайта в интернете. URL-адрес содержит доменное имя и указание пути к странице, включая название файла этой страницы.

Версия определяет, в соответствии с какой версией стандарта HTTP составлен запрос. Указывается как два числа, разделенных точкой, например, 1.1.

3.1.2. Метод HTTP-запроса

HTTP определяет множество методов запроса, которые указывают серверу, как нужно обращаться к запрашиваемому ресурсу, заданному в

URL. Стоит отметить, что метод чувствителен к регистру и его имя следует указывать только в верхнем регистре.

Существуют следующие типы методов HTTP запроса:

- **[GET](#)** – запрашивает представление ресурса. Запросы с использованием этого метода могут только извлекать данные;
- **[HEAD](#)** – запрашивает ресурс так же, как и метод GET, но без тела ответа;
- **[POST](#)** – используется для отправки сущностей к определённому ресурсу. Часто вызывает изменение состояния или какие-то побочные эффекты на сервере;
- **[PUT](#)** – заменяет все текущие представления ресурса данными запроса;
- **[DELETE](#)** – удаляет указанный ресурс;
- **[CONNECT](#)** – устанавливает "туннель" к серверу, определённому по ресурсу;
- **[OPTIONS](#)** – используется для описания параметров соединения с ресурсом;
- **[TRACE](#)** – выполняет вызов возвращаемого тестового сообщения с ресурса;
- **[PATCH](#)** – используется для частичного изменения ресурса.

При написании тестового сценария в Jmeter в качестве методов для формирования запроса, в основном, выступают методы POST и GET для отправки и получения серверных данных соответственно.

3.1.3. Код состояния HTTP-запроса

Код состояния (Status Code) – три цифры (первая из которых указывает на класс состояния), которые определяют результат совершения запроса. Существуют следующие классы состояния:

- **[1xx: Informational](#)** (информационные) – коды, информирующие о процессе передачи данных.
- **[2xx: Success](#)** (успешно) – коды, информирующие о случаях успешного принятия и обработки запроса клиента.
- **[3xx: Redirection](#)** (перенаправление) – коды, сообщающие клиенту, что для успешного выполнения операции необходимо сделать другой запрос, т.е. совершить перенаправление, как правило, по другому URL.
- **[4xx: Client Error](#)** (ошибка клиента) – коды, указывающие на ошибки со стороны клиента.

- [5xx: Server Error](#) (ошибка сервера) – коды, указывающие на случаи неудачного выполнения операции со стороны сервера.

Например, в случае, если был использован метод GET и сервер предоставляет ресурс с указанным идентификатором, то такое состояние задается с помощью кода 200 (OK). Если сервер сообщает о том, что такого ресурса не существует – 404 (Not Found).

3.1.4. Пример HTTP-запроса

В качестве примера обмена информацией между сервером и клиентом, разберем подключение к веб-серверу ELMA в качестве неавторизованного пользователя.

Для начала нам необходимо задать адрес веб-сервера ELMA в адресной строке браузера и нажать на кнопку перехода. После этого происходит соединение с сервером по полученному IP-адресу. После того как соединение установится, браузер передаст серверу следующую информацию:

```
Получение главной страницы ELMA
GET http://myelma:1319/ HTTP/1.1
Host: localhost:1319
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/74.0.3729.157 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: saveAsDefaultComment=true; ElmaCulture=ru-RU
```

Обратите внимание, что данный запрос осуществляется методом GET и состоит из двух частей. В нем осуществляется переход по адресу `http://myelma:1319/`. В заголовках указана информация об узле, на котором размещен сам ресурс, типе http-соединения, о программном обеспечении клиента и его характеристиках, а также о списке доступных форматов ресурса. Тела у GET-запроса, как правило, не имеется, хотя любое сообщение HTTP-запроса может содержать тело сообщения и, следовательно, должно анализироваться с учетом этого. Однако семантика сервера для GET ограничена таким образом, что тело, если оно есть, не имеет семантического значения для запроса, соответственно не будет идентифицировано.

Примечание: чтобы отследить время исполнения определенных запросов и сами запросы в веб-браузере, используется такой инструмент, как сетевой монитор. Он дает представление о запрашиваемых и загружаемых ресурсах в режиме реального времени. Для того чтобы открыть его, необходимо:

1. Открыть консоль нажатием на клавиатуре клавиши **F12** или вызвать контекстное меню любого элемента страницы и выбрать пункт **Просмотреть код**.
2. Перейти на вкладку **Network** для браузера Google Chrome (**Сеть** – для Mozilla Firefox и Internet Explorer).

Лог сетевого монитора начнет автоматически наполняться после совершения какого-либо действия на текущей странице или при переходе на другой ресурс из браузерной строки. Для отображения информации по нужному запросу, необходимо нажать на него в списке запросов (Рис. 6).

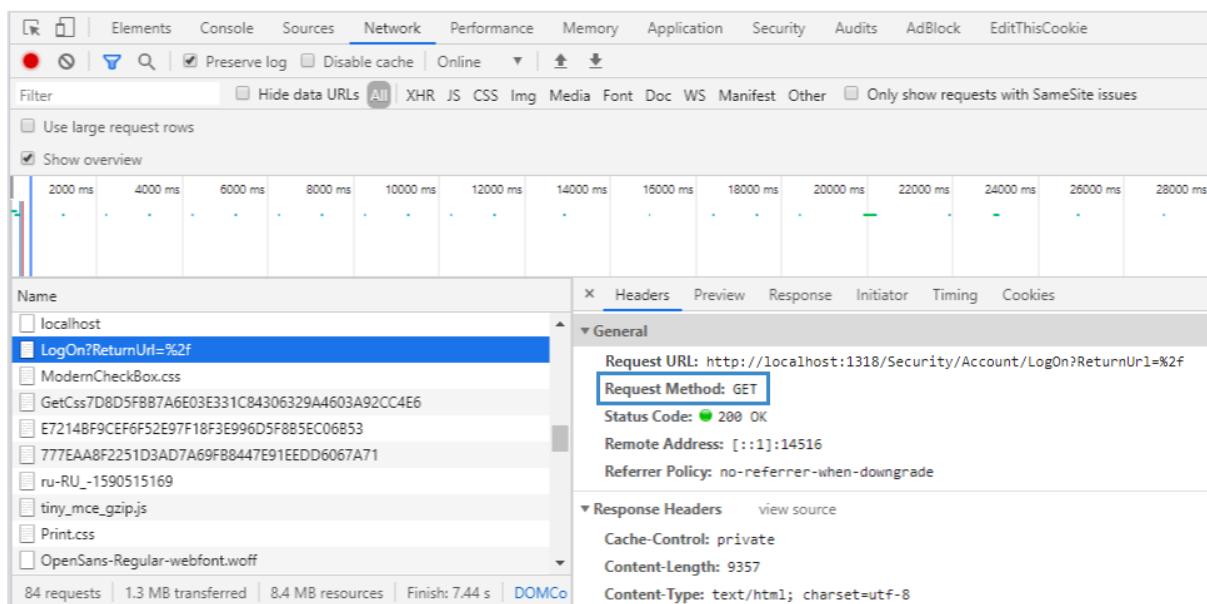


Рис. 6. Отображение информации по запросу

После отправки запроса браузер ожидает, пока пакеты данных дойдут до сервера, которые тот обработает и вернет назад ответ. При попытке перехода на наш сайт был получен следующий ответ:

```
HTTP/1.1 302 Found
Cache-Control: private
Content-Type: text/html; charset=utf-8
Location: /Security/Account/LogOn?ReturnUrl=%2f
Server: Microsoft-IIS/10.0
X-AspNetMvc-Version: 4.0
Set-Cookie: ASP.NET_SessionId=je41h302uinpsy0okopey3rf; path=/; HttpOnly
```

```
Set-Cookie: ElmaCulture=ru-RU; expires=Wed, 20-May-2020 06:28:38 GMT; path=/
Set-Cookie: Elma.V3.Forms.Auth=; expires=Mon, 11-Oct-1999 19:00:00 GMT; path=/; HttpOnly
X-Powered-By: ASP.NET
Date: Tue, 21 May 2019 09:28:38 GMT
Content-Length: 154
Location:/Security/Account/LogOn?ReturnUrl=%2f
```

В полученном ответе сообщается о том, что запрашиваемая страница найдена, но поскольку мы подключаемся от неавторизованного пользователя, сервер перенаправляет нас на страницу авторизации в системе (Рис. 7), на что указывает код состояния (302) и следующая строчка заголовка ответа:

```
Location:/Security/Account/LogOn?ReturnUrl=%2f
```

В данном случае серверу отправляется следующий GET-запрос уже на страницу авторизации в системе:

```
Получение страницы авторизации ELMA
GET http://myelma:1319/Security/Account/LogOn?ReturnUrl=%2f HTTP/1.1
Host: localhost:1319
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/74.0.3729.157 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
```

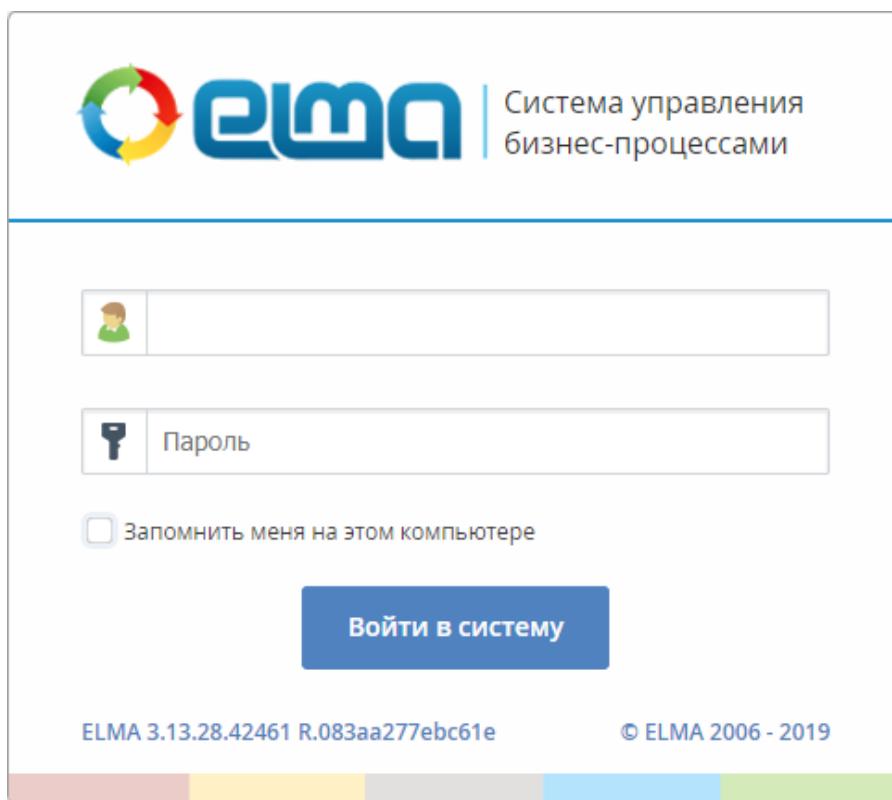


Рис. 7. Страница авторизации ELMA

Для продолжения работы пользователю необходимо ввести свои учетные данные в специальную форму и нажать на кнопку **Войти в систему**. В этом случае на сервер будет отправлены переменные логина и пароля со значением того содержимого, что ввел пользователь. Такая отправка данных будет осуществляться методом POST.

Чтобы обратиться к серверу методом GET, достаточно набрать запрос в URL-адрес, тогда как в методе POST все работает по другому принципу. На странице авторизации в систему для выполнения запроса такого типа нам необходимо нажать на кнопку, в которой имеется атрибут **type="submit"**.

Рассмотрим отправленный нами Post-запрос:

```
Отправка формы с данными авторизации пользователя
POST http://myelma:1319/Security/Account/LogOn?ReturnUrl=%2f HTTP/1.1
Host: localhost:1319
Connection: keep-alive
Content-Length: 328
Cache-Control: max-age=0
Origin: http://localhost:1319
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/74.0.3729.157 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,appli
cation/signed-exchange;v=b3
Referer: http://myelma:1319/Security/Account/LogOn?ReturnUrl=%2f
```

```
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
__RequestVerificationToken=yWc9gZ9WN9LP8SEJZ-hESMZTH6KR_NqsNF4PgGHT9TXb5CbMAaWoM1vxtK-
KdeEMwiZ_Sz2VoxMn0b6aVRmHT9hR4rrmU-
DUBajcvioJ6R7DzJFvYr01hyIiJC0Kn_37i9aYI49QnGwV04BH_mB8Q2;
ASP.NET_SessionId=j0tsspsquwoduw5msg1jbe3s
__RequestVerificationToken=BQ-bMyxpXF8rntoVPmRL0YIkshk3w-
NqKB5zRWiLH_QAfmaESm0dz9407R4V1fHSt-
8a1eDJKDt_NsSqadSK3AsPOz0cQZjMbttW0FxUQYUocu71p4HP1VdvAjiM4g5Ix2HRmEjD6AgkcuHhzaVvAeX4u4bmm
uQa8YRr04VY0E1&login=user1&password=123&rememberMe=false&LogIn=%D0%92%D0%BE%D0%B9%D1%82%D0%
B8+%D0%B2+%D1%81%D0%B8%D1%82%D0%B5%D0%BC%D1%83
```

В отличие от отправленных ранее Get-запросов, в запросе методом Post присутствует тело сообщения, в котором находятся передаваемые на сервер данные, в частности **Токен верификации запроса**, а также учетные данные пользователя и настройка **Запомнить меня на этом компьютере** с выбором значения **Нет**.

Для защиты от **CSRF атак** (англ. Cross-Site Request Forgery – межсайтовая подделка запроса) разработан механизм, который предотвращает выполнение злоумышленником различных действий на веб-сайте путем подделки ссылки и отправки ее от имени авторизованного в системе пользователя. Его принцип работы заключается в том, что для каждой сессии пользователя в HTML-коде запрашиваемой пользователем страницы сайта динамически генерируется дополнительный уникальный ключ, предназначенный для выполнения запросов или **Request Verification Token**. При отправке формы он передается в теле запроса, и перед выполнением каких-либо действий сервер проверяет этот ключ.

Далее следует **аутентификация**, т.е. процедура проверки подлинности введенных пользователем данных. В случае ее успешного прохождения от сервера будет получен ответ с 302 кодом, а значит последующим перенаправлением на ту страницу веб-сервера ELMA, на которую мы изначально обращались до момента авторизации в системе. В данном случае на главную страницу, чему свидетельствует заголовок "Location://" следующего ответа сервера:

```
HTTP/1.1 302 Found
Cache-Control: private
Content-Type: text/html; charset=utf-8
Location: /
Server: Microsoft-IIS/10.0
X-AspNetMvc-Version: 4.0
Set-Cookie: ElmaCulture=ru-RU; expires=Mon, 25-May-2020 10:59:26 GMT; path=/
Set-Cookie:
Elma.V3.Forms.Auth=F9C527306462CBC77AFE93B501D3083017DC24232CA940FAAECC672CD05973E5973F
CB3004EA12FE7DDF5F5C2635F1EEE234CDB74981BD5BF77D260E63D084B64728D4A2663129749F52375C3
BF9D6BCD158E239E62807C677BCDF66A90589EAB9120812E47C010075ED5C5B0682CD8FBA62512002B37
```

```
2580DBE2E3DB78F71CAA83DCCF059751EC66E88182D8295B99D4F780C8F746A57A83C8ABF82A5B89E02C  
52F5C148326810DB0900595B23B11AB7B1CA30C6357A9240006A77590806617ED8893E58942E05434AB47  
1F9FFE862C; path=/; HttpOnly  
X-Powered-By: ASP.NET  
Date: Tue, 21 May 2019 10:59:26 GMT  
Content-Length: 118,
```

Более подробно работа с HTTP-запросами рассматривается в [Главе 4](#) текущего руководства.

Получив представление о понятии и структуре HTTP-запроса, переходим к знакомству с инструментарием самого JMeter.

3.2. Главное окно Jmeter

На Рис. 8 изображено главное окно приложения. В левой части находится рабочее поле программы, представляющее собой древовидную структуру, которая берет начало из основного узла – **TestPlan** (тестовый план).

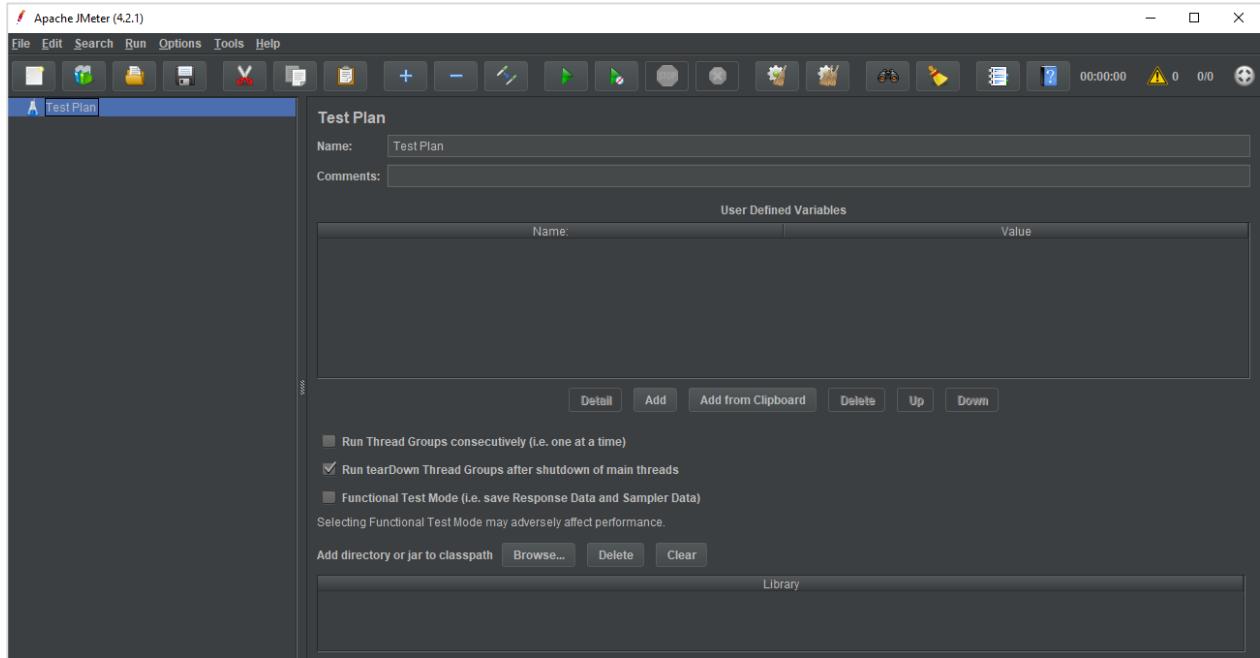


Рис. 8. Главное окно приложения Jmeter

Из этого узла будет происходить выстраивание тестового сценария в иерархическом порядке путем добавления в план тестирования основных его элементов, список которых можно увидеть, вызвав его контекстное меню и наведя курсор на кнопку **Add**, а после этого выбрать нужную категорию (Рис. 9).

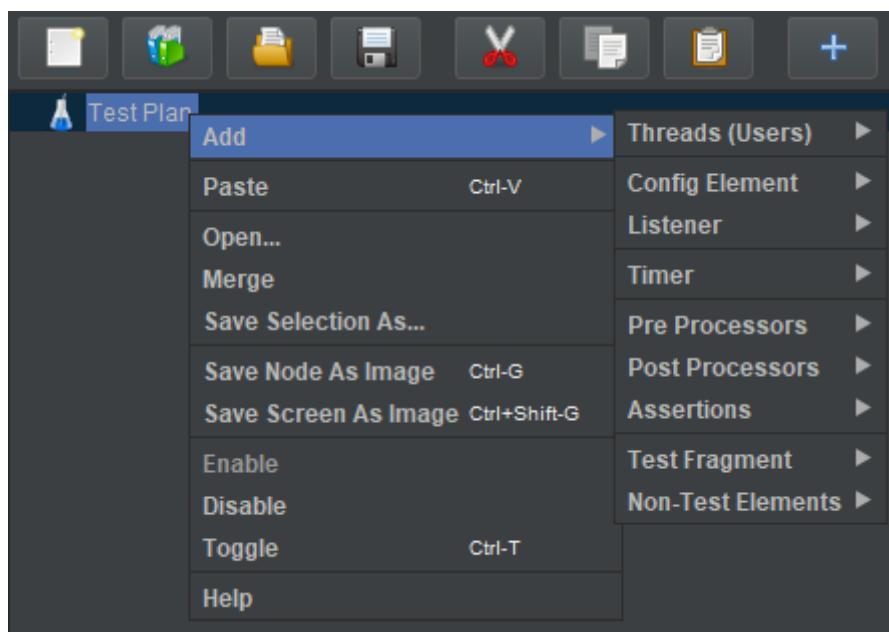


Рис. 9. Порядок выстраивания тестового сценария в приложении Jmeter

Более подробно о ключевых элементах тестового плана будет рассказано ниже.

Справа от рабочего поля расположена форма просмотра текущего выбранного элемента с отображением всех его конфигурируемых настроек и параметров.

Сверху находится панель инструментов, на которой расположены кнопки и индикаторы, помогающие оперировать ходом создания и прохождения теста. К числу основных можно отнести следующие:



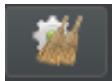
– сохранить тест;



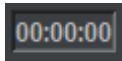
– запустить тест;



– остановить тест;



– очистка всех сгенерированных в Jmeter данных после прохождения теста, а также сброс его индикаторов;



– таймер, показывающий время прохождения текущего теста;



- счетчик, показывающий число ошибок в логе за время прохождения теста. При нажатии на значок, в нижнем углу Jmeter откроется сам лог.



- индикатор, показывающий соотношение загруженных виртуальных пользователей, участвующих в прохождении текущего теста, к их общему заданному числу.

Иногда возникают ситуации, когда необходимо отключать те или иные элементы тестового плана. Для этого необходимо вызвать контекстное меню определенного элемента и выбрать пункт **Disable** или нажать на клавиатуре сочетание клавиш **Ctrl + T**. Для включения элемента обратно необходимо выбрать пункт контекстного меню **Enable** или также нажать сочетание клавиш **Ctrl + T** (Рис. 10).

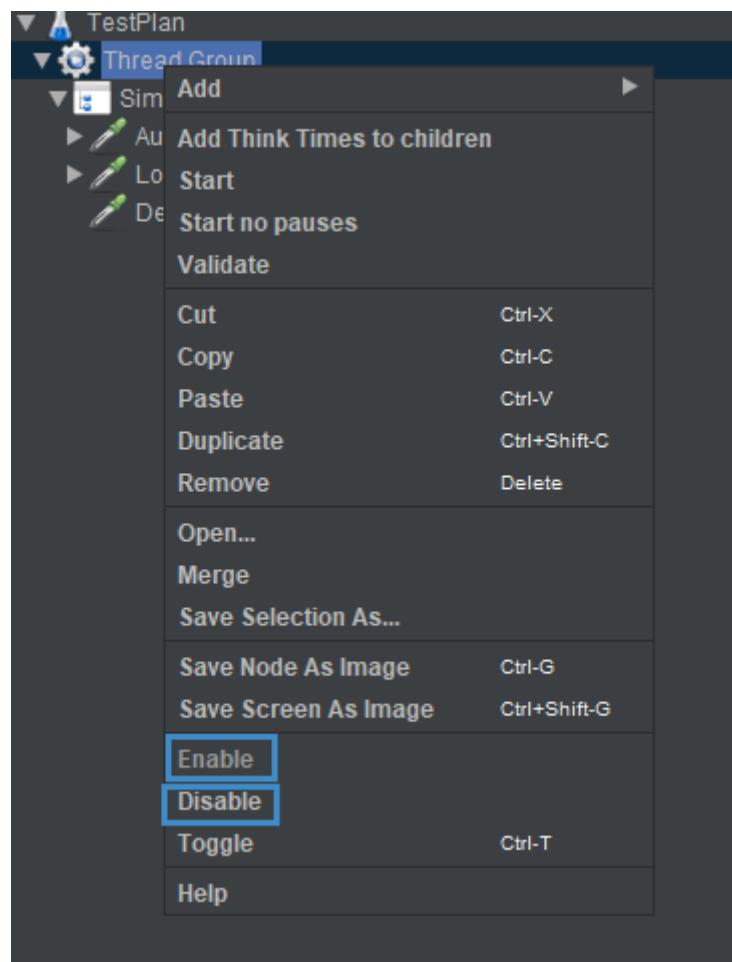


Рис. 10. Возможность отключить элементы тестового плана в приложении Jmeter

3.3. Логирование в JMeter

Для отладки возникающих проблем с самим JMeter или его элементами, в частности, в сценариях пост- и предобработчиков возникает необходимость чтения системных журналов (логов) JMeter, в которые заносится информация об определенных событиях при прохождении тестового плана. При правильной настройке данные журналы могут содержать много полезной информации. В Jmeter используется утилита log4j, позволяющая изменять конфигурационный файл **log4j2.xml**, который находится в каталоге bin Вашего Jmeter, в реальном времени и управлять процедурой записи без перезагрузки приложения. Сам лог-файл, в который записывается информация во время прохождения теста, называется **jmeter.log**.

В графическом интерфейсе JMeter существует специальная панель для просмотра логов, в которой отображаются все события во время прохождения текущего тестового плана. Для ее вызова необходимо нажать

на иконку  (Рис. 11).

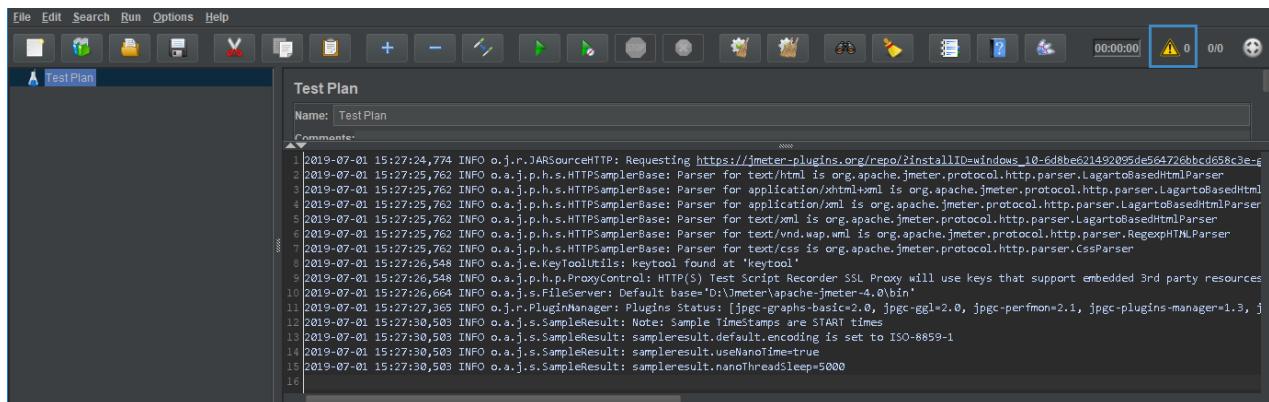


Рис. 11. Иконка вызова панели для просмотра логов

В конфигурационном файле **jmeter.properties**, который расположен в каталоге bin JMeter, для данной панели доступны следующие настройки:

- **#jmeter.loggerpanel.display = true/false** – отвечает за открытие/закрытие панели при каждом запуске JMeter;
- **#jmeter.loggerpanel.enable_when_closed = true/false** – отвечает за получение/игнорирование события журнала, если панель закрыта;
- **#jmeter.loggerpanel maxlen = 1000** – данная опция определяет максимальное количество хранимых строк в панели.

События записываются в журнал согласно уровням. Существует несколько уровней логирования:

- **ERROR** – содержит информацию об ошибках элемента тестового плана во время его прохождения и самого JMeter;
- **WARN** – содержит информацию о событиях предупреждений;
- **INFO** – записываются информационные события;
- **DEBUG** – записываются события отладки;
- **TRACE** – записываются все события.

Самый подробный уровень логирования – **TRACE** (записываются все события), самый низкий – **ERROR** (записываются только ошибки). При повышении уровня логирования все более низкие пишутся тоже.

По умолчанию в Jmeter используется **INFO**. Для включения определенного уровня логирования вызовите меню **Options → Log Level** и выберите из списка необходимое значение (Рис. 12).

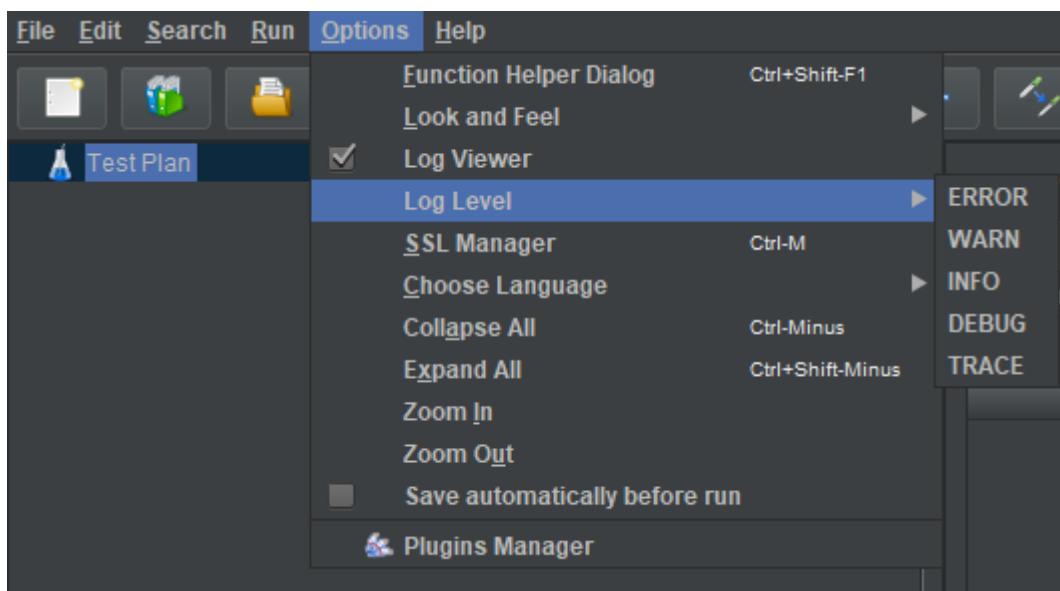


Рис. 12. Включение уровня логирования

При необходимости вывести сообщение в панели просмотра логов в ходе выполнения теста разместите в элементе написания сценариев, например, в JSR223 Preprocessor, код следующего типа:

```
log.info("Тестовое сообщение:" + vars.get("имяПеременнойJMeter"));
```

3.4. Структура тестового плана

Как правило, каждый тестовый сценарий имеет несколько неразрывно связанных друг с другом частей, которые формируются на основании профиля нагрузки, который, в свою очередь, регламентируется уже в самом ТЗ. В Jmeter минимальный тестовый сценарий состоит из [тестового плана](#) (**Test Plan**), [группы потоков](#) (**Thread Group**) и одного или более [сэмплеров](#) (**Samplers**), с помощью которых происходит отправка запросов на тестируемый веб-сервер.

3.4.1. Thread Group

Группа потоков (Thread Group) – основной рабочий элемент, в который записываются сценарии, добавляется различная логика и элементы управления, участвующие в teste (Рис. 13). Элементы данной группы, в частности, сэмплеры (Samplers) и контроллеры (Controllers) являются начальными точками любого тестового плана в Jmeter, и, соответственно, должны располагаться внутри нее. Другие элементы, например, листенеры (Listeners) могут быть размещены непосредственно под тестовым планом и в этом случае будут применяться ко всем группам потоков. Как следует из названия, данный элемент контролирует количество потоков (виртуальных пользователей), которые Jmeter будет использовать для выполнения теста.

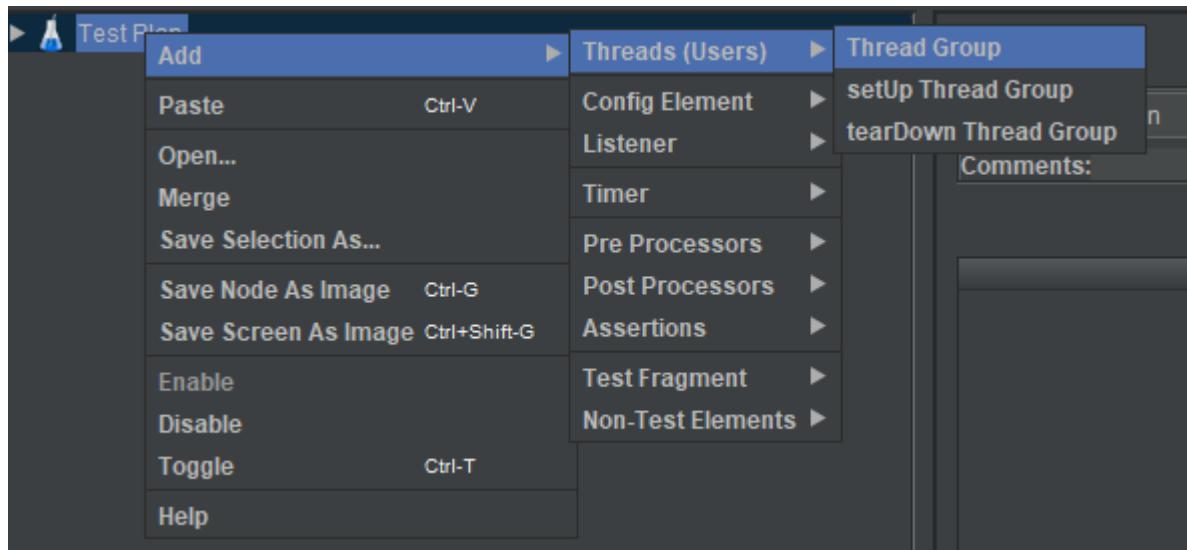


Рис. 13. Добавление группы потоков

Элемент **Thread Group**, позволяющий задавать параметры генерируемой на приложение нагрузки, находится в категории **Threads**

(Users). Основными его параметрами являются [Action to be taken after a Sample Error](#), [Thread Properties](#), [Scheduler Configuration](#) (Рис. 14).

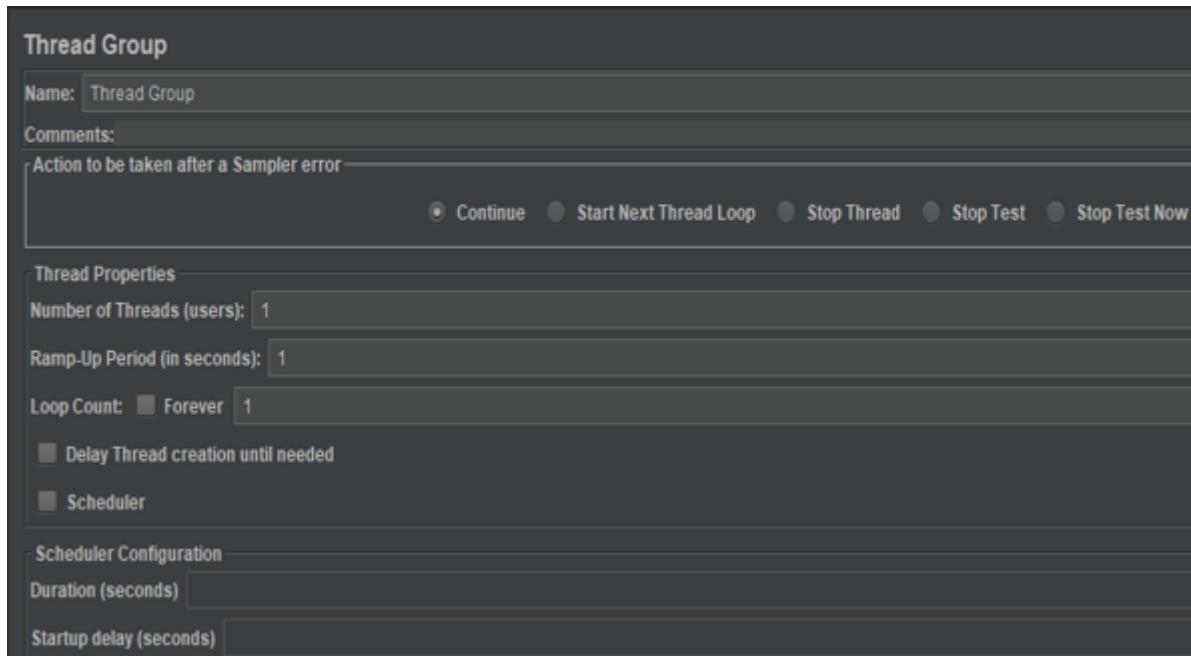


Рис. 14. Основные параметры группы потоков

3.4.1.1. **Action to be taken after a Sample Error**

Если во время прохождения теста какой-либо из запросов вызовет ошибку, можно задать то, как в этом случае поведет себя Jmeter, исходя из следующих возможных опций:

- **Continue** – проигнорировать ошибку и перейти к следующему запросу в древе;
- **Start Next Thread Loop** – завершить цикл в рамках текущего потока и начать следующий;
- **Stop Thread** – остановить текущее выполнение потока;
- **Stop Test** – останавливает выполнение всего теста после завершения выполнения текущих запросов;
- **Stop Test Now** – мгновенно останавливает выполнения всего теста, все текущие запросы прерываются.

3.4.1.2. **Thread Properties**

Данная панель позволяет настроить эмулируемую приложением нагрузку:

- **Number of threads** – количество эмулируемых пользователей, одновременно работающих с приложением;

- **Ramp-up period** – промежуток времени, через который выполняется запуск очередного потока (например, если указать **Number of threads** = 10, а **Ramp-up period** = 100, то это будет означать, что каждые 10 секунд будет запускаться новый поток);
- **Loop count** – количество циклов, в течение которых будет выполняться сценарий внутри Thread Group (**Forever** – сценарий будет выполняться всегда, пока не будет прерван явно);
- **Scheduler** – планировщик времени работы сценария.

3.4.1.3. Scheduler Configuration

Раздел **Scheduler Configuration** (Настройка планировщика) позволяет настроить продолжительность и задержку перед запуском тестового плана. Чтобы сделать данную область конфигурации доступной, необходимо включить параметр **Scheduler** в приведенном выше разделе **Thread Properties**. Панель с настройками планировщика изображена на Рис. 15.

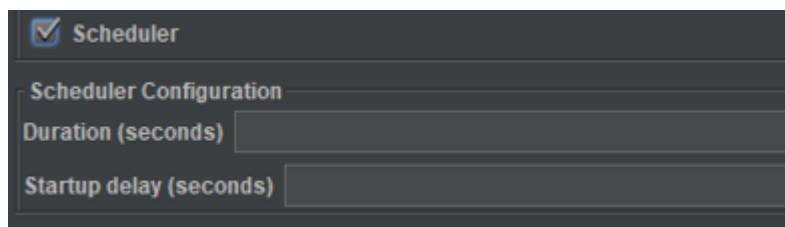


Рис. 15. Раздел "Scheduler Configuration" (Настройка планировщика)

- **Duration (seconds)** – длительность (в секундах). Данная настройка указывает, в течение какого промежутка времени будет выполняться тест. Например, если длительность установлена на 60 секунд, Jmeter будет выполнять тест в течение 60 секунд и завершит его по истечении времени.
- **Startup delay (seconds)** – задержка запуска (в секундах). Данная настройка позволяет установить определенную задержку перед началом теста. Например, если установлено время задержки в 10 секунд, то Jmeter не начнет загружать виртуальных пользователей, пока не истечет 10 секунд с момента запуска теста.

3.4.2. Элементы конфигурации

Элементы конфигурации (Configuration Elements) позволяют задавать глобальные настройки для всего тестового плана, а также объявлять переменные, данные из которых впоследствии уже будут

использованы сэмплерами. Настройки элементов конфигурации можно использовать как для всего сценария, определив его в начале тестового плана, так и для определенных частей плана, разместив его внутри конкретного контроллера или сэмплера.

К числу наиболее важных элементов относятся следующие:

- [**CSV Data Set Config**](#);
- [**HTTP Request Defaults**](#);
- [**User Defined Variables**](#);
- [**HTTP Header Manager**](#);
- [**JDBC Connection Configuration**](#);
- [**HTTP Cache Manager**](#);
- [**HTTP Cookie Manager**](#);
- [**FTP Request Defaults**](#);
- [**Counter**](#).

3.4.2.1. CSV Data Set Config

CSV Data Set Config используется для чтения данных из текстового файла или файла формата .CSV.

Например, если требуется провести нагрузочное тестирование с использованием 100 уникальных пользователей, то необходимо подготовить данные в CSV-файле, содержащие записи 100 пользователей системы, с указанием в столбик их учетных данных. Таким образом, с помощью данного элемента конфигурации на протяжении всего теста для каждого потока вы сможете использовать данные из этого файла, записанные в конкретные локальные переменные Jmeter, и, объявляя эти переменные в запросах/сэмплерах (Рис. 16).

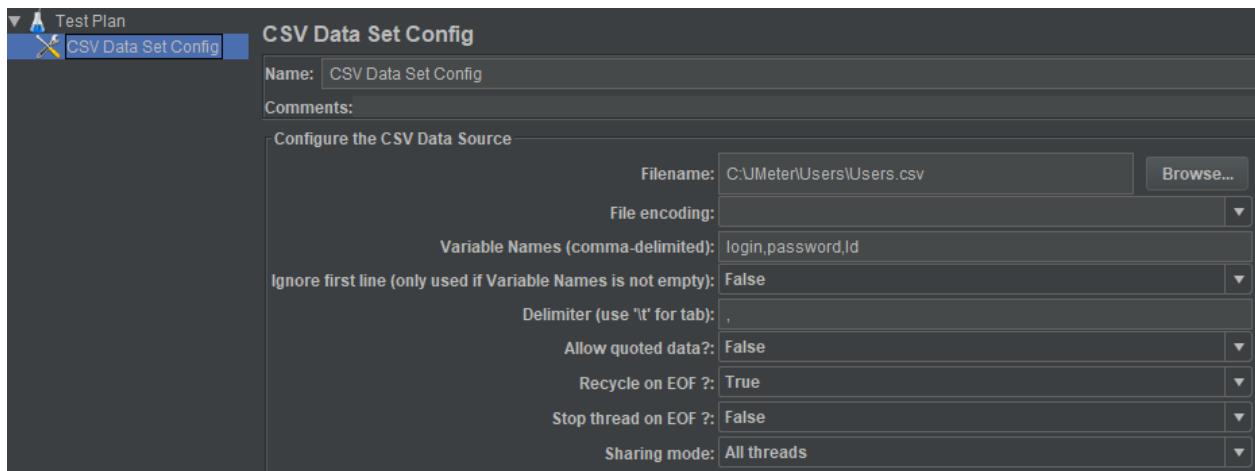


Рис. 16. Элемент конфигурации "CSV Data Set Config"

Добавьте CSV Data Set Config в тот узел, где он будет использоваться.

- **FileName** – необходимо указать путь к текстовому файлу или CSV-файлу, в котором хранятся данные с пользователями;
- **Variable Names** – необходимо задать через запятую имена переменных для значений в той же последовательности, в которой расположены соответствующие им столбцы текстового/CSV-файла. Впоследствии в сэмплерах их можно будет объявлять, как "\$login" / "\$password" / "\$id";
- **Delimiter** – необходимо указать разделитель, который использован в вашей CSV-файле (или текстовом файле) для логического отделения учетных данных пользователя друг от друга;
- **Sharing mode** – режим обмена. Данные элемента конфигурации будут использоваться между всеми потоками (Threads) или только в рамках определенного потока или группы (Thread Group).

На Рис. 17 приведен пример CSV-файла, который содержит учетные данные пользователей, а также их Id, для сценария входа в систему:

	A	B	C	D	E
1	User1,user1pw,2				
2	User2,user2pw,3				
3	User3,,4				
4	User4,user4pw,5				

Рис. 17. Пример CSV-файла

В данном случае, в качестве пароля пользователя User3 будет использоваться пустое значение.

3.4.2.2. HTTP Request Defaults

Элемент **HTTP Request Defaults** позволяет задать параметры соединения с веб-сервером по умолчанию, которые будут использоваться во всех элементах HTTP Request Sampler при условии, что в них не задано своих настроек.

Например, необходимо отправить 100 HTTP-запросов на один и тот же сервер. Если задать необходимые параметры в самом элементе HTTP Request Defaults и разместить его в начале плана тестирования, указанные настройки будут использоваться для каждого [HTTP Request Sample](#), что исключит необходимость ввода одной и той же информацию вручную и позволит избежать дублирования данных в тестах (Рис. 18).

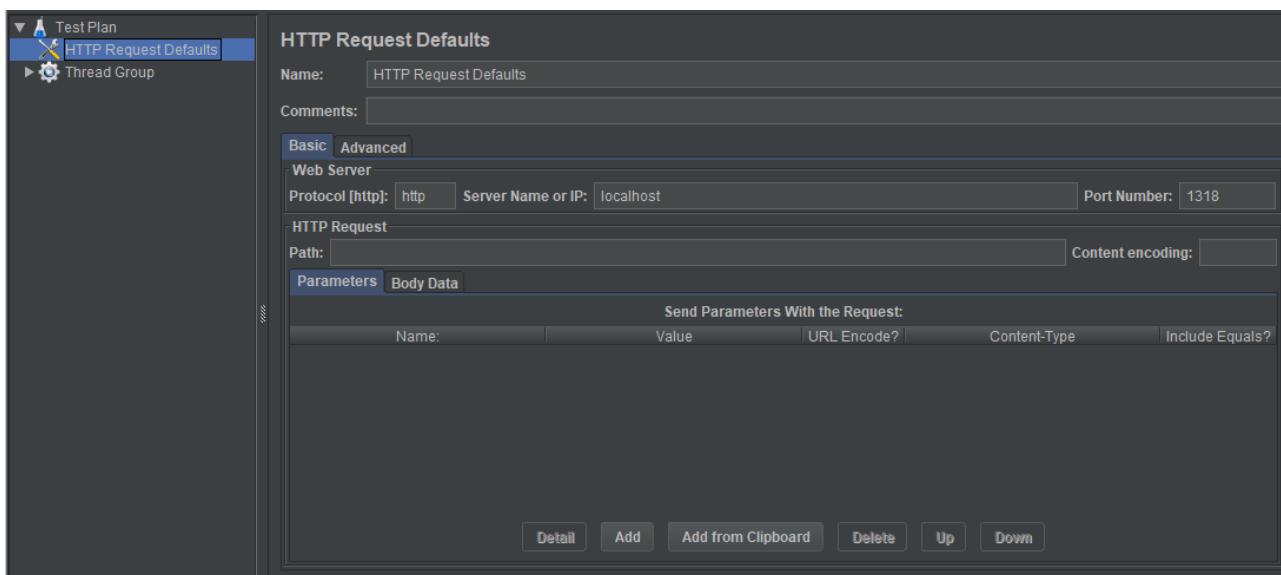


Рис. 18. Элемент конфигурации "HTTP Request Defaults"

- **Web Server** – необходимо указать тип протокола, доменное имя / IP-адрес и порт веб-сервера
- **Content encoding** – необходимо ввести стандарт кодирования текста. Желательно использовать UTF-8 (Юникод).
- **Send Parameters With the Request** – в данном поле определяются параметры, которые будут отправлены в каждом [HTTP Request Sampler](#).

3.4.2.3. User Defined Variables

Элемент **User Defined Variables (UDV)** позволяет задать набор глобальных переменных и их значения по умолчанию, которые будут использованы в тестовом плане (Рис. 19). На протяжении всего тестового

сценария в них можно записывать значения, извлекаемые с помощью пост- и предобработчиков запросов. Для простоты и открытости теста рекомендуется, чтобы данный элемент размещался в начале тестового плана (Рис. 19).

Примечание: все переменные UDV инициализируются при старте, независимо от расположения элемента в тестовом плане.

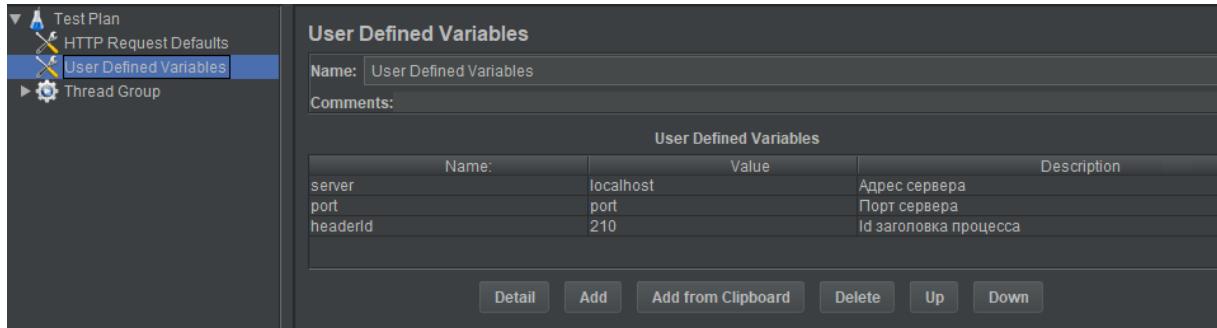


Рис. 19. Элемент "User Defined Variables (UDV)"

Панель **User Defined Variables**:

- **Name** – имя переменной для обращения к ней в teste;
- **Value** – значение переменной;
- **Description** – описание переменной. Данное поле используется в информационных целях, его необходимо заполнять для большей прозрачности и более легкого восприятия вашего тестового плана.

Данные переменные могут быть записаны в любое поле любого элемента тестового плана, а также использоваться внутри специальных функций, однако нужно помнить, что некоторые поля не допускают случайные строки, поскольку ожидают на вход числа.

Ссылка на переменную в тестовом элементе задается в формате **`${имя переменной}`**. Например, для параметризации адреса `http://localhost:1319` с помощью указанных на Рис. 20 переменных необходимо в поле, где он используется, ввести следующее значение: **`http://${server}:${port}`**.

Примечание: для отображения значения переменных во время отладки теста используется элемент Debug Sampler, результат выполнения которого и хранимые им данные можно увидеть в листенере View Result Tree на вкладке **Response Data**.

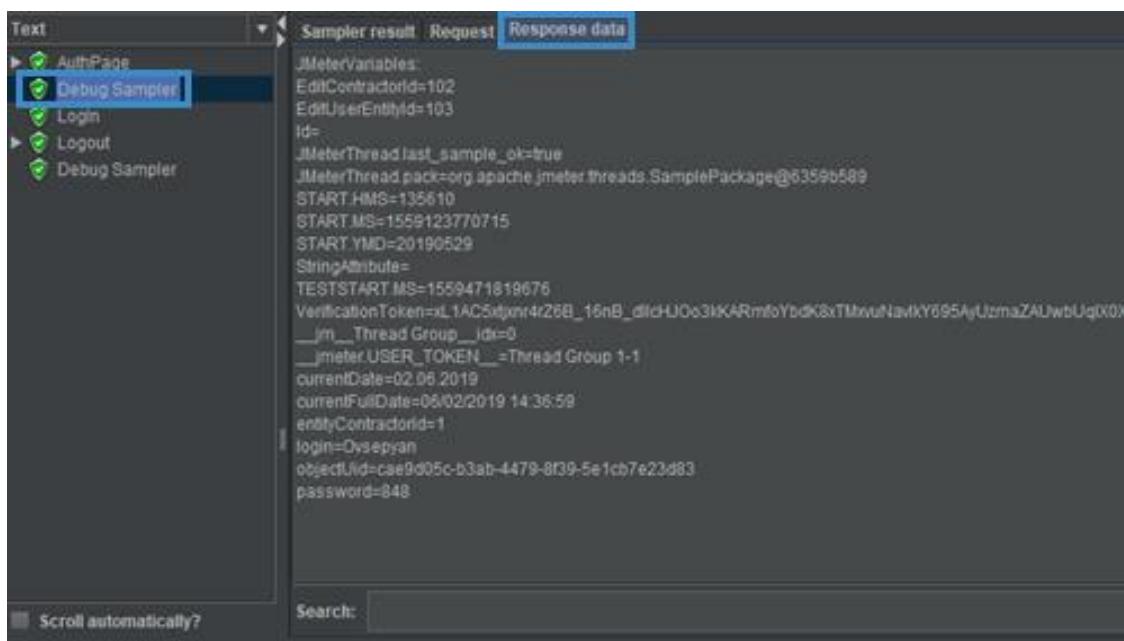


Рис. 20. Элемент "Debug Sampler"

3.4.2.4. HTTP Header Manager

Каждый раз, когда браузер отправляет запрос на веб-сервер, к запросу прикрепляются заголовки с дополнительной информацией, содержащие информацию о браузере, кодировке, предпочтительном языке получаемой страницы, перенаправлении и т.д. Обычно HTTP-заголовки выглядят таким образом:

Name	Value
Accept-Language	en-En
Accept	text/html, application/xhtml+xml, */*
User-Agent	Mozilla/5.0(compatible: MSIE 9.0; Windows NT 6.1; WOW 64; Trid)
Accept-Encoding	gzip, deflate
Referer	http://192.168.100.52\$(APPPATH)/main.nsf/welcome?openform

Элемент **HTTP Header Manager** позволяет добавлять или переопределять такие заголовки для отправляемого запроса в JMeter. Добавленные в него записи передаются в качестве списка данных для сэмплера (Рис. 21). Далее имя каждой записи переданного списка сверяется с существующими именами заголовков, и в случае их соответствия значение записи будет передано в этот заголовок. При этом, если переданная запись будет содержать пустое значение, заголовок содержащий ее будет удален из списка и соответственно в запросе передан не будет.

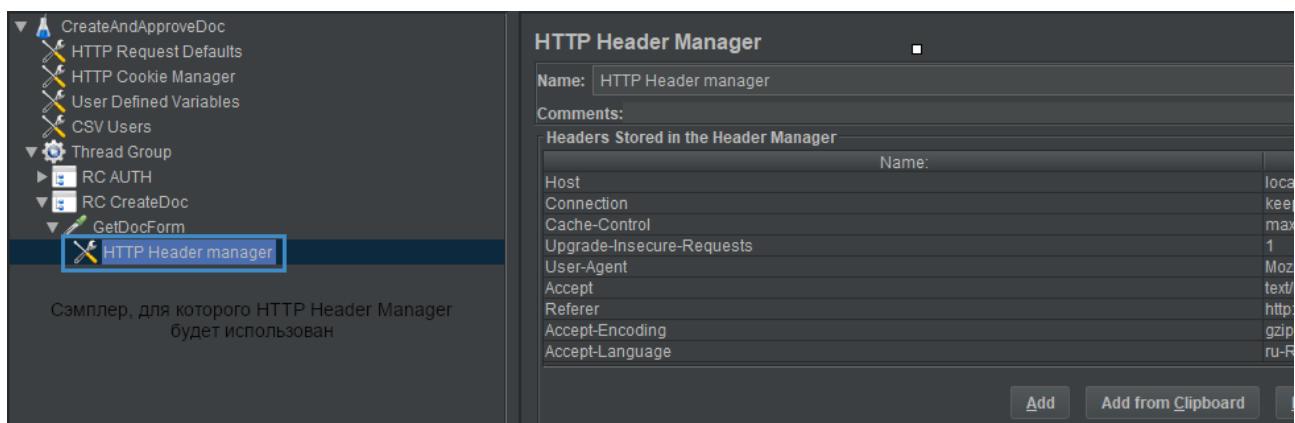


Рис. 21. Элемент "HTTP Header Manager"

При записи тестового сценария с помощью элемента HTTP(s) Test Script Recorder с включенной настройкой **Capture HTTP Headers** или другого внешнего прокси, для каждого запроса автоматически добавляется элемент HTTP Header Manager с уже заданным в нем набором параметров.

3.4.2.5. JDBC Connection Configuration

JDBC – это интерфейс прикладного программирования, который определяет, как клиент может получить доступ к базе данных, соответственно элемент JDBC Connection Configuration используется для настройки соединения Jmeter с определенной базой данных. Для отправки самих запросов используется элемент JDBC Request (Рис. 22).

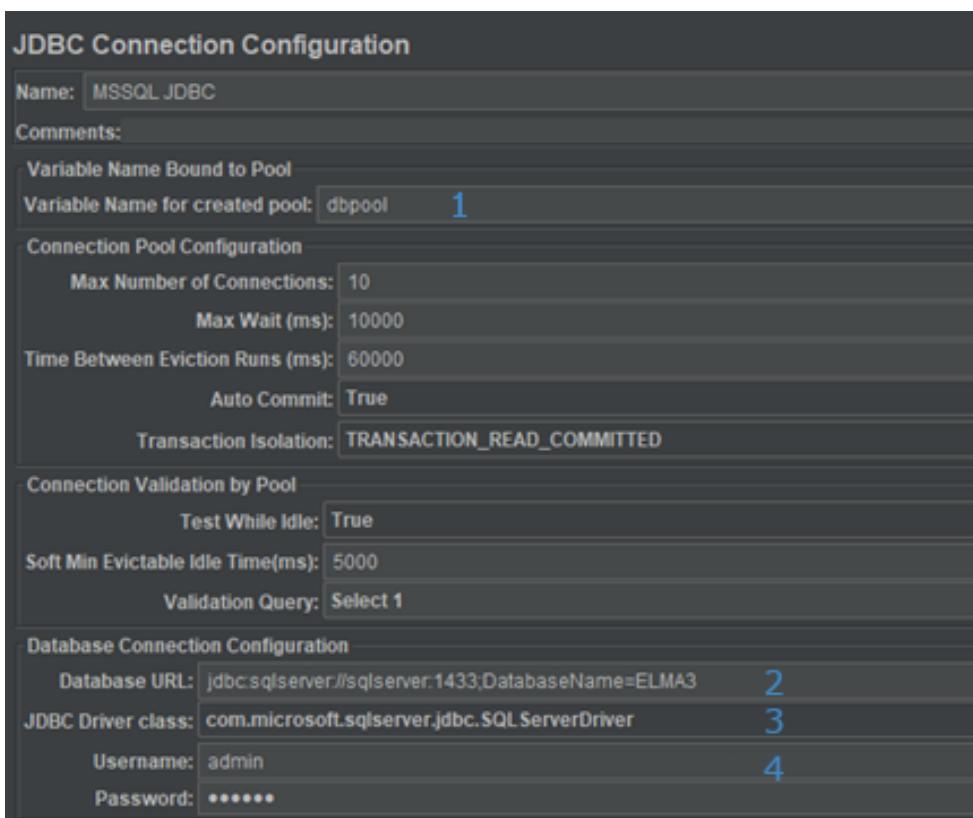


Рис. 22. JDBC Connection Configuration

- **Variable Name** – необходимо задать имя, которое будет использоваться в JDBC Request (Sampler) для доступа к пулу сессий.
- **Database URL** – URL-адрес соединения с базой данных. При использовании MS SQL в качестве СУБД, он выглядит следующим образом:

jdbc:sqlserver://[serverName\[instanceName][:portNumber]][;property=value[:property=value]], где:

- **jdbc:sqlserver://** (обязательно) известен как подпротокол и является константой;
- **serverName** (необязательно) является адресом сервера, с которым выполняется соединение. Это может быть DNS, IP-адрес, локальный узел или 127.0.0.1 локального компьютера. Имя сервера необходимо указать в коллекции свойств, если оно не указано в URL-адресе соединения;
- **instanceName** (необязательно) является экземпляром, с которым выполняется соединение с serverName. Подключение выполняется к экземпляру по умолчанию, если не указано другое;

- **portNumber** (необязательно) является портом, с которым выполняется соединение с **serverName**. Значение по умолчанию – 1433. Если соединение выполняется с портом по умолчанию, в URL-адресе необязательно указывать порт или символ ':' перед ним;
- **property** (необязательно) представляет собой одно или несколько свойств соединений. Как правило, указывается имя базы данных, с которой выполняется соединение, т.е. вместо параметра `property = value` необходимо написать `databaseName = название базы данных`.
- **JDBC Driver class** – класс драйвера JDBC, в котором реализован код работы с конкретной СУБД. Для MS SQL из списка необходимо выбрать `com.microsoft.sqlserver.jdbc.SQLServerDriver`.
- Учетные данные для подключения к СУБД:
 - **Username** – имя пользователя, у которого есть доступ к СУБД;
 - **Password** – пароль для доступа к СУБД.

ВАЖНО! При параметризации значений параметров адреса соединения с базой данных, а также учетных данных для подключения к СУБД необходимо инициализировать механизм встроенных свойств из командной строки или из файла **user.properties**, расположенному в **Путь до папки Jmeter\bin**.

3.4.2.6. HTTP Cache Manager

В веб-браузерах существует несколько механизмов, которые позволяют сократить поступающий интернет-трафик. Одним из таких механизмов является кэширование ресурсов, загруженных из сети, для использования последующих запросов без нагрузки веб-сервера. Именно поэтому крайне важно воспроизвести такое же поведение при создании теста производительности, для чего и используется элемент HTTP Cache Manager.

Во время прохождения теста с сервера запрашивается определенный URL-адрес, Jmeter, с помощью данного элемента, сохраняет результаты запроса в ОЗУ (Рис. 23).

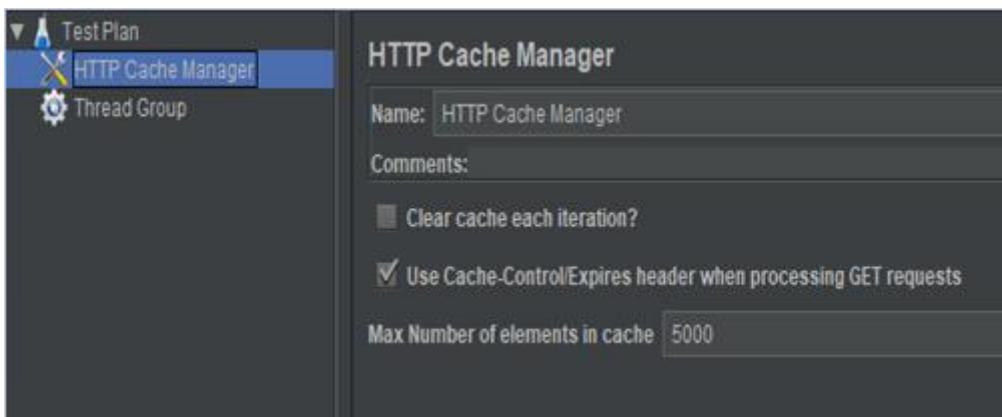


Рис. 23. Элемент "HTTP Cache Manager"

- **Clear cache each iteration?** – при выборе данного параметра кэш будет очищаться при старте каждой новой итерации.
- **Use Cache-Control/Expires header when processing GET requests** – при выборе данного параметра будет установлен контроль сохраненного в Jmeter кэша путем проверки его актуальности. Для каждого Get-запроса, имеющего HTTP Cache Manager с заданной настройкой в качестве элемента верхнего уровня, проверяется заголовок Expires кэша, в котором задано время истечения его актуальности. Если время, заданное в этом заголовке, превышает текущую дату, то сэмплер будет обработан мгновенно, возвращая кэшированный контент и не запрашивания URL-адрес удаленного веб-сервера. Данная настройка предназначена для эмуляции поведения веб-браузера.
- **Max Number of elements in cache** – максимальное число элементов, которые могут находиться в кэше (по умолчанию – 5000).

3.4.2.7. HTTP Cookie Manager

Для хранения фрагментов текста, в которых содержится информация о сессиях входа, веб-браузеры в системе используют **Куки** (Cookies). В Jmeter, инструмент обработки данных cookie, собранных во время прохождения теста, называется **HTTP Cookie Manager** (Рис. 24).

Элемент HTTP Cookie Manager имеет 3 функции:

1. Эмулирует работу веб-браузера в плане отправки и хранения куки. Если в ответе на HTTP-запрос содержится куки, то Cookie Manager ее сохраняет и будет использовать для каждого последующего запроса на конкретный веб-сервер. В каждом потоке Jmeter имеется "область

хранения куки". Соответственно, при тестировании веб-сайта, который использует куки для хранения информации об определенных сессиях, каждый поток Jmeter будет иметь собственную сессию.

2. Позволяет сохранять полученные куки в качестве переменных Jmeter. Для сохранения куки в качестве переменных установите свойство "CookieManager.save.cookies=true". Перед сохранением имена куки содержат префикс "COOKIE_", что позволяет избежать случайного повреждения локальных переменных. Для возвращения к исходному поведению, необходимо установить свойство "CookieManager.name.prefix =".
3. Возможность добавления куки вручную. При их ручном добавлении они станут доступным для всех потоков Jmeter. У таких куки имеется дата истечения срока актуальности.



Рис. 24. Элемент "HTTP Cookie Manager"

- **Clear cookies each iteration?** – при выборе данного параметра куки будут очищаться при старте каждой новой итерации.

3.4.2.8. FTP Request Defaults

Элемент **FTP Request Defaults** позволяет задать параметры соединения с FTP-сервером по умолчанию, которые будут использоваться во всех элементах FTP Request Sampler при условии, что в них не задано своих настроек (Рис. 25).



Рис. 25. Элемент "FTP Request Defaults"

- **Server Name or IP / Port Number** – доменное имя / IP-адрес FTP-сервера и порт;
- **Remote File** – путь до файла на удаленном сервере для скачивания. По умолчанию это поле пустое, его необходимо использовать только в том случае, если файл объявлен глобально;
- **Local File** – путь до файла в локальной файловой системе (где расположен сам Jmeter), который будет загружен на FTP-сервер. По умолчанию это поле пустое, его необходимо использовать только в том случае, если файл объявлен глобально;
- **Local File Contents** – содержимое локального файла. Данное поле позволяет задать содержимого исходного файла;
- **Get(RETR)** – установите данный параметр, если необходимо скачать файл с FTP-сервера;
- **Put(STOR)** – установите данный параметр, если необходимо загрузить файл на FTP-сервер;
- **Use Binary mode?** – устанавливает бинарный режим для передачи данных. При скачивании/загрузке простых текстовых файлов данный параметр включать не нужно, в любом другом случае данный параметр необходимо установить.

3.4.2.9. Counter

Элемент **Counter** создает динамически изменяющуюся числовую характеристику, как правило, задачей которой является отследить число повторений того или иного элемента теста на текущий момент (Рис. 26).

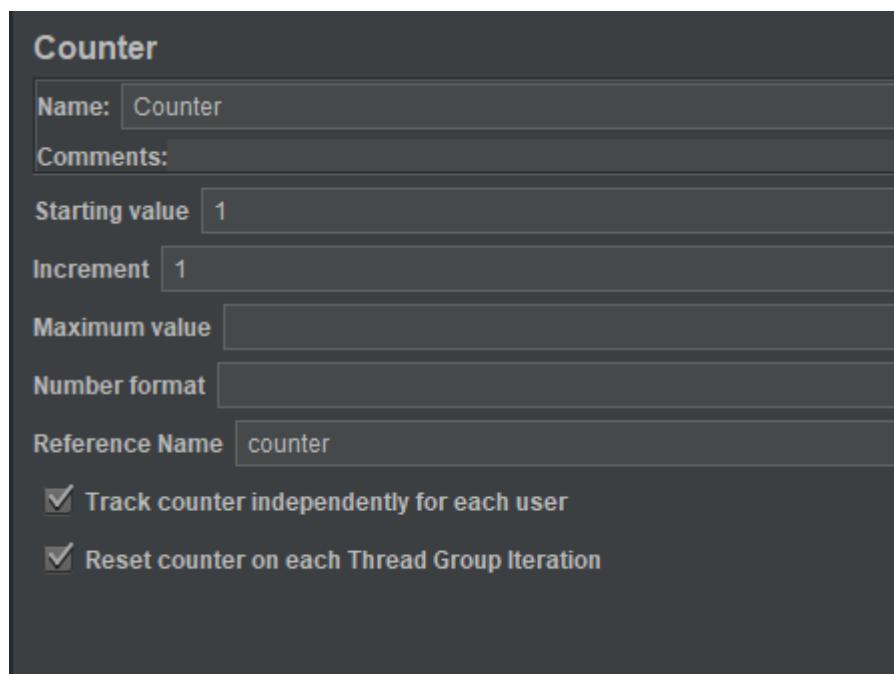


Рис. 26. Элемент "Counter"

- **Starting value** – стартовое значение, с которого будет начинаться отсчет;
- **Increment** – значение данного поля будет прибавляться к текущему значению Counter при каждом его срабатывании в teste;
- **Maximum value** – максимальное допустимое значение Counter. Если оставить данное поле пустым, то его значение будет увеличиваться бесконечно;
- **Number format** – задает формат выходного числа. Например, если ввести в данное поле "00", то значения Counter будут иметь префикс из двух нулей, например, 001, 002, 003 и т.д;
- **Reference Name** – определяет имя переменной Jmeter, в которой будет храниться значение Counter для его дальнейшего использования в тестовом сценарии.

3.4.3. Контроллеры

Контроллеры (Controllers) в JMeter являются элементами логики и предназначены для управления прохождением теста. Всего имеется два типа контроллеров: **Сэмплеры (Samplers)** и **логические контроллеры (Logic Controllers)**.

3.4.4. Сэмплеры

Сэмплеры (Samplers) используются для обращения Jmeter к серверу путем отправки запросов. Сэмплеры, по сути, это реальные запросы, которые Jmeter отправляет на тестируемый веб-сервер. Каждый сэмплер генерирует один или несколько результатов, которые имеют определенные атрибуты (успех/неудача, время выполнения, размер данных и т.д.) и могут быть отображены в различных листенерах. К числу наиболее важных сэмплеров относятся:

- [HTTP Request](#);
- [FTP Request](#);
- [SMTP Sampler](#);
- [JDBC Request](#);
- [Debug Sampler](#).

3.4.4.1. HTTP Request

HTTP Request Sampler используется для отправки HTTP/HTTPS-запросов на веб-сервер (Рис. 27).

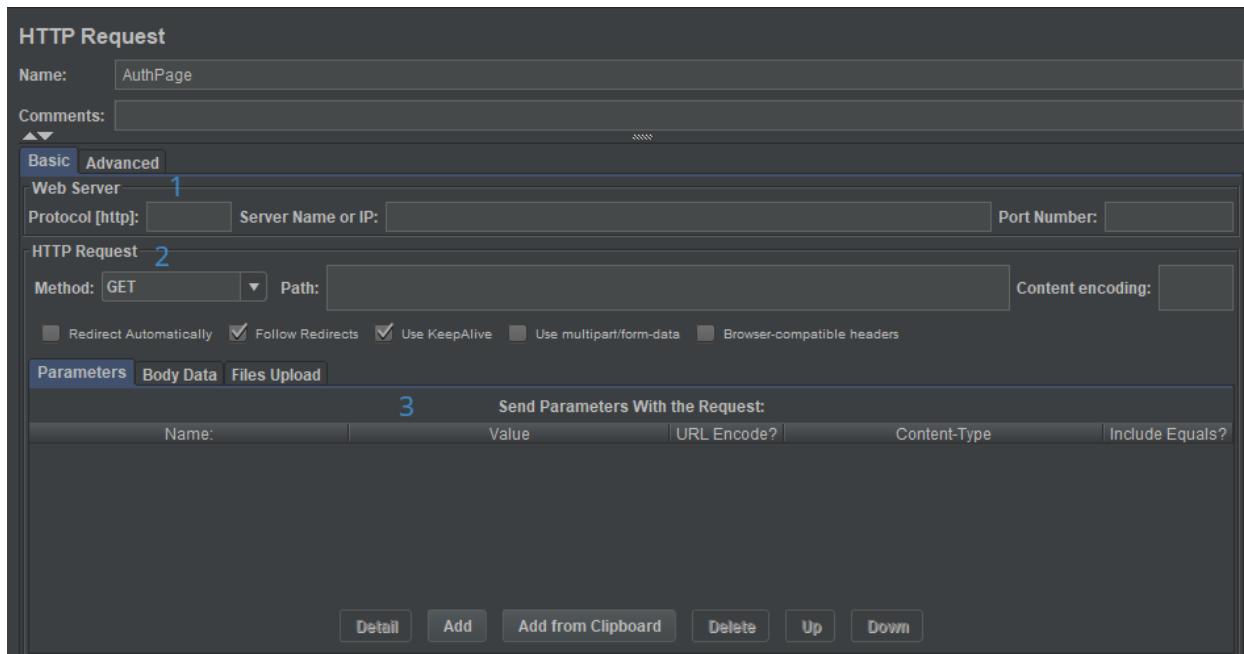


Рис. 27. HTTP Request Sampler

- Панель **Web Server** – необходимо ввести тип протокола, ввести доменное имя / IP-адрес веб-сервера и порт сервера;
- Панель **HTTP Request** – необходимо выбрать метод запроса, ввести путь к запрашиваемому ресурсу;

- **Send Parameters With the Request** – в данном поле определяются параметры, отправляемые в запросе. При использовании инструментов автоматической записи сценария, в добавленных с помощью них запросах уже имеются необходимые параметры, если сам тип запроса предполагает их передачу. В них лишь будет необходимо задать определенный уровень параметризации, чтобы во время прохождения самого теста нужные данные, передаваемые в запросах, формировались динамически.

Примечание: при отправке нескольких запросов на один и тот же веб-сервер следует рассмотреть возможность использования элемента HTTP Request Defaults, чтобы не приходилось вводить одну и ту же информацию для каждого HTTP-запроса.

3.4.4.2. *FTP Request*

FTP Request Sampler позволяет отправлять FTP-запрос типа "Скачать файл" или "Загрузить файл" на FTP-сервер (Рис. 28).

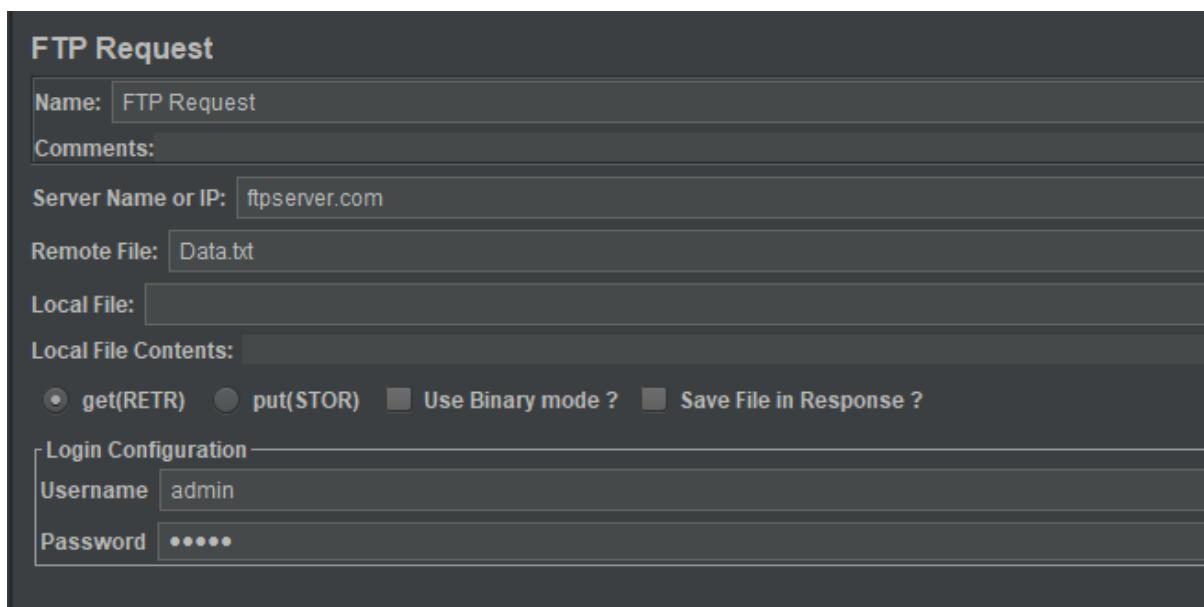


Рис. 28. *FTP Request Sampler*

- **Server Name or IP / Port Number** – доменное имя / IP-адрес FTP-сервера;
- **Remote File** – имя файла для извлечения;
- **Get(RETR)** – необходимо выбрать, если хотите скачать файл с сервера;
- **Put(STOR)** – необходимо выбрать, если хотите загрузить файл на сервер;
- **Login Configuration** – учетные данные пользователя FTP-сервера.

3.4.4.3. JDBC Request

JDBC Request Sampler полезен при тестировании баз данных. С его помощью можно передать SQL-запрос в любую базу данных, поддерживающую протокол [JDBC](#).

Для использования данного сэмплера необходимо добавить в Test Plan связанный с ним конфигурационный элемент с настройками подключения к серверу БД (**JDBC Connection Configuration**), который находится в категории **Config Element**.

JDBC Request Sampler выглядит следующим образом (Рис. 29).

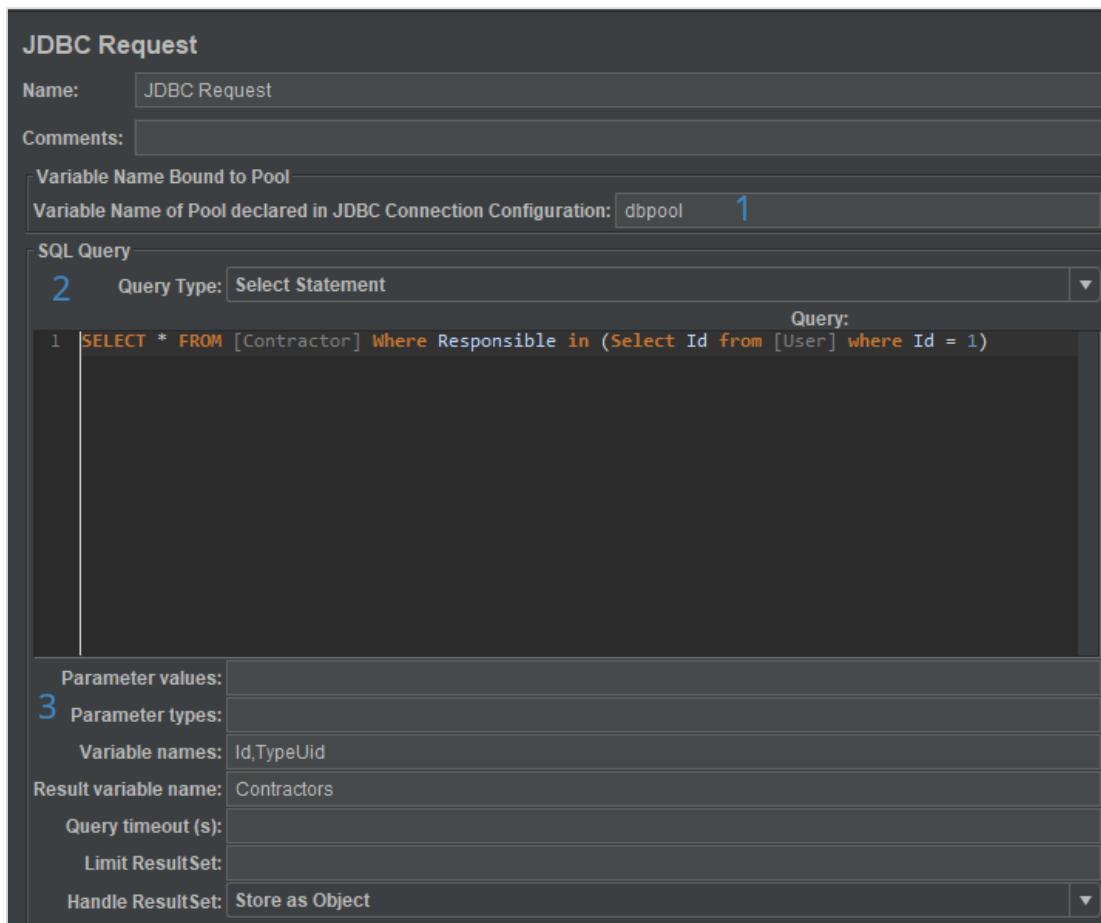


Рис. 29. JDBC Request Sampler

- **Variable Name** – название переменной, с которой связан пул сессий. Оно должно совпадать с именем переменной, определенной в JDBC Connection Configuration;
- **Query Type** – все часто используемые типы операторов (например, нельзя использовать оператор SELECT, если ваш запрос обновляет структуру базы данных);

- **SQL Query** – поле для написания SQL-запроса. На рисунке выше показан запрос с типом Select;
- **Parameter values** – список параметров SQL-запроса при условии, что он параметризован. В качестве разделителя используется запятая;
- **Parameter types** – список типов параметров SQL. В качестве разделителя используется запятая. Их следует писать в верхнем регистре, например: INTEGER, VARCHAR и т.д;
- **Variable Names** – список переменных для хранения выбранных значений данных столбцов, которые возвращаются из базы данных. В качестве разделителя используется запятая. Имена столбцов задаются в том же порядке, в котором они отображаются, например, в качестве результата выборки с помощью оператора SELECT в самой БД. При перечислении столбцов для пропуска ненужных достаточно не задавать им имени в данном поле, например: Id, TypeUid, CreationDate;
- **Result variable name** – название локальной переменной Jmeter с типом ключ-значение, из которой можно вынимать возвращенный набор данных (Variable Names);
- **Query timeout (s)** – максимальная длительность ожидания результата запроса (в секундах);
- **Handle ResultSet** – указать тип переменной для хранения результатов запроса.

Примечание: при параметризации значений параметров адреса соединения с базой данных, а также имени пользователя и пароля необходимо инициализировать механизм встроенных свойств из командной строки или из файла **user.properties**.

3.4.4.4. SMTP Sampler

SMTP Sampler используется для проверки почтового сервера, на который он отправляет email-сообщения по протоколу SMTP/SMTPS (Рис. 30).

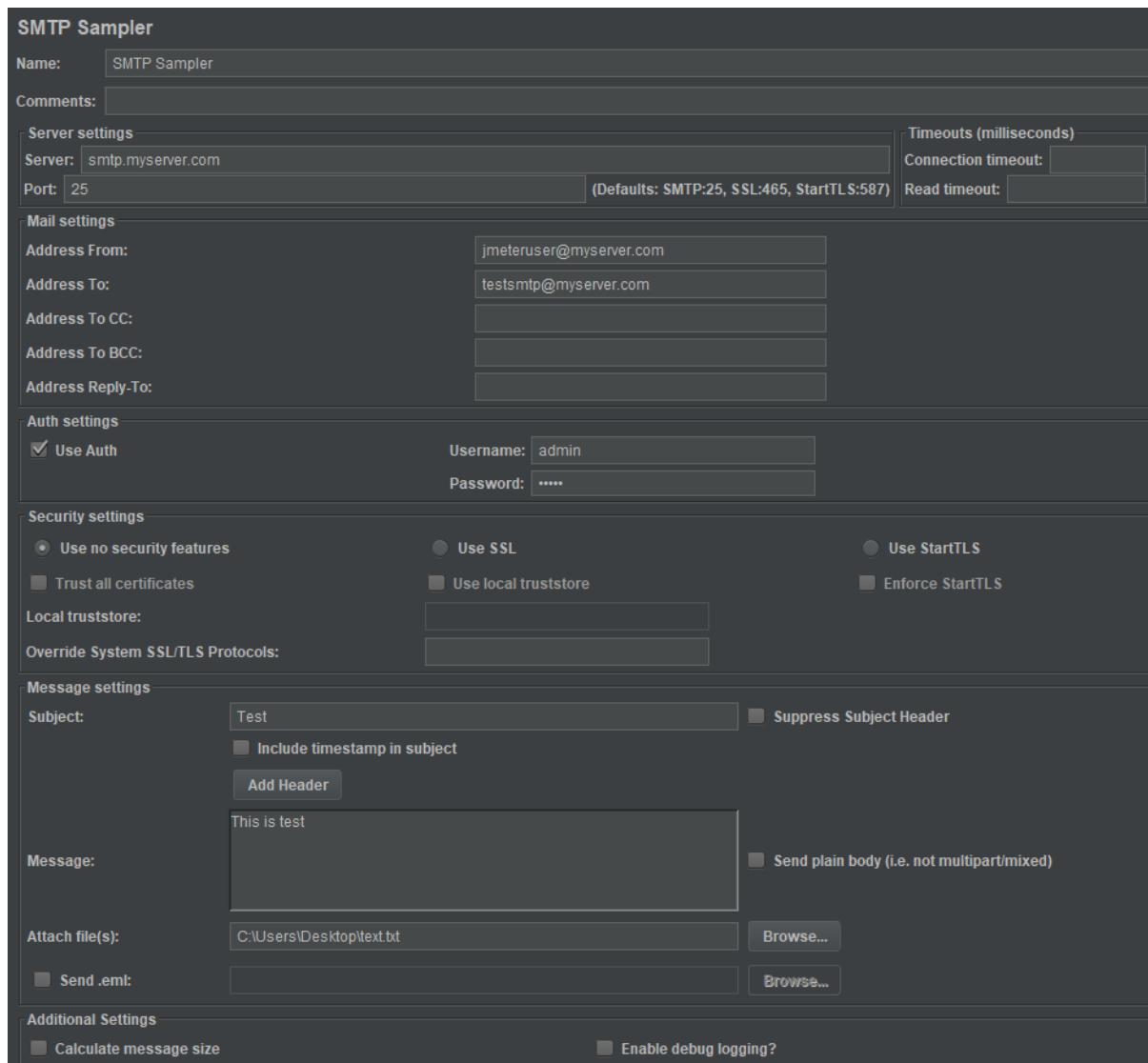


Рис. 30. SMTP Sampler

- **Server settings** – необходимо ввести адрес и порт почтового сервера;
- **Timeouts (milliseconds)** – если Jmeter не отправит email-сообщение в течение времени, указанного в данном поле, то сэмплер завершится по тайм-ауту;
- **Mail settings** – настройки email-адреса для отправки электронной почты:
 - **Address From** – необходимо указать адрес почты, откуда будет отправлен email;
 - **Address To** – необходимо указать адрес почты, куда будет отправлен email;
- **Auth settings** – учетные данные пользователя, от которого будет отправлен email;

- **Security settings** – при использовании шифрования канала связи необходимо выбрать соответствующий протокол текстового обмена;
- **Message settings** – настройки самого email, которые включают в себя тему, тело сообщения и прикрепленный файл(ы).

3.4.4.5. Debug Sampler

Элемент **Debug Sampler** используется для отображения значений, используемых в ходе прохождения теста переменных и свойств.

Данный элемент рекомендуется размещать по прохождению какого-то логического этапа тестового сценария, как правило, внутри контроллера рядом с другими элементами, в которых происходит вычисление/изменение каких-либо тестовых данных (например, пост- и предобработчиками) для отладки и последующих корректировок самого тестового сценария (Рис. 31).

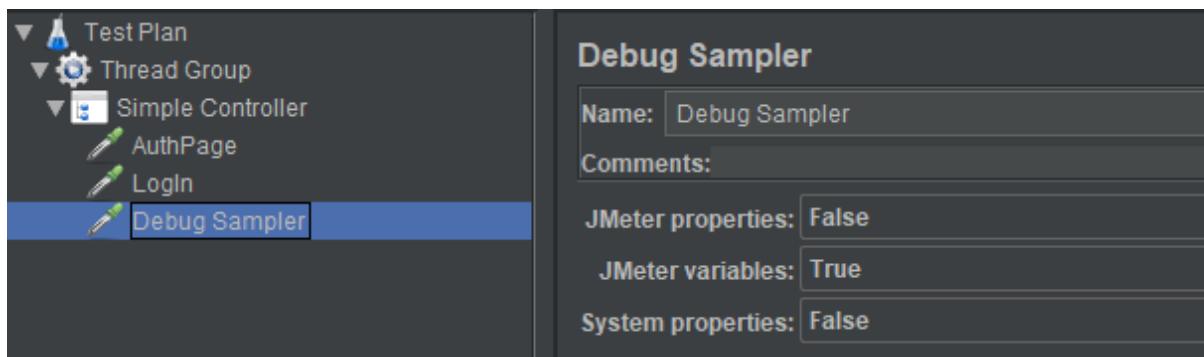


Рис. 31. Debug Sampler

- **Jmeter properties** – включает/выключает отображение информации о заданных в Jmeter свойствах в логе;
- **Jmeter variables** – включает/выключает отображение информации о заданных локальных переменных Jmeter в логе;
- **System properties** – включает/ выключает отображение информации о системных свойствах в логе.

Собранную данным сэмплером информацию можно увидеть в листенере View Results Tree во вкладке **Response Data**.

Примечание: при проведении нагрузочного тестирования рекомендуется отключать все элементы Debug Sampler.

3.4.5. Логические контроллеры

Логический контроллер (Logic Controller) позволяет выстроить порядок обработки сэмплеров в тестовом плане. **Логические контроллеры**

решают, когда и каким образом отправлять запрос на веб-сервер. Они могут быть размещены только внутри Thread Group.

К числу наиболее важных логических контроллеров относятся:

- [Loop Controller](#);
- [Random Controller](#);
- [Recording Controller](#);
- [RunTime Controller](#);
- [If Controller](#);
- [While Controller](#);
- [Simple Controller](#).

3.4.5.1. Loop Controller

Loop Controller позволяет запускать находящиеся в нем сэмплеры/запросы определенное количество раз или бесконечно (если установлен флагок **Forever**) (Рис. 32).

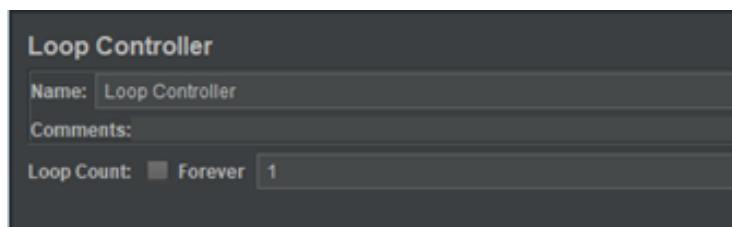


Рис. 32. Loop Controller

- **Loop Count Forever** – при установке данного параметра находящиеся внутри контроллера элементы будут выполняться бесконечное число раз;
- Если задать в поле **Loop Count** определенного числа, то находящиеся внутри контроллера элементы будут выполнены указанное число раз.

Например: в тестовом плане установлено значение "2" для **Thread Group Loop Count**, для **Loop Controller Loop Count** также установлено значение "2" и 3 запроса находятся внутри **Loop Controller**. Таким образом, за полное прохождение теста Jmeter отправит в общей сложности 12 запросов:

Количество раз, которые будет выполняться тестовый сценарий внутри Thread Group (2) * количество циклов Loop Controller'a (2) * количество запросов внутри Loop Controller'a (3).

3.4.5.2. Random Controller

Random Controller позволяет запускать находящиеся внутри него сэмплеры/запросы в случайном порядке для каждого цикла (Рис. 33).

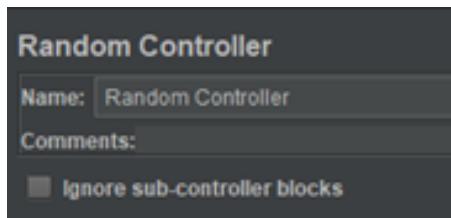


Рис. 33. Random Controller

- **Ignore sub-controller blocks** – при включении данного параметра, **Random controller** будет обрабатывать вложенные в него контроллеры как отдельные элементы запроса и разрешать только один запрос на контроллер за раз.

3.4.5.3. Recording Controller

В данный элемент будут сохраняться все действия, которые будет записывать элемент **HTTP(S) Test Script Recorder** (для него **Recording Controller** используется по умолчанию) (Рис. 34).

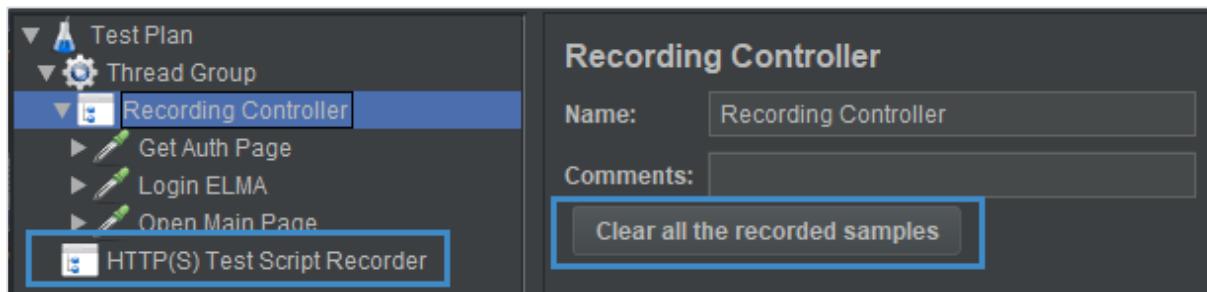


Рис. 34. Recording Controller

Внутри контроллера формируются записанные сэмплеры/запросы.

- **Clear all the recorded samples** – данная кнопка очищает все сэмплеры/запросы, записанные внутри Recording Controller.

3.4.5.4. RunTime Controller

RunTime Controller контролирует выполнение расположенных внутри сэмплеров/запросов в течение заданного времени (Рис. 35).

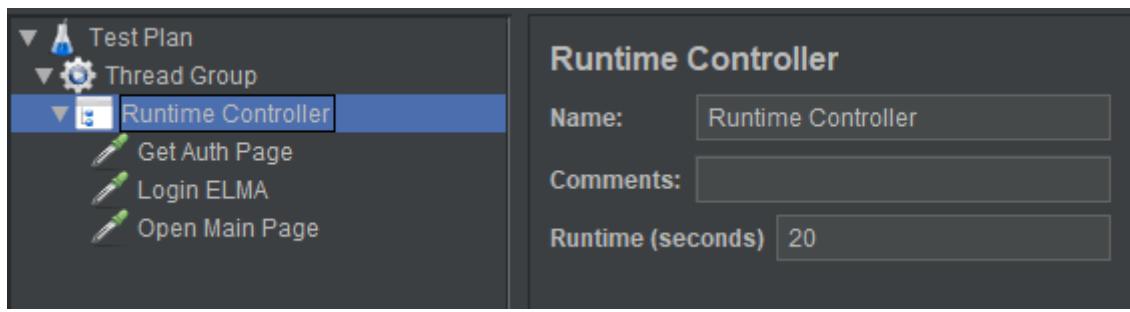


Рис. 35. RunTime Controller

- **Runtime (seconds)** – поле для ввода времени (в секундах), в течение которого будут выполняться расположенные внутри контроллера сэмплеры/запросы.

3.4.5.5. If Controller

IF Controller позволяет задать условие выполнения вложенных в него элементов. По умолчанию условие в элементе **If Controller** проверяется только один раз при входе в контроллер, но существует настройка, при которой заданное условие будет проверяться для каждого элемента, расположенного внутри него.

Рекомендуемой настройкой проверки выражения внутри данного элемента (она же настройка по умолчанию) является "**Interpret Condition as Variable Expression**" (описать условие в качестве выражения, содержащего переменные), при этом само условие можно задать двумя способами:

1. Использовать переменную, содержащую true или false. Например, выражение `${JmeterThread.last_sample_ok}` будет проверять, было ли успешным прохождение предыдущего запроса (Sample) (Рис. 36).

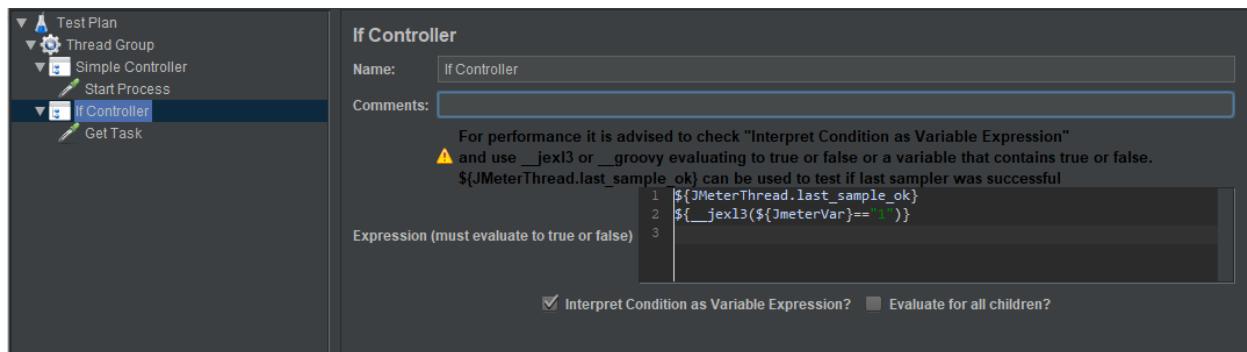


Рис. 36. Пример использования условия в элементе "If Controller"

2. Использовать встроенную в Jmeter функцию `(${__jexl3()})` для вычисления выражения, которое должно возвращать true или false.

Evaluate for all children? – при выборе данного параметра проверка условия будет производиться как при входе в сам контроллер, так и для всех его вложенных элементов.

3.4.5.6. While Controller

While Controller позволяет задать поведение тестового плана, которое происходит при определенном условии и лучше всего подходит для сложных сценариев, которые имитируют реалистичное поведение пользователя. Элементы внутри контроллера выполняются ноль, один или несколько раз в зависимости от того, сколько раз выполняется заданное в нем условие. При завершении всех вложенных элементов контроллера, еще раз перепроверяется его условие. Если условие выполнено – то все вложенные элементы контроллера выполняются снова до тех пор, пока условие не будет нарушено.

При необходимости в While Controller существует возможность установить фиксированное число повторений для заданного набора действий. Для этого внутри While Controller необходимо разместить элемент Counter, который находится в категории Config Element. Counter создает динамически изменяющую числовую характеристику, главная задача которой отследить число повторений того или иного элемента теста на текущий момент.

Пример:

1. Настроим элемент Counter следующим образом:

- Starting value – 1;
- Increment – 1;
- Maximum value – не задано;
- Reference Name – counter.

2. Зададим следующее условие для While Controller с использованием javaScript (Рис. 37).

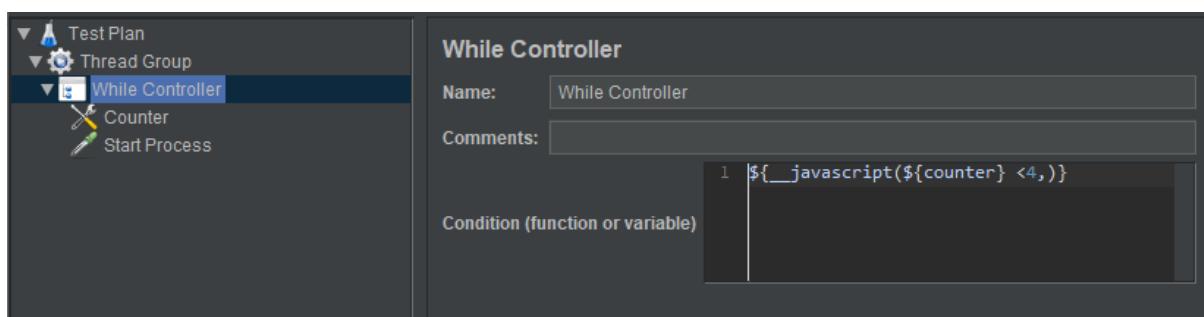


Рис. 37. Условие для While Controller с использованием javaScript

Таким образом, в данном примере элементы, расположенные внутри **While Controller**, будут выполнены 4 раза.

3.4.5.7. *Simple Controller*

Simple Controller предназначен для хранения сэмплеров и других логических контроллеров в заданном порядке. В этом заключается его основная задача и, по сравнению с другими контроллерами, другой функциональности он не имеет (Рис. 38).

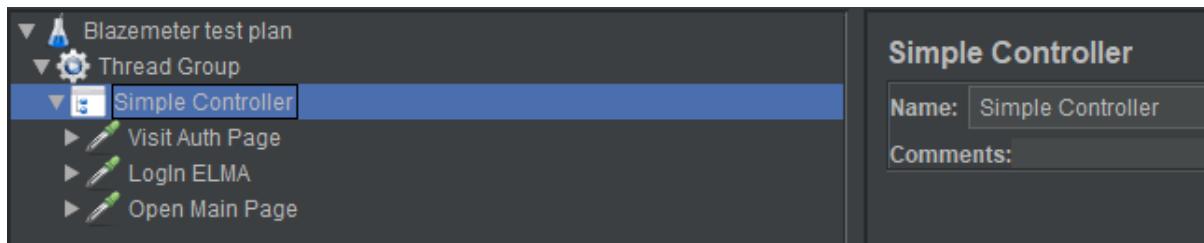


Рис. 38. *Simple Controller*

Сэмплеры, расположенные внутри Simple Controller, идут в строго заданном порядке.

3.4.6. Препроцессоры

Препроцессоры (Pre-Processor) – это элементы Jmeter, которые используются для выполнения действий до выполнения запросов сэмплера в тестовом сценарии. Препроцессоры могут использоваться для различных задач тестирования производительности, таких как выборка данных из базы данных, установка тайм-аута между выполнением сэмплера или перед генерацией тестовых данных.

Среди самых важных препроцессоров можно выделить следующие:

- [BeanShell PreProcessor](#);
- [JSR223 PreProcessor](#);
- [JDBC PreProcessor](#).

3.4.6.1. *BeanShell PreProcessor*

Предположим, что нам необходимо протестировать POST-запрос отправки формы запуска экземпляра процесса, в котором необходимо передать параметр **Название экземпляра процесса**. Этот параметр может быть любой случайной строкой. Можно использовать статическое значение для данной строки для всех запросов, но это сделало бы тест менее

реалистичным, чем реальный сценарий, поскольку название экземпляра процесса формируются на основание определенных контекстных данных, и, как правило, изменяется для каждого пользователя. Для этого необходимо воспользоваться элементом написания сценариев на языке BeanShell, который является скриптовым языком для Java, **BeanShell PreProcessor**, для генерации случайной строки и дальнейшего ее использования в сэмплере. Для этого его необходимо разместить в качестве вложенного элемента данного сэмплера (Рис. 39).

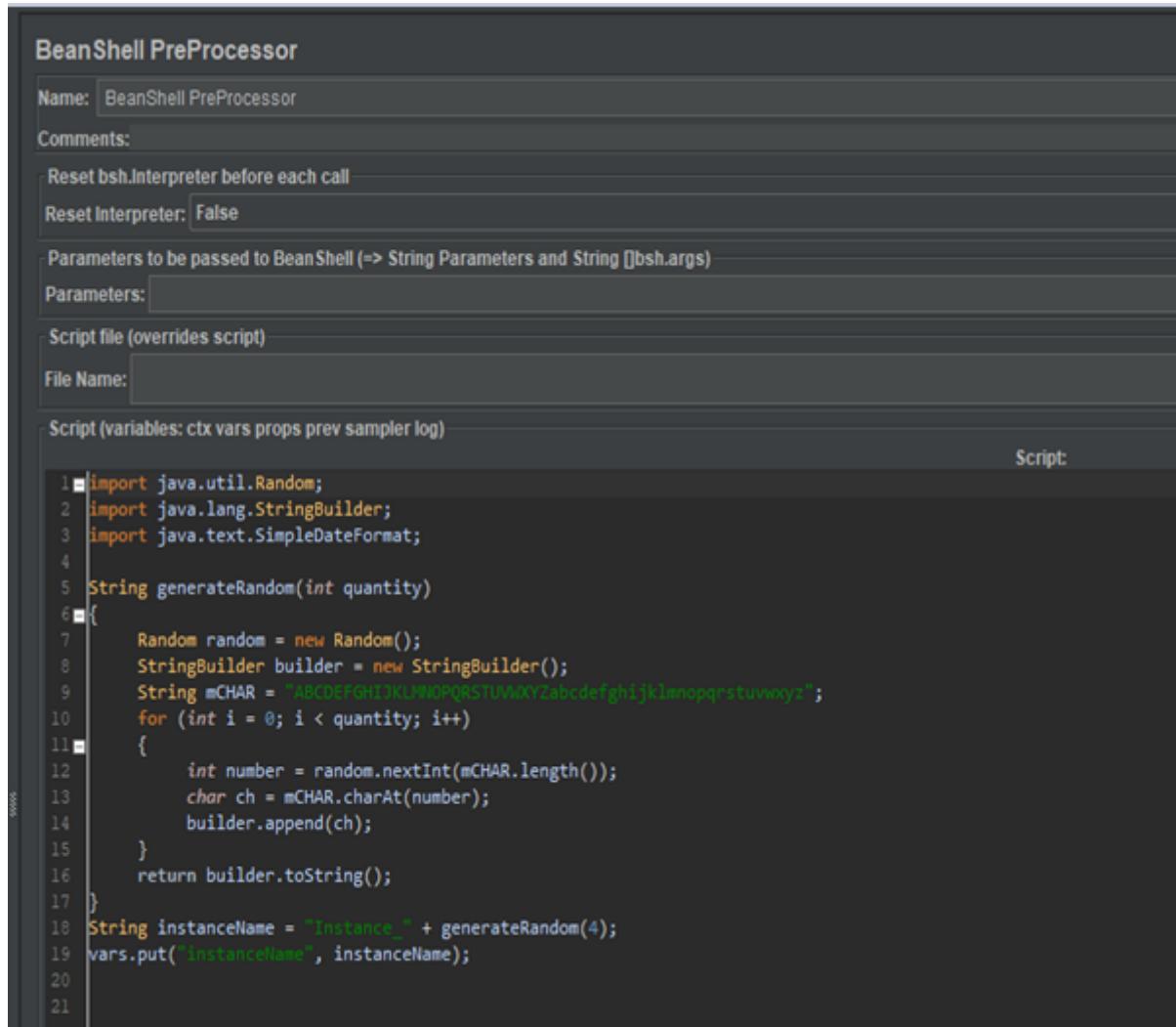


Рис. 39. BeanShell PreProcessor

- **Reset bsh.Interpreter before each call** сбрасывает перед каждым вызовом, тем самым очищает занятую им память. Установка данного параметра в "True" может быть полезна для требовательных к памяти сценариев, так как повторные вызовы данного элемента могут занимать много памяти;

- **Parameters** – JMeter parameters that will be passed to the BeanShell script
– параметры Jmeter, которые будут переданы в скрипт BeanShell. Необходимо помнить, что переменные Jmeter не могут быть использованы в BeanShell PreProcessor, если они не указаны в этом поле конфигурации. Если параметр указан, его можно использовать в препроцессоре следующим образом: String BS_Variable_Name = vars.get("JMeterVariable");
- **File Name** – путь к внешнему скрипту BeanShell, который необходимо запустить. Сценарий препроцессора BeanShell может быть внешним или внутренним. Если он внутренний, вы можете записать его в поле **Сценарий** препроцессора BeanShell.

3.4.6.2. JSR223 PreProcessor

Препроцессор JSR223 – другой вариант написания сценариев с использованием препроцессоров. JSR223 (от англ. Java Specification Request – запрос на спецификацию Java) имеет схожую функциональность с BeanShell PreProcessor, однако основное его отличие в том, что вы можете использовать дополнительные языки сценариев, такие как: ecmascript, groovy, java, javascript, jexl и nashorn.

Примечание: при создании тестового плана в Jmeter качестве элемента написания сценариев рекомендуется использовать именно его, поскольку он обеспечивает более высокую производительность по сравнению с BeanShell PreProcessor.

Чтобы показать JSR223 препроцессор в действии, перепишем ранее созданный скрипт для создания случайной строки наименования экземпляра процесса, который был использован в BeanShell.

Примечание: в данном примере сценарий будет написан на JavaScript, но рекомендуется использовать Groovy (Рис. 40).

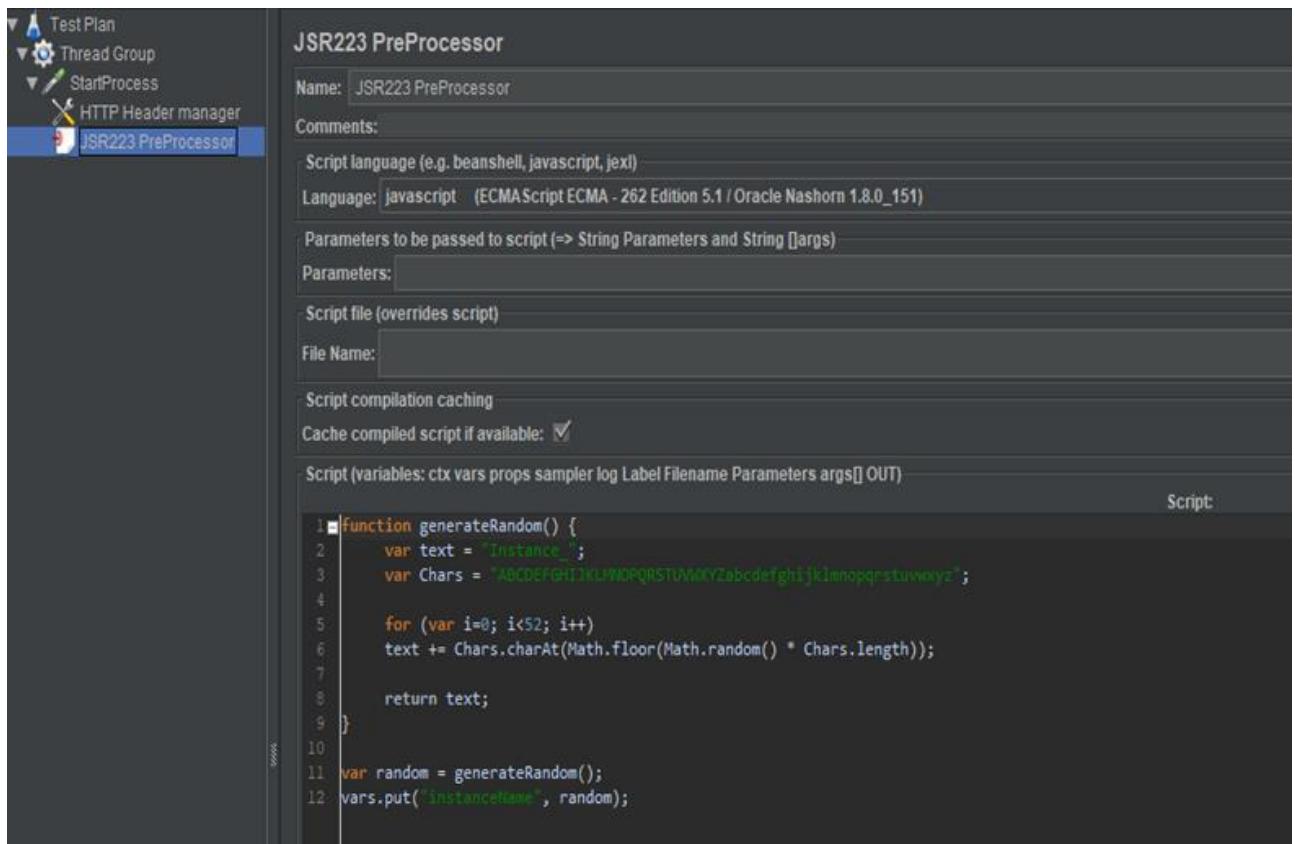


Рис. 40. Использование скрипта для препроцессора JSR223

- **Language** – выбор языка написания сценария (Рис. 41).

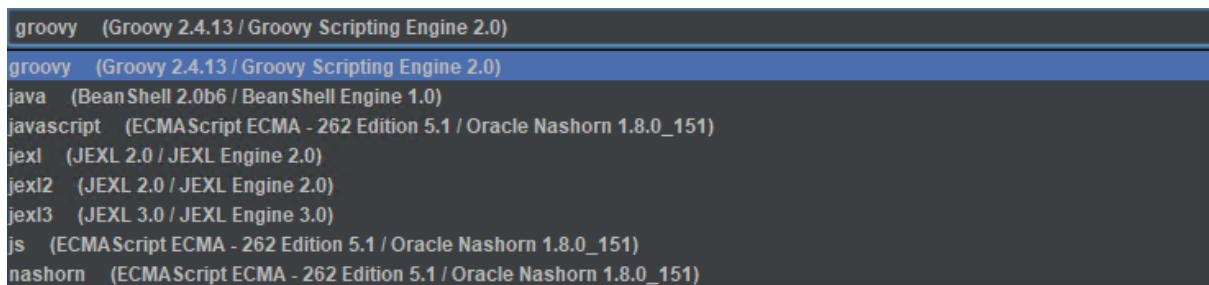


Рис. 41. Выбор языка написания сценария

- **Parameters to be passed to script** – параметры Jmeter, которые будут переданы в скрипт JSR223. Используйте пробел в качестве разделителя при передаче нескольких переменных.
В самом скрипте к переменным, записанным в разделе **Parameters**, можно обратиться в виде аргументов в формате **args[0]**, где **[0]** – индекс первой переменной раздела (при передаче нескольких переменных, разделенных пробелом, вы можете ссылаться на вторую, как **args[1]**, третью – **args[2]** и так далее);

- **File Name** – путь к внешнему скрипту JSR223, который необходимо запустить. Сценарий препроцессора JSR223 может быть внешним или внутренним. Если он внутренний, вы можете записать его в поле **Сценарий** препроцессора JSR223;
- **Cache compiled script if available** – при использовании данной функции включается кэширование скомпилированного кода, что может значительно повысить производительность. Рекомендуется всегда использовать данную функцию для языка написания сценариев, который поддерживает такое кэширование, например, **Groovy**.

Примечание: при использовании данной настройки убедитесь, что не используете переменные JMeter в коде скрипта напрямую, поскольку кэширование будет принимать только первое значение для них и при последующем выполнении скрипта оно изменяться не будет. Вместо этого используйте [параметры](#).

3.4.6.3. JDBC PreProcessor

Элемент **JDBC PreProcessor** позволяет отправить SQL-запрос с использованием определенного оператора прямо перед запуском сэмплера. Настройки данного элемента идентичны настройкам [JDBC Request Sampler](#).

3.4.7. Пост-обработчики

Пост-обработчики (Post-Processors) – элементы Jmeter, которые выполняются после завершения сэмплеров и используются, главным образом, для обработки данных ответа сервера и извлечения конкретного значения разметки для его последующего использования в teste. Если вам необходимо использовать пост-обработчик для определенного сэмплера, то необходимо добавить его в качестве вложенного элемента данного сэмплера.

Далее перечислены наиболее часто используемые пост-обработчики:

- [BeanShell PostProcessor](#);
- [JSR223 PostProcessor](#);
- [JDBC PostProcessor](#);
- [CSS/JQuery Extractor](#);
- [Xpath Extractor](#);

- [Regular Expression Extractor](#);
- [Debug PostProcessor](#).

3.4.7.1. BeanShell PostProcessor

Настройки данного элемента такие же, как у [BeanShell PreProcessor](#) с разницей лишь в том, что элемент будет выполняться после завершения связанного с ним сэмплера.

3.4.7.2. JSR223 PostProcessor

Настройки данного элемента такие же, как у [JSR223 PreProcessor](#) с разницей лишь в том, что элемент будет выполняться после завершения связанного с ним сэмплера.

3.4.7.3. JDBC PostProcessor

Настройки данного элемента такие же, как у [JDBC Request Sampler](#) с разницей лишь в том, что элемент будет выполняться после завершения связанного с ним сэмплера.

3.4.7.4. CSS/JQuery Extractor

Элемент **CSS/JQuery Extractor** является одним из наиболее удобных инструментов для извлечения и обработки данных из тела ответа сэмплера, благодаря общепринятым синтаксису CSS. Созданные в нем с помощью CSS-селекторов или JQuery выражения извлекают все совпадающие значения из HTML-кода страницы ответа и записывают их в заданную локальную переменную Jmeter, которая может быть использована в последующих запросах.

Элемент необходимо разместить внутри сэмплера, тело ответа которого необходимо отправить в CSS/JQuery Extractor для анализа (Рис. 42).

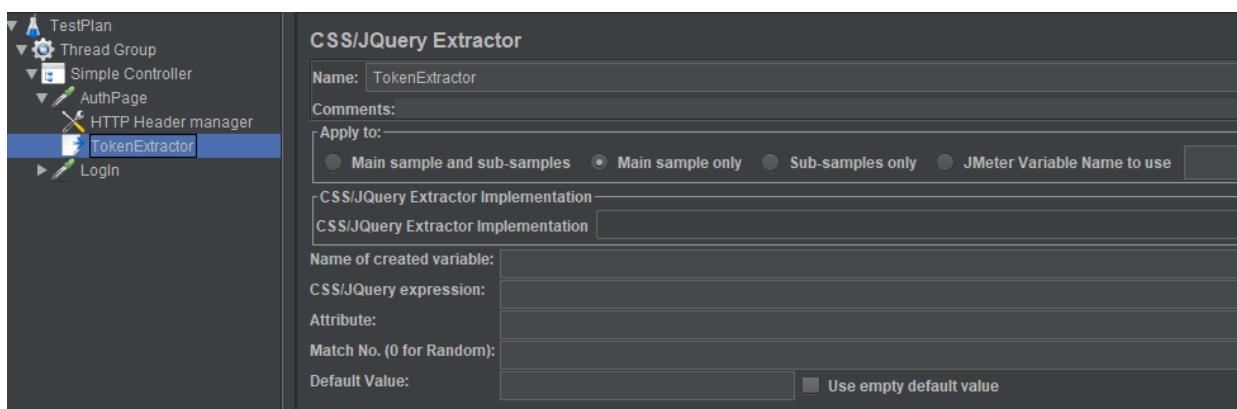


Рис. 42. Элемент "CSS/JQuery Extractor"

- **Apply to** – данный параметр полезен, если у текущего сэмплера имеются дочерние сэмплеры:
 - **Main sample and sub-samples** – применять Extractor к основному сэмплеру и к вложенным сэмплерам;
 - **Main sample only** – применять Extractor только к основному сэмплеру;
 - **Sub-samples only** – применять Extractor только к вложенным сэмплерам;
 - **Jmeter Variable Name to use** – применить Extractor к указанной переменной Jmeter;
- **CSS/JQuery Extractor Implementaion** – существуют две реализации CSS/JQuery Extractor: JSOUP (используется по умолчанию) и JODD. Хотя между ними нет существенного различия, некоторые элементы синтаксиса могут отличаться;
- **Name of created variable** – имя переменной Jmeter, в которую будет записано полученное значение;
- **CSS/JQuery Expression** – выражение, которое будет использовано для извлечения совпадающих значений;
- **Attribute** – название атрибута html, который будет использован для извлечения значения из элементов кода страницы, соответствующих заданному выше выражению;
- **Match No. (0 for Random)** – в зависимости от значения данного параметра CSS/JQuery селектором будет выбран:
 - 0 – случайный среди всех найденных результат;
 - 1 – первый (самый верхний) результат в массиве; 2 – второй (если результатов 2 и более) и т.д;
- **Default Value** – значение по умолчанию переменной JMeter в случае, если селектор не будет найден.

3.4.7.5. Xpath Extractor

Элемент **Xpath Extractor** – инструмент для извлечения и обработки данных из тела ответа сэмплера с помощью заданного в нем Xpath выражения. Он оказывается весьма полезным, когда с помощью других экстракторов получить необходимую информацию достаточно трудно. Например, в случае, когда на странице имеется несколько похожих тегов без атрибутов, но с разными значениями Рис. 43.

```
<div id="numbers">
    <div>uniq_value_01</div>
    <div>uniq_value_02</div>
    <div>uniq_value_03</div>
</div>
```

Рис. 43. Случай с похожими тегами без атрибутов, но с разными значениями

Элемент необходимо разместить внутри сэмплера, тело ответа которого необходимо отправить в **Xpath Extractor** для анализа (Рис. 44).

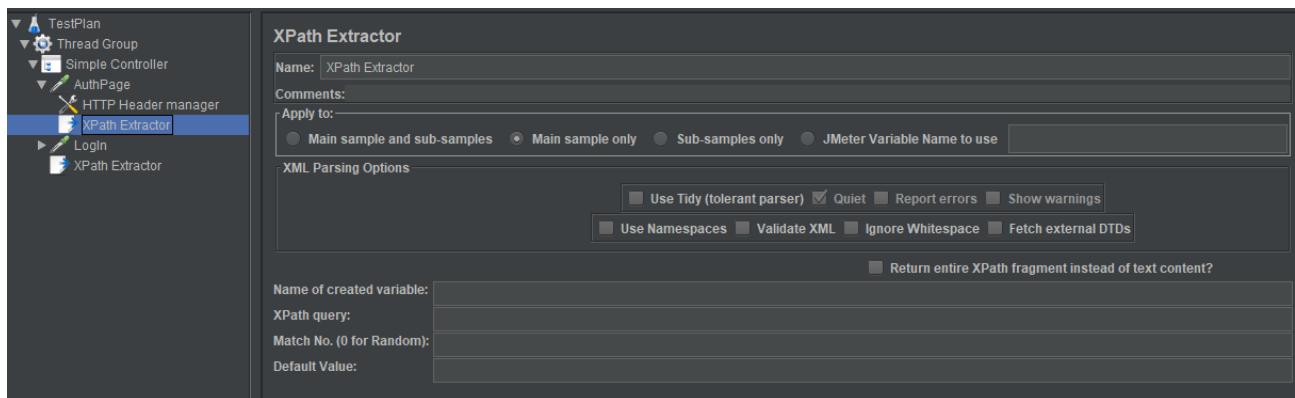


Рис. 44. Элемент "XPath Extractor"

- **Apply to** – данный параметр полезен, если у текущего сэмплера имеются дочерние сэмплеры:
 - **Main sample and sub-samples** – применять Extractor к основному сэмплеру и к вложенным сэмплерам;
 - **Main sample only** – применять Extractor только к основному сэмплеру;
 - **Sub-samples only** – применять Extractor только к вложенным сэмплерам;
 - **Jmeter Variable Name to use** – применить Extractor к указанной переменной Jmeter;
- **XML Parsing Options** –варианты парсинга XML:
 - **Use Tidy (tolerant parser)** – необходимо задать данный параметр, чтобы преобразовать приходящий HTML-код страницы в XHTML (XML-совместимый HTML). Таким образом, используйте его при получении неверных XML-ответов;
- **Return entire XPath fragment instead of text content?** – при выборе данного параметра вместо текстового содержимого вернется сам

фрагмент целиком. Например //title вместо "Добро пожаловать в систему ELMA!" вернет "<title>Добро пожаловать в систему ELMA! - ELMA</title>".

- **Name of created variable** – имя переменной Jmeter, в которую будет записано полученное значение;
- **XPath query** – выражение XPath, которое будет использовано для извлечения совпадающих значений;
- **Match No. (0 for Random)** – в зависимости от значения данного параметра Xpath селектором будет выбран:
 - 0 – случайный среди всех найденных результатов;
 - 1 – первый (самый верхний) результат в массиве; 2 – второй (если результатов 2 и более) и т.д.;
- **Default Value** – значение по умолчанию переменной JMeter в случае, если локатор не будет найден.

3.4.7.6. Regular Expression Extractor

Регулярные выражения (англ. *regular expressions*) – формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов, которые указывают, что должна быть найдена некоторая необычная вещь, или влияют на другие части регулярного выражения, повторяя или изменяя их значение. Для поиска используется строка-образец (шаблон), состоящая из символов и метасимволов и задающая правило поиска. Для манипуляций с текстом дополнительно задаётся строка замены, которая также может содержать в себе специальные символы.

В регулярных выражениях используются следующие метасимволы: . ^ \$ * + ? { [] \ | ()

- **Метасимволы соответствия**

[] – используются для определения класса символов, являющегося набором символов, с которыми ищутся совпадения. Символы могут быть перечислены в качестве диапазона, обозначенного первым и последним символом, разделенным знаком -. Например, [abc] будет соответствовать любому из символов a, b или c, что является тем аналогичным выражению [a-c].

^ – данный метасимвол добавляется, чтобы находить соответствие символов вне определенного класса. Например, выражение [^5] соответствует любому символу, кроме 5.

**** – данный метасимвол используется для экранирования метасимволов, чтобы их можно было использовать в шаблонах. Например, если нужно найти соответствие [или], для того, чтобы лишить их своей особой роли, перед ними нужно поставить \, тем самым экранировать их: \[или]\.

Также с его помощью задаются специальные последовательности символов, например:

\d – соответствует любой цифре; эквивалент класса [0-9].

\D – соответствует любому нечисловому символу; эквивалент класса [^0-9].

\s – соответствует любому символу whitespace (пробел, перевод строки и табуляция); эквивалент [\t\n\r\f\v].

\S – соответствует любому не-whitespace символу; эквивалент [^\t\n\r\f\v]

\w – соответствует любой букве или цифре; эквивалент [a-zA-Z0-9_].

\W – наоборот; эквивалент [^a-zA-Z0-9_].

Такие последовательности могут включены в класс символов. Например, [\s,.] является выражением, которое будет соответствовать любому whitespace-символу или запятой, или точке.

. – метасимвол, который соответствует всем символам, кроме символа новой строки.

- **Метасимволы повторения**

***** – метасимвол повторения, который указывает, что предыдущий символ может быть сопоставлен 0 и более раз, вместо одного сравнения. Например, ca*t будет соответствовать ct (0 повторений), cat (1 повторение), caaat (3 повторения а) и тд.

+ – метасимвол повторения, который повторяет последовательность сравнения один или более раз. В отличие от него, метасимвол * требует соответствия необходимой части ноль или более раз, т.е повторяемое может и не присутствовать вовсе. Для аналогичного примера ca+t будет

сопоставляться cat (1 повторение) или, например, caaat (3 повторения a), но никак не ct.

? – метасимвол повторения, проверяющий наличие совпадения ноль или один раз. Например, home-?brew соответствует, как homebrew, так и home-brew.

{m,n} – метасимвол повторения, где m и n – целые числа. Такой определитель означает, что здесь должно быть не менее m и не более n повторений. Например, выражение a/{1,3}b соответствует a/b, a//b и a///b, но не может быть ab (строкой, в которой нет слэшей) или a///b (строкой, в которой их 4).

Элемент Regular Expression Extractor предназначен для обработки и извлечения данных из тела ответа сэмплера с помощью заданного в нем регулярного выражения (Рис. 45).

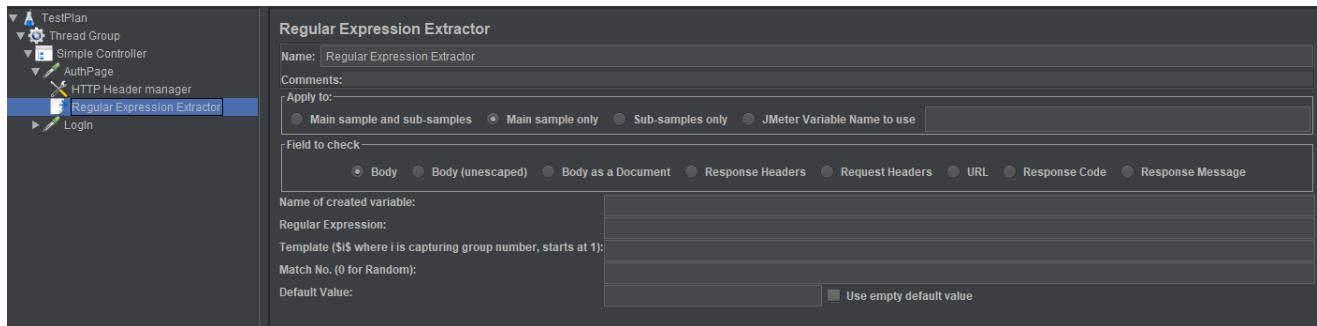
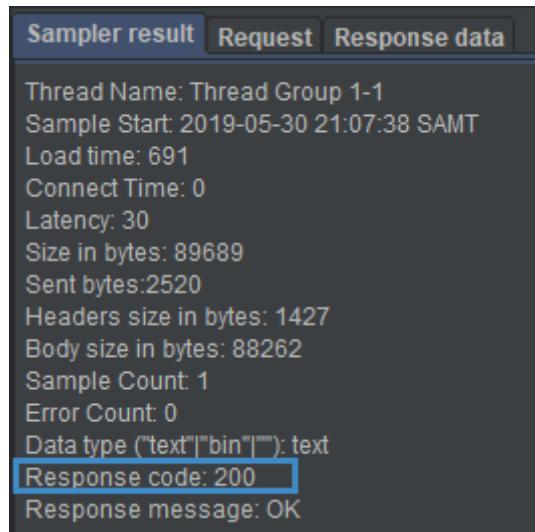


Рис. 45. Элемент "Regular Expression Extractor"

- **Apply to** – данный параметр полезен, если у текущего сэмплера имеются дочерние сэмплеры:
 - **Main sample and sub-samples** – применять Extractor к основному сэмплеру и к вложенным сэмплерам;
 - **Main sample only** – применять Extractor только к основному сэмплеру;
 - **Sub-samples only** – применять Extractor только к вложенным сэмплерам;
 - **Jmeter Variable Name to use** – применить Extractor к указанной переменной Jmeter;
- **Field to check** – данная настройка позволяет установить, какая часть ответа сэмплера должна быть проанализирована:
 - **Body** – тело ответа. Будет проанализировано весь контент веб-страницы, за исключением заголовков;

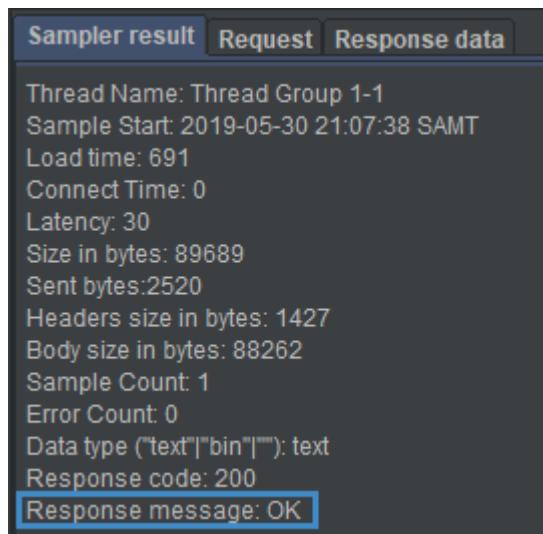
- **Body (unesaped)** – тело ответа с заменой всех символов экранирования;
- **Response Headers** – будут проанализированы только заголовки ответа;
- **Request Headers** – будут проанализированы только заголовки запроса;
- **URL** – будет проанализирован URL запроса;
- **Response Code** – будет проанализирован код ответа, например, 200 (Рис. 46).



Sampler result	Request	Response data
Thread Name: Thread Group 1-1		
Sample Start: 2019-05-30 21:07:38 SAMT		
Load time: 691		
Connect Time: 0		
Latency: 30		
Size in bytes: 89689		
Sent bytes: 2520		
Headers size in bytes: 1427		
Body size in bytes: 88262		
Sample Count: 1		
Error Count: 0		
Data type ("text" "bin" ""): text		
Response code: 200		
Response message: OK		

Рис. 46. Код ответа

- **Response Message** – будет проанализировано только сообщение в ответе, например, OK (Рис. 47).



Sampler result	Request	Response data
Thread Name: Thread Group 1-1		
Sample Start: 2019-05-30 21:07:38 SAMT		
Load time: 691		
Connect Time: 0		
Latency: 30		
Size in bytes: 89689		
Sent bytes: 2520		
Headers size in bytes: 1427		
Body size in bytes: 88262		
Sample Count: 1		
Error Count: 0		
Data type ("text" "bin" ""): text		
Response code: 200		
Response message: OK		

Рис. 47. Сообщение в ответе

- **Name of created variable** – имя переменной Jmeter, в которую будет записано полученное значение;
- **Regular Expression** – поле для ввода регулярного выражения, по которому будет осуществлен анализ;
- **Template** – шаблон, в котором необходимо выбрать группу, которую необходимо извлечь из результатов анализа регулярного выражения. **\$1\$** – извлекает группу 1, **\$2\$** – извлекает группу 2 и так далее. **\$0\$** – извлекает все результаты целиком. Например, если в ответе имеется слово header, а регулярное выражение задано “**(head)(er)**”, при применении шаблона **\$1\$** в выходной переменной Вы получите “head”, при **\$2\$** – Вы получите “er”. Тогда как если Вы примените шаблон **{0}**, то получите “header” целиком.
- **Match No.** – при наличии нескольких соответствий заданному регулярному выражению позволяет выбрать, какой конкретно вариант следует извлекать. При выборе значения “0” – вариант будет выбран случайно.

3.4.7.7. Debug PostProcessor

Элемент **Debug PostProcessor** выполняет те же самые функции, что и Debug Sampler, за исключением лишь того, что его можно использовать в качестве дочернего элемента определенного сэмплера, чтобы анализировать только его информацию (Рис. 48).

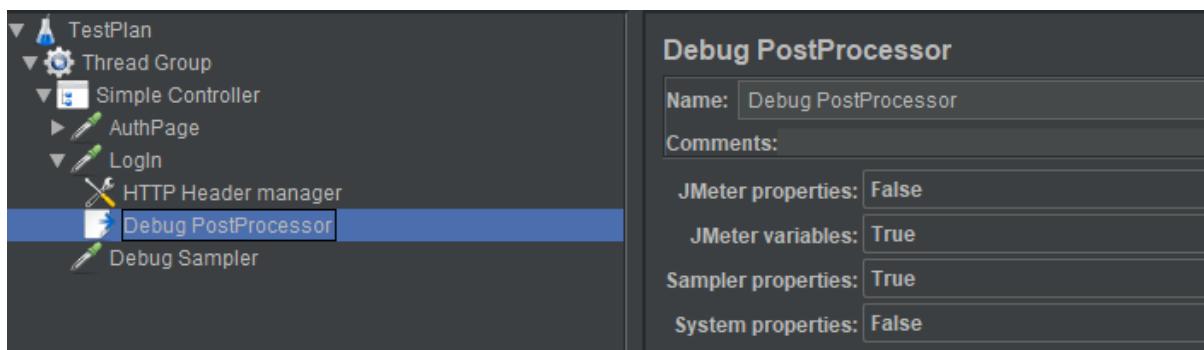


Рис. 48. Элемент “Debug PostProcessor”

- **Jmeter properties** – включает/выключает отображение информации о заданных в Jmeter свойствах в логе;
- **Jmeter variables** – включает/выключает отображение информации о заданных локальных переменных Jmeter в логе;
- **Sampler properties** – включает/выключает отображение информации о свойствах родительского сэмплера в логе;

- **System properties** – включает/ выключает отображение информации о системных свойствах в логе.

Собранную данным элементом информацию можно увидеть в листенере View Results Tree во вкладке **Response Data**.

3.4.8. Таймеры

Отправляя запросы на веб-сервер, Jmeter не применяет какую-либо задержку между выполнением сэмплеров. Однако для получения реалистичных результатов проведения тестирования производительности необходимо опираться на реальное время, которое пользователь затрачивает до выполнения следующей операции, тем самым имитируя опыт реального сетевого взаимодействия между пользователем и веб-сервером. Для имитации пользовательского ожидания в Jmeter используют **таймеры (Timers)**. В тестовом элементе таймер размещается, как правило, между сэмплерами для применения заданного в нем времени ожидания.

Примечание: при необходимости проведения тестирования производительности в режиме стресс-теста время на обдумывание не задается.

К числу наиболее важных таймеров относятся:

- [**Constant Timer**](#);
- [**Uniform Random Timer**](#);
- [**Gaussian Random Timer**](#);
- [**JSR223 Timer**](#);
- [**BeanShell Timer**](#).

3.4.8.1. Constant Timer

Элемент **Constant Timer** применяется для генерации постоянной задержки перед выполнением каждого запроса (Рис. 49).



Рис. 49. Элемент "Constant Timer"

- **Thread Delay (in milliseconds)** – задать значение задержки (в миллисекундах), которая будет применяться перед выполнением запроса. Например, при вводе в данное поле значения 3000 будет добавлена 3-секундная задержка. В данном поле также можно использовать функцию JMeter или локальную переменную.

3.4.8.2. Uniform Random Timer

Элемент **Uniform Random Timer** применяется для генерации случайной задержки перед выполнением каждого запроса (Рис. 50).

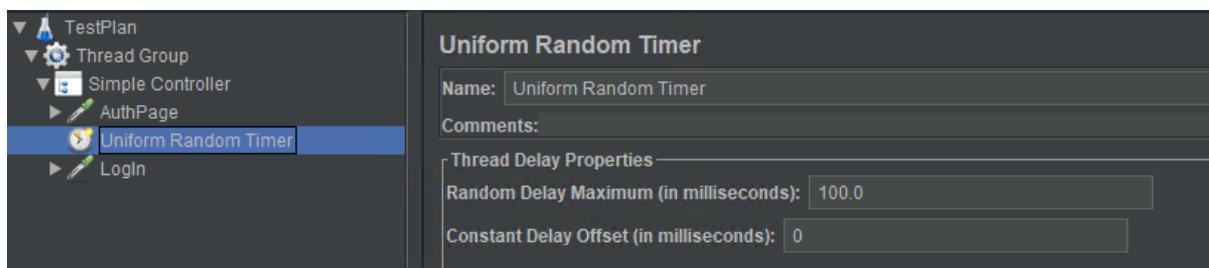


Рис. 50. Элемент "Uniform Random Timer"

- **Random Delay Maximum (in milliseconds)** – задать максимальное случайное время задержки (в миллисекундах). Например, при вводе в данное поле значения 2000 время задержки будет выбираться из диапазона от 0 до 2 секунд;
- **Constant Delay Offset (in milliseconds)** – задать постоянное время задержки (в миллисекундах), которое будет добавлено к случайному времени.

3.4.8.3. Gaussian Random Timer

Элемент **Gaussian Random Timer** применяется для генерации случайной задержки перед выполнением каждого запроса. В отличие от Uniform Random Timer, к постоянному времени задержки, заданному в данном элементе, применяется случайное отклонение на основе функции распределения Гаусса (GF) (Рис. 51).

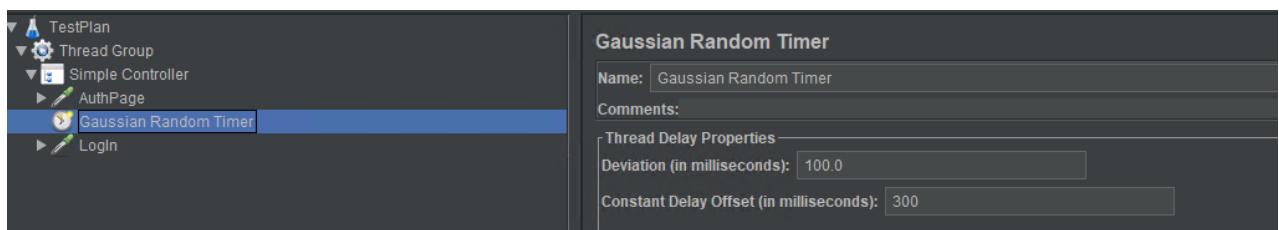


Рис. 51. Элемент "Gaussian Random Timer"

- **Deviation (in milliseconds)** – задать значение отклонения (в миллисекундах);
- **Constant Delay Offset (in milliseconds)** – задать постоянное время задержки (в миллисекундах).

Например: при значении Deviation в 100 мс, а Constant Delay Offset – 300 мс, то для 68% всех задержек таймер сгенерирует случайное число в диапазоне от 200 до 400 мс, что следует из формулы:

Constant Delay Offset – GF * Deviation), где значение GF колеблется от -1 до 1.

3.4.8.4. JSR223 Timer

JSR223 Timer – таймер, задаваемый с помощью сценария. В нем нужно самостоятельно реализовать логику вычисления задержки между выполнением запросов, используя один из поддерживаемых языков сценариев, таких как Groovy, Beanshell, java, javascript и тд. Использование данного таймера полезно в случае, если необходимо вычислить время ожидания на основе какого-то уникального алгоритма, который в настоящее время отсутствует в Jmeter. С помощью данного таймера можно создать собственную реализацию такого алгоритма (Рис. 52).

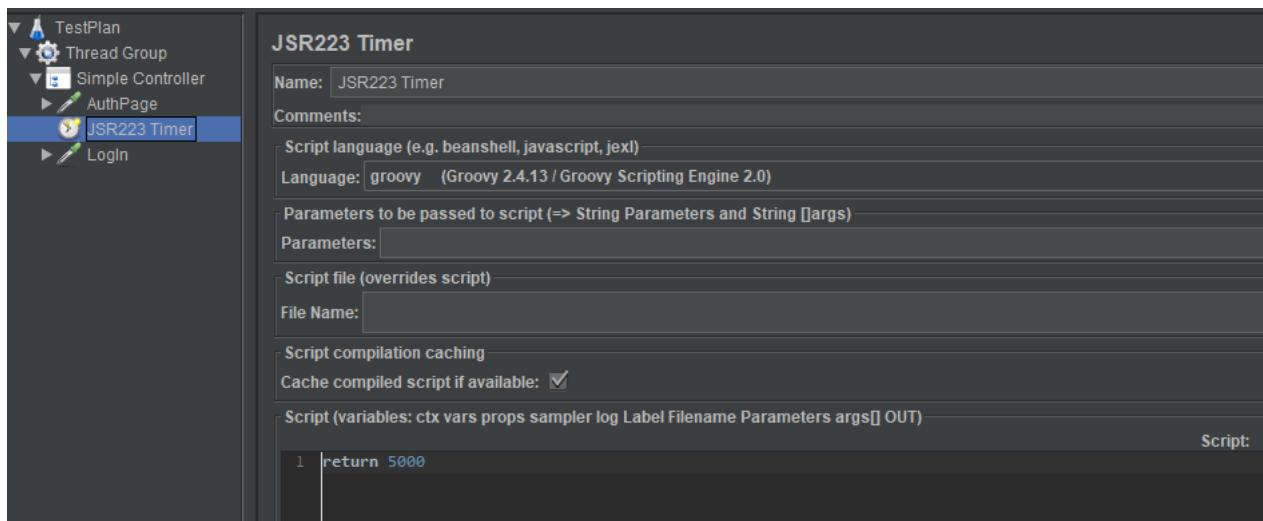


Рис. 52. JSR223 Timer

Настройки данного элемента идентичны настройкам, разобранным в [JSR223 Pre-Processor](#).

В качестве примера, разберем простейший код на языке Groovy:

return <delay time>, где <delay time> - время задержки в миллисекундах.

Таким образом, если в поле сценария написать **return 5000**, то при применении данного таймера в тестовом плане будет сгенерирована задержка в размере 5 секунд.

3.4.8.5. BeanShell Timer

BeanShell Timer – таймер, задаваемый с помощью сценария на языке BeanShell. В нем нужно самостоятельно реализовать логику вычисления задержки между выполнением запросов, используя один из поддерживаемых языков сценариев. Использование данного таймера полезно в случае, если необходимо вычислить время ожидания на основе какого-то уникального алгоритма, который в настоящее время отсутствует в Jmeter (Рис. 53).

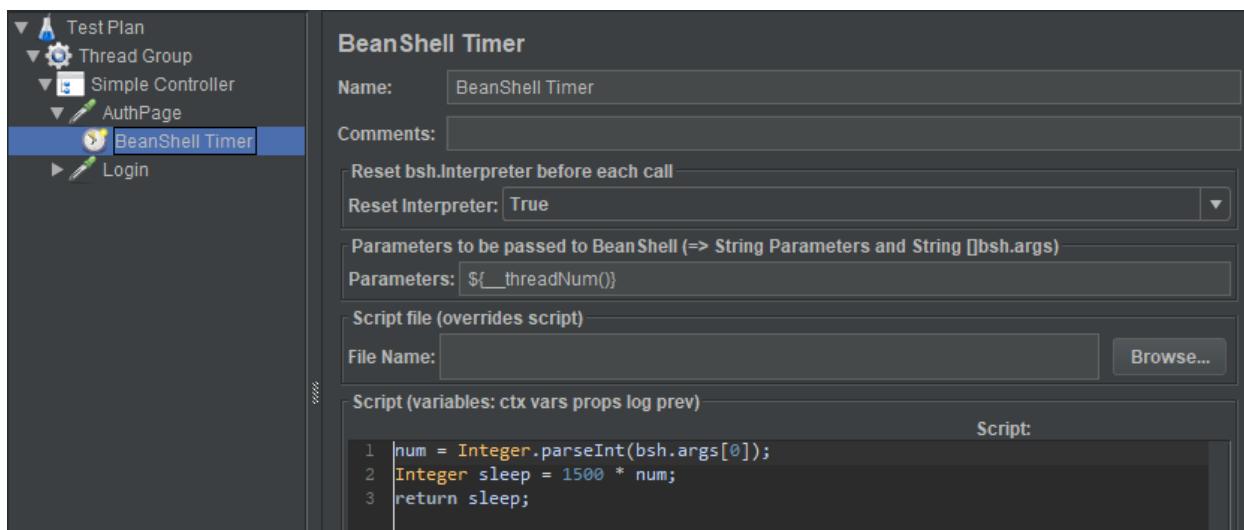


Рис. 53. BeanShell Timer

Настройки данного элемента идентичны настройкам, рассмотренным для [BeanShell Pre-processor](#).

ВАЖНО! В качестве элемента написания сценариев рекомендуется использовать JSR223, а Groovy – в качестве языка сценария. Это обусловлено более быстрой работой скрипtingа на языке Groovy и, соответственно, более высокой производительностью.

3.4.9. Элементы подтверждения

Элемент подтверждения (Assertion) используются в качестве инструмента проверки в Jmeter. Он проверяет ответ сэмплера на основе заданного в нем условия, определяющее провал или успешное выполнение сэмплера. Теоретически, Assertion можно отнести к пост-обработчикам,

поскольку он также выполняется после получения ответа от сервера, а затем проверяет ожидаемый результат с фактическим. Чтобы применить Assertion к определенному сэмплеру, необходимо добавить его в качестве вложенного элемента этого сэмплера.

При использовании Assertion необходимо помнить следующее:

1. Не следует использовать слишком много таких элементов, так как это может повлиять на пропускную способность запроса во время прохождения теста.
2. Заданный шаблон проверки Assertion (особенно текстовые сообщения) должен быть уникальным на странице, чтобы анализируемые в ответе данные проверялись корректно.
3. Assertion может использоваться в качестве глобального элемента для всего тестового плана, который будет применяться ко всем сэмплерам. В таком случае в качестве проверяемого поля обычно выступает **код ответа** (response code).

К числу наиболее важных элементов подтверждения можно отнести следующие:

- [**Response Assertion**](#);
- [**Size Assertion**](#);
- [**Duration Assertion**](#);
- [**JSR223 Assertion**](#).

3.4.9.1. Response Assertion

Элемент **Response Assertion** используется для проверки наличия в ответе веб-сервера конкретной строки (шаблона) или определенных полей, таких как код ответа, сообщение ответа и т.д. Строкой шаблона может выступать число, слово, выражение и т.д., которые могут находиться либо в ответе сервера, либо в самом запросе JMeter.

В качестве примера разберем следующий случай: при запуске экземпляра бизнес-процесса должно происходить автоматическое перенаправление на страницу первой задачи по этому процессу. Если такого не произошло, значит возникли определенные трудности: бизнес-процесс не успел запуститься в силу возможной нагрузки на веб-сервер, либо при его запуске произошла ошибка. Для подтверждения того, что после отправки формы запуска процесса мы действительно переместились на его первую задачу, добавим Assertion со следующими настройками (Рис. 54).

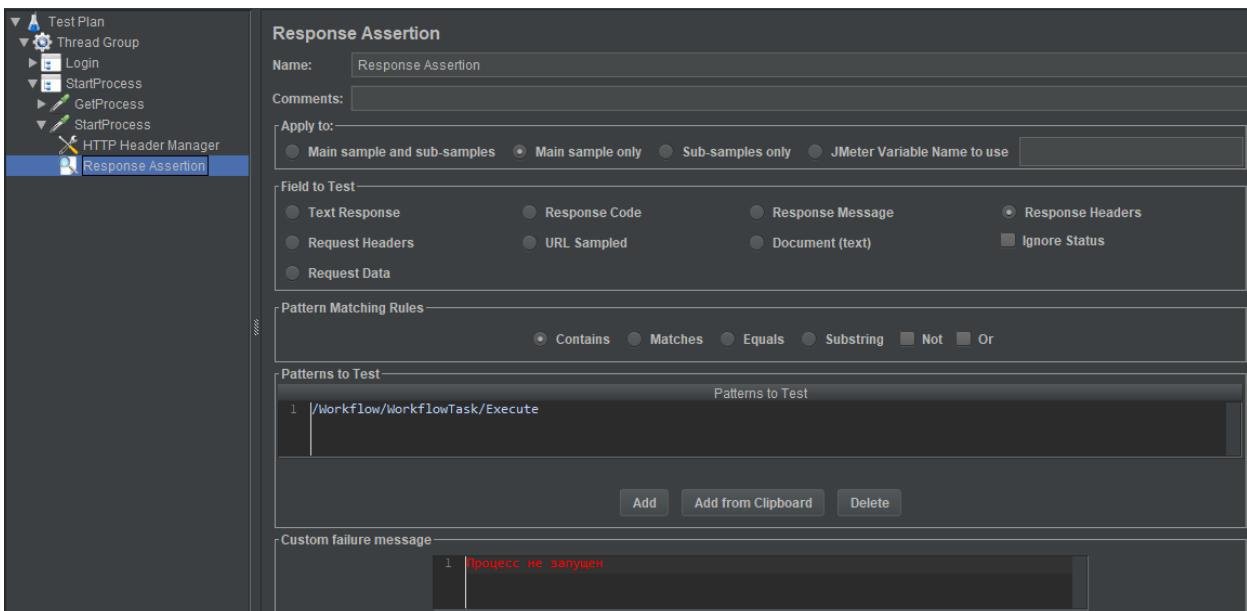


Рис. 54. Элемент "Response Assertion"

- **Apply to** – данный параметр полезен, если у текущего сэмплера имеются дочерние сэмплеры:
 - **Main sample and sub-samples** – применять Extractor к основному сэмплеру и к вложенным сэмплерам;
 - **Main sample only** – применять Extractor только к основному сэмплеру;
 - **Sub-samples only** – применять Extractor только к вложенным сэмплерам;
 - **Jmeter Variable Name to use** – применить Extractor к указанной переменной Jmeter;
- **Field to check** – данная настройка позволяет установить, какая часть ответа сэмплера должна быть проанализирована:
 - **Text Response** – включение данного параметра указывает JMeter искать заданную строку шаблона в теле ответа;
 - **Response code** – при включении данного параметра будет проанализирован код ответа, например, 200 (Рис. 55);

Sampler result	Request	Response data
Thread Name: Thread Group 1-1		
Sample Start: 2019-05-30 21:07:38 SAMT		
Load time: 691		
Connect Time: 0		
Latency: 30		
Size in bytes: 89689		
Sent bytes: 2520		
Headers size in bytes: 1427		
Body size in bytes: 88262		
Sample Count: 1		
Error Count: 0		
Data type ("text" "bin" ""): text		
Response code: 200		
Response message: OK		

Рис. 55. Элемент "Response Assertion". Response code

- **Response Message** – при включении данного параметра будет проанализировано только сообщение ответа, например, OK (Рис. 56);

Sampler result	Request	Response data
Thread Name: Thread Group 1-1		
Sample Start: 2019-05-30 21:07:38 SAMT		
Load time: 691		
Connect Time: 0		
Latency: 30		
Size in bytes: 89689		
Sent bytes: 2520		
Headers size in bytes: 1427		
Body size in bytes: 88262		
Sample Count: 1		
Error Count: 0		
Data type ("text" "bin" ""): text		
Response code: 200		
Response message: OK		

Рис. 56. Элемент "Response Assertion". Response Message

- **Response Headers** – при включении данного параметра поиск заданной строки шаблона будет происходить в заголовках ответа, включая данные куки;
- **Request Headers** – при включении данного параметра поиск заданной строки шаблона будет происходить в заголовках запроса;
- **URL Sampled** – при включении данного параметра в качестве области поиска будет служить только URL;

- **Document (text)** – включение данного параметра позволяет Jmeter искать заданную строку шаблона в самом документе, возвращаемом веб-сервером;
- **Ignore Status** – провал или прохождение элемента Assertion формируется из двух частей. Сначала проверяется сам код ответа и при условии, что он успешный (например, 200), только тогда JMeter начинает проверять заданное в Assertion условие. Таким образом, успех Assertion является возможным как при успешном коде ответа, так и прохождении самого условия. При включении параметра **Ignore Status** статус ответа принудительно становится успешным, даже если он попадает в категорию неудачных ответов от сервера (например, 4XX или 5XX) и приступает к проверке самого условия;
- **Response Data** – при включение данного параметра будет проверяться строка, доступная в теле запроса, который JMeter отправляет на сервер.
- **Pattern Matching Rules** – данный параметр устанавливает правило соответствия заданному в строке условию:
 - **Contains** – функция неполного соответствия. Проверяет, содержат ли проверяемые данные элементы, заданные в строке шаблона;
 - **Matches** – в этом случае проверяемые данные должны полностью соответствовать строке шаблона;
 - **Equals** – в этом случае данные будут проверяться с учетом регистра. Например, workflowTask и WorkflowTask будут являться разными строками;
 - **SubString** – в этом случае JMeter будет проверять подстроку из полученной строки;
 - **Not** – включение данного параметра указывает JMeter, что проверяемые данные не должны содержать заданный шаблон строки. Например, в качестве поля для проверки выбран **Response Code**, в качестве правила соответствия был выбран параметр "Not", а значение строки шаблона – "500". В этом случае все Assertions будут завершаться успешно до получения кода состояния "500";
 - **Or** – включение данного параметра целесообразно в случае, если в качестве условия у вас задано несколько строк шаблона

и при наличии какой-либо из этих строк в проверяемых данных Assertion должен пройти проверку;

- **Patterns to Test** – поле для написания строк шаблона. Каждая строка проверяется отдельно. Если проверка одной из строк завершается неудачей, то остальные уже не проверяются;
- **Custom Failure Message** – поле для ввода сообщения об ошибке, которое будет отображено в случае, если Assertion завершится неудачей.

Таким образом, для нашего сценария в качестве поля для проверки был выбран параметр **Response Headers**, поскольку сообщение о перенаправление на задачу процесса приходит именно в заголовках ответа. Правилом проверки условия был выбран метод **Contains**, а сама строка шаблона проверки выглядит следующим образом: **/Workflow/WorkflowTask/Execute/**, при не нахождении соответствия которой мы получим сообщение "Процесс не запустился". После выполнения сэмплера с помощью листенера View Result Tree, на вкладке **Assertion Result**, мы сможем отследить, выполнилось условие нашего Assertion или нет (Рис. 57).

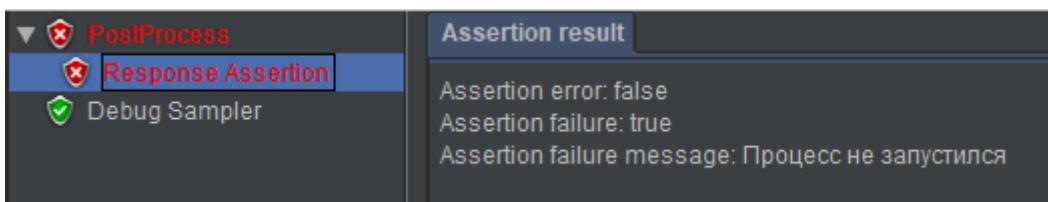


Рис. 57. Вкладка "Assertion Result"

3.4.9.2. Size Assertion

Элемент **Size Assertion** используется в основном, когда необходимо проверить размер ответа целиком или его определенной части, такой как заголовок, код и т. д (Рис. 58). Данный элемент помогает выявить проблему, связанную с пропускной способностью. Простым сценарием, для которого можно добавить элемент Size Assertion, является сценарий скачивания файла.

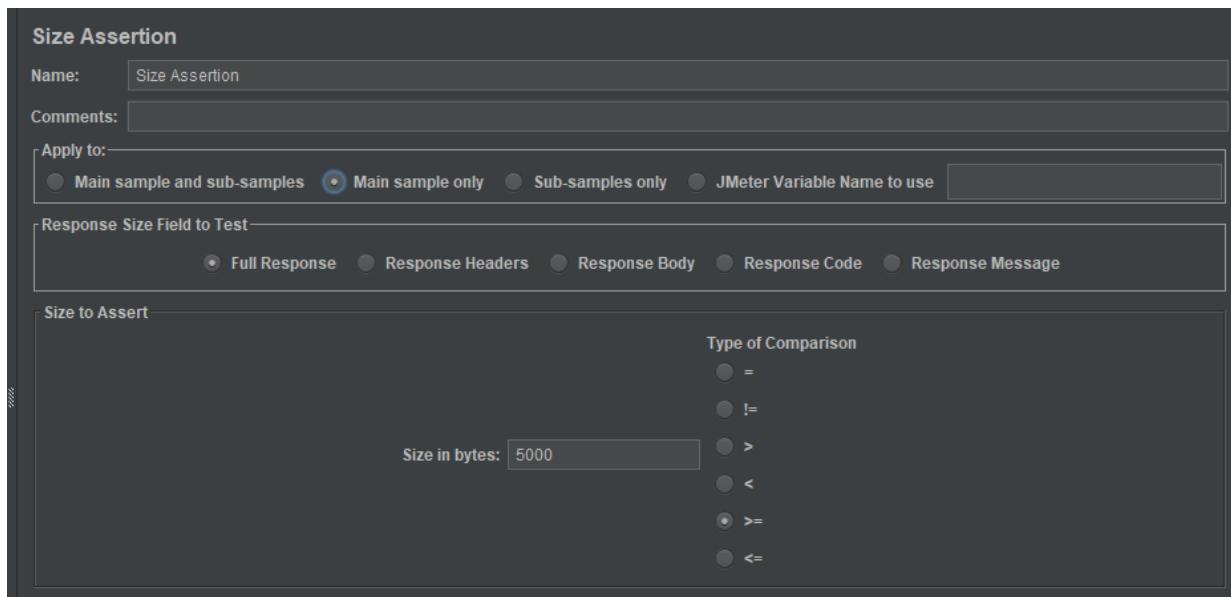


Рис. 58. Элемент "Size Assertion"

- **Apply to** – данный параметр полезен, если у текущего сэмплера имеются дочерние сэмплеры:
 - **Main sample and sub-samples** – применять Extractor к основному сэмплеру и к вложенным сэмплерам;
 - **Main sample only** – применять Extractor только к основному сэмплеру;
 - **Sub-samples only** – применять Extractor только к вложенным сэмплерам;
 - **Jmeter Variable Name to use** – применить Extractor к указанной переменной Jmeter;
- **Field to check** – данная настройка позволяет установить, какая часть ответа сэмплера должна быть проанализирована;
- **Response Size Field To Test:**
 - **Full Response** – при выборе данного параметра проверяется размер как заголовков ответа, так и тела ответа, а полученная сумма сравнивается с ожидаемым размером, заданным в панели "Size to Assert";
 - **Response Headers** – при выборе данного параметра проверяется только размер заголовков;
 - **Response Code** – при выборе данного параметра проверяется размер кода ответа;
 - **Response Message** – при выборе данного параметра проверяется размер сообщения ответа;

- **Size to Assert** – ожидаемый размер ответа необходимо указать в байтах в поле **Size in bytes**, который сравнивается с фактическим размером ответа с использованием условия, выбранного в поле **Type of Comparison**. Значение данного поля может быть параметризовано;
- **Type of Comparison** – в JMeter предоставлены всевозможные методы сравнения: равно (=), не равно (!=), больше (>), меньше (<), больше или равно (>=), меньше или равно (<=) для сравнения фактического и ожидаемого размера ответа.

3.4.9.3. Duration Assertion

Элемент **Duration Assertion** используется, главным образом, когда известно время отклика запроса и, на основании этих данных, необходимо проверить конкретный сэмплер (Рис. 59).

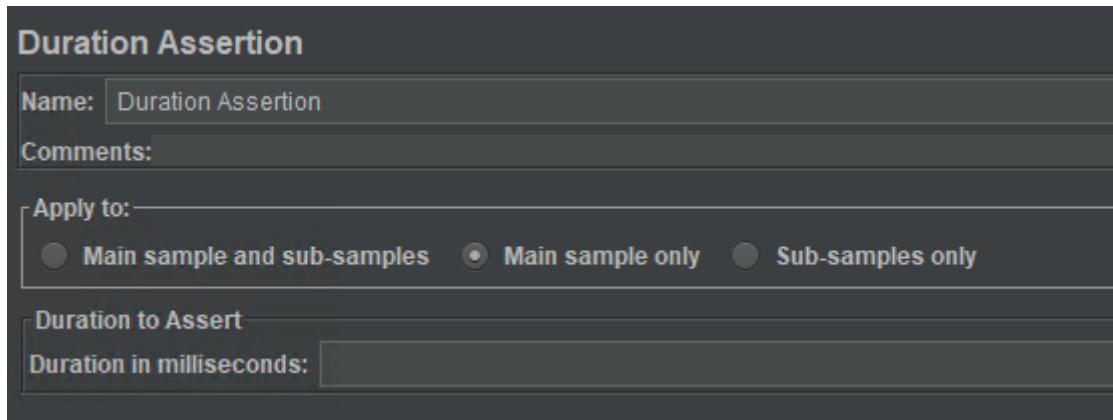


Рис. 59. Элемент "Duration Assertion"

- **Apply to** – данный параметр полезен, если у текущего сэмплера имеются дочерние сэмплеры:
 - **Main sample and sub-samples** – применять Extractor к основному сэмплеру и к вложенным сэмплерам;
 - **Main sample only** – применять Extractor только к основному сэмплеру;
 - **Sub-samples only** – применять Extractor только к вложенным сэмплерам;
- **Duration to Assert** – ожидаемое время отклика необходимо указать в поле **Duration in milliseconds**, которое будет сравниваться с фактическим временем отклика и на основании этого будет принято соответствующее решение. Если фактическое время будет больше ожидаемого, то Assertion будет провален, в противном случае, Assertion будет помечен как пройден.

Ожидаемое время отклика должно быть задано в миллисекундах. Его значение может быть параметризовано.

3.4.9.4. JSR223 Assertion

JSR223 Assertion является элементом подтверждения, задаваемым с помощью сценария с использованием одного из поддерживаемых языков сценариев, таких как Groovy, BeanShell, java и т.д.

В качестве примера разберем сценарий, который будет проверять код ответа и, в случае несоответствия, возвращать нам сообщение об ошибке (Рис. 60).

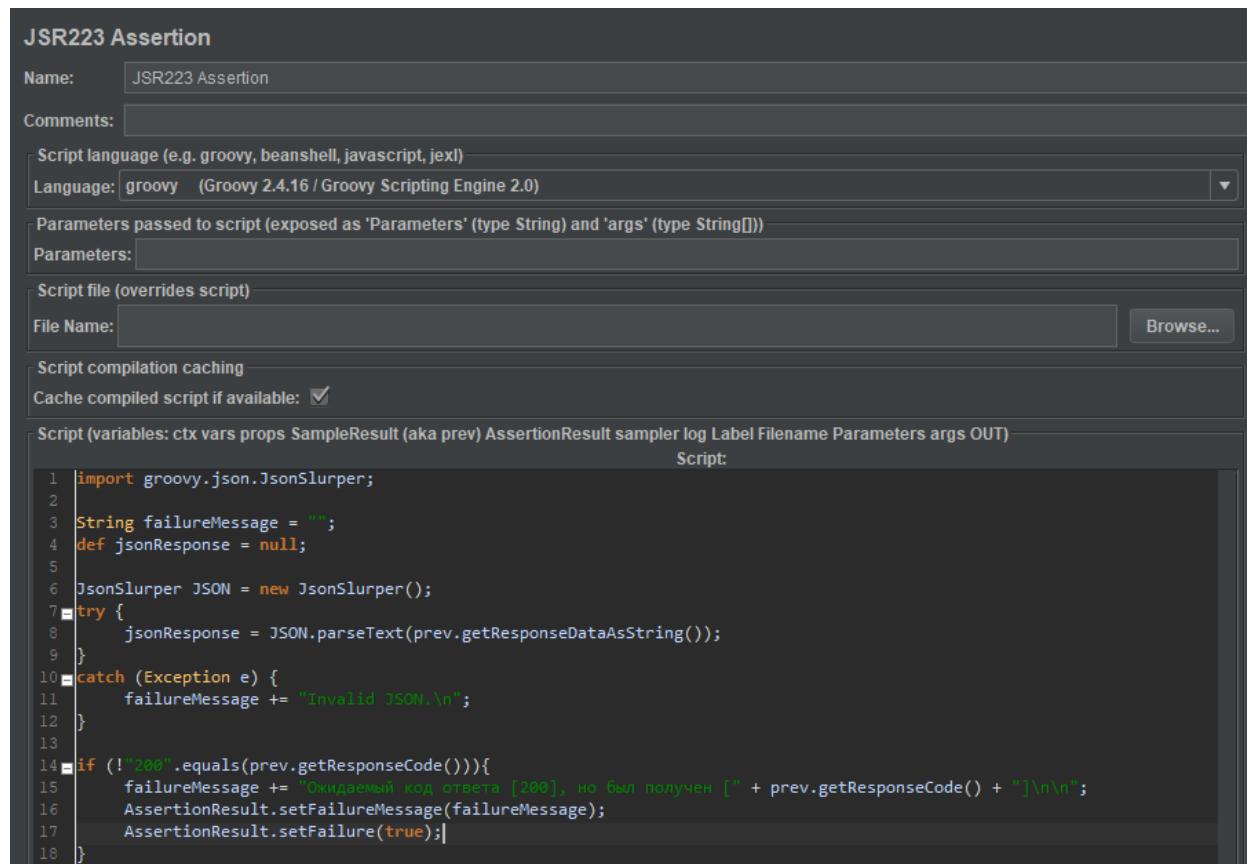


Рис. 60. JSR223 Assertion

В качестве языка написания сценария будет использоваться **Groovy**. Для начала декларируем пространство имен, в котором имеется необходимый класс для синтаксического анализа строки в формате JSON – "JSONSlurper".

Инициализируем необходимые переменные: строка **failureMessage** будет служить для вывода необходимой нам ошибки, в случае провала

Assertion, тогда как объект **jsonResponse** необходим для записи обработанного ответа JSON.

Далее мы используем конструкцию **try...catch**, чтобы обработать некую исключительную ситуацию во время исполнения кода, например, когда нам вернется некорректный JSON либо не придет ответ вследствие ошибки сервера. Соответственно, в блоке **try** мы получаем ответ на предыдущий запрос и конвертируем его в объект JSON. В случае неудачи выполняется блок **catch**, который возвращает ошибку. Далее, проверяется код предыдущего ответа: если код ответа не равен 200 (OK), то выдается информация об ошибке и статус Assertion устанавливается на "провален".

Информацию о выполнении Assertion можно найти в листенере View Results Tree на вкладке **Assertion result**. При этом Assertion не будет отображаться, если проверка его условия завершилась успехом (Рис. 61).

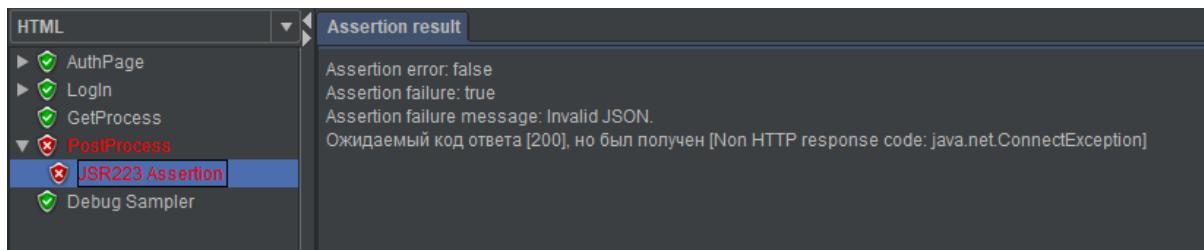


Рис. 61. View Results Tree. Вкладка "Assertion result"

3.4.10. Листенеры

Листенеры (listeners) – элементы тестового плана, которые используются для просмотра и анализа информации, собранной JMeter во время прохождения теста, в табличной или графической форме. В них также собраны и сгруппированы различные метрики производительности, с помощью которых могут быть выявлены ограничители производительности системы. Листенеры в JMeter находятся в категории **Listeners** и, как правило, добавляются в качестве вложенного элемента **Thread Group** для анализа данных исполняемых в ней сэмплеров во время прохождения теста.

При необходимости проанализировать только сэмплеры, размещенные внутри определенного логического контроллера, добавьте интересующих вас листенер в качестве дочернего элемента данного контроллера (Рис. 62).

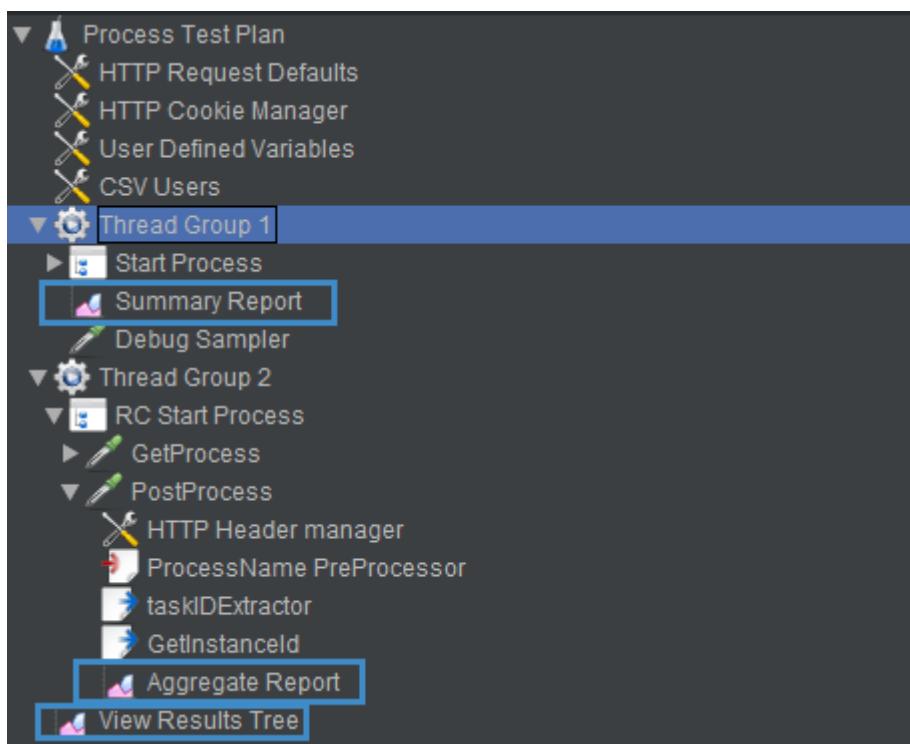


Рис. 62. Листенеры "Summary Report", "Aggregate Report", "View Result Tree"

- Листенер **Summary Report** собирает данные из всех сэмплеров, находящихся в Thread Group 1.
- Листенер **Aggregate Report** собирает данные из сэмплеров, находящихся в логическом контроллере Start Process.
- Листенер **View Result Tree** собирает данные из всех сэмплеров текущего тестового плана.

Стоит отметить, что почти все листенеры имеют возможность записывать результаты в файл, что позволяет выбрать более подходящий формат для их дальнейшего анализа. Одними из самых востребованных листенеров являются:

- [**View Results Tree**](#);
- [**Aggregate Report**](#);
- [**Aggregate Graph**](#);
- [**Simple Data Writer**](#);
- [**Summary Report**](#).

3.4.10.1. View Results Tree

Элемент **View Results Tree** отображает результаты выполнения сэмплеров, а также связанных с ними элементов подтверждения (Assertions) в виде древовидной структуры в том же порядке, в котором они были

инициализированы в teste. Помимо результатов выполнения сэмплеров, в листенере также представлены параметры и данные для каждого из них. Например, для каждой запроса, отправляемого с помощью элемента HTTP Sampler, листенер View Results Tree возвращает объект, содержащий в себе параметры отправляемого запроса, а также полученный на него ответ от веб-сервера. Эти данные отображаются на соответствующих вкладках:

- **Sampler result** (результат выполнения сэмплера) – данная вкладка содержит код ответа, его заголовки, куки и информацию о времени, задержке, размера ответа в байтах и количестве ошибок (Рис. 63).

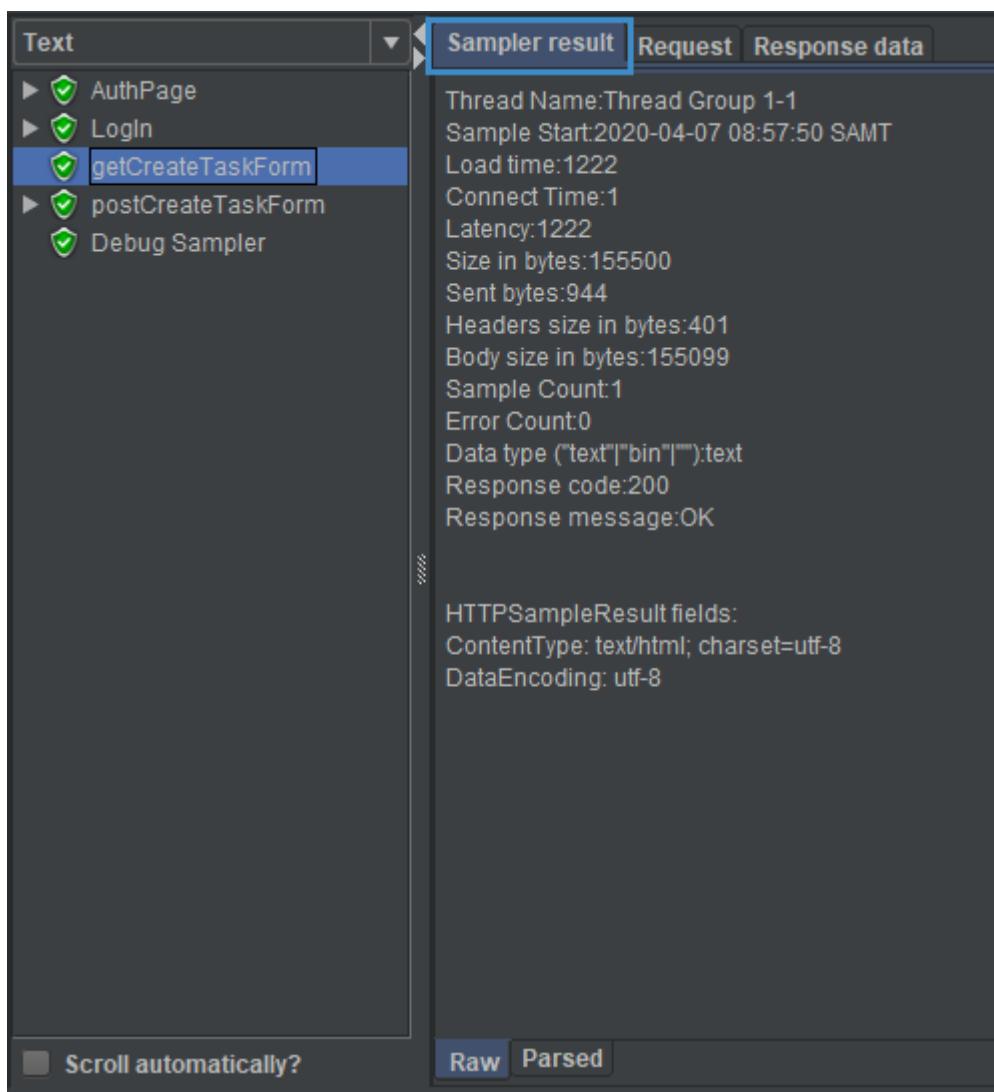


Рис. 63. Результат выполнения сэмплера во View Results Tree. Вкладка "Sampler result"

- **Request** (запрос) – на данной вкладке отображается информация об отправленном JMeter запросе, его заголовках, URL, методе HTTP и данных cookies (Рис. 64).

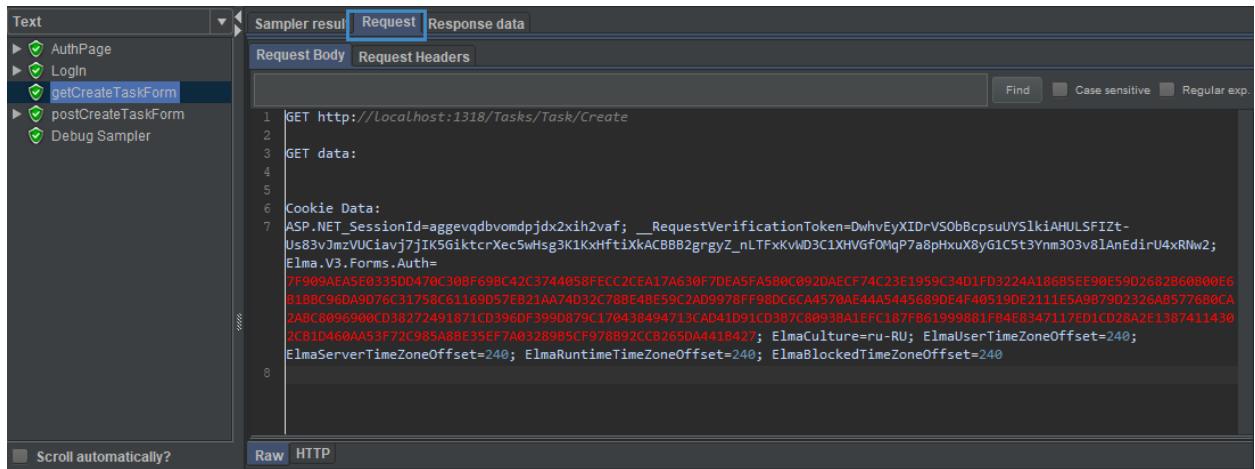


Рис. 64. Результат выполнения сэмплера во View Results Tree. Вкладка "Request"

- **Response data** (данные ответа) – на данной вкладке отображается тело ответа (Рис. 65).

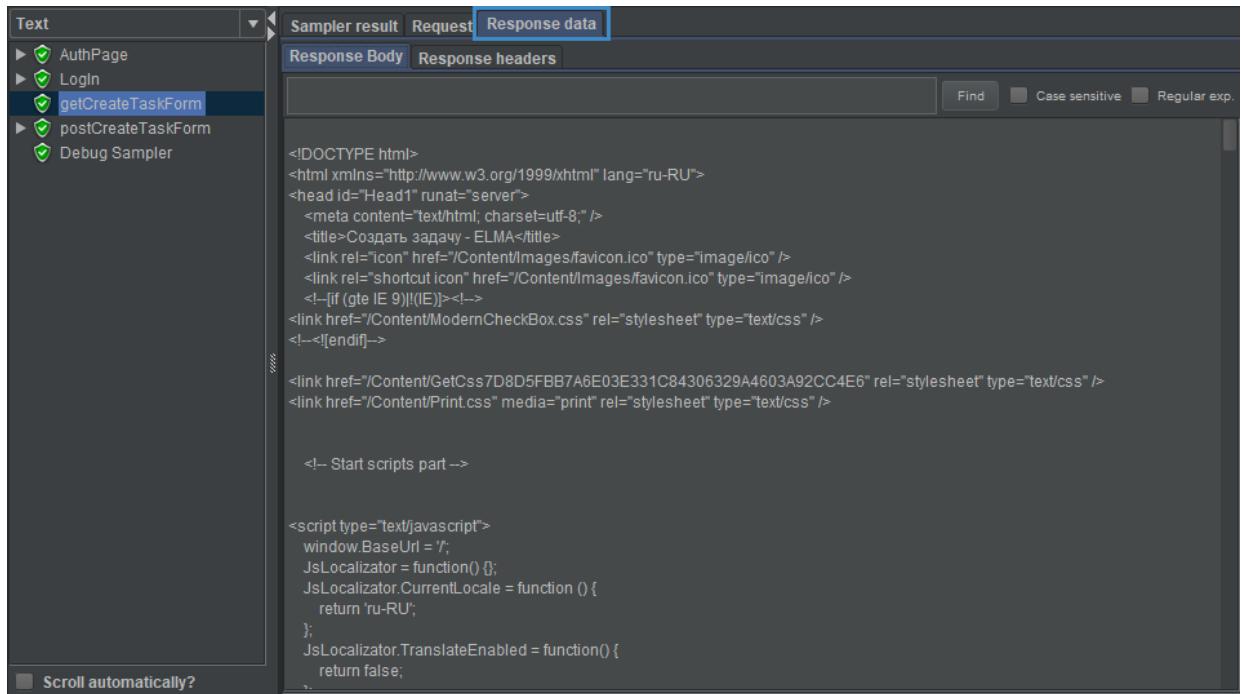


Рис. 65. Результат выполнения сэмплера во View Results Tree. Вкладка "Response data"

Примечание: отображение зеленого значка рядом с именем сэмплера означает, что запрос дошел до сервера и был им обработан. Если имя сэмплера подсвеченено красным, а рядом с ним находится соответствующая иконка, это означает, что в качестве кода ответа он получил коды состояния 4xx и 5xx или был провален элементом подтверждения.

Одна из самых полезных функций листенера View Result Tree – наличие переключателя формата отображаемой информации на вкладке **Response data**. С ее помощью можно преобразовать обычный текст в HTML или XML-формат (Рис. 66).

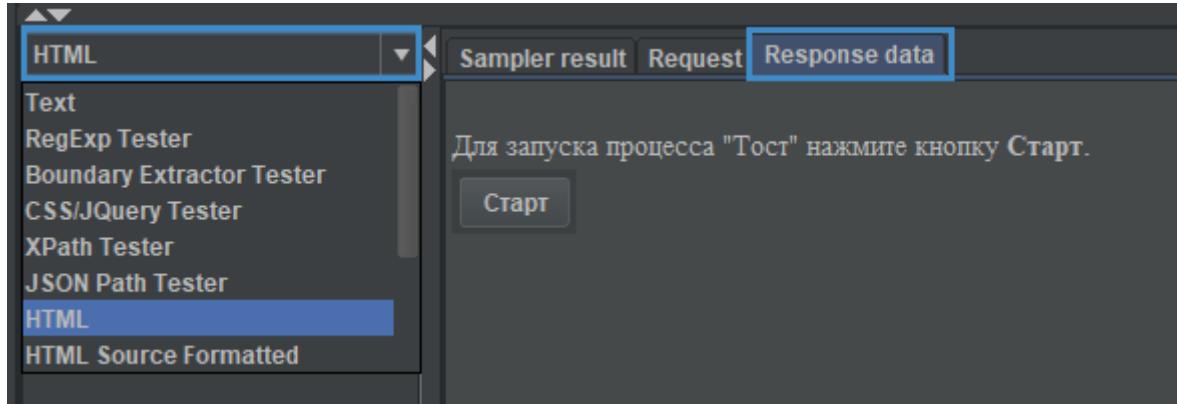


Рис. 66. Изменение вида отображения результатов в элементе "View Results Tree".
Вкладка "Response data"

Хотя HTML-формат будет отображаться без подгруженных стилей, этого все равно достаточно для общего понимания, что происходило на странице в тот момент времени.

Во View Results Tree также отображаются завершившиеся провалом элементы подтверждения (Assertions), в которых сравниваются ожидаемые и фактические результаты на вкладке **Assertion result** (Рис. 67). Если Assertion завершается успехом, то он не отображается.

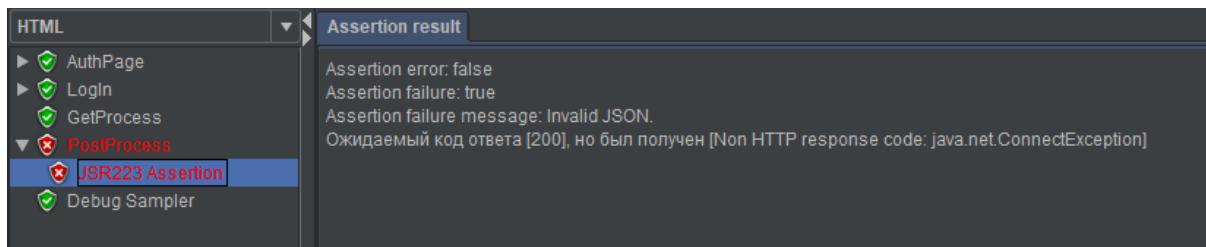


Рис. 67. Результат выполнения Assertion во View Results Tree. Вкладка "Assertion result"

ВАЖНО! Элемент View Results Tree не должен быть активен при проведении самого нагружочного тестирования, так как потребляет большое количество памяти и CPU. Используйте его только при отладке и проверке тестового плана или при выполнении тестового сценария с небольшим количеством потоков (виртуальных пользователей).

3.4.10.2. Aggregate Report

Элемент **Aggregate Report (Агрегированный отчет)** представляет собой сводную таблицу со статическими показателями выполнения каждого уникального сэмплера в тестовом плане. Одними из основных показателей данного элемента, которые отличают его от других отчетов такого типа, являются **перцентили (Percentile/ Line)** – время, за которое отклик получает соответствующая доля запросов (Рис. 68).

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
AuthPage	2	16	13	19	19	19	13	19	0,00%	3,8/sec	38,17	0,53
Login	2	22	22	23	23	23	22	23	0,00%	3,8/sec	62,27	9,42
Debug Samp...	2	0	0	1	1	1	0	1	0,00%	3,9/sec	1,40	0,00
TOTAL	6	13	13	22	23	23	0	23	0,00%	10,9/sec	98,16	9,67

Рис. 68. Элемент "Aggregate Report"

- **Write results to file / Read from file** – в данном поле можно указать имя файла, в который будут записаны данные отчеты после прохождения теста. Наличие файла необязательно, достаточно указать его имя и выбрать расположение, где он будет автоматически создан после выполнения данного элемента. Также, с помощью кнопки **Browse** Вы можете выбрать уже существующий файл, содержащий данные предыдущих тестовых прогонов, и информация с данного файла будет загружена в таблицу.
- **Log/Display only** – поле, в котором можно задать параметр, в соответствии с которым будет производиться запись информации в таблицу. **Errors** – будут показаны сэмплеры, завершившиеся ошибками, **Successes** – будут показаны сэмплеры, завершившиеся успехом. Если не включать ни один из этих параметров, то в таблице будут отображены все выполненные сэмплеры.
- **Configure** – при нажатии на кнопку выводится окно настроек, в которых можно задать состав выводимых в файле данных (Рис. 69).

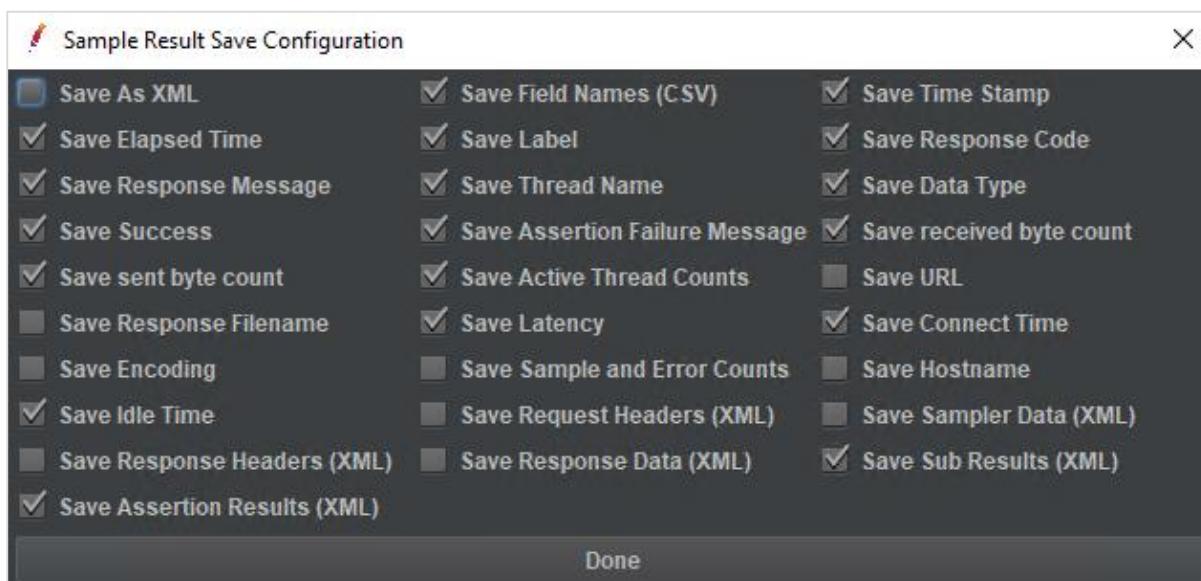


Рис. 69. Окно настроек Aggregate Report

- **Save Table Data** – кнопка для экспорта таблицы отчета в CSV-файл с возможностью отображения в нем заголовков таблицы, выбрав параметр **Save Table Header**.

Агрегированный отчет имеет следующие показатели:

- **Метка (Label)** – метка для запроса;
- **Количество запросов (# Samples)** – количество запросов с такой меткой;
- **Среднее (Average)** – среднее время;
- **Медиана (Median)** – время, за которое гарантировано выполнилось 50% запросов;
- **90% Line** – время, за которое гарантировано выполнилось 90% запросов (90th percentile);
- **95% Line** – время, за которое гарантировано выполнилось 95% запросов (95th percentile);
- **99% Line** – время, за которое гарантировано выполнилось 99% запросов (99th percentile);
- **Min** – минимальное время запроса;
- **Max** – максимальное время запроса;
- **Error %** – процент запросов с ошибкой;
- **Производительность (Throughput)** – количество запросов в секунду;
- **Получено (Received) KB/sec;**
- **Отправлено (Sent) KB/sec.**

3.4.10.3. Aggregate Graph

Элемент **Aggregate Graph** аналогичен листенеру [Aggregate Report](#), только помимо табличных данных с информацией о метриках производительности, в нем также отображается гистограмма, которую впоследствии можно выгрузить в PNG-файл (Рис. 70).

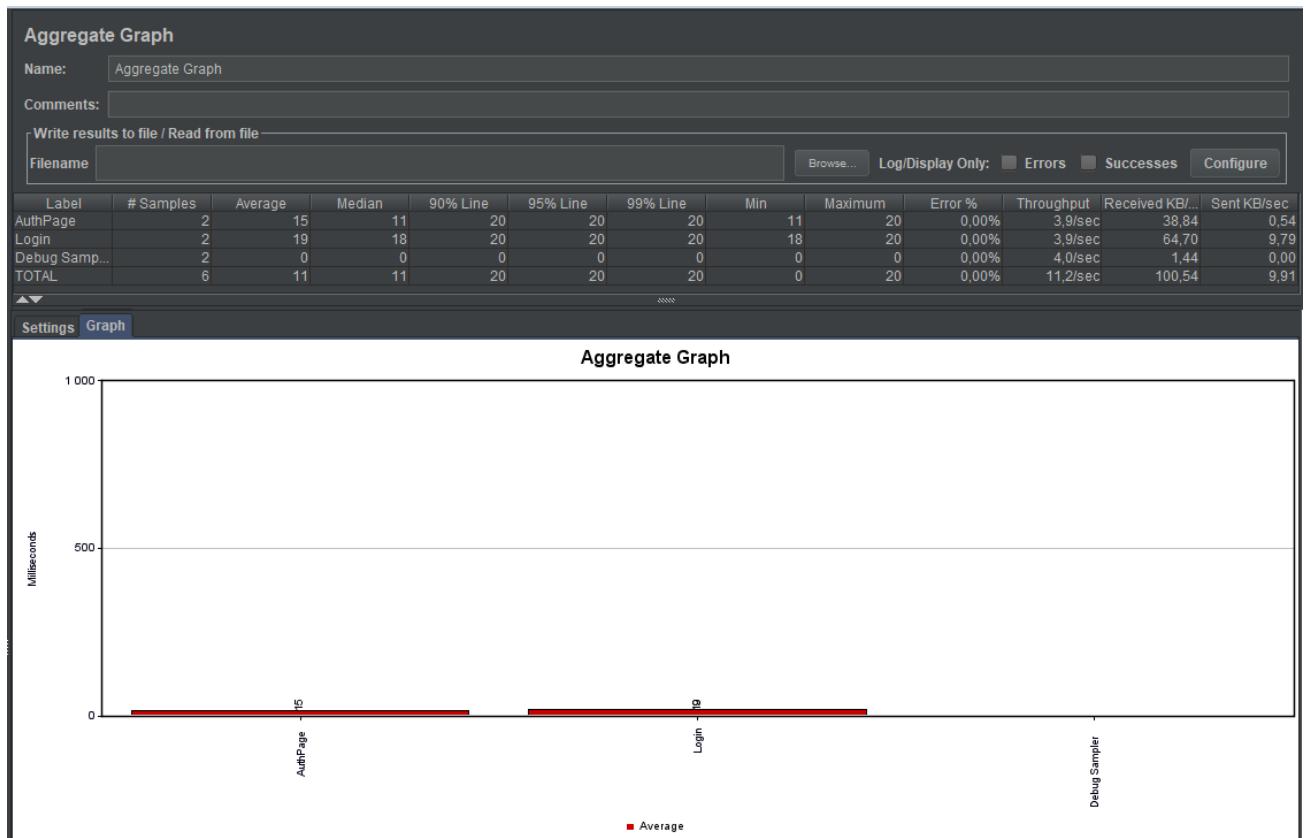


Рис. 70. Элемент "Aggregate Graph"

3.4.10.4. Simple Data Writer

Элемент **Simple Data Writer** записывает результаты прохождения теста в файл. Он не отображает результаты в графическом интерфейсе JMeter (Рис. 71).

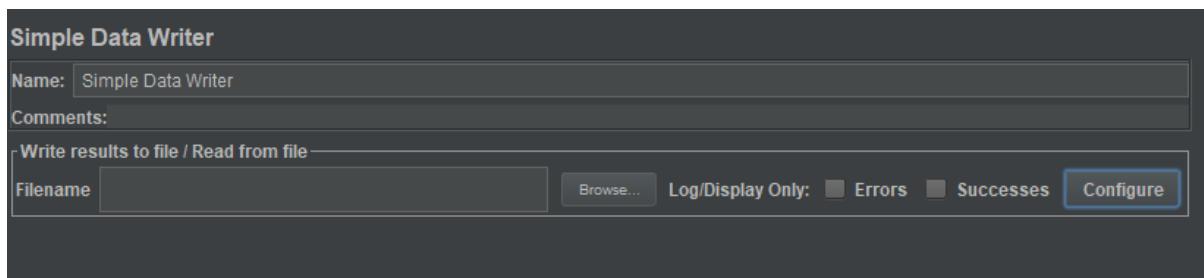


Рис. 71. Элемент "Simple Data Writer"

- **Write results to file / Read from file** – в данном поле можно указать имя файла, в который будут записаны данные отчеты после прохождения теста. Наличие файла необязательно, достаточно указать его имя и выбрать расположение, где он будет автоматически создан после выполнения данного элемента.
- **Log/Display only** – поле, в котором можно задать параметр, в соответствии с которым будет производиться запись информации в таблицу. **Errors** – будут показаны сэмплеры, завершившиеся ошибками, **Successes** – будут показаны сэмплеры, завершившиеся успехом. Если не включать ни один из этих параметров, то в таблице будут отображены все выполненные сэмплеры.
- **Configure** – при нажатии на кнопку выводится окно настроек, в которых можно задать состав выводимых в файле данных (Рис. 72).

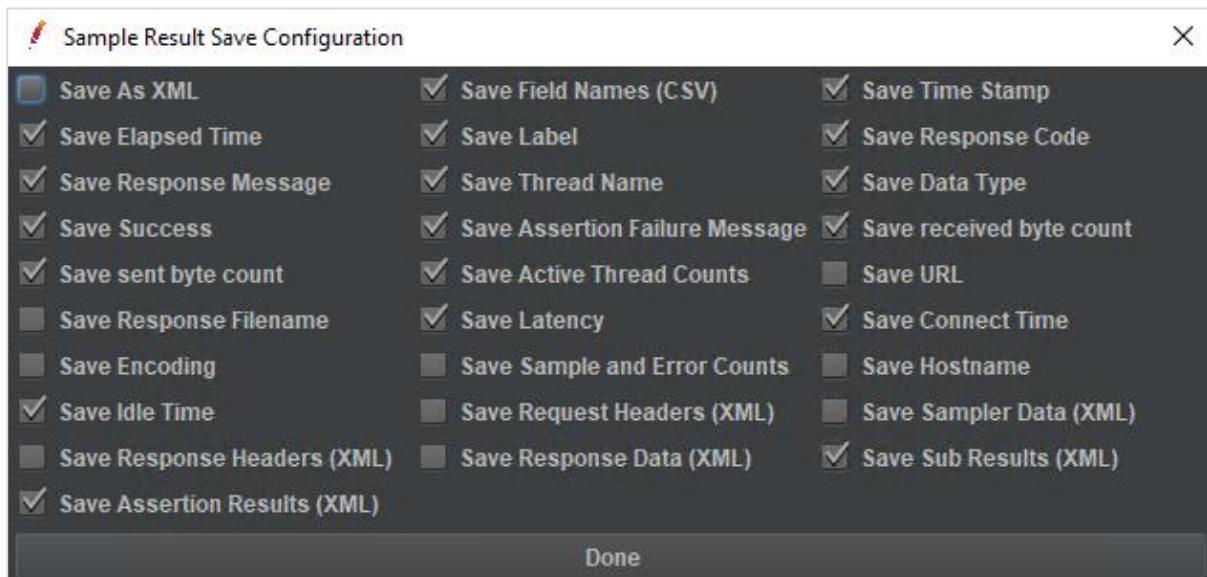


Рис. 72. Окно настроек Simple Data Writer

3.4.10.5. Summary Report

Элемент **Summary Report** представляет собой сводную таблицу со статистическими показателями выполнения каждого уникального сэмплера в тестовом плане. Данный отчет напоминает [Aggregate Report](#), за исключением того, что он расходует меньше памяти и, среди его показателей, не имеются [перцентили](#) (Рис. 73).

Summary Report										
Name:	Summary Report									
Comments:										
Write results to file / Read from file-										
Filename	<input type="text"/>				<input type="button" value="Browse..."/>	<input type="checkbox"/> Log/Display Only:	<input type="checkbox"/> Errors	<input type="checkbox"/> Successes	<input type="button" value="Configure"/>	
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
AuthPage	2	15	11	20	4,50	0,00%	3,9/sec	38,84	0,54	10182,0
Login	2	19	18	20	1,00	0,00%	3,9/sec	64,70	9,79	16994,0
Debug Sampler	2	0	0	0	0,00	0,00%	4,0/sec	1,44	0,00	365,0
TOTAL	6	11	0	20	8,67	0,00%	11,2/sec	100,54	9,91	9180,3

Include group name in label? Save Table Header

Рис. 73. Элемент "Summary Report"

Его настройки идентичны [настройкам](#) агрегированного отчета.

Summary Report имеет следующие показатели:

- **Метка (Label)** – метка для запроса;
- **Количество запросов (# Samples)** – количество запросов с такой меткой;
- **Среднее (Average)** – среднее время;
- **Min** – минимальное время запроса;
- **Max** – максимальное время запроса;
- **Std Dev.** – стандартное отклонение набора показателей;
- **Error %** – процент запросов с ошибкой;
- **Производительность (Throughput)** – количество запросов в секунду;
- **Получено (Received) KB/sec;**
- **Отправлено (Sent) KB/sec.**

Примечание: элемент Summary Report также используется для генерации на его основе DashBoard Report, настройка которого будет рассмотрена в [соответствующей главе данного руководства](#).

3.4.11. Порядок выполнения элементов

Элементы тестового плана Jmeter имеют следующий порядок выполнения:

1. Элементы конфигурации.
2. Препроцессоры.
3. Таймеры.
4. Сэмплеры.
5. Пост-обработчики (если результат выполнения сэмплера не null).
6. Элементы подтверждения (если результат выполнения сэмплера не null).

7. Листенеры (если результат выполнения сэмплера не null).

Обратите внимание, что таймеры, элементы подтверждения, пост и препроцессоры обрабатываются только при наличии сэмплера, к которому они применяются. Логические контроллеры и сэмплеры обрабатываются в том порядке, в котором они отображаются в дереве тестового плана. Другие тестовые элементы обрабатываются в зависимости от своего типа и области, в которой они расположены.

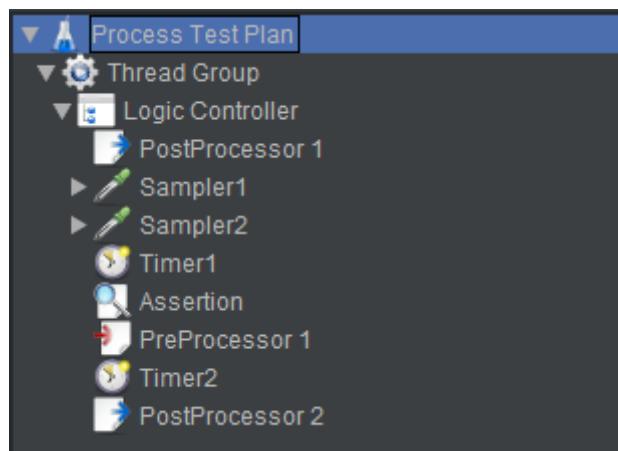


Рис. 74. Пример тестового плана

Например, для тестового плана, указанного на Рис. 74, порядок выполнения элементов будет следующим:

1. Pre-Processor 1.
2. Timer 1.
3. Timer 2.
4. Sampler 1.
5. Post-Processor 1.
6. Post-Processor 2.
7. Assertion.
8. Pre-Processor 1.
9. Timer 1.
10. Timer 2.
11. Sampler 2.
12. Post-Processor 1.
13. Post-Processor 2.
14. Assertion.

Глава 4. Сценарий тестирования производительности

Изначально для проведения тестирования производительности формируется ТЗ, исходя из которого будет описан тестовый сценарий. Уровень нагрузки, регламентируемый в ТЗ, должен определяться профилем нагрузки, содержащим следующую информацию:

- количество эмулируемых пользователей;
- перечень операций, выполняемых эмулируемыми пользователями через соответствующие интерфейсы;
- интенсивность операций, выполняемых эмулируемыми пользователями;
- продолжительность теста.

В качестве примера для проведения нагрузочного тестирования рассмотрим следующий случай:

Небольшая компания преимущественно занимается обработкой входящей корреспонденции. После получения письма от контрагента ответственный менеджер создает в системе соответствующий документ и отправляет его на последующее рассмотрение руководителем. В среднем в системе одновременно работает 30 бизнес-пользователей, создающих в день до 800 документов и отправляющих их на последующее рассмотрение. Необходимо знать, какое максимальное число пользователей может одновременно работать в системе, соответствуя ее критериям производительности.

Проводим следующие расчеты:

1. Количество операций документооборота в день для каждого пользователя $800 \times 3 / 30 = 80$.
2. В час каждый пользователь выполняет $80 / 8 = 10$ операций.
3. Определяем период повторения (интенсивность) операции для каждого пользователя $60 / 10 = 6$ минут (360 секунд).

Итак, после проведения необходимых расчетов приступаем к написанию самого сценария нагрузочного тестирования в JMeter.

ВАЖНО! Jmeter необходимо разместить на отдельной машине, в противном случае результаты тестирования с большой вероятностью не будут соответствовать реальным значениям, поскольку сама работа клиента

нагрузочного тестирования будет использовать системные ресурсы, тем самым увеличивая время выполнения операций, отображаемое в метриках производительности.

4.1. Подготовка к записи сценария нагрузочного тестирования

Запустив **JMeter** в графическом режиме, первое, что необходимо сделать на главной странице приложения, – это сохранить текущий тест-план на жесткий диск компьютера. Для этого необходимо нажать на



соответствующую иконку. В открывшемся окне ввести название тестового плана и папку, куда он будет сохранен, и нажать кнопку **Save**.

Название каждого тестового плана должно отражать сценарий, который в нем проверяется: это облегчает его восприятие при чтении, а также ускоряет сам поиск в архиве при его повторном использовании (Рис. 75). В нашем случае мы будем эмулировать создание и ознакомление с документом, исходя из этого, зададим тест-плану соответствующее название – **CreateAndReviewDoc**.

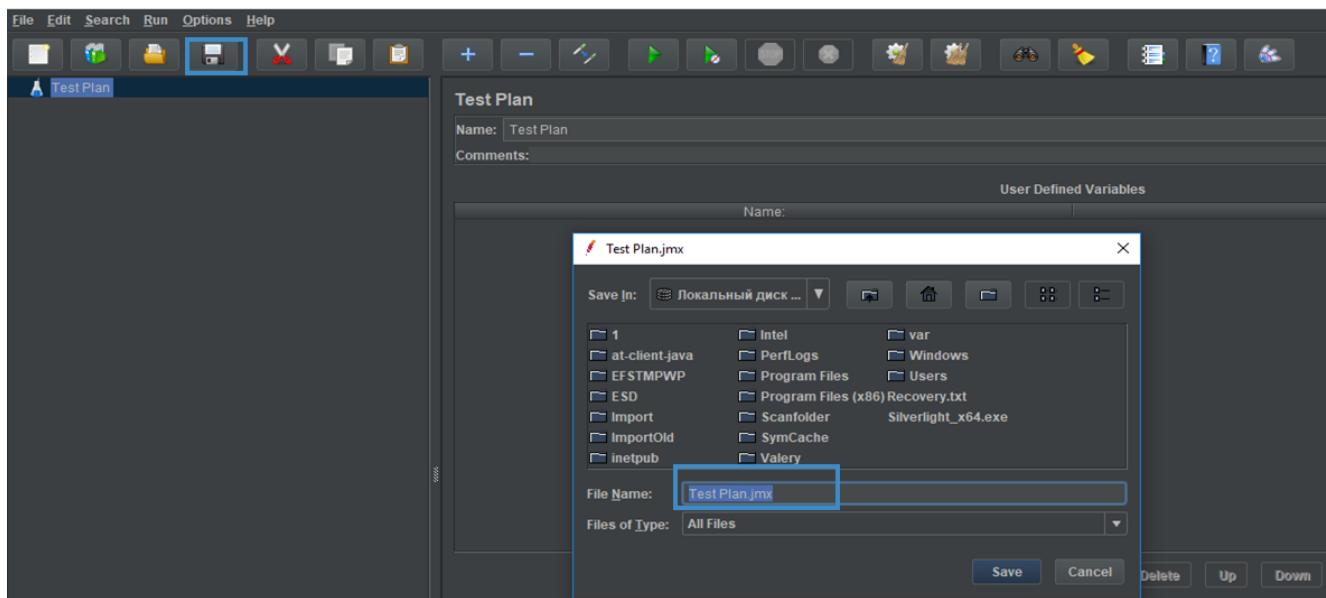


Рис. 75. Сохранение нового тестового плана

Примечание: если вам необходимо открыть уже готовый тестовый план, в главном окне JMeter выберите пункт меню **File - Open**. В открывшемся окне укажите путь до.jmx файла, содержащего тестовый план.

Далее переходим к структурированию самого тестового плана и начнем с добавления в план элементов конфигурации. Для этого необходимо вызвать контекстное меню текущего тестового плана и выбрать **Add-Config Element** (Рис. 76).

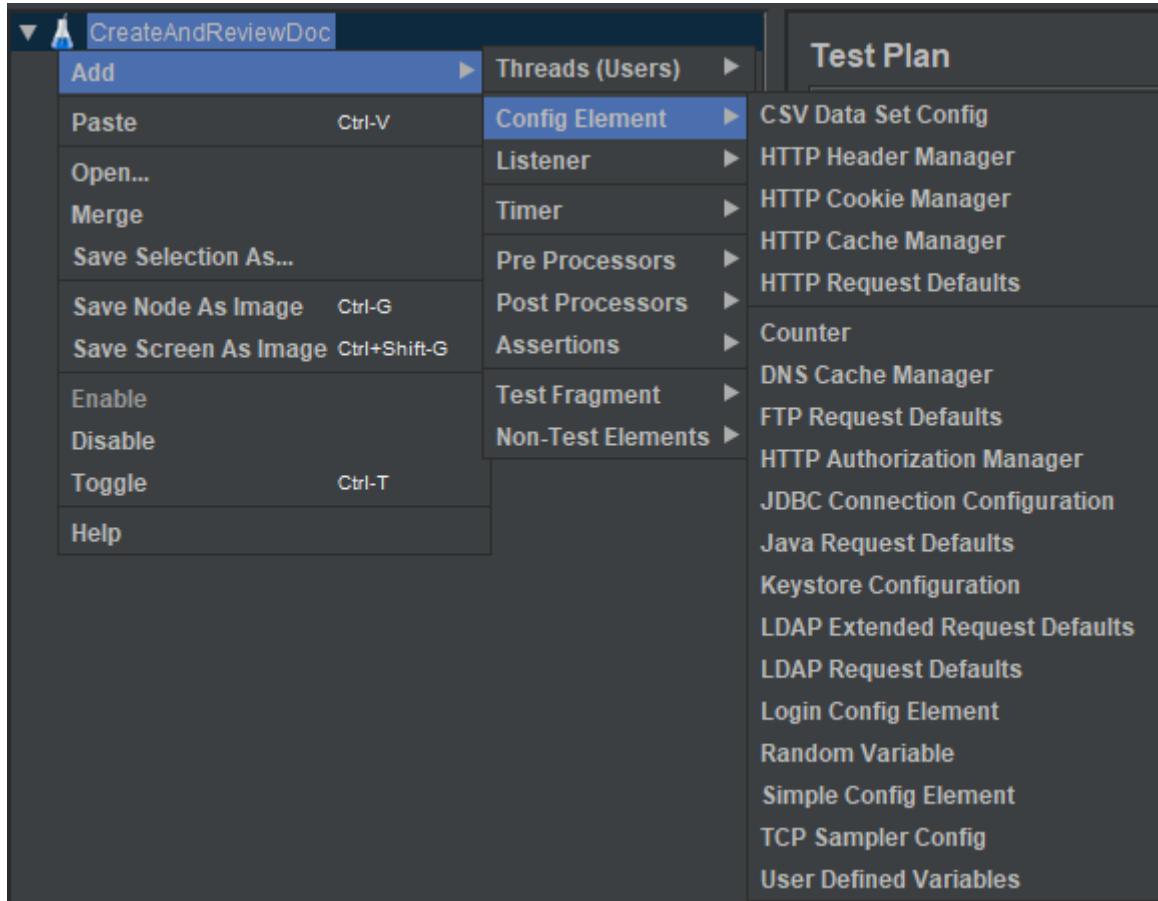


Рис. 76. Добавление в план элементов конфигурации

Добавим в наш тестовый план следующие элементы:

- **HTTP Request Defaults** – в нем будут заданы параметры соединения с веб-сервером по умолчанию, которые будут использоваться для HTTP-сэмплеров (Рис. 77).

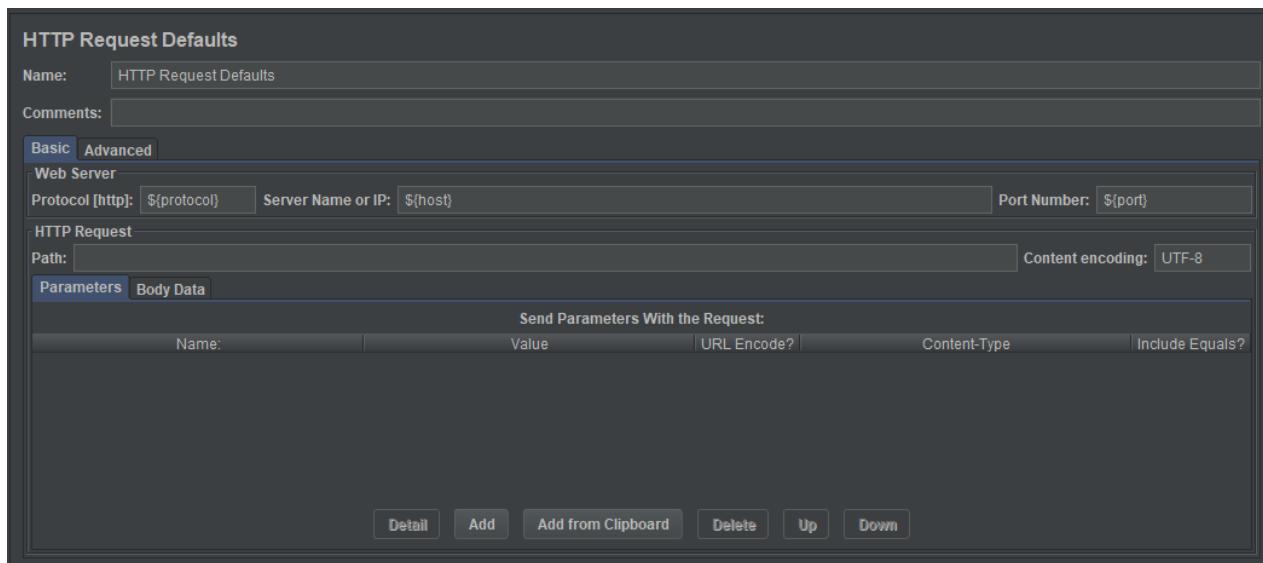


Рис. 77. HTTP Request Defaults

Поля **Protocol**, **Server Name or IP** и **Port Number** необходимо параметризовать путем добавления ссылок на переменные JMeter, в которых будут определены их значения, что позволит запускать созданный нами тест-план для разных веб-серверов, изменяя лишь значения данных переменных.

- **User Defined Variables** – в нем будут заданы глобальные переменные, которые будут использоваться в teste (Рис. 78).

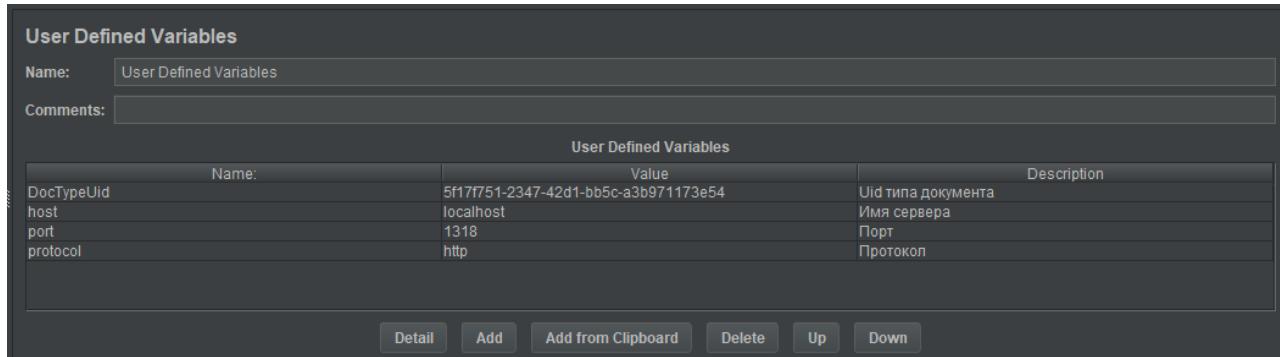


Рис. 78. User Defined Variables

Создадим 4 статические параметра, нажав на кнопку **Add** и введя их имя, значение и описание: **docTypeUID** – UID типа документа, экземпляр которого будет создаваться в соответствующем запросе; **server** – имя тестируемого веб-сервера; **port** – порт тестируемого веб-сервера; **protocol** – используемый протокол.

- **HTTP Cache Manager** – будет использоваться для кэширования Jmeter результатов запросов (Рис. 79).

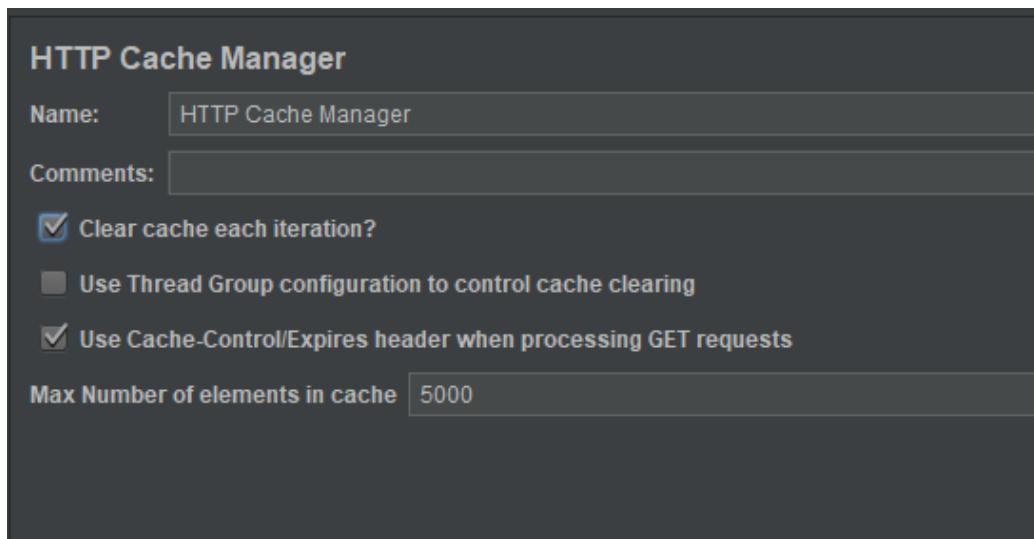


Рис. 79. HTTP Cache Manager

Установим флажок на **Clear cache each iteration?**, чтобы очищать кэш при каждой итерации тестирования. Остальные настройки оставим по умолчанию.

- [HTTP Cookie Manager](#) – необходим, чтобы эмулировать обмен куки между браузером и веб-сервером ELMA (Рис. 80).

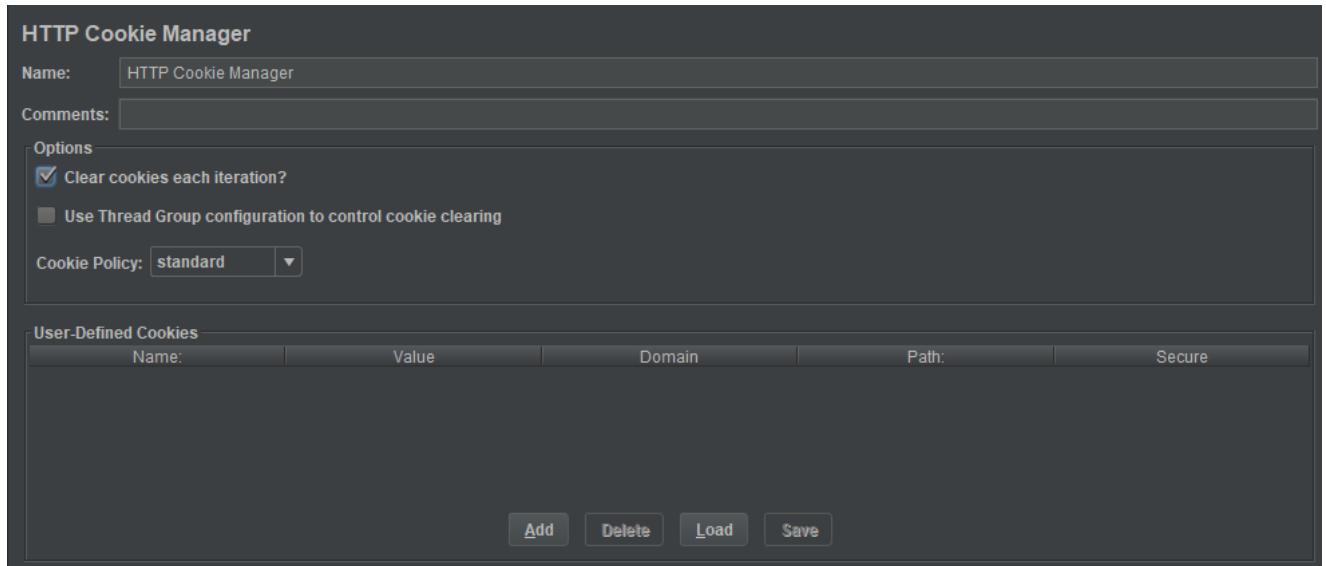


Рис. 80. HTTP Cookie Manager

Установим флажок на **Clear cookies each iteration?**, чтобы очищать куки при каждой итерации тестирования.

- [CSV Data Set Config](#) – в нем мы укажем путь до CSV-файла с учетными данными пользователей и определим переменные для их хранения (Рис. 81).

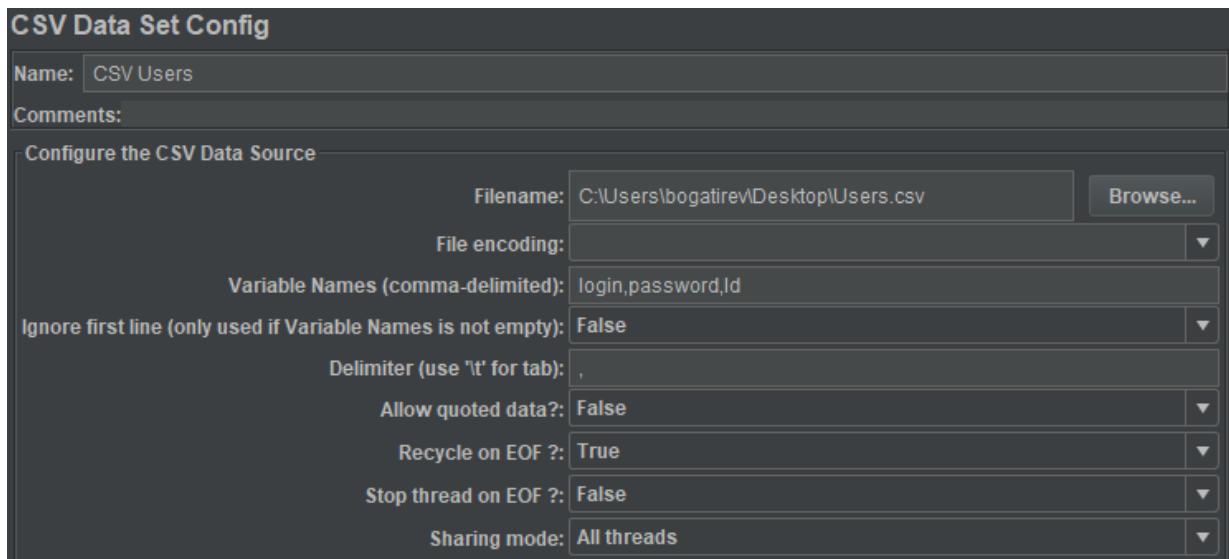
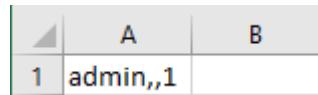


Рис. 81. CSV Data Set Config

Примечание: для упрощения сценария создание документа, его отправка на рассмотрение и последующее рассмотрение будет проводиться одним пользователем (администратором ELMA), поскольку при тестировании производительности бизнес-логика не проверяется, соответственно, загружаемый CSV-файл будет содержать единственную

запись:  . Таким образом, в зависимости от настроек Thread Group, JMeter создаст определенное число активных сессий конкретного пользователя.

Далее добавим элемент [Thread Group](#), в которую будет записан наш тестовый сценарий (Рис. 82).

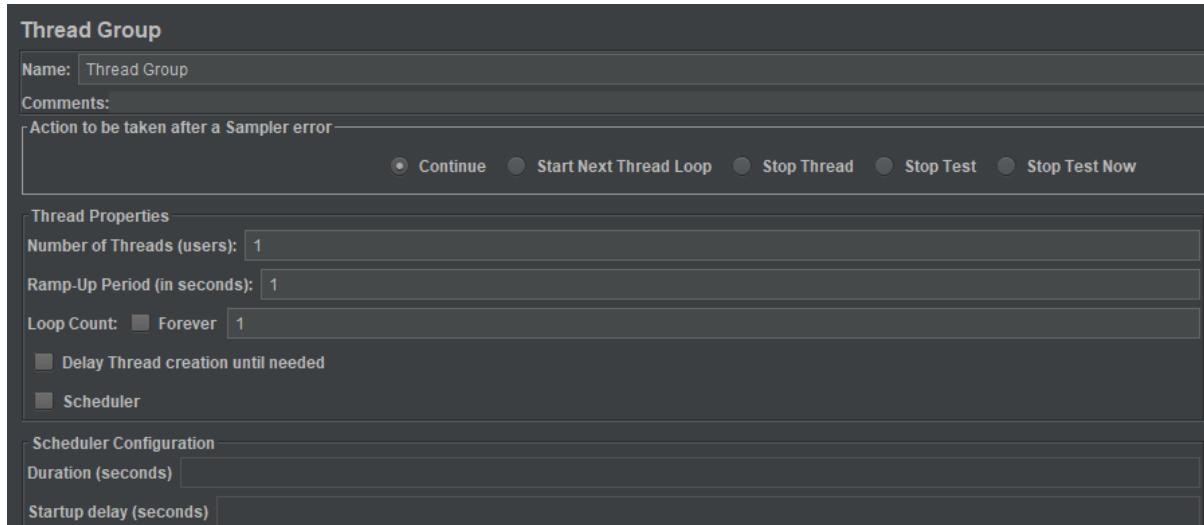


Рис. 82. Thread Group

Во время проверки и отладки теста оставим настройки Thread Group без изменения. При проведении уже самого нагрузочного тестирования сконфигурируем элемент, исходя из имеющегося профиля нагрузки.

На текущий момент наш тестовый план будет выглядеть следующим образом (Рис. 83).



Рис. 83. Тестовый план

Если брать в расчет, что выполнение большинства операций в ELMA подразумевает отправку нескольких взаимосвязанных запросов, например, операция создания документа включает в себя получение формы создания документа и ее последующей отправки, то при моделировании тестового сценария данные операции будут разделены на группы связанных запросов с помощью логических контроллеров. К тому же, несмотря на то, что авторизация пользователей происходит, как правило, заранее, до наступления часа активности, проведение нагрузочного тестирования без операции авторизации для определенного пользователя невозможно, поэтому эту операцию тоже необходимо включить в тесты, как и операцию выхода из системы, чтобы не накапливались активные сессии. Исходя из написанного выше, наш тестовый сценарий будет включать в себя выполнение следующих запросов:

1. Авторизация:

- получение страницы авторизации в системе;
- отправка формы с учетными данными пользователя.

2. Создание документа:

- получение формы создания документа;
- отправка формы создания документа.

3. Отправка документа на ознакомление:

- получение формы задачи "Отправка на ознакомление";
- отправка формы задачи "Отправка на ознакомление".

4. Ознакомление с документом:

- получение формы задачи ознакомления;
- отправка формы задачи ознакомления.

5. Логаут:

- завершение сеанса авторизации пользователя.

Сценарий тестирования производительности может быть записан как с помощью внутреннего прокси-сервера JMeter, который запускается из элемента HTTP(S) Test Script Recorder, так и с использованием сторонних инструментов, имеющих в себе функционал прокси-сервера, как, например, Fiddler или FireBug. Для начала разберем запись сценария с помощью элемента HTTP(S) Test Script Recorder.

ВАЖНО! К моменту составления сценария нагрузочного теста система должна иметь настроенное окружение. Конкретно для этого случая участвующему в сценарии пользователю должен быть настроен доступ к разделу **Документооборот** и выдан полный доступ ко всем документам.

4.2. Запись с помощью HTTP(S) Test Script Recorder

Прокси-сервер (Proxy) – компонент, выполняющий роль посредника между пользователем и целевым сервером. Весь трафик, проходящий между браузером и целевым сервером, будет перехвачен прокси. Элемент HTTP(S) Test Script Recorder представляет из себя прокси-сервер, позволяющий сохранить все проходящие через него запросы в указанный в нем узел тестового плана (Рис. 84).

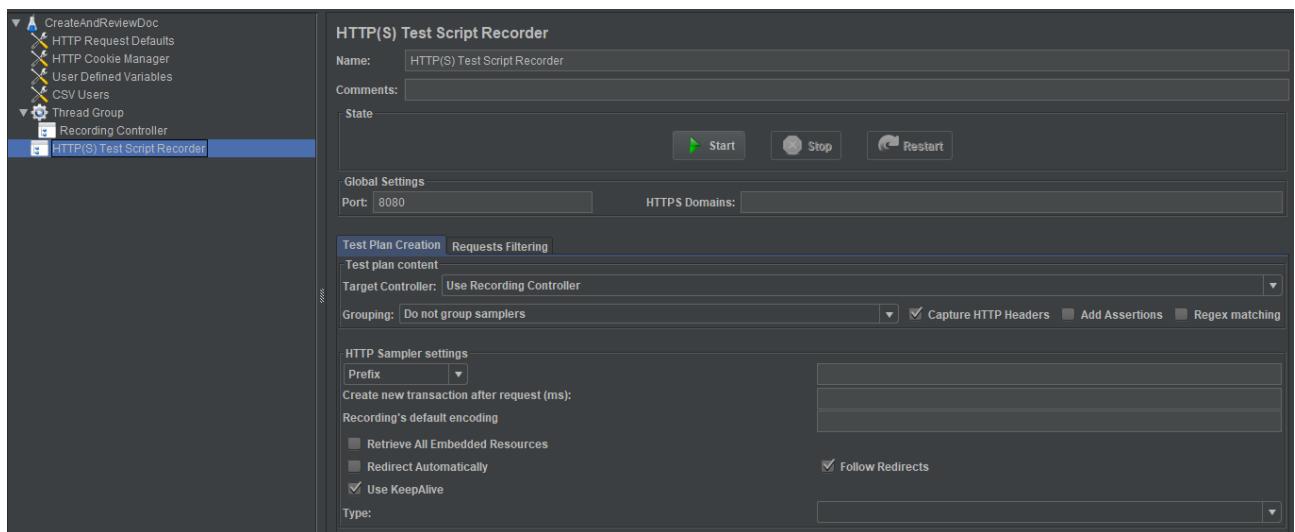


Рис. 84. Элемент "HTTP(S) Test Script Recorder"

- Панель **Global Settings**:
 - **Port** – любой незанятый порт, на котором будет работать прокси-сервер, прослушивая все HTTP-запросы;
 - **HTTPS Domains** – список доменных имен (хостов) для HTTPS. Необходим для генерации SSL-сертификатов для всех серверов, для которых используется протокол HTTPS, трафик с которых необходимо записывать;
- Панель **Test Plan Creation**:
 - **Target Controller** – если в данном поле используется значение **Use Recording Controller**, то все запросы, проходящие через прокси, будут записываться в первый попавшийся Recording Controller в нашем тест-плане, соответственно, на момент запуска прокси, данный контроллер уже должен в нем находиться. Несмотря на то, что существует возможность записывать результаты прямо в Thread Groups либо в сам

- элемент HTTP(S) Script Recorder, для этих целей все равно рекомендуется использовать Recording Controller, потому что так лучше отслеживается и формируется структура теста;
- **Grouping** – группирует http-запросы по их принадлежности к одной и той же странице;
 - **Capture HTTP Headers** – при установке данного параметра будет создаваться элемент HTTP Headers Manager с автоматически сформированными заголовками связанного с ним http-запроса, дочерним элементом которого он будет являться. Данный параметр необходимо включать, поскольку большинство передаваемых веб-браузером заголовков являются важными.
 - Панель **HTTP Sampler settings**:
 - **Use Keep-Alive** – используйте данный параметр, чтобы сигнализировать о том, что соединение должно оставаться открытым для дальнейших сообщений (по умолчанию, для HTTP 1.1.);
 - **Follow Redirects** – данный параметр отвечает за автоматическое создание связанных подзапросов для запроса, отправленного сэмплером, для которого предусмотрен редирект.

В дополнение к общим настройках, в данном элементе есть возможность задать фильтрацию запросов, поскольку некоторые из них не представляют важности для нагрузки. Это могут быть отдельные страницы, или, например, все js-скрипты и т.д. Фильтрацию запросов можно задать во вкладке **Requests Filtering**, воспользовавшись настройкой **URL Patterns To Include**, задав в него фильтр(ы) для отсева нежелательных запросов в виде регулярных выражений (Рис. 85).

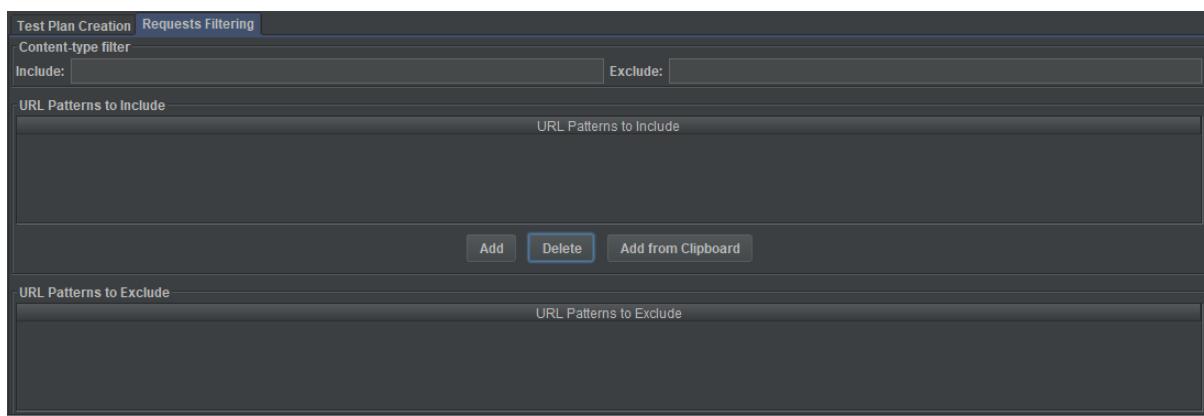


Рис. 85. Настройка URL Patterns To Include

Примеры регулярных выражений для фильтрации:

- `.*\png.*;`
- `.*\gif.* ;`
- `.*\jpg.*;`
- `.*\php.*;`
- `.*\jsp.*;`
- `.*\html.*;`
- `.*\css.*;`
- `.*\svg.*;`
- `.*\js.*.`

Пока оставим настройки фильтрации пустыми и запустим прокси-сервер JMeter.

Для запуска прокси необходимо нажать на кнопку  расположенную в панели **State** элемента HTTP(s) Test Script Recorder. После нажатия на кнопку появиться диалоговое окно с просьбой установить корневой сертификат корневого центра сертификации (CA) для возможности прослушивания HTTP-s трафика. Поскольку сайт в нашем примере, трафик с которого необходимо записывать, поддерживает протокол HTTP, установка данного сертификата не требуется и достаточно нажать на кнопку **OK** (Рис. 86).

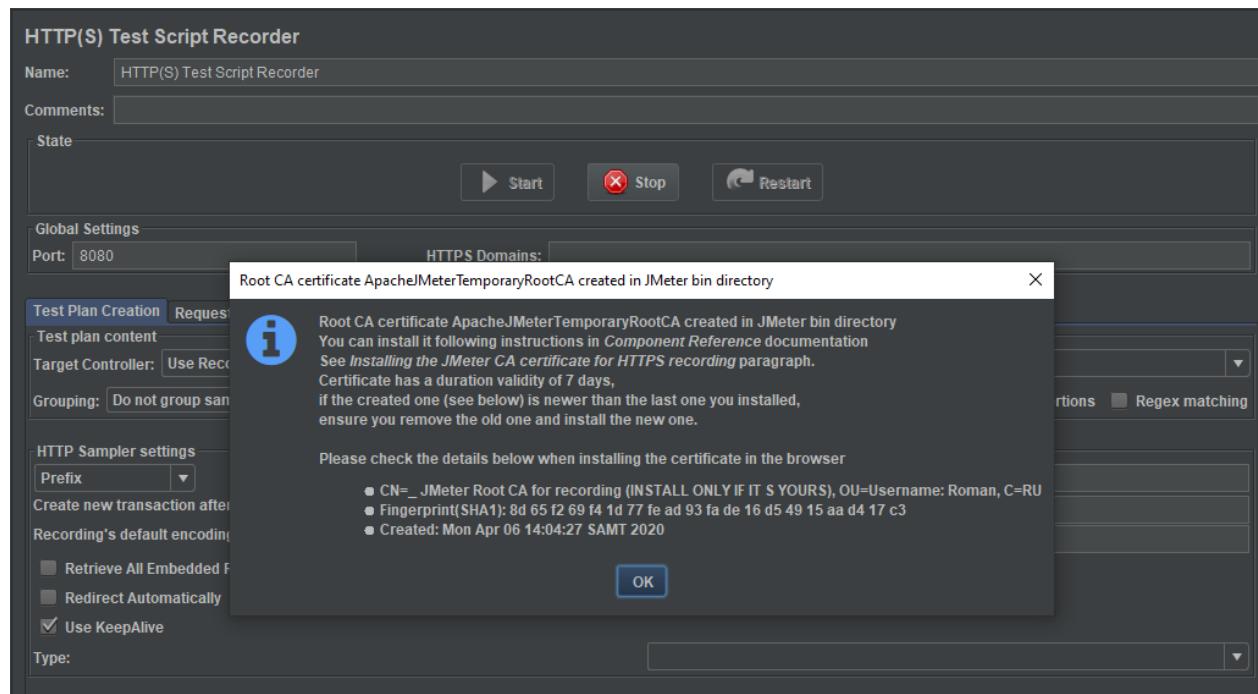


Рис. 86. Окно запуска с помощью HTTP(s) Test Script Recorder

Следующим шагом является настройка прокси-сервера в самом браузере.

ВАЖНО! Для перехвата трафика с сайтов, использующих HTTPS протокол, необходимо установить SSL-сертификат. Его настройка и установка разобраны в главе "[Настройка HTTPS](#)".

4.2.1. Настройка HTTPS

На сайтах, использующих HTTPS протокол, происходит шифрование передаваемых данных с помощью SSL-сертификата, который гарантирует отсутствие шпионов (прокси) между браузером и сервером, на который вы обращаетесь.

Для того чтобы прокси-сервер JMeter смог расшифровать HTTPS трафик, необходимо установить сертификат доверенного центра сертификации JMeter. Данный сертификат создается после нажатия на кнопку **Start** в папке **bin** вашего JMeter и называется ApacheJMeterTemporaryRootCA.crt.

Примечание: по умолчанию генерируемый с помощью JMeter сертификат действует в течение 7 дней. Однако существует способ изменить его срок действия. Для этого выполните следующее:

1. Закройте JMeter.
2. Откройте файл **jmeter.properties**, расположенный в папке bin вашего Jmeter.
3. Найдите следующие строки:

```
# The default validity for certificates created by JMeter
proxy.cert.validity=7
```

4. Измените значение строки proxy.cert validity. Например, если вы хотите, чтобы сертификат действовал в течение года, введите proxy.cert.validity=365. Сохраните файл.
5. Удалите существующий сертификат ApacheJMeterTemporaryRootCA.crt.
6. Запустите Jmeter.

Далее переходим к установке самого сертификата.

4.2.1.1. Установка центра сертификации в Mozilla Firefox

Чтобы установить доверенный сертификат JMeter в Firefox необходимо выполнить следующее:

1. Откройте главное меню браузера и перейдите в **Настройки** (Рис. 87).

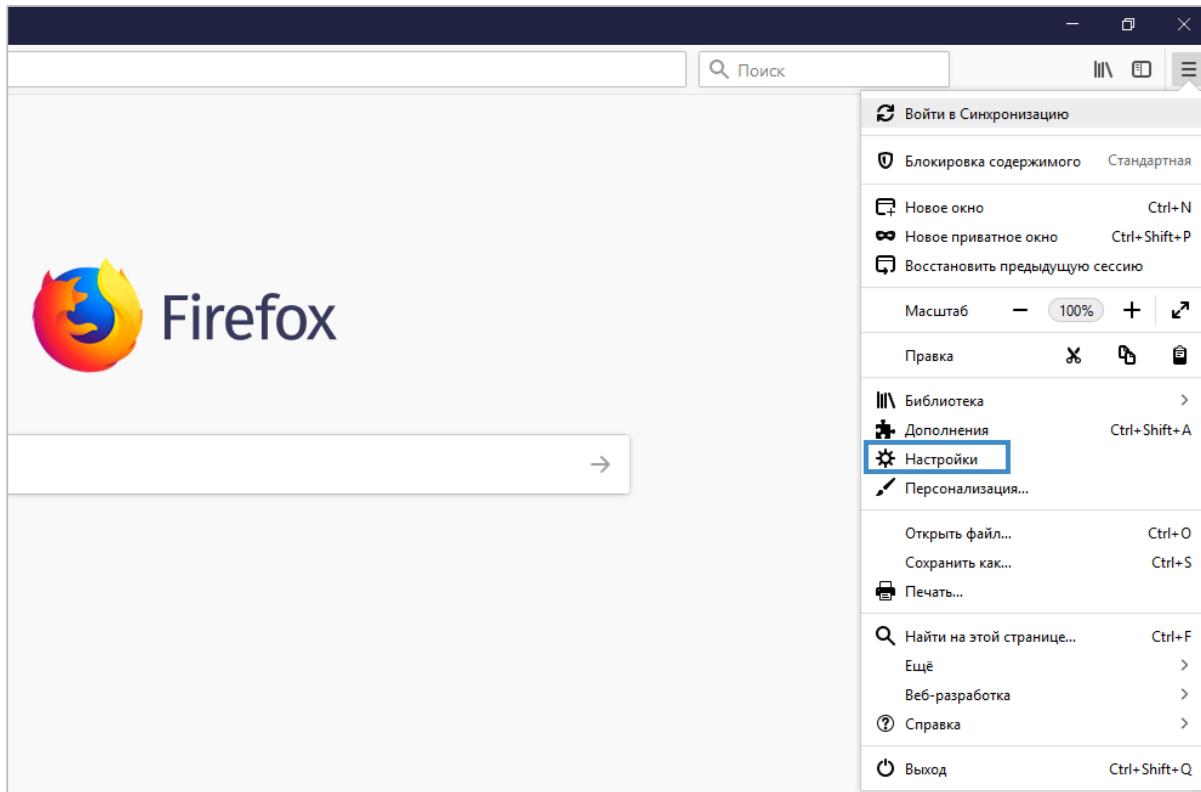


Рис. 87. Главное меню браузера Mozilla Firefox

2. В разделе **Приватность и защита** прокрутите страницу вниз до панели **Сертификаты** и нажмите **Просмотр сертификатов** (Рис. 88).

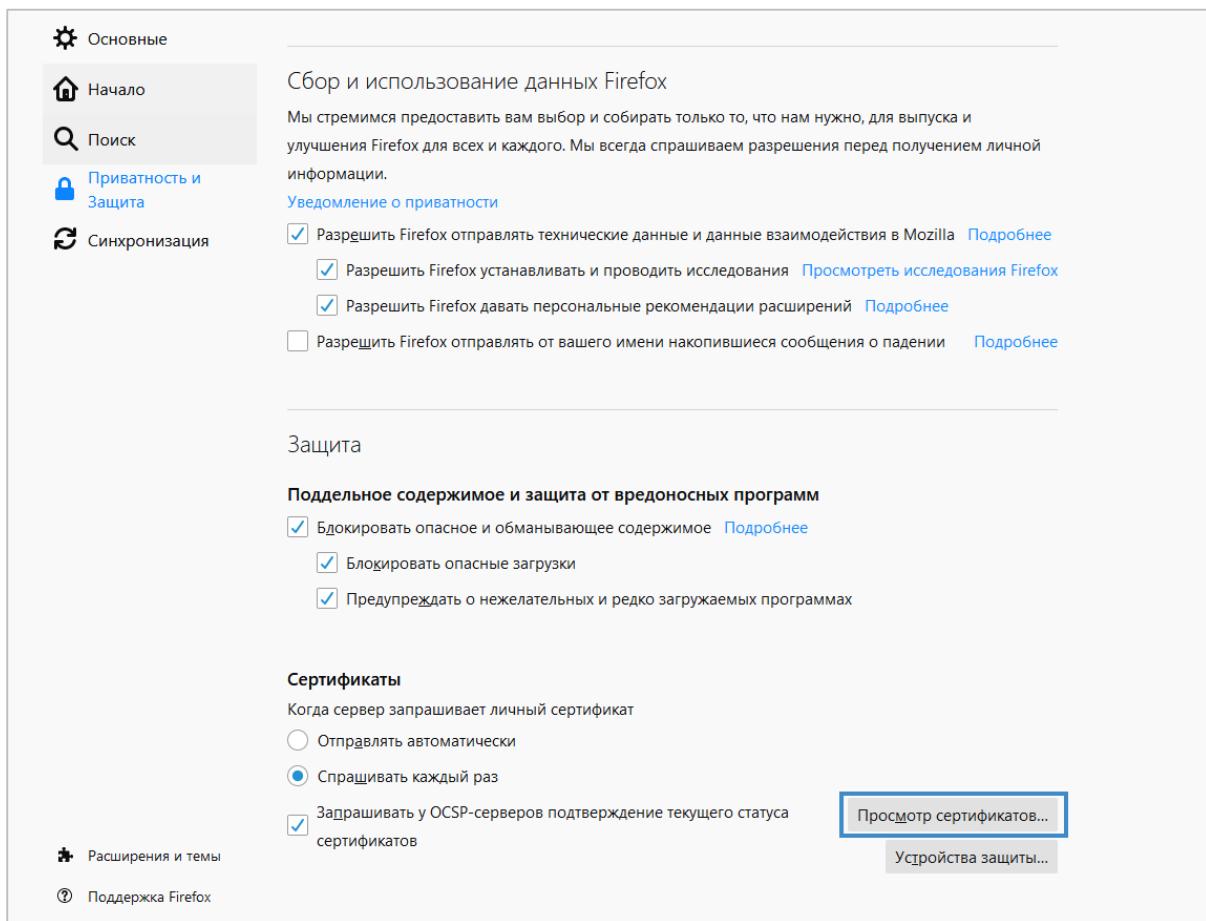


Рис. 88. Приватность и защита. Сертификаты. Просмотр сертификатов

3. В открывшемся окне на вкладке **Центры сертификации** нажмите на кнопку **Импортировать** (Рис. 89).

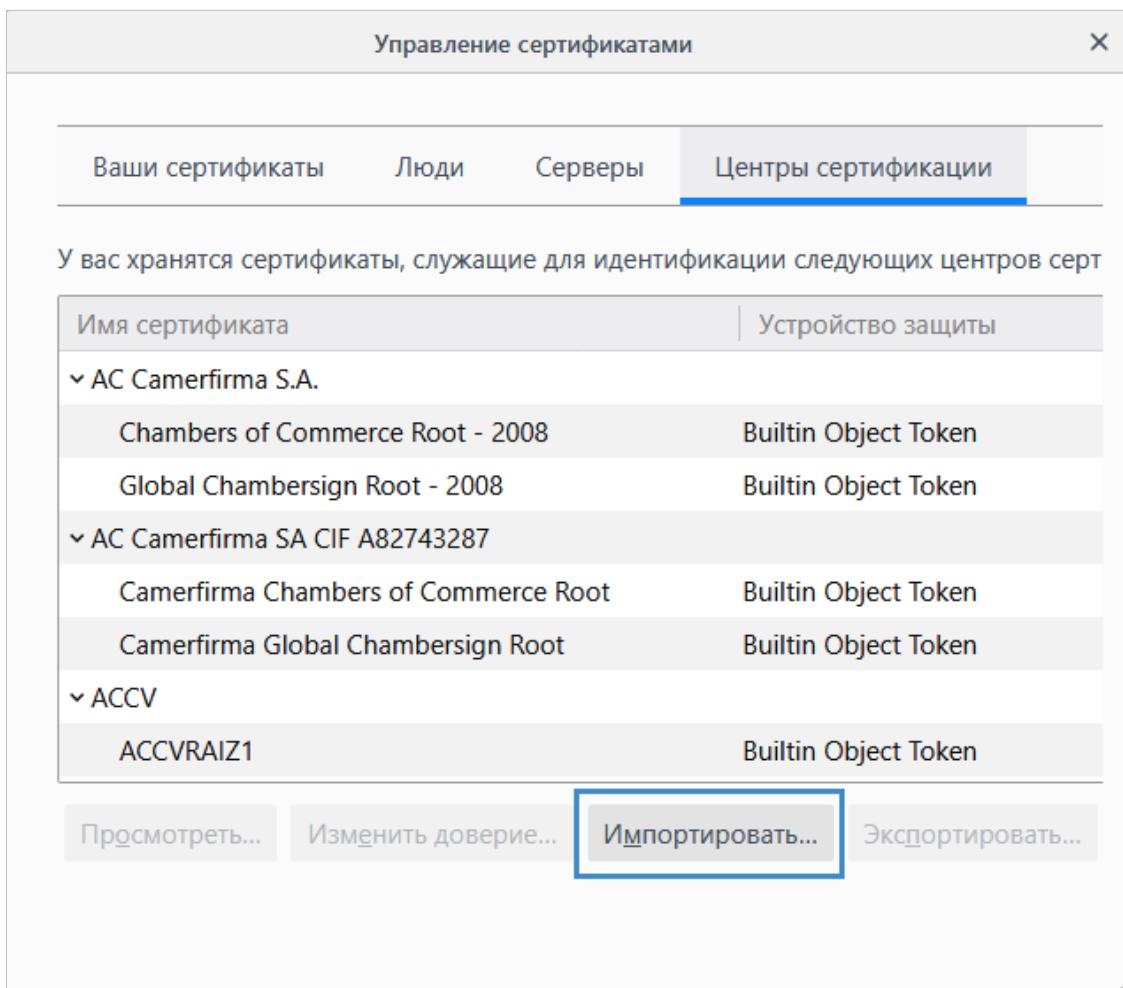


Рис. 89. Центры сертификации. Кнопка "Импортировать"

4. Выберите **ApacheJMeterTemporaryRootCA.crt**, расположенный в папке **bin** вашего JMeter. В открывшемся окне поставьте флажок на **Доверять при идентификации веб-сайтов** (Рис. 90).

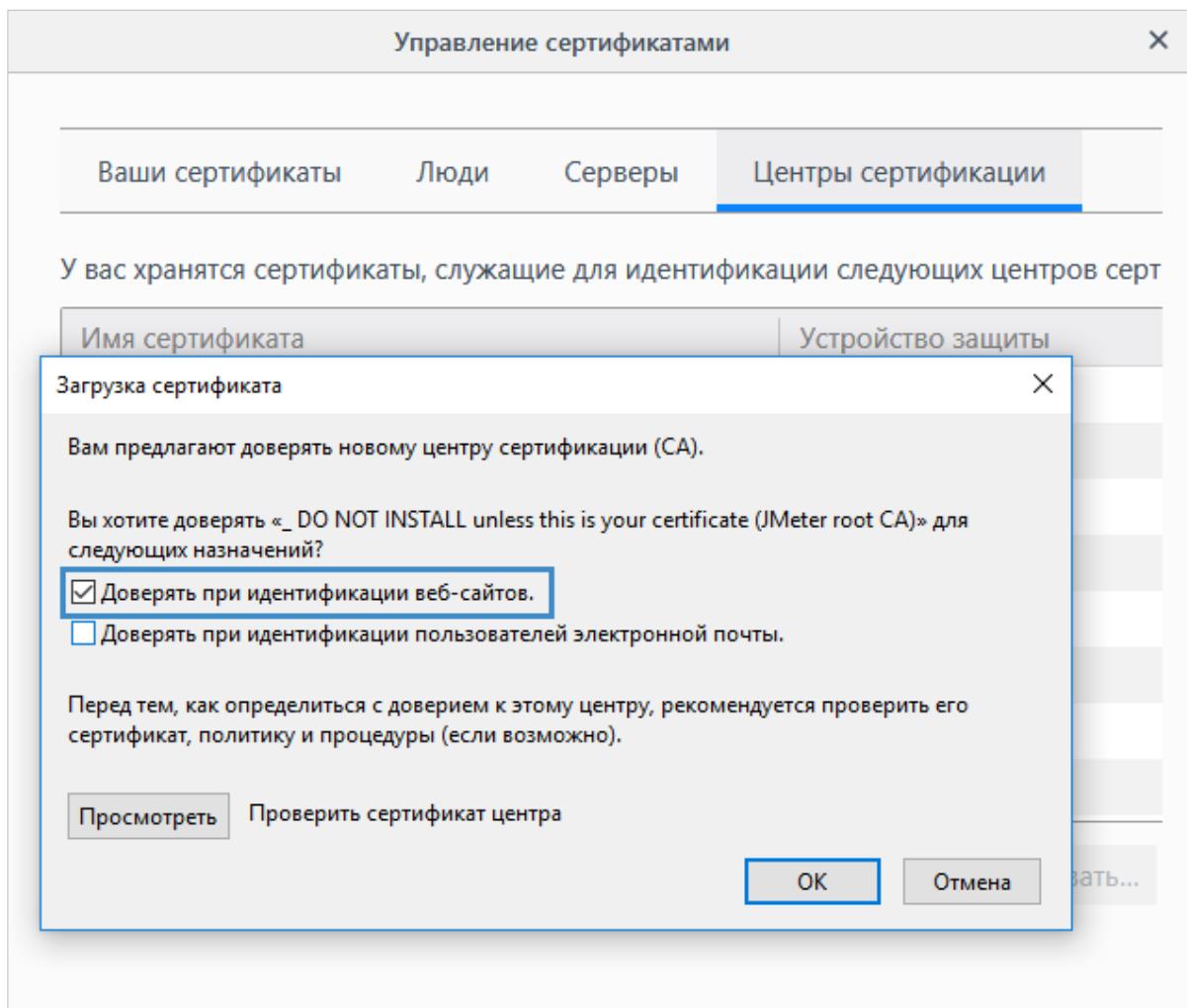


Рис. 90. Окно загрузки сертификатов

5. Загруженный сертификат должен отобразиться на вкладке **Центры сертификации** (Рис. 91).

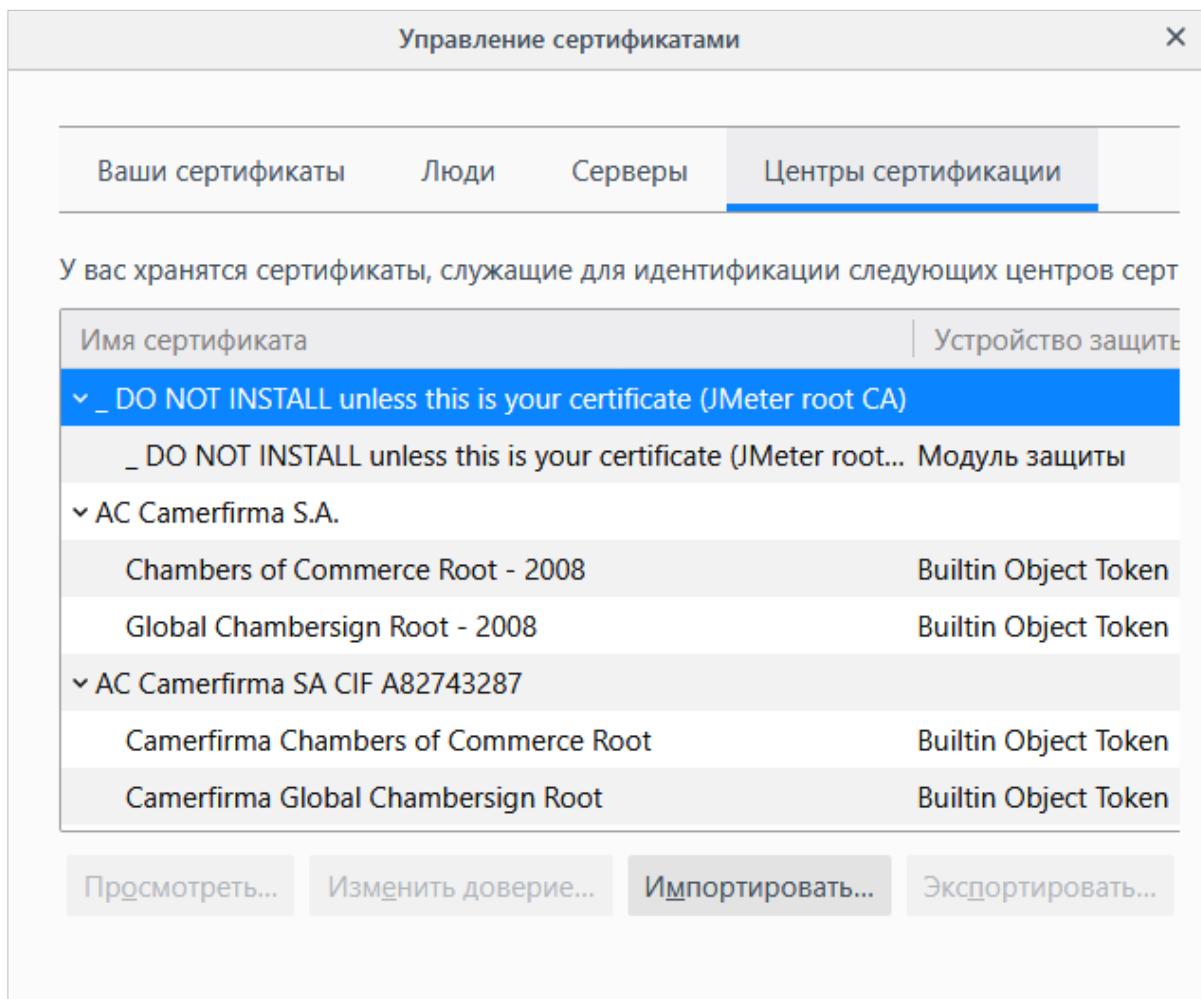


Рис. 91. Управление сертификатами

6. Закройте диалоговое окно. Сертификат готов к работе.

4.2.1.2. Установка сертификата в Google Chrome

Чтобы установить доверенный сертификат JMeter в Google Chrome, необходимо выполнить следующее:

1. Откройте главное меню браузера и перейдите в **Настройки** (Рис. 92).

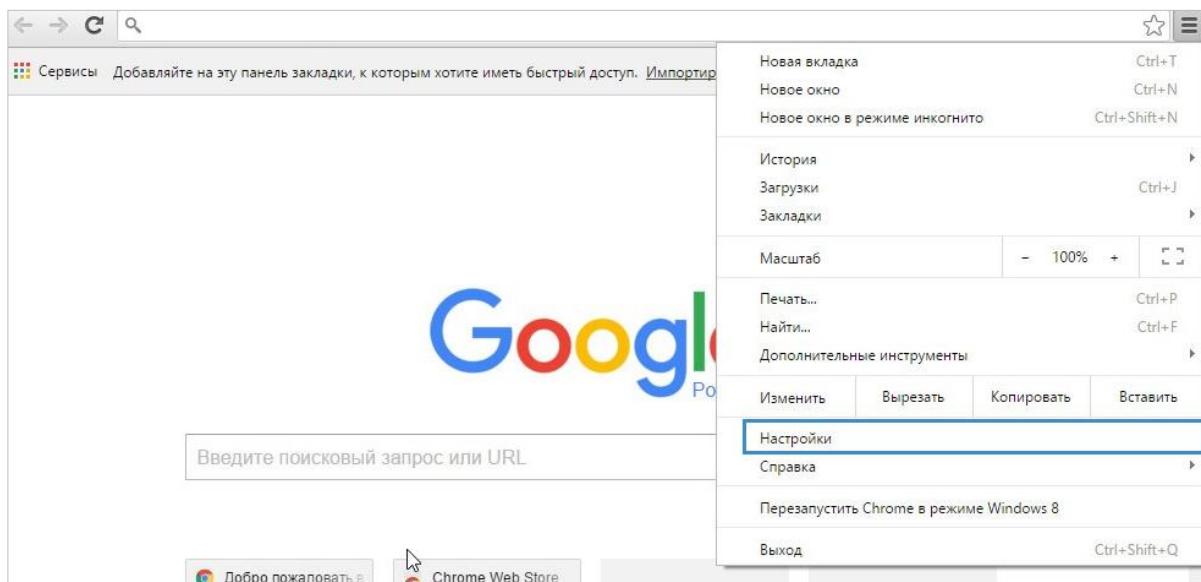


Рис. 92. Веб-браузер Google Chrome. Настройки

2. В поле поиска настроек введите "безопасность" и нажмите **Настройте сертификаты**, чтобы в зависимости от типа используемой ОС открыть управление настройками ее сертификатов (Рис. 93).

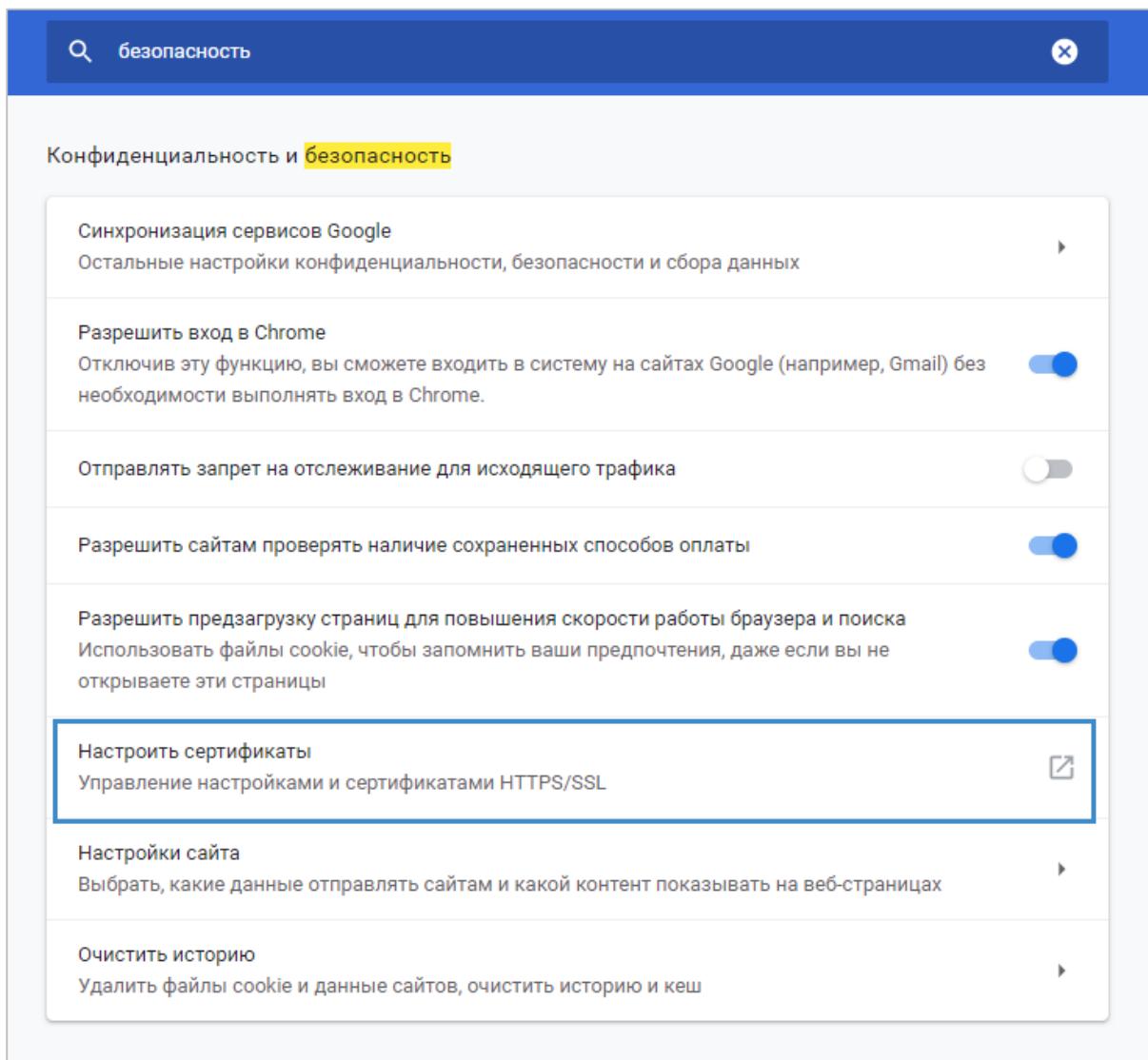


Рис. 93. Настройка безопасности в веб-браузере Google Chrome

3. В открывшемся окне перейти на вкладку **Доверенные корневые центры сертификации** и нажмите на кнопку **Импорт** (Рис. 94).

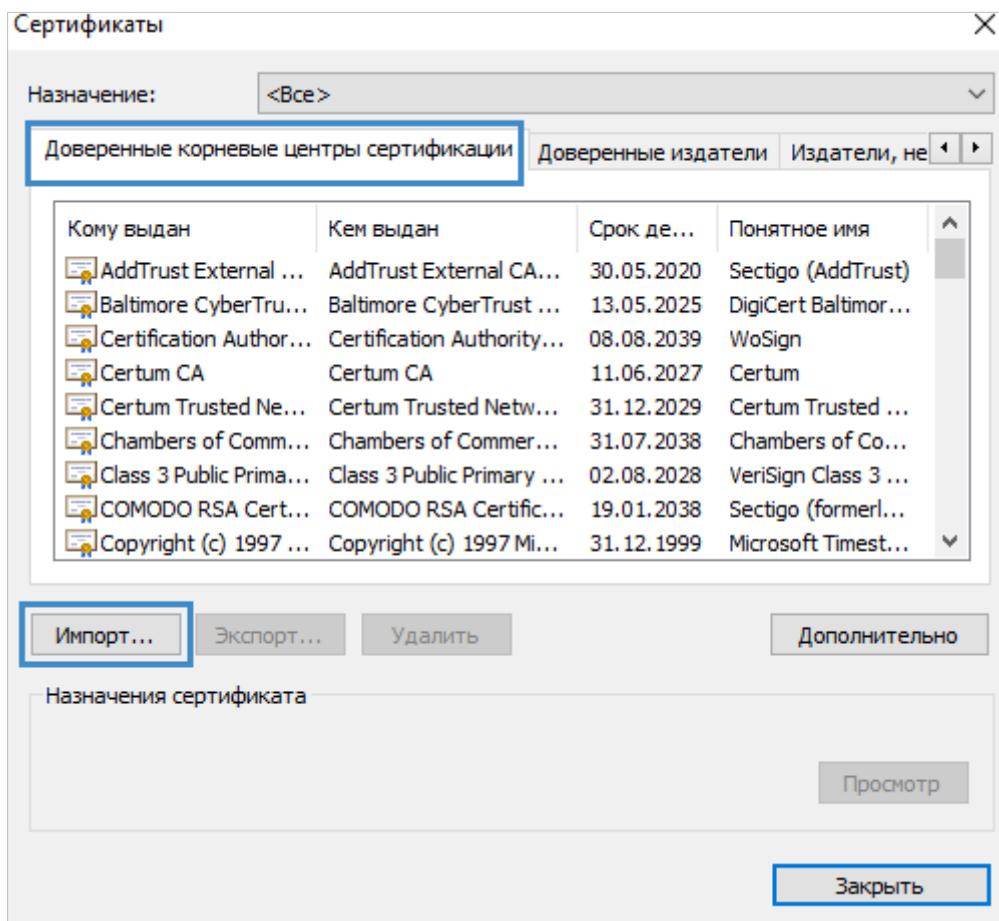


Рис. 94. Доверенные корневые центры сертификации. Импорт

4. Откроется мастер импорта сертификатов. В качестве импортируемого файла выберите **ApacheJMeterTemporaryRootCA.crt**, расположенный в папке **bin** вашего JMeter. Следующие шаги мастера оставляйте без изменений (Рис. 95).

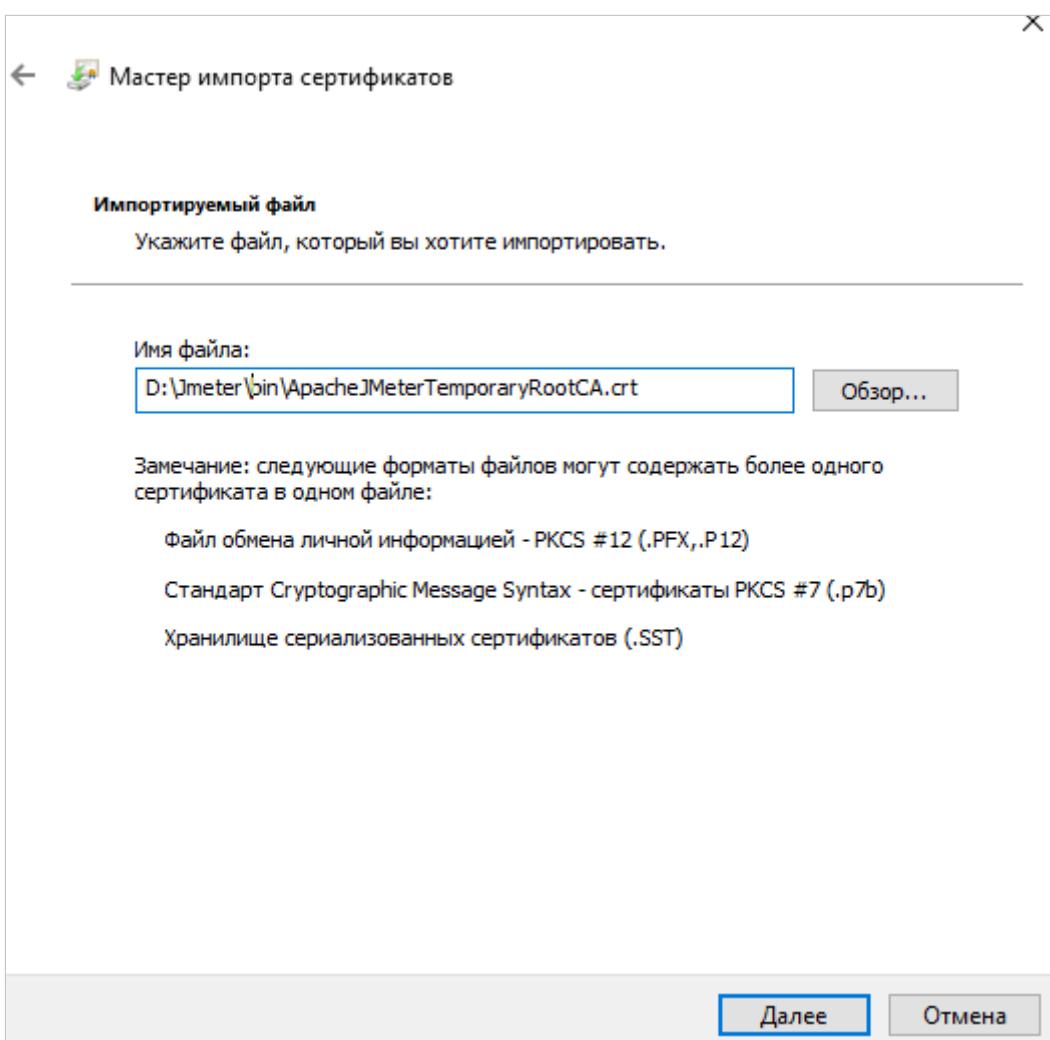


Рис. 95. Мастер импорта сертификатов

5. На последнем этапе процесса импорта появится диалоговое окно с разрешением на установку сертификата. Нажмите на кнопку **Да** (Рис. 96).

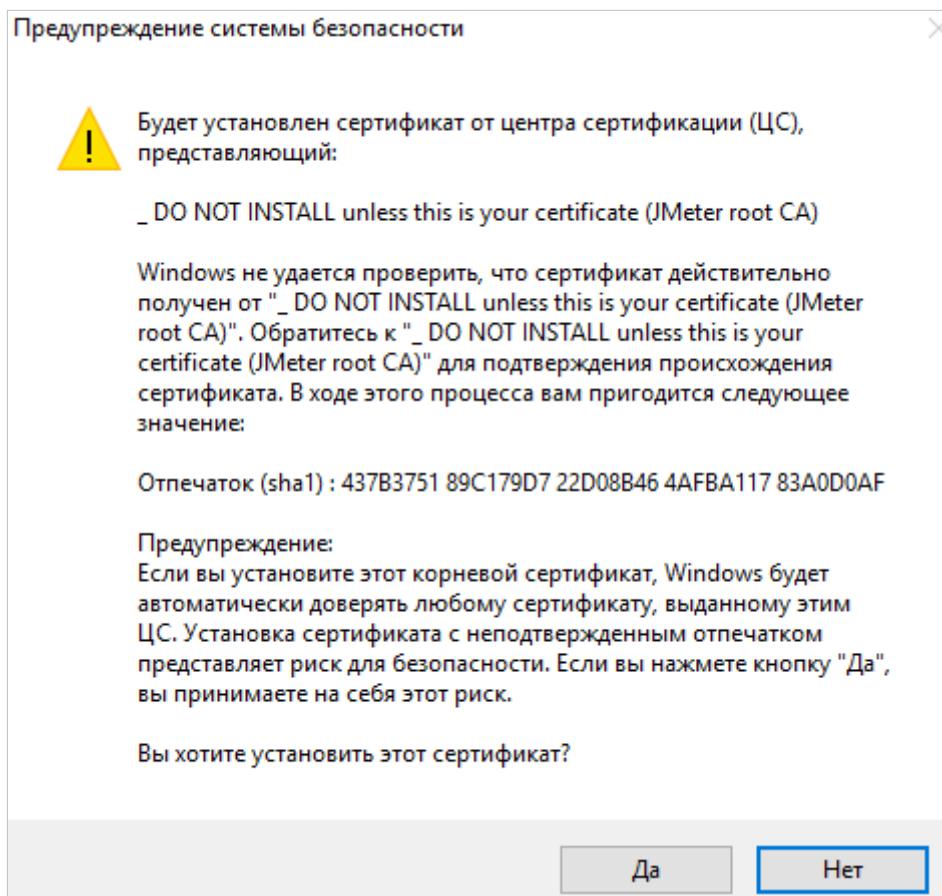


Рис. 96. Предупреждение системы безопасности

6. После завершения процесса импорта сертификат должен отобразиться в списке доверенных корневых центров сертификации (Рис. 97).

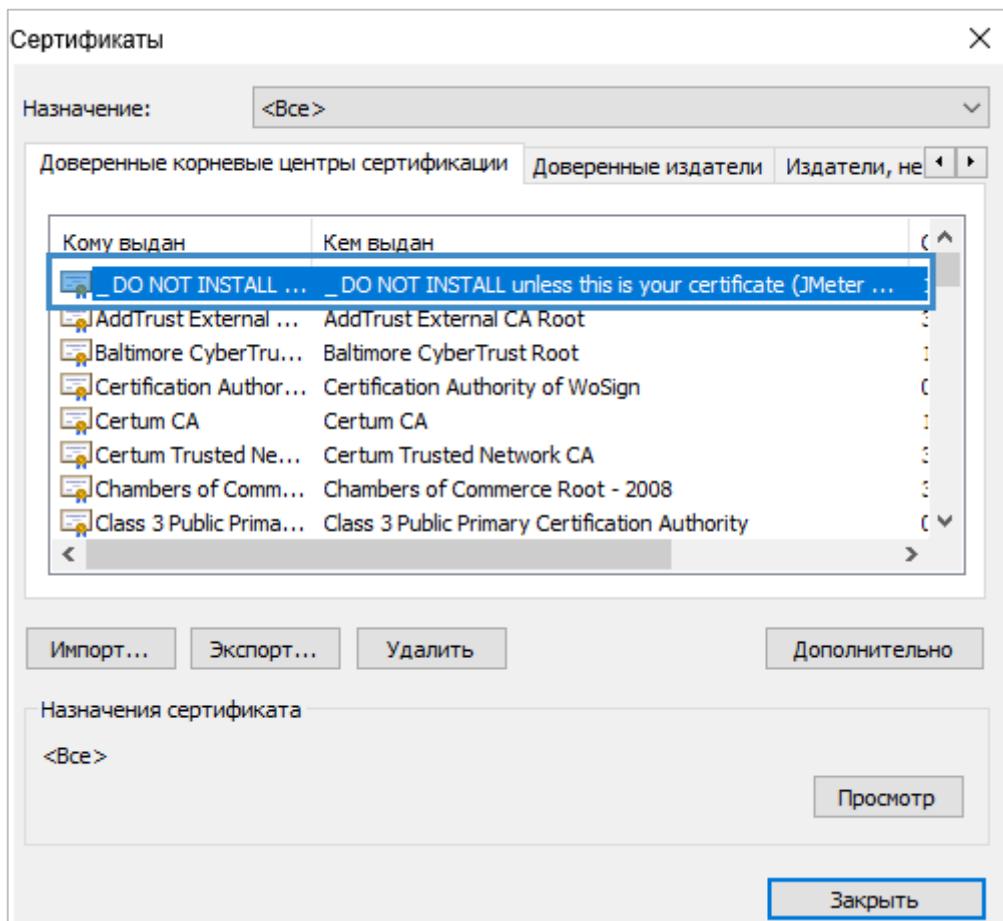


Рис. 97. Список доверенных корневых центров сертификации

4.2.2. Настройка прокси

4.2.2.1. Firefox

Чтобы включить прокси-сервер Jmeter в Firefox, выполните следующее:

1. Откройте главное меню браузера и перейдите в **Настройки** (Рис. 98).

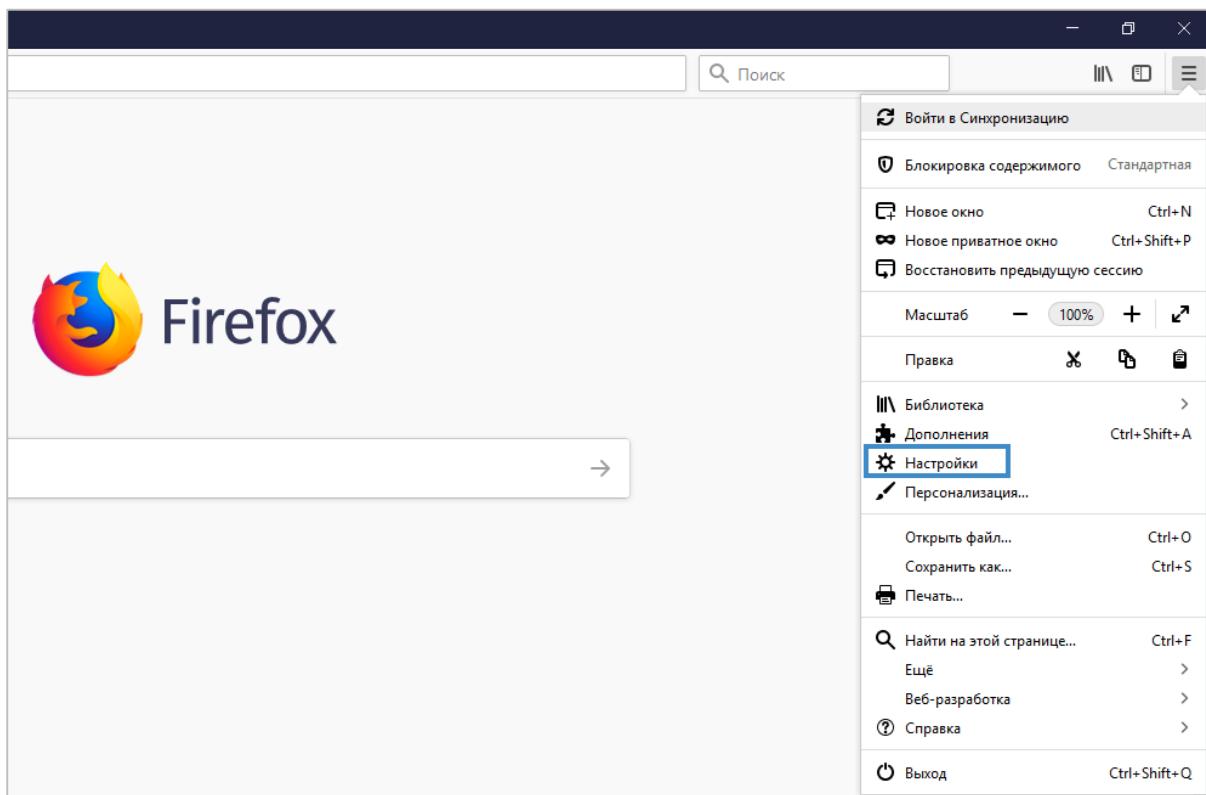


Рис. 98. Веб-браузер Firefox. Настройки

2. В разделе **Основные** прокрутите страницу вниз до панели **Параметры сети** и нажмите на кнопку **Настроить** (Рис. 99).

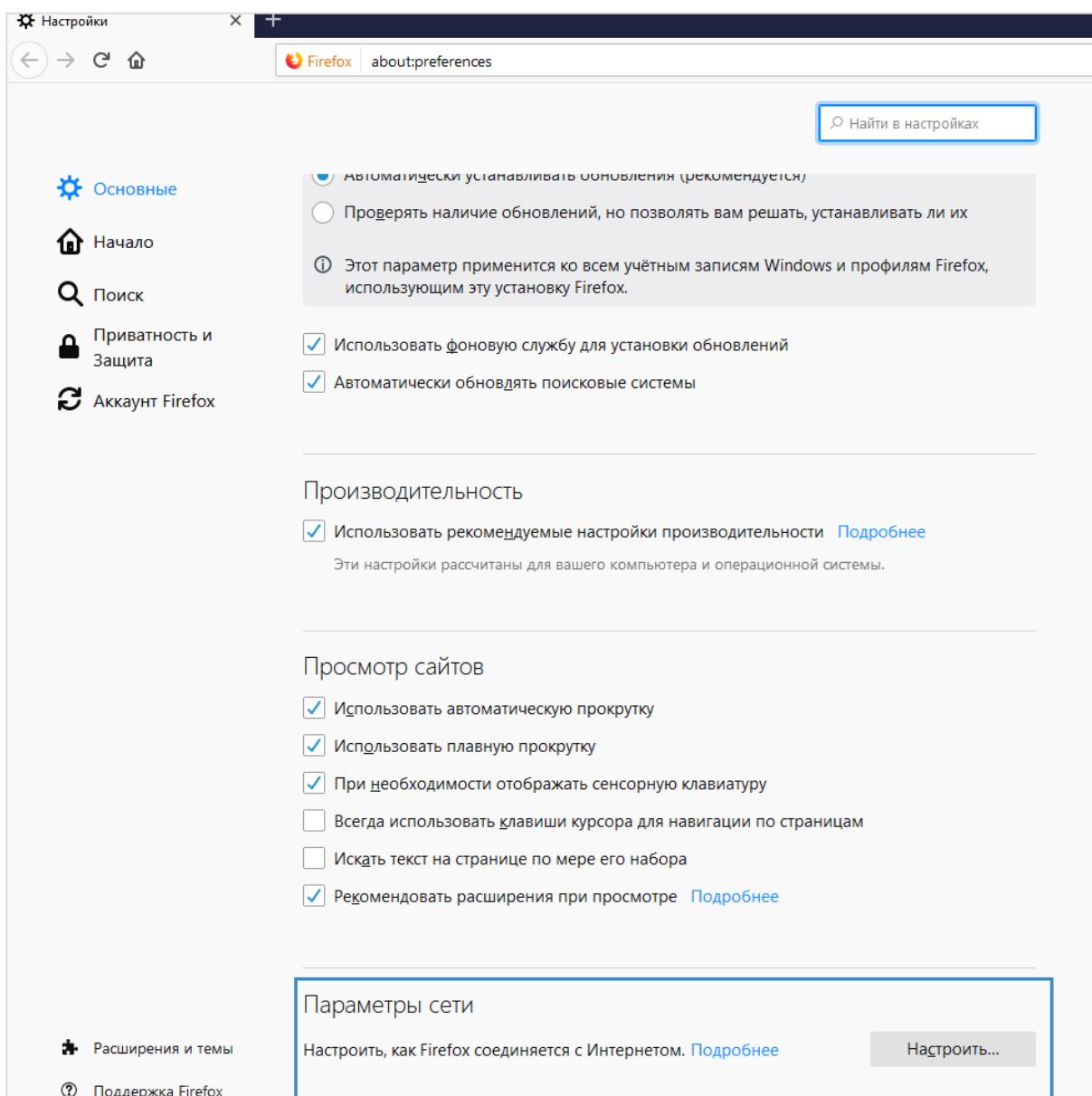


Рис. 99. Основные. Параметры сети

- Выберите **Ручная настройка прокси** и в поле **HTTP прокси** введите "localhost", в качестве порта установите порт, указанный в настройках **HTTP(s) Test Script Recorder** (в нашем случае – 8080). Остальные настройки оставьте по умолчанию (Рис. 100).

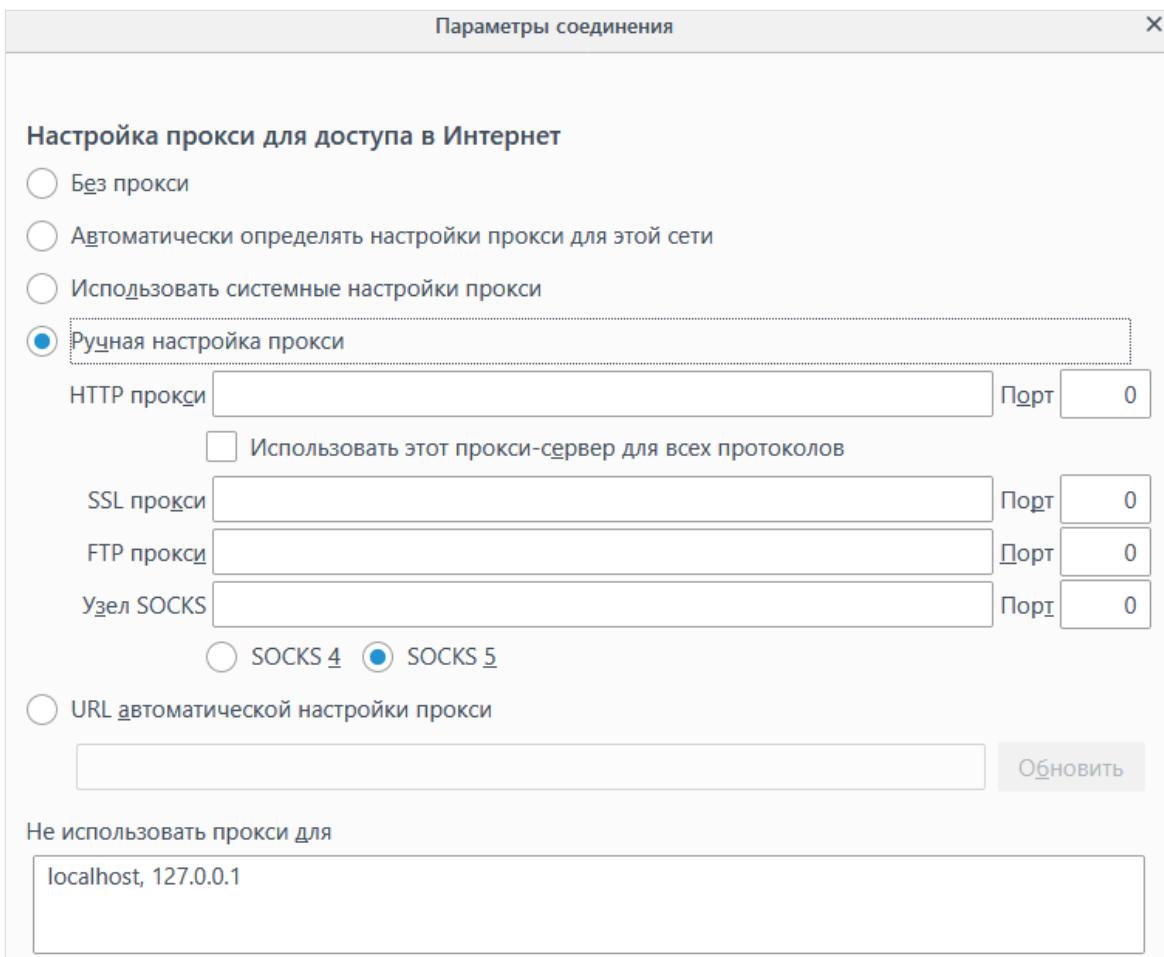


Рис. 100. Параметры соединения

Примечание: если прокси используется для чтения трафика с локального сайта, то в поле **Не использовать прокси для** очистите "localhost" и "127.0.0.1".

4. Нажмите **OK**.

4.2.2.2. Google Chrome

Чтобы включить прокси-сервер Jmeter в Google Chrome, выполните следующее:

1. Откройте главное меню браузера и перейдите в **Настройки** (Рис. 101).

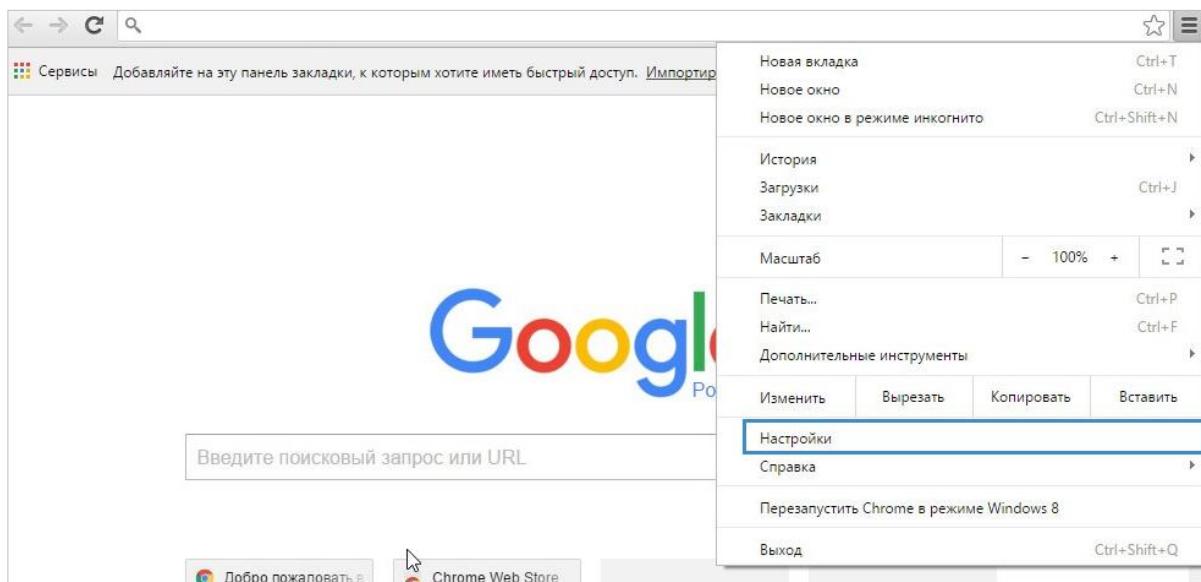


Рис. 101. Веб-браузер Google Chrome. Настройки

2. В поле поиска настроек введите "прокси" и нажмите на кнопку **Открыть настройки прокси-сервера для компьютера**, которая в зависимости от типа используемой ОС откроет настройки ее сетевых подключений (Рис. 102).

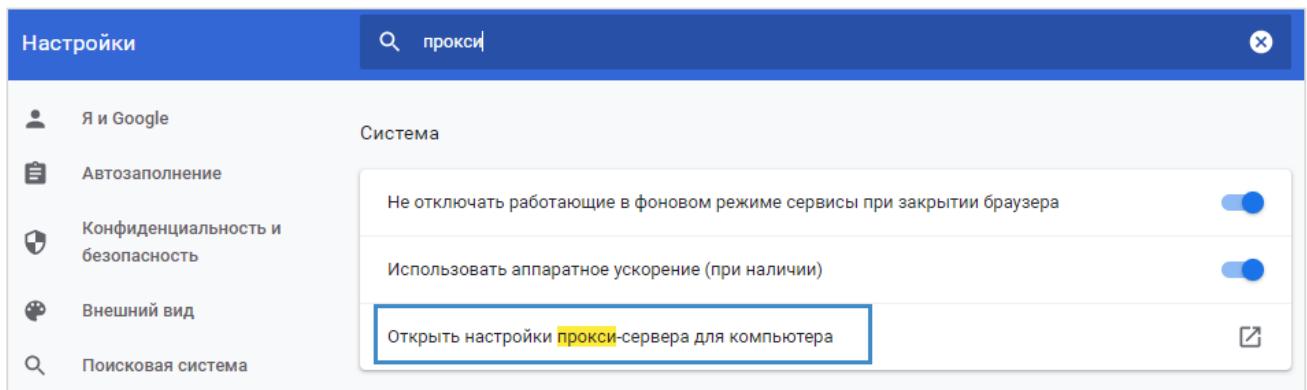


Рис. 102. Настройки прокси-сервера

3. В панели **Настройка параметров локальной сети** нажмите на кнопку **Настройка сети** (Рис. 103).

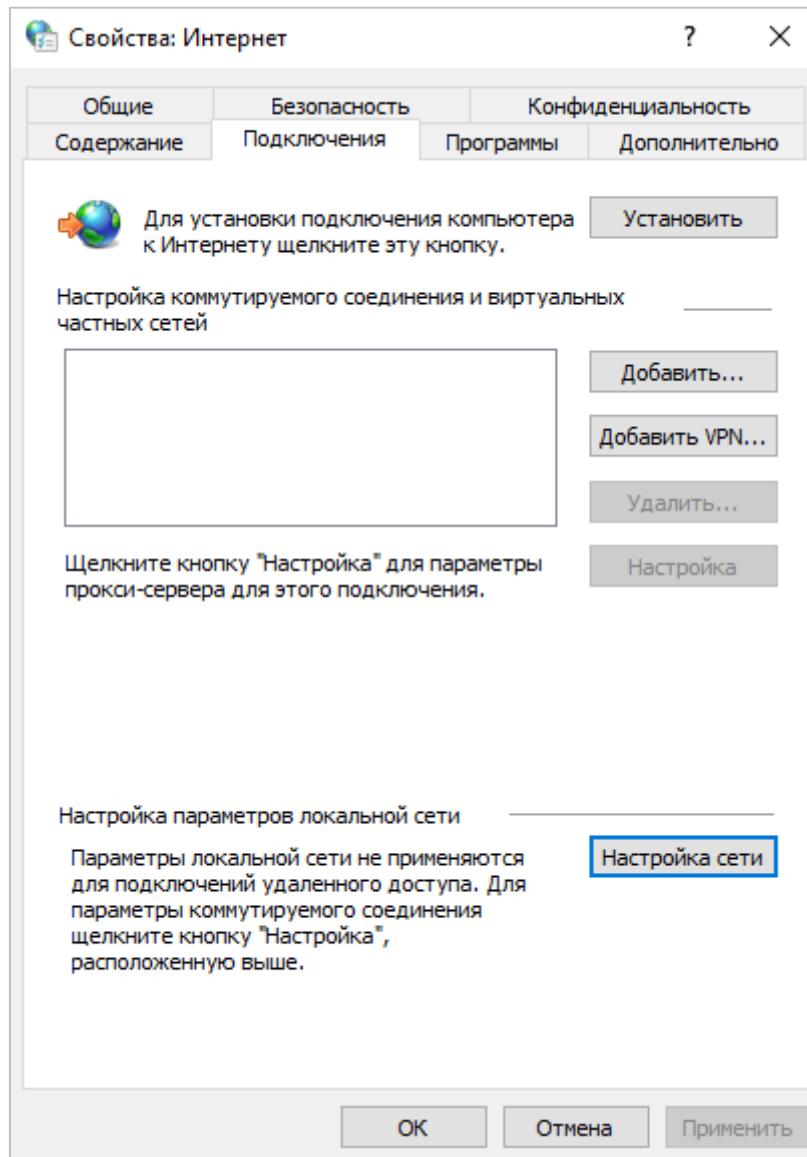


Рис. 103. Свойства: Интернет. Настройка параметров локальной сети. Настройка сети

4. В открывшемся окне выберите **Использовать прокси-сервер для локальных подключений** и нажмите на кнопку **Дополнительно** (Рис. 104).

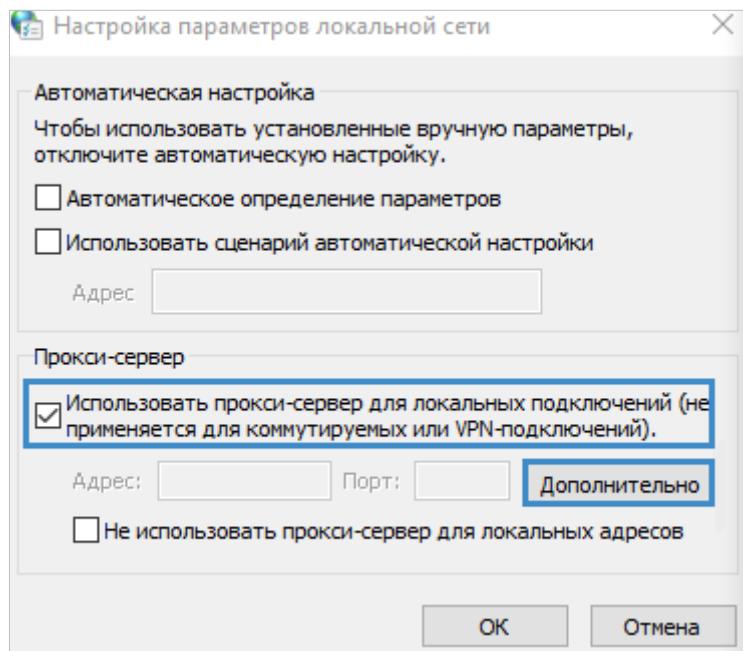


Рис. 104. Настройка параметров локальной сети

Примечание: если прокси используется для чтения трафика с локального сайта, убедитесь, что параметр **Не использовать прокси-сервер для локальных адресов** не включен.

5. В окне параметров прокси-сервера в поле **HTTP** и **Secure** (если используется через SSL) в поле адреса прокси-сервера введите 127.0.0.1, в качестве порта установите порт, указанный в настройках **HTTP(s) Test Script Recorder** (в нашем случае – 8080). Остальные настройки оставьте по умолчанию (Рис. 105).

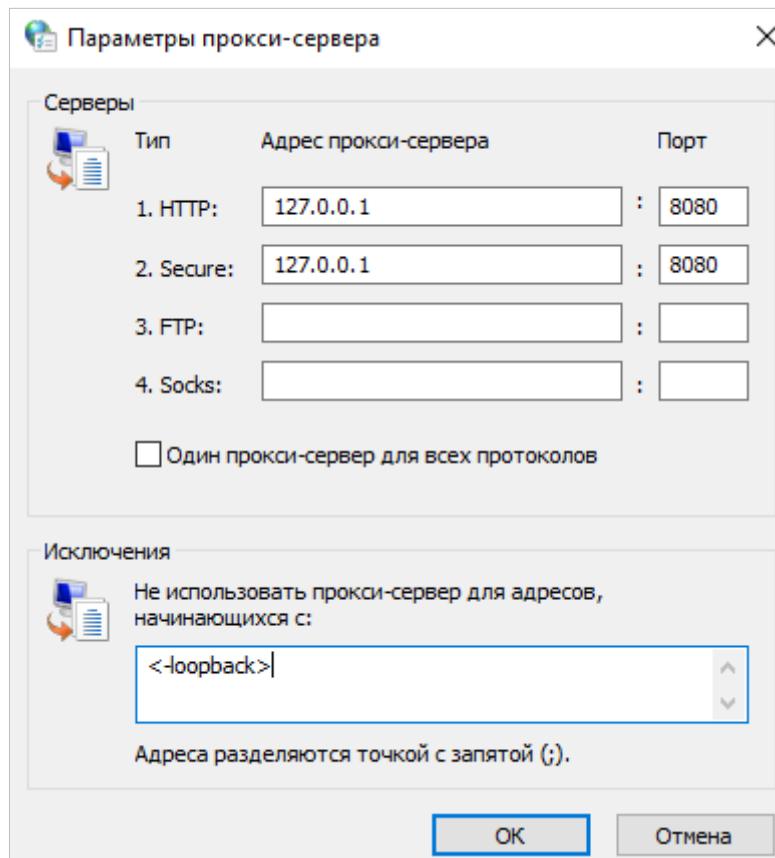


Рис. 105. Параметры прокси-сервера

6. Закройте все окна, нажав на кнопку **OK**.

4.2.3. Запись тестового сценария

После того как все настроено, переходим к записи самого сценария нагрузочного тестирования. Возвращаемся в JMeter, в элемент [HTTP\(s\) Test Script Recorder](#). Каждую операцию плана, изложенного в главе 4.1, желательно записывать в отдельный [Recording Controller](#), тем самым задавая логический порядок всем частям сценария. В этом поможет настройка [Target Controller](#), в которой необходимо выбрать из списка необходимый контроллер. Далее нажимаем на кнопку **Start** для запуска прокси и начинаем выполнять действия в браузере, следуя пунктам плана:

1. Авторизация пользователя:

- в адресную строку вводим адрес тестируемого сервера;
- вводим логин и пароль. Нажимаем **Войти в систему**;
- удостоверяемся, что пользователь авторизован.

После проделанных действий возвращаемся в элемент HTTP(s) Test Script Recorder и нажимаем на кнопку **Stop**. Тем временем наш Recording Controller наполнился данными (Рис. 106)

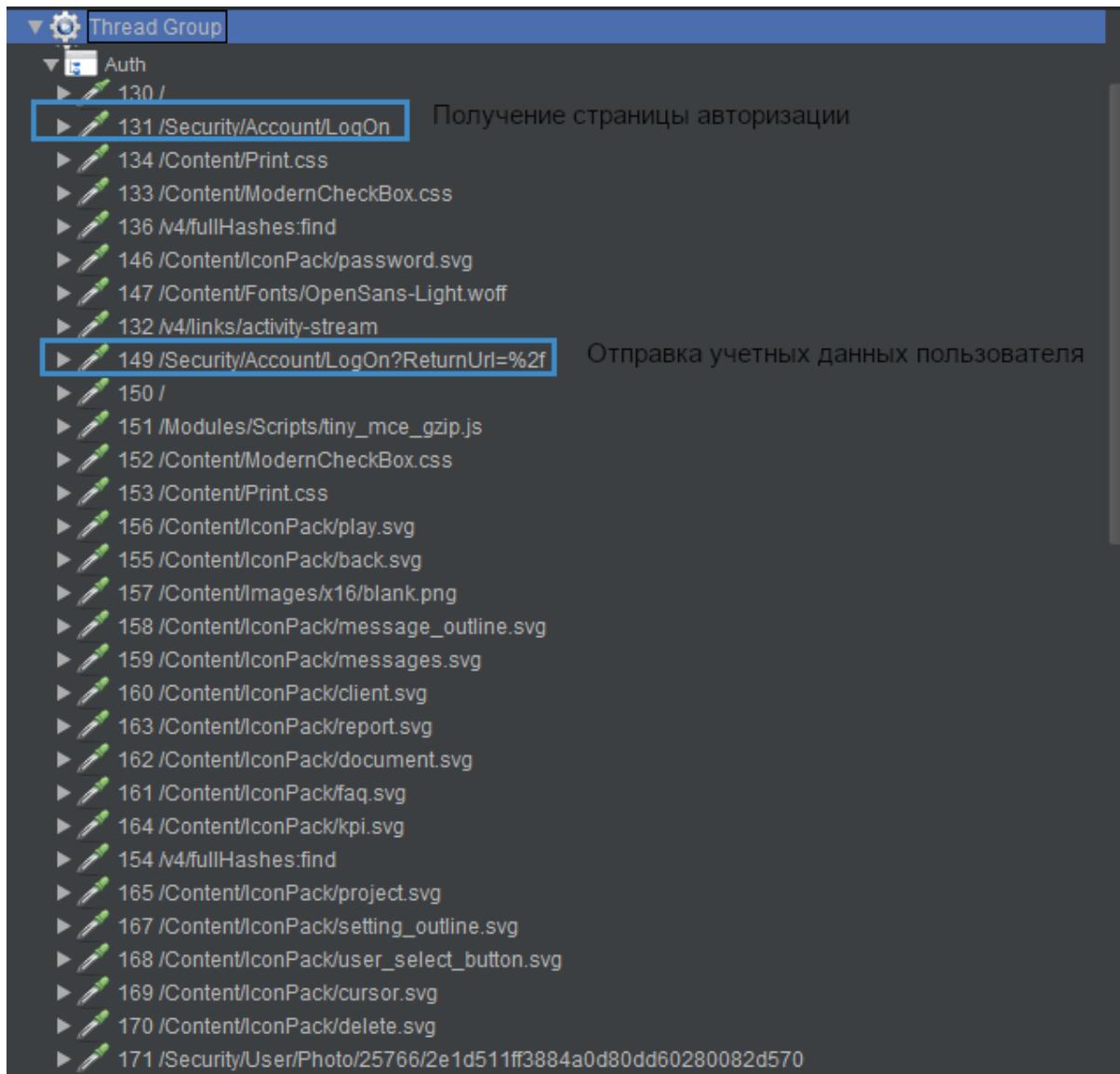


Рис. 106. Наполнение данными Recording Controller

Записанные в Recording Controller запросы являются элементами, называемыми [HTTP Request Sampler](#). Как можно заметить, среди записанных элементов есть те, которые оканчиваются на .js, .png, .svg, .ico, .css, представляющие собой запросы на загрузку различных картинок и стилей.

Как правило, такие запросы излишни и сильно засоряют сценарий, поэтому самым оптимальным вариантом является убрать их из плана и оставить только те запросы, которые нужны для проверки нагрузки на конкретный объект тестирования. Для дальнейшего отсеивания таких запросов необходимо включить фильтрацию запросов. Для этого

необходимо перейти на вкладку **Requests filtering** элемента HTTP(s) Test Script Recorder и в поле **URL Patterns to Exclude** задать шаблоны, по которым они будут отбираться. Примеры шаблонов фильтрации можно посмотреть [здесь](#).

Все действия, совершаемые пользователем для входа в систему, происходят на странице авторизации, соответственно, нас будут интересовать запросы, имеющие строчку "Security/Account/LogOn" в качестве URL. Среди полученных результатов имеется 2 подходящих, и они расположены в том же порядке, в котором были совершены действия на странице, т.е. сверху низ. Чтобы окончательно убедиться в этом, необходимо перейти в каждый из запросов (Рис. 107).

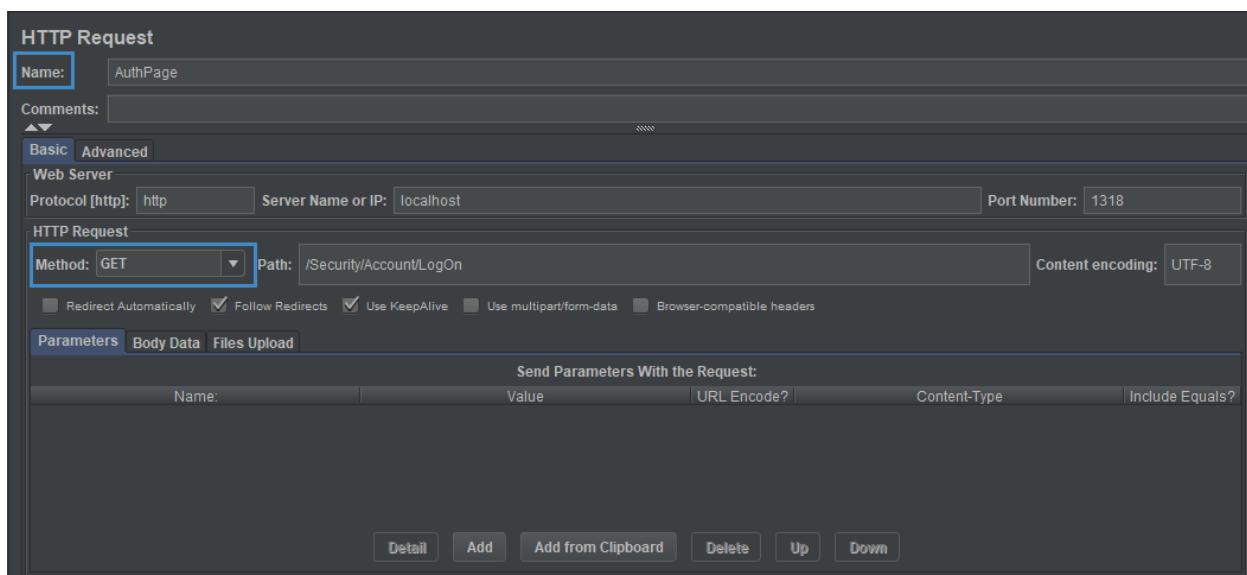


Рис. 107. HTTP Request получения страницы авторизации

Как и предполагалось, верхний запрос выполняется методом GET и представляет собой получение страницы авторизации. Для лучшего восприятия нашего тестового плана в поле **Name** зададим данному элементу соответствующее название – "AuthPage" (Рис. 108).

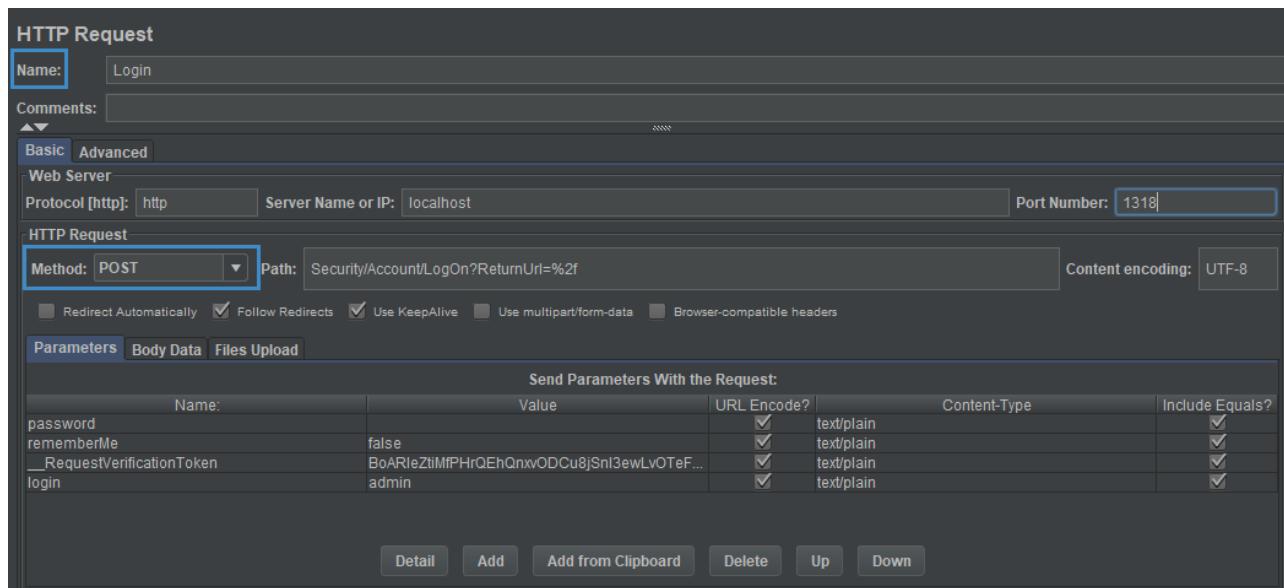


Рис. 108. Конфигурирование HTTP Request Sampler

Данный запрос выполняется уже методом POST и отправляется по нажатию кнопки на странице авторизации, передавая серверу в теле данные, показанные на рисунке. В поле **Name** зададим элементу соответствующее название – "LogIn".

Таким образом, данная группа запросов инициализирует создание активной сессии виртуального пользователя в системе, в ходе которой он будет выполнять последующие действия с документом. Дальнейшее их конфигурирование будет разобрано в главе [4.4.4](#). Также благодаря настройке **Capture HTTP Headers** внутри каждого записанного сэмплера находится элемент HTTP Header manager с уже заданным в нем набором передаваемых заголовков.

После удаления лишних элементов наш тестовый план будет выглядеть следующим образом (Рис. 109).

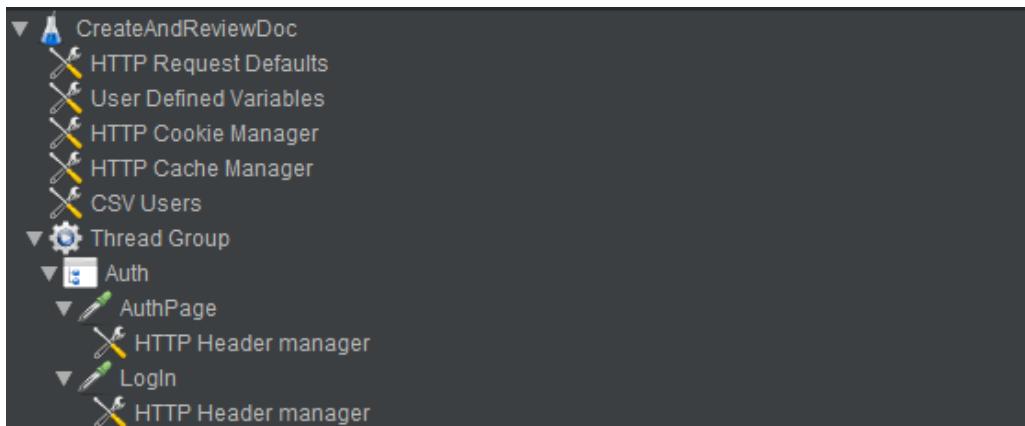


Рис. 109. Тестовый план

2. Создание документа.

Авторизовавшись, мы переходим к созданию документа. Добавим в Jmeter еще один элемент Recording Controller и переименуем его в CreateDoc. Переходим в HTTP(s) Test Script Recorder, в поле **Target Controller** из списка выбираем добавленный нами контроллер. Нажимаем на кнопку **Start** и начинаем выполнять следующие действия в браузере:

- на главной странице системы нажимаем на кнопку **Создать документ**;
- из списка документов выбираем необходимый тип документа и нажимаем на кнопку **Создать**;
- на форме документа заполняем поля **Название** и **Предмет договора**, а также загружаем файл-версию документа;
- нажимаем на кнопку **Сохранить**.

После проделанных действий возвращаемся в элемент HTTP(s) Test Script Recorder и нажимаем на кнопку **Stop**. Как мы видим, в этот раз, благодаря фильтрации в Recording Controller, записалось куда меньше лишних элементов (Рис. 110).

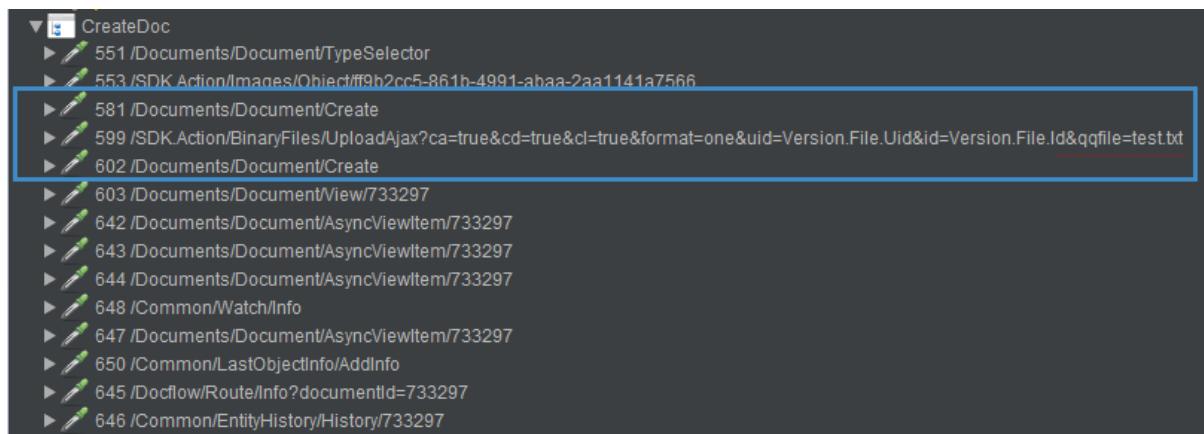


Рис. 110. Результаты записи создания документа через HTTP(s) Test Script Recorder

Из всего списка результатов нас интересует только 3 запроса: 2 из них – GET и POST – имеют строчку "Documents/Document/Create" в качестве URL, поскольку наши действия происходили на форме создания документа, последний совершается методом POST и имеет в URL строку "SDK.Action/BinaryFiles/UploadAjax" и используется для загрузки файла в качестве версии документа. Поскольку в нашем тестовом сценарии версию документа мы заполняем, данный запрос нам понадобится. Зададим для всех трех запросов соответствующие названия: **GetDocForm** – для GET-

запроса получения формы создания документа, **UploadFile** – для POST-запроса загрузки файла-версии документа, **PostDoc** – для POST-запроса отправки формы создания документа.

Удаляем лишние элементы из тестового плана, а также убеждаемся, что оставшиеся запросы идут в правильном порядке.

3. Отправка документа на ознакомление

Согласно плану, после создания документа нам необходимо отправить его на ознакомление. Снова добавляем в Jmeter элемент Recording Controller и переименовываем его в CreateAcquaintanceTask. Переходим в HTTP(s) Test Script Recorder, в поле **Target Controller** из списка выбираем добавленный нами контроллер. Нажимаем кнопку **Start** и начинаем выполнять следующие действия в браузере:

- в верхнем меню карточки созданного нами документа нажимаем на кнопку **Отправить** – **Отправить на ознакомление**;
- на форме отправки на ознакомление заполняем необходимые поля: сроком ознакомления устанавливаем завтрашнее число, в поле **Кому** выбираем администратора ELMA (в зависимости от профиля нагрузки, для выполнения определенных действий в системе могут быть заданы разные пользовательские роли. В нашем случае все операции будут осуществляться одним пользователем). Остальные атрибуты оставим по умолчанию;
- нажимаем на кнопку **Отправить**.

После проделанных действий возвращаемся в элемент HTTP(s) Test Script Recorder и нажимаем на кнопку **Stop** (Рис. 111).

```

▼ CreateAcquaintanceTask
  ► 654 /Documents/Document/View/733297
  ► 668 /Documents/Document/AsyncViewItem/733297
  ► 666 /Documents/Document/AsyncViewItem/733297
  ► 667 /Documents/Document/AsyncViewItem/733297
  ► 665 /Documents/Document/AsyncViewItem/733297
  ► 671 /Common/Watch/Info
  ► 669 /Docflow/Route/Info?documentId=733297
  ► 670 /Common/EntityHistory/History/733297
  ► 701 /Common/LastObjectInfo/AddInfo
  ► 700 /Docflow/Acquaintance/Create
    └─ 717 /Content/IconPack/document.svg
    └─ 715 /Content/IconPack/message_outline.svg
    └─ 718 /Content/IconPack/search.svg
  ► 716 /Security/User/Photo/8697/2cbe4537/bf94d3eb2a02cbdc3b6fb87
  ► 719 /SDK.Action/Images/Object/af117256-0703-4fac-b126-ff823e808d5f
  ► 720 /SDK.Action/Images/Object/a1ea16e1-4ba4-4a82-8e87-84cf57df72d
  ► 721 /SDK.Action/Images/Object/4b6678a3-4f9a-4d7c-88f3-316ec58a1e4e
  ► 724 /Content/IconPack/messages_all_written.svg
  ► 722 /SDK.Action/Images/Object/24894223-0db6-405c-b40c-b29d229ad4a3
  ► 729 /Content/IconPack/user_select_button.svg
  ► 728 /Content/IconPack/delete.svg
  ► 732 /Content/IconPack/cursor.svg
  ► 735 /Common/LastObjectInfo/AddInfo
  ► 736 /Security/User>SelectWithGroups?checkReplacement=True&showBlock=False&selectPopupid=ComplexExecutor_Workers_PrefixedVa
  ► 739 /Content/IconPack/arrow_up.svg
  ► 700 /Docflow/Acquaintance/Create
  ► 700 /Documents/Document/View/910

```

Рис. 111. Результаты записи создания задачи ознакомления через HTTP(s) Test Script Recorder

Среди записанных результатов в Recording Controller нас интересуют запросы, имеющие строку "Docflow/Acquaintance/Create" в своем URL, поскольку в данной группе запросов мы будем создавать и отправлять задачу ознакомления документа. Зададим для запросов соответствующие названия: **GetSendToAcqForm** – для GET-запроса получения формы отправки на ознакомление, **PostSendToAcqForm** – для POST-запроса отправки формы отправки на ознакомление.

Удаляем все лишние элементы тестового плана и переходим к записи его заключительной части.

4. Ознакомление с документом

Финальным этапом нашего плана (не считая логаута) будет выполнение созданной ранее задачи ознакомления с документом. Добавим в Jmeter элемент Recording Controller и переименуем его в ExecuteAcqTask. Далее перейдем в HTTP(s) Test Script Recorder, в поле **Target Controller** из списка выбираем добавленный нами контроллер. Нажимаем на кнопку **Start** и начинаем выполнять следующие действия в браузере:

- переходим в созданную нами ранее задачу ознакомления;

- нажимаем на кнопку верхнего меню **Ознакомлен**;
- в появившемся окне вводим комментарий и нажимаем на кнопку **Сохранить**.

После выполнения действий возвращаемся в элемент HTTP(s) Test Script Recorder и нажимаем на кнопку **Stop** (Рис. 112).



Рис. 112. Результаты записи выполнения задачи ознакомления через HTTP(s) Test Script Recorder

Среди записанных результатов нас интересуют запросы, имеющие строку "Docflow/Acquaintance/Execute" в своем URL, поскольку данная группа запросов будет отвечать за выполнение задачи ознакомление с документом. Зададим для запросов соответствующие названия: **GetAcqTask** – для GET-запроса получения формы задачи ознакомления, **PostAcqTask** – для POST-запроса отправки формы задачи ознакомления.

Удаляем все лишние элементы тестового плана и переходим к выполнению операции выхода из системы.

5. Логаут

После того как основная часть нашего тестового плана завершена, осуществим выход из системы текущим пользователем, чтобы в ней не накапливались лишние сессии. Для лучшей наглядности эту операцию также желательно поместить в отдельный блок, поэтому добавим элемент Recording Controller и переименуем его в Logout. Далее переходим в HTTP(s) Test Script Recorder, в поле **Target Controller** из списка выбираем добавленный нами контроллер. Нажимаем на кнопку **Start** и выполняем следующее в браузере: раскрываем меню пользователя и нажимаем на кнопку **Выйти** (Рис. 113).



Рис. 113. Результаты записи выполнения логаута через HTTP(s) Test Script Recorder

В данном случае нас интересует запрос, имеющий строку "Security/Account/LogOff" в своем URL.

Итак, после записи всех этапов плана наш тестовый сценарий будет выглядеть следующим образом (Рис. 114).

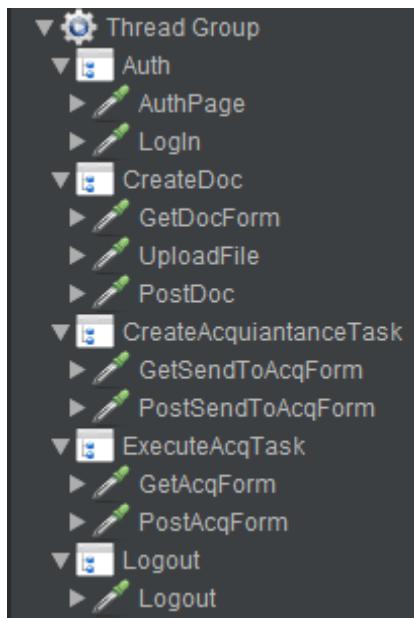


Рис. 114. Тестовый сценарий

- **Auth** – авторизация:
 - **AuthPage** – получение страницы авторизации;
 - **Login** – авторизация пользователя;
- **CreateDoc** – создание документа:
 - **GetDocForm** – получение формы создания документа;
 - **PostDoc** – отправка формы создания документа;
- **CreateAcquaintanceTask** – отправка документа на ознакомление:
 - **GetSendToAcqForm** – получение формы задачи отправки на ознакомление;
 - **PostSendToAcqForm** – отправка формы задачи отправки на ознакомление;
- **ExecuteAcqTask** – ознакомление с документом:
 - **GetAcqForm** – получение формы задачи ознакомления;
 - **PostAcqForm** – отправка формы задачи ознакомления;
- **Logout** – выход из системы:
 - **Logout** – логаут пользователя.

Следующим этапом написания нагрузочного сценария будет его настройка и отладка.

4.3. Запись с помощью Fiddler



Fiddler – прокси-сервер, который работает с HTTP(s) трафиком между клиентом и удаленным сервером и позволяет просматривать и менять его. Благодаря своей простоте, понятности и многофункциональности данный инструмент является хорошей альтернативой HTTP(S) Test Script Recorder, а в некоторых случаях, например, в возможности получить более расширенную информацию как по отправленным запросам, так и по ответам на них от сервера, а также в возможности фильтровать отдельные категории запросов из уже записанного пула запросов, его превосходит.

4.3.1. Установка

Для установки Fiddler выполните следующие действия:

1. Перейдите на официальный сайт <https://www.telerik.com/download/fiddler>.
2. Заполните следующую форму (Рис. 115).

1 Performance Testing

2 myemail@ya.ru

3 Country

4 I accept the Fiddler End User License Agreement

Download for Windows

Рис. 115. Форма для заполнения перед скачиванием Fiddler

- **1** – необходимо выбрать из списка, для каких целей будет использоваться Fiddler, например, для тестирования производительности (Performance Testing);
- **2** – необходимо указать e-mail;
- **3** – необходимо выбрать из списка страну проживания;
- **4** – необходимо согласиться с условиями пользовательского лицензионного соглашения на использование Fiddler, установив флажок.

3. Нажать на кнопку **Download for Windows**.

Также существует возможность скачать Fiddler для Linux или MacOS, нажав на соответствующую ссылку на странице (Рис. 116).

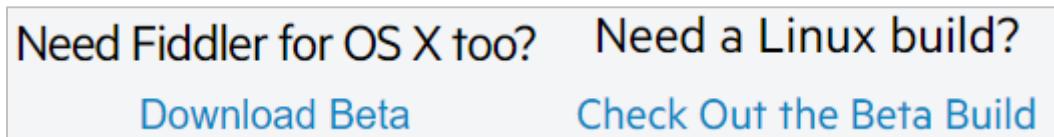


Рис. 116. Скачивание Fiddler для Linux или MacOS

4. После завершения загрузки запустите файл **.exe** для установки Fiddler. Следуйте указаниям установочной программы.
5. После завершения установки нажмите **Close**.

4.3.2. Запуск

У Fiddler есть 2 способа подключения:

1. Просто запустить его. Он автоматически будет работать для программ, использующих WinINET. Например, для таких как Google Chrome, Internet Explorer, приложения MS Office.
2. Явно поставить Fiddler как прокси для браузера по адресу 127.0.0.1:8888 (адрес по умолчанию). Например, для Mozilla Firefox сделать это можно [так](#).

ВАЖНО! При использовании в браузере данных настроек после закрытия Fiddler их необходимо будет отключить, иначе вместо загрузки страницы браузер будет показывать следующую ошибку (Рис. 117).

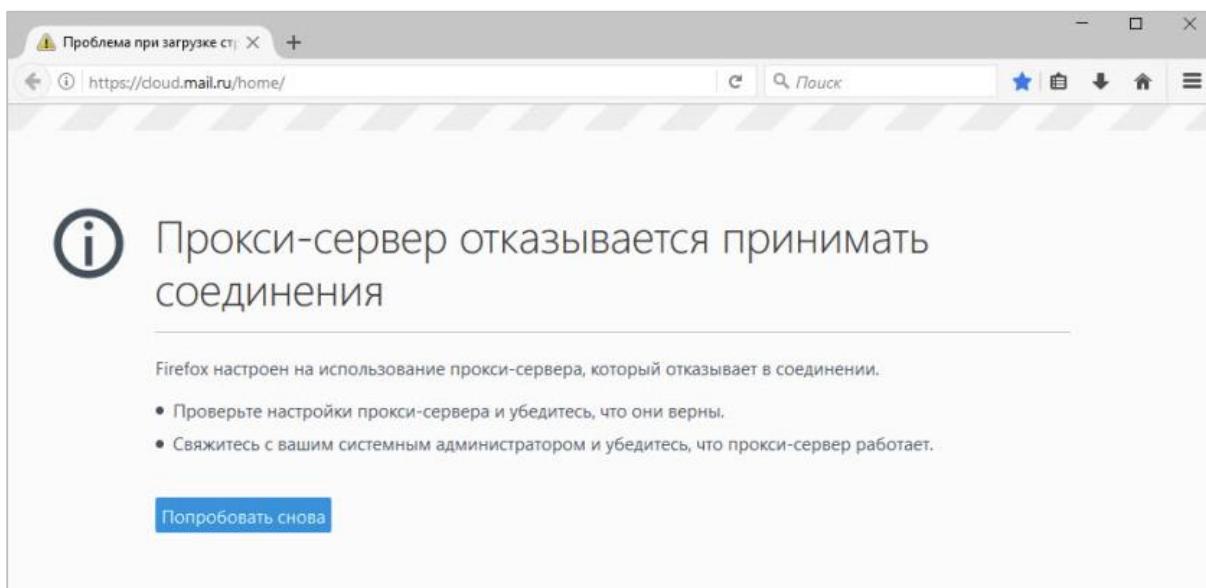


Рис. 117. Ошибка при загрузке страницы в браузере

Примечание: по умолчанию Fiddler использует порт 8888, при этом если данный порт занят другим приложением, то при запуске Fiddler появится окно, в котором будет предложено использовать вместо него другой случайный порт (Рис. 118).

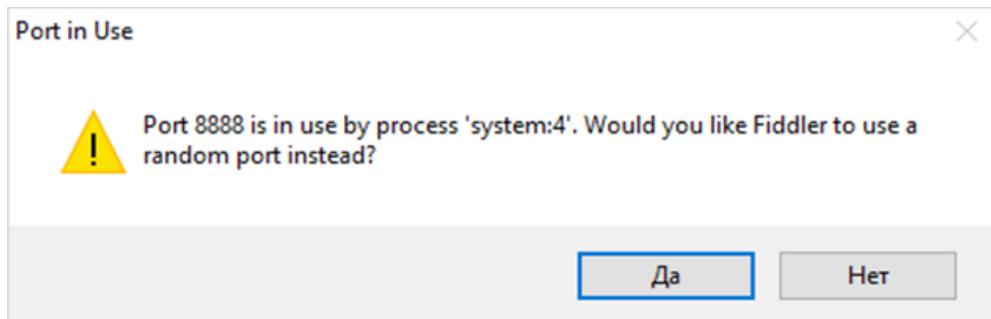


Рис. 118. Окно с предложением использовать случайный порт

Чтобы посмотреть, какой порт использует Fiddler, необходимо открыть меню **Tools** из главного меню программы и выбрать пункт **Options**, в появившемся окне перейти на вкладку **Connections** (Рис. 119).

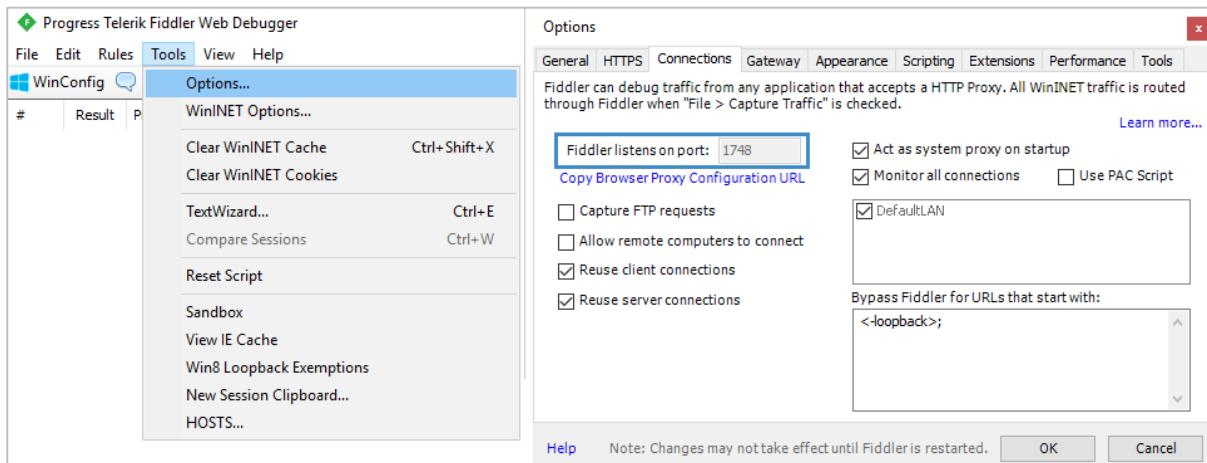


Рис. 119. Просмотр порта, который использует Fiddler

После запуска Fiddler становится промежуточным звеном между клиентом и целевым сервером, соответственно все запросы от вашего браузера к сайту будут на время сеанса Fiddler проходить через него. Вы будете видеть абсолютно все HTTP-запросы, например, к изображениям, CSS, JS и прочим веб-ресурсам. Для каждого запроса и ответа выводятся параметры, диагностируя которые можно судить об оптимальности или неоптимальности поведения веб-сайта.

Главное окно приложения выглядит следующим образом (Рис. 120).

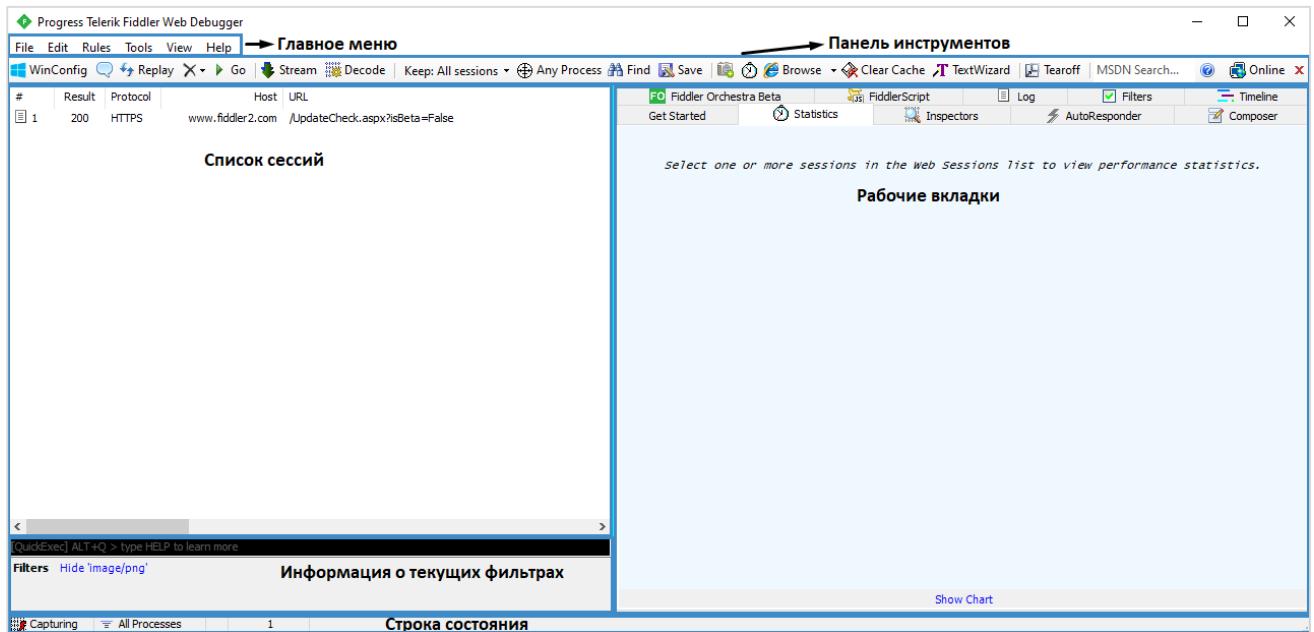


Рис. 120. Главное окно приложения

Главное меню программы предназначено для доступа к настройкам соединения, созданию правил, с помощью которых можно модифицировать

запросы или ответы сервера, а также к информации о самой программе и т.д. В панели инструментов находятся кнопки и индикаторы, помогающие оперировать ходом записи сессий и последующей работой с ними. В **строке состояния**, расположенной снизу, находится информация об общем числе записанных запросов, а также надпись о том, включена ли в текущий момент сама запись (Capturing). Слева находится **список сессий**, в котором можно просматривать отправленные на сервер запросы в виде пиктограммы типа данных, статуса, адреса, размера и другой информации. Справа от списка сессий располагаются следующие **рабочие вкладки** с отображением всех конфигурируемых настроек и параметров:

- **Statistics** – при выборе одного или нескольких запросов из списка сессий, на данной вкладке отображается информация об их выполнении: общее количество запросов, размер, фактическое время его выполнения с датами начала и завершения, код ответа, размер ответа на них в байтах и т.д;
- **Inspectors** – при выборе определенного запроса из списка сессий, на данной вкладке отображается развернутая информация как о запросе (строка запроса, его заголовки и содержимое), так и о полученном на него ответе от сервера.

Расположенные в Inspectors вложенные вкладки позволяют увидеть данные по запросу и его ответу в различных текстовых форматах, например, TextView, SyntaxView, Raw, Json, XML (Рис. 121).



Рис. 121. Вложенные вкладки в *Inspectors*

На данных вкладках также есть поле для полнотекстового поиска, с помощью которого можно отыскать необходимый элемент в выводимых данных.

- **AutoResponder** – позволяет подставлять файлы с локального диска вместо передачи запроса на сервер;
- **Composer** – позволяет составить запрос на сервер вручную, а также настроить;
- **Fiddler Orchestra Beta** – инструмент для удаленной отладки веб-приложений. С его помощью можно отслеживать, что отправляется и принимается на каждом этапе прохождения веб-сценария любой сложности;

- **FiddlerScript** – основной скрипт, задающий функционал для Fiddler. При его редактировании можно добавить или удалить пункты меню, колонки в списке запросов и многое другое. Язык программирования JS/Script.NET, который здесь используется, может взаимодействовать с Windows в полном объеме, включая коммуникацию с базой данных, Word, Excel;
- **Log** – лог Fiddler, в который заносится информация об определенных событиях, в частности об ошибках, за время его использования;
- **Filters** – на данной вкладке есть возможность настроить фильтрацию запросов, например, выбрать определенный сервер, с которого будет собираться трафик, а также отсеивать запросы, содержащие в своем URL различные элементы, например .png, .gif и т.д., либо наоборот выводить только те запросы, в URL которых они присутствуют;
- **Timeline** – позволяет визуализировать записываемый приложением HTTP(s) трафик в виде диаграммы.

4.3.3. Настройка HTTPS

По умолчанию Fiddler не имеет доступа к содержимому HTTPS-запросов, поскольку является прокси. Чтобы его получить, необходимо установить специальный сертификат, с помощью которого Fiddler сможет расшифровать запросы и затем отправить их дальше. Для этого выполните следующее:

1. В главном меню программы нажмите на кнопку **Tools** и выберите пункт **Options**.
2. На вкладке **HTTPS** установите параметр **Capture HTTPS CONNECTs**.
3. Установите флажок **Decrypt HTTPS traffic**, после чего появится всплывающее окно, которое предложит установить доверие корневому сертификату Fiddler. Нажмите на кнопку **Yes** (Рис. 122).

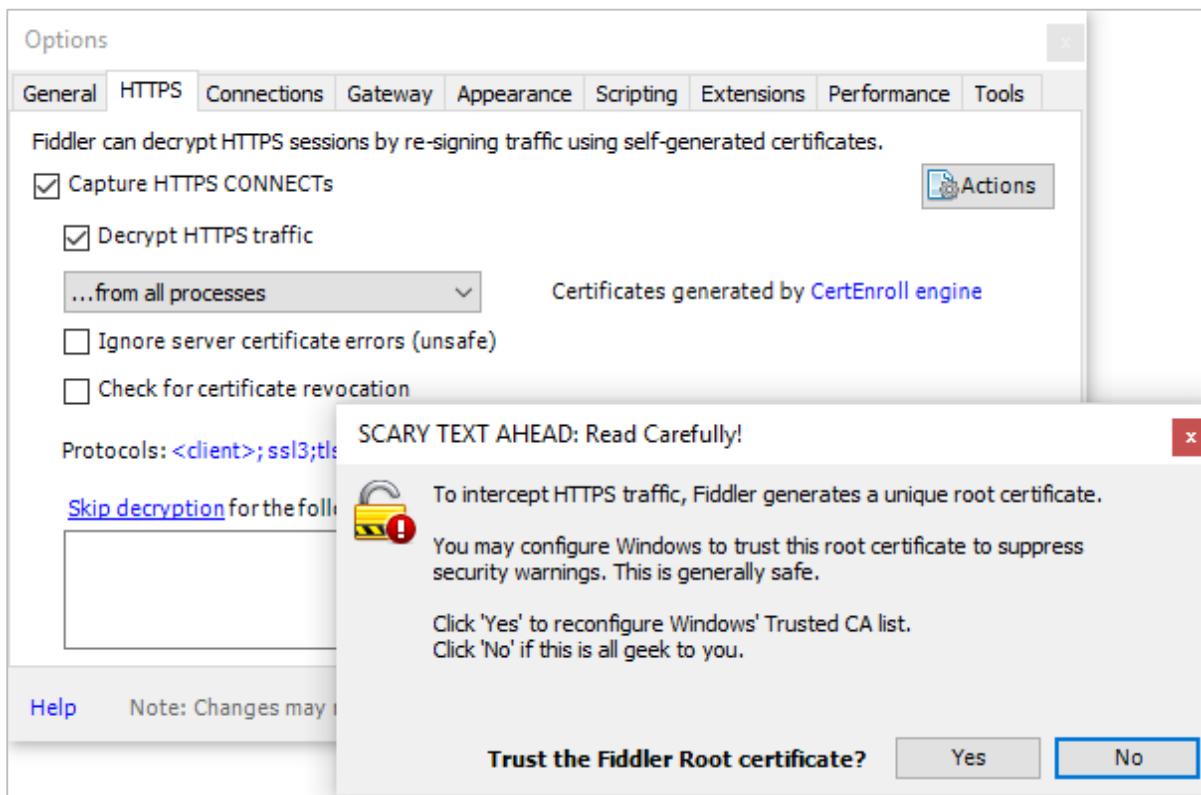


Рис. 122. Установка сертификата

4. Далее появится диалоговое окно с разрешением на установку сертификата. Нажмите на кнопку **Да** (Рис. 123).

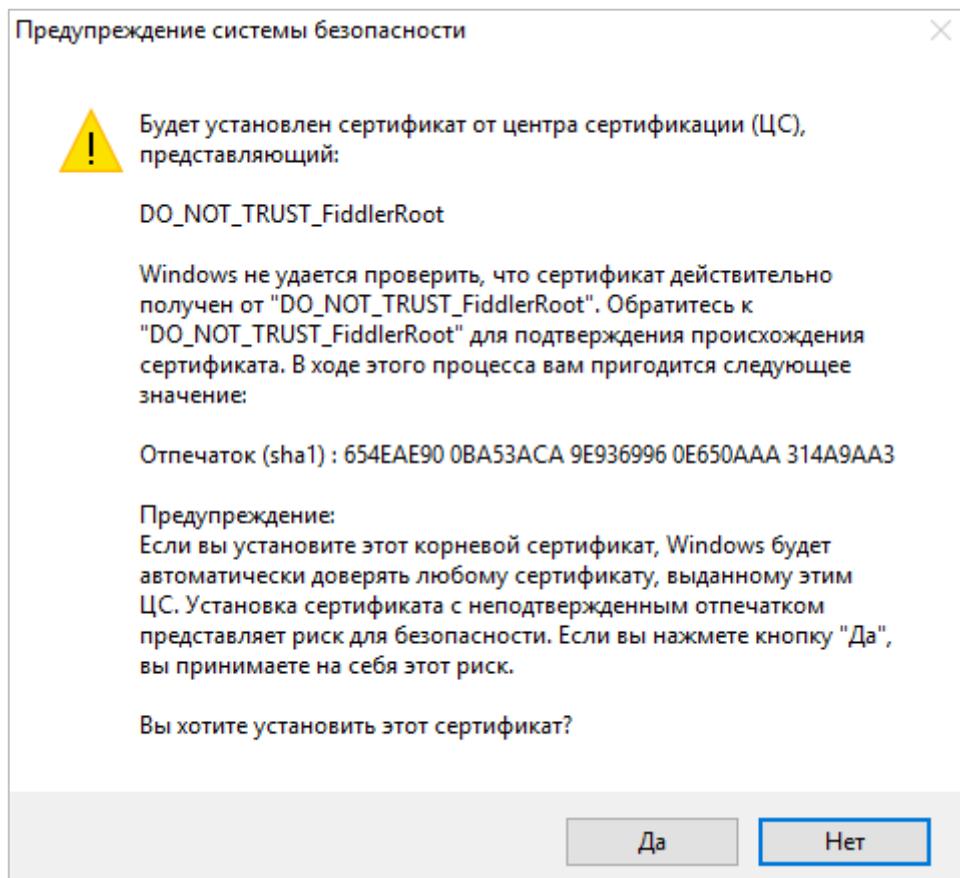


Рис. 123. Предупреждение системы безопасности

5. После установки должно появится окно, информирующее об успешном добавлении сертификата в список доверенных корневых центров сертификации для локального компьютера (Рис. 124).

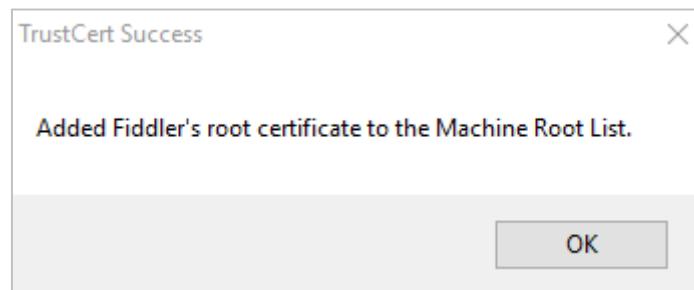


Рис. 124. Окно, информирующее об успешном добавлении сертификата

4.3.4. Запись сценария в Fiddler

Каждую операцию плана, изложенного в главе [4.1](#), в JMeter желательно записывать в отдельный контроллер, тем самым задавая логический порядок всем частям сценария. В данном случае мы будем использовать элемент Simple Controller, куда мы передадим записанные с помощью Fiddler и сконвертированные в .jmx (для возможности их чтения

самим JMeter) запросы. В Jmeter в Thread Group добавим элемент Simple Controller и переименуем его в Auth. После этого переходим в Fiddler.

Для начала настроим фильтрацию запросов, чтобы в список сессий не попадали ненужные нам данные. Для этого перейдем в рабочую вкладку **Filters** и установим флажок **Use filters** (Рис. 125).

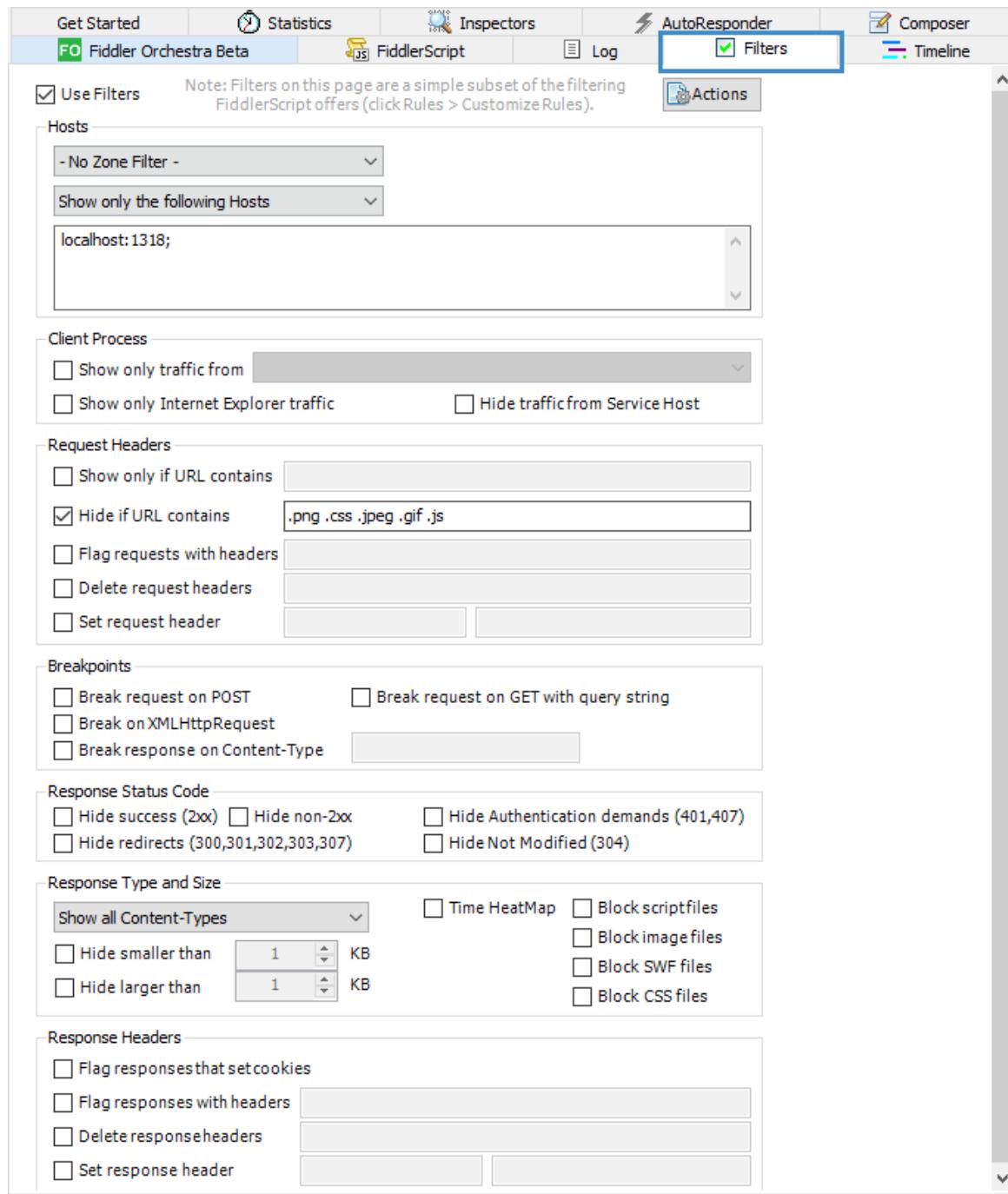


Рис. 125. Настройка фильтрации запросов. Вкладка "Filters"

В выпадающем списке выбираем значение **Show only the following Hosts** и в поле пишем адрес веб-сервера ELMA, в нашем случае – localhost:1318.

Далее переходим в панель **Request Headers** и устанавливаем флагок на **Hide if URL contains**. В поле вводим расширения, разделяя их пробелом: **.png .js .gif .css .ico**. Если URL запроса будет содержать указанные расширения, то он будет отсеян и в общий пул запросов, отображаемый в списке сессий, не попадет.

Сделав необходимые настройки, переходим к включению данного фильтра. Для этого нажимаем на кнопку **Actions** и выбираем пункт выпадающего меню **Run Filterset now** (Рис. 126).



Рис. 126. Включение фильтра. Кнопка "Actions"

Таким образом, теперь в список сессий будет попадать трафик только с сайта **localhost:1318**, отсеивая при этом запросы, содержащие в своем URL расширения **.png, .js, .gif, .css, .ico**.

Перед тем как начать выполнять действия в браузере, необходимо убедиться, что включена запись трафика в самом Fiddler. В том случае, если запись трафик ведется, в строке состояния приложения будет отображаться надпись "Capturing" (Рис. 127).



Рис. 127. Страна состояния приложения. Надпись "Capturing"

Если надпись не отображается, то включить запись трафика можно перейдя в меню **File** из главного меню программы и выбрав пункт **Capture Traffic** или нажав на клавишу **F12** (Рис. 128).

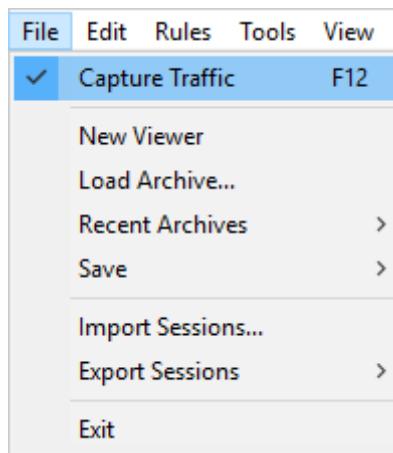


Рис. 128. Включение записи трафика

Для прекращения записи трафика необходимо снять флажок с **Capture Traffic** или нажать на клавишу **F12**.

Теперь все готово к записи сценария. Начинаем выполнять действия в браузере, следуя пунктам плана:

1. Авторизация пользователя:

- в адресную строку вводим адрес тестируемого сервера;
- вводим логин и пароль. Нажимаем **Войти в систему**;
- удостоверяемся, что пользователь авторизован.

После проделанных действий возвращаемся в Fiddler и нажимаем на клавишу **F12** для прекращения записи. Тем временем в списке сессий у нас появились следующие данные (Рис. 129).

#	Result	Protocol	Host	URL	Body	Caching	Content-Type	Process
5	302	HTTP	localhost:1318	/	154	private	text/html; c...	chrome...
6	200	HTTP	localhost:1318	/Security/Account/LogOn?ReturnUrl=%2f	9 419	private	text/html; c...	chrome...
25	302	HTTP	localhost:1318	/Security/Account/LogOn?ReturnUrl=%2f	118	private	text/html; c...	chrome...
26	200	HTTP	localhost:1318	/	91 868	public, ...	text/html; c...	chrome...
59	200	HTTP	localhost:1318	/SDK.Action/Images/Object/555a587d-25c1-4f99-ad6-3176b4fc9a...	484	private...	image/png	chrome...
60	200	HTTP	localhost:1318	/SDK.Action/Images/EnumObject/5ab21887-fc5f-4d16-8bbf-6818499...	237	private...	image/png	chrome...
61	200	HTTP	localhost:1318	/SDK.Action/Images/Object/c30a3592-30c9-4802-88a3-858663a64c...	595	private...	image/png	chrome...
63	200	HTTP	localhost:1318	/SDK.Action/Images/Object/0e40f1c2-94ae-4885-b9aa-7fc460b7e59...	3 046	private...	image/png	chrome...
96	200	HTTP	localhost:1318	/SDK.Action/Personalization/PortletContent?portletId=1c695b3d-b4...	4 814	public, ...	text/html; c...	chrome...
97	200	HTTP	localhost:1318	/SDK.Action/Personalization/PortletContent?portletId=fac6d43c-1f95...	427 968	public, ...	text/html; c...	chrome...
98	200	HTTP	localhost:1318	/SDK.Action/Personalization/PortletContent?portletId=42322f50-995...	2 585	public, ...	text/html; c...	chrome...
99	200	HTTP	localhost:1318	/SDK.Action/Personalization/PortletContent?portletId=240d0b1b-488...	47 388	public, ...	text/html; c...	chrome...
107	200	HTTP	localhost:1318	/Common/AutoComplete/GetData?objectUid=%7B466D9202-CD97-4...	145 207	private...	application/...	chrome...
118	200	HTTP	localhost:1318	/SDK.Action/Images/Object/856324ea-7106-40a9-9f69-c0e08593d9...	3 100	private...	image/png	chrome...
120	200	HTTP	localhost:1318	/SDK.Action/Images/Object/e1155ae5-c6de-4b7d-8160-5337ed8b56...	3 097	private...	image/png	chrome...
121	200	HTTP	localhost:1318	/SDK.Action/Images/Object/dedec971-594a-4028-9781-f218b4658b...	473	private...	image/png	chrome...

Рис. 129. Список сессий. Запросы с типом содержимого "image/png"

Как мы видим, несмотря на включенную нами фильтрацию, в пул попали ненужные для нашего теста запросы с типом содержимого **image/png**. Данные запросы не были отфильтрованы, поскольку тип

содержимого передается в заголовке "Content-Type", а не в URL запроса. Для того чтобы в дальнейшем включить данный тип запроса в фильтр, необходимо вызвать контекстное меню любого из этих элементов и выбрать пункт **Filter Now → Hide 'image/png'** (Рис. 130).

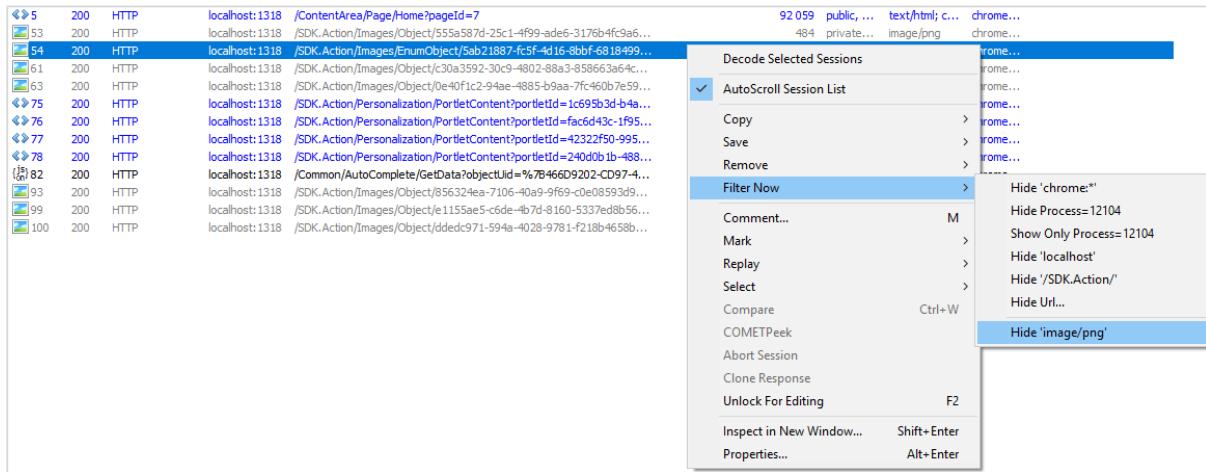


Рис. 130. Включение типа запроса в фильтр

После проделанных действий из списка сессий пропадут все похожие запросы и снизу появится поле **Filters**, в котором будет указан добавленный нами фильтр для отсеивания последующих запросов такого типа (Рис. 131).

#	Result	Protocol	Host	URL	Body	Caching	Content-Type	Process	Comments	Custom
1	200	HTTP	localhost:1318	/Security/Account/LogOn	9 405	private	text/html; c...	chrome...		
20	302	HTTP	localhost:1318	/Security/Account/LogOn	118	private	text/html; c...	chrome...		
21	200	HTTP	localhost:1318	/	91864	public, ...	text/html; c...	chrome...		
80	200	HTTP	localhost:1318	/SDK.Action/Personalization/PortletContent?portletId=1c695b3d-b4a...	4 814	public, ...	text/html; c...	chrome...		
81	200	HTTP	localhost:1318	/SDK.Action/Personalization/PortletContent?portletId=facd43c-1f95...	427968	public, ...	text/html; c...	chrome...		
82	200	HTTP	localhost:1318	/SDK.Action/Personalization/PortletContent?portletId=42322f50-995...	2 585	public, ...	text/html; c...	chrome...		
83	200	HTTP	localhost:1318	/SDK.Action/Personalization/PortletContent?portletId=240d0b1b-488...	47388	public, ...	text/html; c...	chrome...		
102	200	HTTP	localhost:1318	/Common/AutoComplete/GetData?objectUid=%7B466D9202-CD97-4...	145 207	private...	application/...	chrome...		

Рис. 131. Список сессий. Поле "Filters"

Все действия, совершаемые пользователем для входа в систему, происходят на странице авторизации, соответственно, нас будут интересовать запросы, имеющие строчку "Security/Account/LogOn" в качестве URL. Среди полученных результатов имеется 2 подходящих запроса, переданные методами GET и POST соответственно. Чтобы окончательно убедиться в этом, в списке сессий по очереди выделим каждый из этих запросов и перейдем в подкладку **Headers** рабочей вкладки **Inspectors**. Как видно на Рис. 132, верхний запрос выполняется методом GET и представляет собой получение страницы авторизации.

The screenshot shows the Fiddler application interface. In the main pane, there is a list of network requests. One request is highlighted with a blue border, showing its details. The request is a GET to `/Security/Account/LogOn`. The Headers tab is selected, showing the following content:

```

Request Headers
GET /Security/Account/LogOn HTTP/1.1
Cache-Control: max-age=0
Client
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36
Cookies
Cookie
__RequestVerificationToken=ElKfbE6e3Y8dTAbk1GaOyX9b6uSS4uFhiAwGYYHgld03g5nGGZ903V32pGVOPtC-9VAaqp9GIWtaG2TCI9NC1O5bnHKq7
ASP.NET_SessionId=yfisuvnsyzuvv1ejz154wvo
ElmaCulture=ru-RU
ElmaLastUserName=admin
saveAsDefaultComment=true
Security
Upgrade-Insecure-Requests: 1
Transport
Connection: keep-alive
Host: localhost:1318
  
```

Рис. 132. Запрос, выполняющийся методом GET. Вкладка "Inspectors". Подвкладка "Headers"

Следующий после него запрос выполняется уже методом POST (Рис. 133) и отправляется при нажатии кнопки на странице авторизации, передавая серверу в теле данные, которые можно увидеть в подвкладке **TextView** (Рис. 134).

The screenshot shows the Fiddler application interface. A POST request for account logon is highlighted. The Headers tab is selected, showing the following content:

```

Request Headers
POST /Security/Account/LogOn HTTP/1.1
Cache-Control: max-age=0
Client
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36
Cookies
Cookie
__RequestVerificationToken=ElKfbE6e3Y8dTAbk1GaOyX9b6uSS4uFhiAwGYYHgld03g5nGGZ903V32pGVOPtC-9VAaqp9GIWtaG2TCI9NC1O5bnHKq7
ASP.NET_SessionId=yfisuvnsyzuvv1ejz154wvo
ElmaCulture=ru-RU
ElmaLastUserName=admin
saveAsDefaultComment=true
Entity
Content-Length: 325
Content-Type: application/x-www-form-urlencoded
Miscellaneous
Referer: http://localhost:1318/Security/Account/LogOn
Security
  
```

Рис. 133. Запрос, выполняющийся методом POST. Вкладка "Inspectors". Подвкладка "Headers"

The screenshot shows the Fiddler application interface with the **TextView** tab selected. The raw data sent in the POST request is displayed:

```

RequestVerificationToken=008MR-
o4ODLbxu3DVpW5G1L1q04Y9qj7CL2zQ-0Lkd183mvyqw_YvA3NlYvrAxWndjAgmlbsXtA5TkOllq_pEAH63G0Ju09qvWld1ERcZTbZFcYMG6uXlhZa4WI2lIV6GLYpd322WJoGOITGRZ0DINnSmjyWnohE01
&login=admin&password=&rememberMe=false&Login=%D0%92%D0%BE%D0%B9%D1%82%D0%B8+&D0%B2+&D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%83
  
```

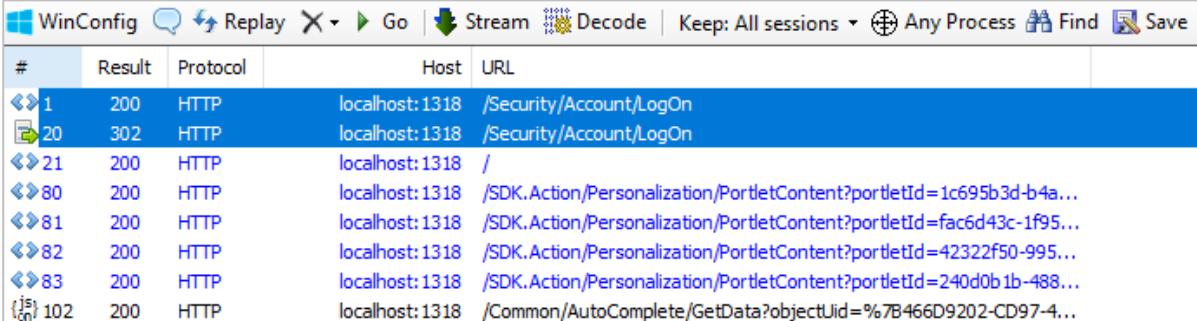
Рис. 134. Данные, передаваемые серверу. Вкладка "Inspectors". Подвкладка "TextView"

После нахождения интересующей нас группы запросов их необходимо передать в наш тестовый план в Jmeter. Для этого необходимо воспользоваться экспортом выбранных сессий в файл с расширением HAR (HTTP Archive – файл архива в формате JSON, использующийся для записи

событий взаимодействия веб-браузера с сайтом), который в дальнейшем требуется сконвертировать в JMX файл, чтобы открыть его в JMeter.

Для этого выполняем следующие действия:

- в списке сессий выделяем необходимые для экспорта запросы (Рис. 135);



#	Result	Protocol	Host	URL
1	200	HTTP	localhost:1318	/Security/Account/LogOn
20	302	HTTP	localhost:1318	/Security/Account/LogOn
21	200	HTTP	localhost:1318	/
80	200	HTTP	localhost:1318	/SDK.Action/Personalization/PortletContent?portletId=1c695b3d-b4a...
81	200	HTTP	localhost:1318	/SDK.Action/Personalization/PortletContent?portletId=fac6d43c-1f95...
82	200	HTTP	localhost:1318	/SDK.Action/Personalization/PortletContent?portletId=42322f50-995...
83	200	HTTP	localhost:1318	/SDK.Action/Personalization/PortletContent?portletId=240d0b1b-488...
102	200	HTTP	localhost:1318	/Common/AutoComplete/GetData?objectUid=%7B466D9202-CD97-4...

Рис. 135. Выделение в списке сессий запросов, необходимых для экспорта

- в главном меню программы открываем меню **File** и нажимаем **Export Sessions → Selected Sessions** (Рис. 136);

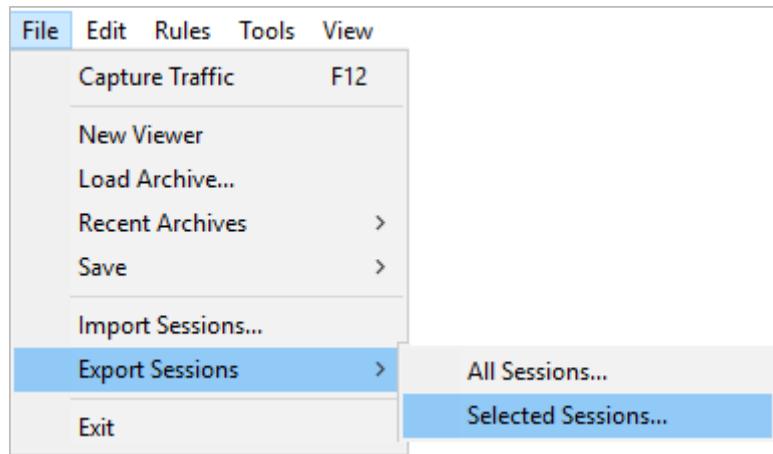


Рис. 136. Экспорт выбранных сессий Меню "File"

- в появившемся окне из списка выбираем **HTTPArchive v.1.2** и нажимаем на кнопку **Next** (Рис. 137);

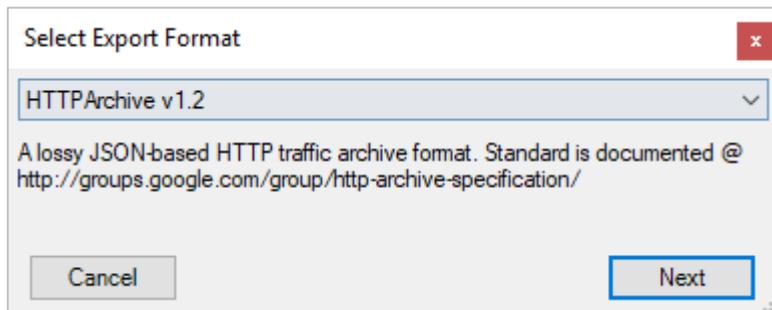


Рис. 137. Выбор формата "HTTPArchive v.1.2"

- далее необходимо выбрать папку, куда сохранится файл с расширением HAR и задать имя файла (Рис. 138);

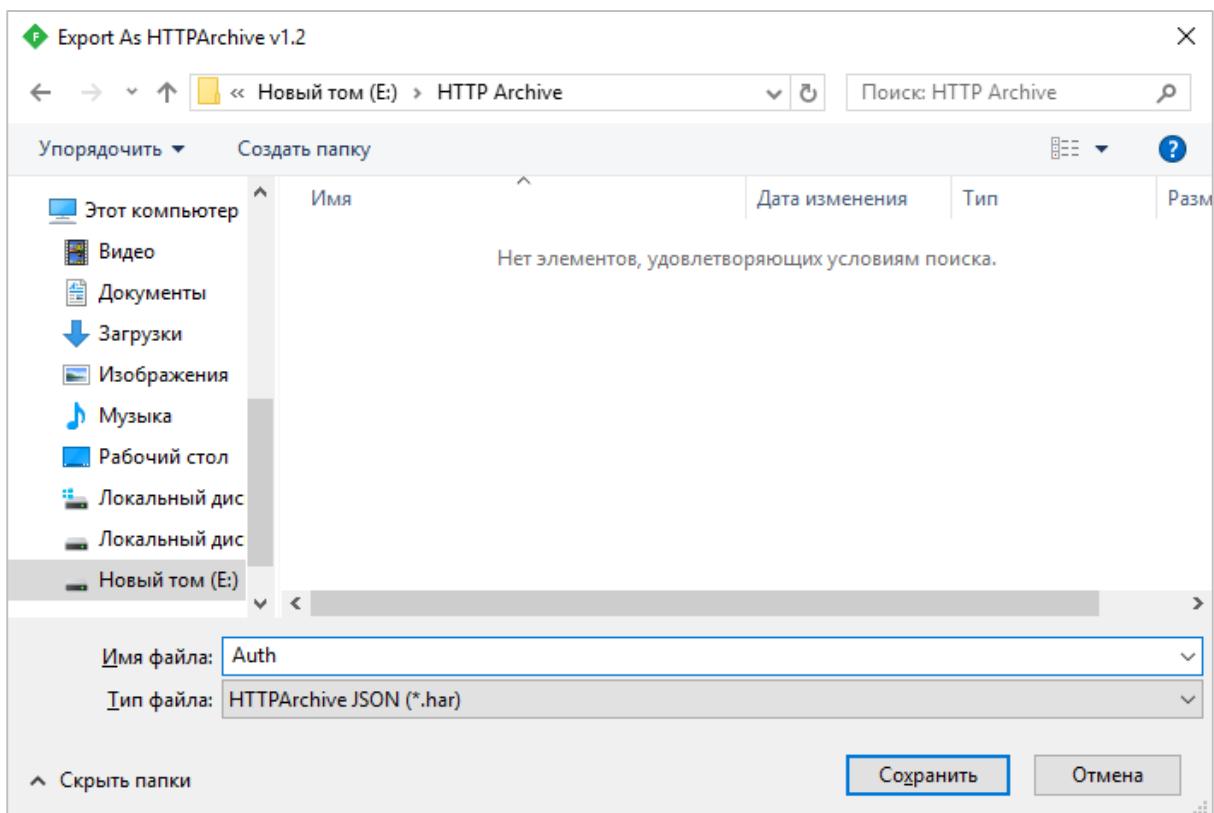


Рис. 138. Сохранение файла с расширением HAR

- переходим на [сайт для конвертации файлов в jmx формат](#);
- на форме конвертера нажимаем на кнопку **Choose File** и выбираем сохраненный нами ранее файл с расширением HAR, далее нажимаем на кнопку **Convert** (Рис. 139);

Рис. 139. Форма конвертера

- после завершения конвертации нажимаем на кнопку **Download JMX File** (Рис. 140);

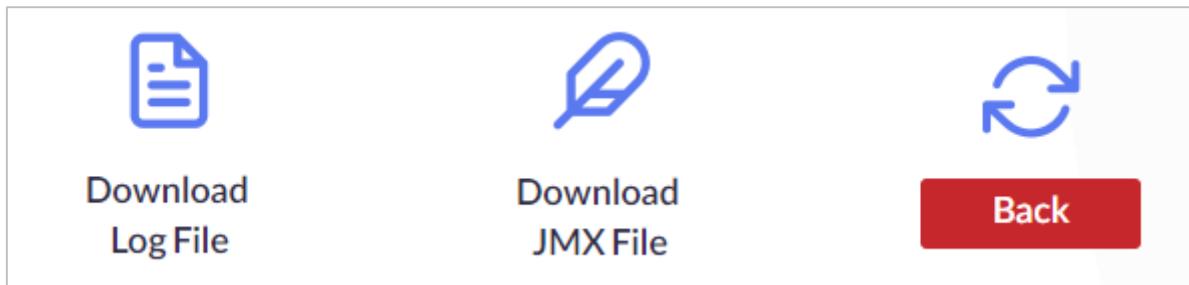


Рис. 140. Скачивание сформированного JMX файла

- запускаем Jmeter;

ВАЖНО! Если JMeter с готовым тестовым планом уже запущен, то необходимо открыть еще одно окно приложения, чтобы текущий тестовый план не перезаписался после импорта JMX.

- в главном меню Jmeter открыть меню **File** и нажать **Open**;
- в появившемся окне выбрать загруженный JMX файл и нажать на кнопку **Open** или дважды кликнуть на него (Рис. 141).

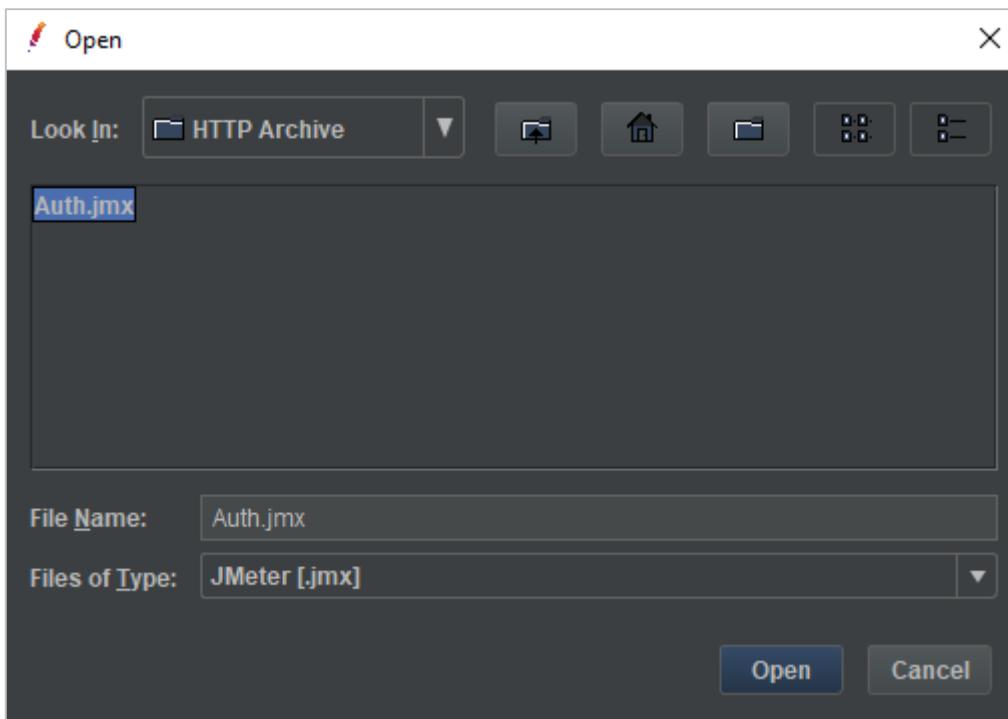


Рис. 141. Открытие JMX файла

- после открытия файла требуется скопировать записанные сэмплеры в уже имеющийся тестовый план (Рис. 142).

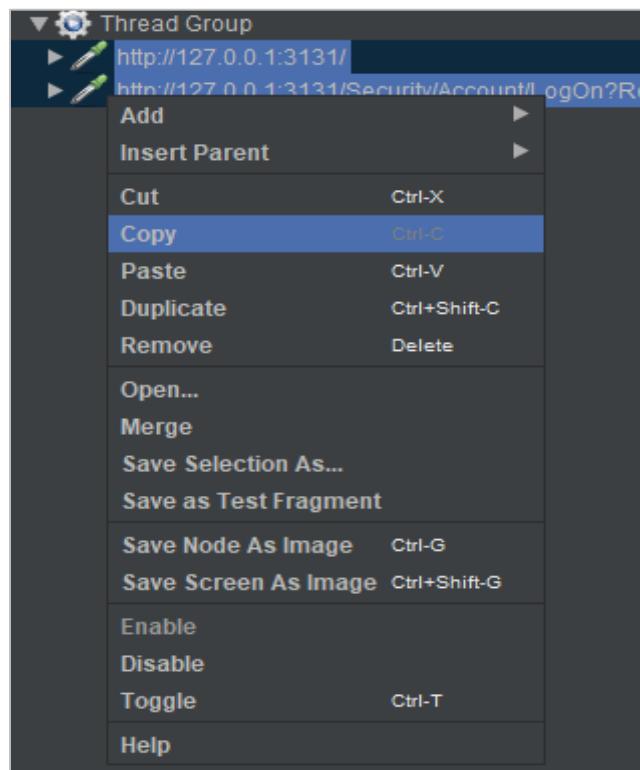


Рис. 142. Копирование записанных сэмплеры в тестовый план

В нашем случае мы копируем их в специально созданный заранее Simple Controller с именем "Auth". После копирования для лучшего восприятия нашего тестового плана переименуем GET-запрос получения страницы авторизации в Auth, а отправки данных – в LogIn (Рис. 143).

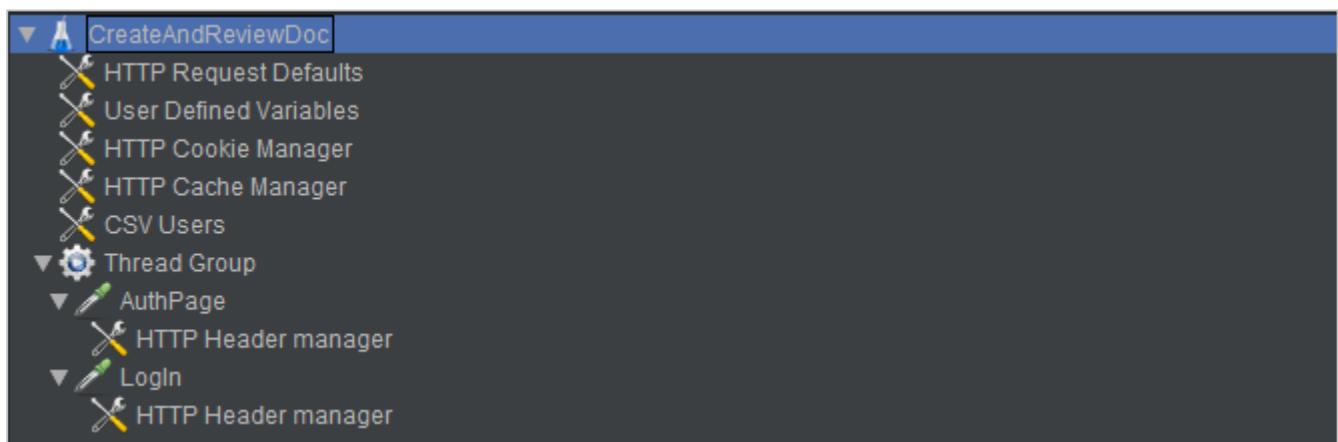


Рис. 143. Тестовый план

2. Создание документа

Авторизовавшись, мы переходим к созданию документа. Добавим в Jmeter еще один элемент Simple Controller, в котором в дальнейшем разместим необходимые для совершения данной операции запросы, и

переименуем его в CreateDoc. В Fiddler нажимаем на клавишу **F12** до появления в строке состояния надписи "Capturing" и выполняем следующие действия в браузере:

- на главной странице системы нажимаем на кнопку **Создать документ**;
- из списка документов выбираем необходимый тип документа и нажимаем на кнопку **Создать**;
- на форме документа заполняем поля **Название** и **Предмет договора**, а также загружаем файл-версию документа;
- нажимаем на кнопку **Сохранить**;

После проделанных действий возвращаемся в Fiddler и нажимаем на клавишу **F12** для остановки записи.

В списке сессий нас интересует 3 запроса: 2 из них – GET и POST – имеют строчку "Documents/Document/Create" в качестве URL, поскольку наши действия происходили на форме создания документа, последний совершается методом POST и имеет в URL строку "SDK.Action/BinaryFiles/UploadAjax" и используется для загрузки файла в качестве версии документа. Поскольку в нашем тестовом сценарии версию документа мы заполняем, данный запрос нам понадобится (Рис. 144).

#	Result	Protocol	Host	URL
10	200	HTTP	localhost:1318	/Documents/Document/TypeSelector?folderId=¶meters=&_=15...
14	200	HTTP	localhost:1318	/Documents/Document/Create
50	200	HTTP	localhost:1318	/SDK.Action/BinaryFiles/UploadAjax?ca=true&cd=true&d=true&form...
64	302	HTTP	localhost:1318	/Documents/Document/Create
65	200	HTTP	localhost:1318	/Documents/Document/View/1213
121	200	HTTP	localhost:1318	/Documents/Document/AsyncViewItem/1213?viewMode=00000000-0...
122	200	HTTP	localhost:1318	/Documents/Document/AsyncViewItem/1213?viewMode=00000000-0...
123	200	HTTP	localhost:1318	/Documents/Document/AsyncViewItem/1213?viewMode=00000000-0...
124	200	HTTP	localhost:1318	/Documents/Document/AsyncViewItem/1213?viewMode=00000000-0...
126	200	HTTP	localhost:1318	/Docflow/Route/Info?documentId=1213
127	200	HTTP	localhost:1318	/Common/EntityHistory/History/1213?actionObject=5f17f751-2347-4...
128	200	HTTP	localhost:1318	/Common/Watch/Info?typeUid=5f17f751-2347-42d1-bb5c-a3b97117...
141	200	HTTP	localhost:1318	/Common/LastObjectInfo/AddInfo

Рис. 144. Запросы, имеющие строки "Documents/Document/Create" и "SDK.Action/BinaryFiles/UploadAjax" в URL

Переносим выбранные запросы в JMeter с помощью процедуры экспорта и задаем для них соответствующие названия: **GetDocForm** – для GET-запроса получения формы создания документа, **UploadFile** – для POST-запроса загрузки файла-версии документа, **PostDoc** – для POST-запроса отправки формы создания документа.

3. Отправка документа на ознакомление

Согласно плану, после создания документа нам необходимо отправить его на ознакомление. Снова добавляем в Jmeter элемент Simple Controller и переименовываем его в CreateAcquaintanceTask. Убеждаемся, что в строке состояния Fiddler надпись "Capturing" активна, и начинаем выполнять действия в браузере:

- в верхнем меню карточки созданного нами документа нажимаем кнопку **Отправить – Отправить на ознакомление**;
- на форме отправки на ознакомление заполняем необходимые поля: сроком ознакомления устанавливаем завтрашнее число, в поле **Кому** выбираем администратора ELMA (в зависимости от профиля нагрузки, для выполнения определенных действий в системе могут быть заданы разные пользовательские роли. В нашем случае все операции будут осуществляться одним пользователем). Остальные атрибуты оставим по умолчанию.
- нажимаем кнопку **Отправить**;

После проделанных действий возвращаемся в Fiddler и нажимаем на клавишу **F12** для остановки записи.

Среди записанных результатов нас интересуют запросы, имеющие строку "Docflow/Acquaintance/Create" в своем URL, поскольку в данной группе запросов мы будем создавать и отправлять задачу ознакомления документа (Рис. 145).

#	Result	Protocol	Host	URL
1	200	HTTP	localhost:1318	/Common/LastObjectInfo/AddInfo
2	200	HTTP	localhost:1318	/Docflow/Acquaintance/Create
60	200	HTTP	localhost:1318	/Common/LastObjectInfo/AddInfo
61	200	HTTP	localhost:1318	/Security/User>SelectWithGroups?checkReplacement=True&showBloc...
66	200	HTTP	localhost:1318	/Content/Fonts/OpenSans-SemiboldItalic-webfont.woff
69	200	HTTP	localhost:1318	/Security/PermissionManagement/NeedAddPermissionToEntity?permis...
70	302	HTTP	localhost:1318	/Docflow/Acquaintance/Create
71	200	HTTP	localhost:1318	/Documents/Document/View/1213
133	200	HTTP	localhost:1318	/Documents/Document/AsyncViewItem/1213?viewMode=00000000-0...
134	200	HTTP	localhost:1318	/Documents/Document/AsyncViewItem/1213?viewMode=00000000-0...
135	200	HTTP	localhost:1318	/Documents/Document/AsyncViewItem/1213?viewMode=00000000-0...
136	200	HTTP	localhost:1318	/Documents/Document/AsyncViewItem/1213?viewMode=00000000-0...
137	200	HTTP	localhost:1318	/Docflow/Route/Info?documentId=1213
138	200	HTTP	localhost:1318	/Common/EntityHistory/History/1213?actionObject=5f17f751-2347-4...
139	200	HTTP	localhost:1318	/Common/Watch/Info?typeUid=5f17f751-2347-42d1-bb5c-a3b97117...
147	200	HTTP	localhost:1318	/Common/LastObjectInfo/AddInfo

Рис. 145. Запросы, имеющие строку "Docflow/Acquaintance/Create" в URL

Импортируем выбранную группу запросов в JMeter и задаем для них соответствующие названия: **GetSendToAcqForm** – для GET-запроса

получения формы отправки на ознакомление, **PostSendToAcqForm** – для POST-запроса отправки формы отправки на ознакомление.

4. Ознакомление с документом

Финальным этапом нашего плана (не считая операции логаута) будет выполнение созданной ранее задачи ознакомления с документом. Добавим в Jmeter элемент Simple Controller и переименуем его в ExecuteAcqTask. Убеждаемся, что в строке состояния Fiddler надпись "Capturing" активна, и начинаем выполнять действия в браузере:

- переходим в созданную нами ранее задачу ознакомления;
- нажимаем кнопку верхнего меню **Ознакомлен**;
- в появившемся окне вводим комментарий и нажимаем кнопку **Сохранить**;

После проделанных действий возвращаемся в Fiddler и нажимаем на клавишу **F12** для остановки записи.

Среди записанных результатов нас интересуют запросы, имеющие строку "Docflow/Acquaintance/Execute" в своем URL, поскольку данная группа запросов будет отвечать за выполнение задачи ознакомление с документом (Рис. 146).

#	Result	Protocol	Host	URL
8	200	HTTP	localhost:1318	/Docflow/Acquaintance/Execute/5152
71	200	HTTP	localhost:1318	/Tasks/Task/HasActiveSubTasks?taskId=5152&_=1561727365976
72	200	HTTP	localhost:1318	/Common/LastObjectInfo/AddInfo
74	302	HTTP	localhost:1318	/Docflow/Acquaintance/Execute/5152
75	200	HTTP	localhost:1318	/Documents/Document/View/1213
76	200	HTTP	localhost:1318	/Tasks/Task/MarkRead/5152
121	200	HTTP	localhost:1318	/Docflow/Route/Info?documentId=1213
122	200	HTTP	localhost:1318	/Documents/Document/AsyncViewItem/1213?viewMode=00000000-0...
123	200	HTTP	localhost:1318	/Documents/Document/AsyncViewItem/1213?viewMode=00000000-0...
124	200	HTTP	localhost:1318	/Documents/Document/AsyncViewItem/1213?viewMode=00000000-0...
125	200	HTTP	localhost:1318	/Documents/Document/AsyncViewItem/1213?viewMode=00000000-0...
126	200	HTTP	localhost:1318	/Common/EntityHistory/History/1213?actionObject=5f17f751-2347-4...
127	200	HTTP	localhost:1318	/Common/Watch/Info?typeId=5f17f751-2347-42d1-bb5c-a3b97117...
154	200	HTTP	localhost:1318	/Common/LastObjectInfo/AddInfo

Рис. 146. Запросы, имеющие строку "Docflow/Acquaintance/Execute" в URL

Переносим выбранные запросы в JMeter с помощью процедуры экспорта и задаем для них соответствующие названия: **GetAcqTask** – для GET-запроса получения формы задачи ознакомления, **PostAcqTask** – для POST-запроса отправки формы задачи ознакомления.

5. Логаут

После того как основная часть нашего тестового плана завершена, осуществим выход из системы текущим пользователем, чтобы в ней не накапливались лишние сессии. Для лучшей наглядности эту операцию также желательно поместить в отдельный блок, поэтому добавим элемент Simple Controller и переименуем его в Logout. Далее включаем запись трафика в Fiddler и в браузере, на главной странице ELMA, выполняем следующее действие:

- Раскрываем меню пользователя и нажимаем на кнопку **Выйти**.

После проделанных действий возвращаемся в Fiddler и нажимаем на клавишу **F12** для остановки записи.

В данном случае нас интересует запрос, имеющий строку "Security/Account/LogOff" в своем URL (Рис. 147).

#	Result	Protocol	Host	URL
4	302	HTTP	localhost:1318	/Security/Account/LogOff
5	200	HTTP	localhost:1318	/Security/Account/LogOn

Рис. 147. Запрос, имеющий строку "Security/Account/LogOff" в URL

Переносим последний запрос в Jmeter и переименовываем его в "Logout".

Итак, после записи всех этапов плана наш тестовый сценарий будет выглядеть следующим образом (Рис. 148):

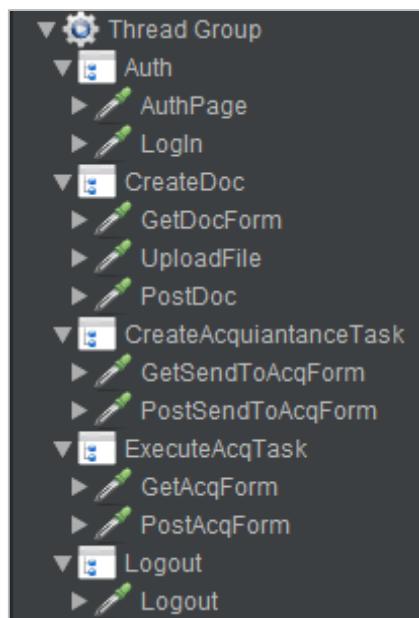


Рис. 148. Тестовый сценарий

- **Auth** – авторизация:
 - **AuthPage** – получение страницы авторизации;
 - **LogIn** – авторизация пользователем;
- **CreateDoc** – создание документа:
 - **GetDocForm** – получение формы создания документа;
 - **PostDoc** – отправка формы создания документа;
- **CreateAcquaintanceTask** – отправка документа на ознакомление:
 - **GetSendToAcqForm** – получение формы задачи отправки на ознакомление;
 - **PostSendToAcqForm** – отправка формы задачи на ознакомление;
- **ExecuteAcqTask** – ознакомление документа:
 - **GetAcqForm** – получение формы задачи ознакомления;
 - **PostAcqForm** – отправка формы задачи ознакомления;
- **Logout** – выход из системы:
 - **Logout** – логаут пользователя.

Следующим этапом написания нагрузочного сценария будет его настройка и отладка.

4.4. Настройка и отладка сценария нагрузочного тестирования

Под отладкой понимается процесс, позволяющий получить сценарий, который безошибочно функционирует исходя из заданных входных параметров профиля нагрузки для достижения цели тестирования производительности.

В ходе отладки вашего тестового сценария необходимо следовать следующим пунктам:

- при внесении изменений в тест-план, его необходимо сохранять через пункт меню **File – Save**;
- для запуска и прогонки теста нужно выбрать пункт меню **Run – Start**, предварительно указав в настройках Thread Group число пользователей – 1;
- после прогонки теста результат можно посмотреть в элементе View Results Tree. В случае успеха в таблице рядом с запросом будет поле с зеленой галочкой, иначе – с красной, что будет сигнализировать о том, что произошла ошибка;
- при ошибках нужно перейти в модуль View Results Tree и выяснить, какие запросы вызывают ошибку. Затем, если возможно, соответствующий запрос нужно исправить или удалить из тест-кейса;
- для очистки результатов предыдущего тестирования нужно выбрать пункт меню **Run – Clear All** или нажать на соответствующую кнопку панели инструментов Jmeter;
- повторять предыдущие шаги до тех пор, пока тест не будет пройдет успешно.

Для начала добавим в тестовый план элемент для отладки View Results Tree, в котором можно будет отследить выполнение того или иного этапа сценария, все ли запросы отправлены корректно, и получены ли корректные ответы.

На текущий момент наш тестовый план будет выглядеть следующим образом (Рис. 149).

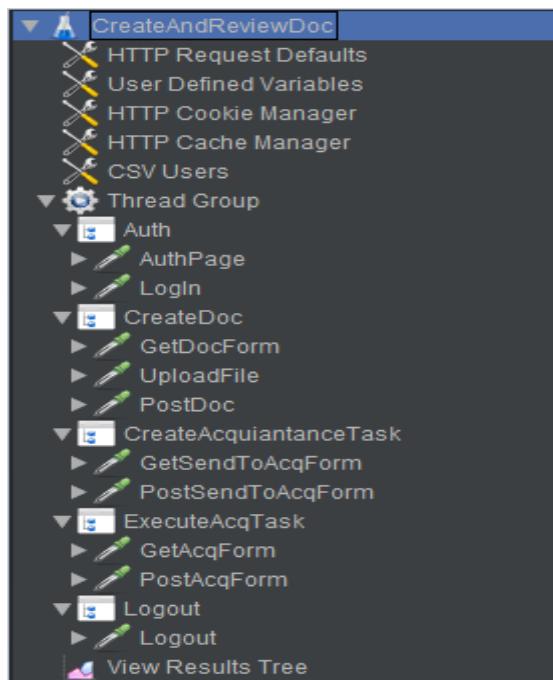


Рис. 149. Тестовый план

После этого перейдем в настройки каждого из имеющихся сэмплеров и очистим поля **Protocol [http]**, **Server Name or IP** и **Port Number**, поскольку мы задали их в элементе конфигурации [HTTP Request Defaults](#) (Рис. 150).

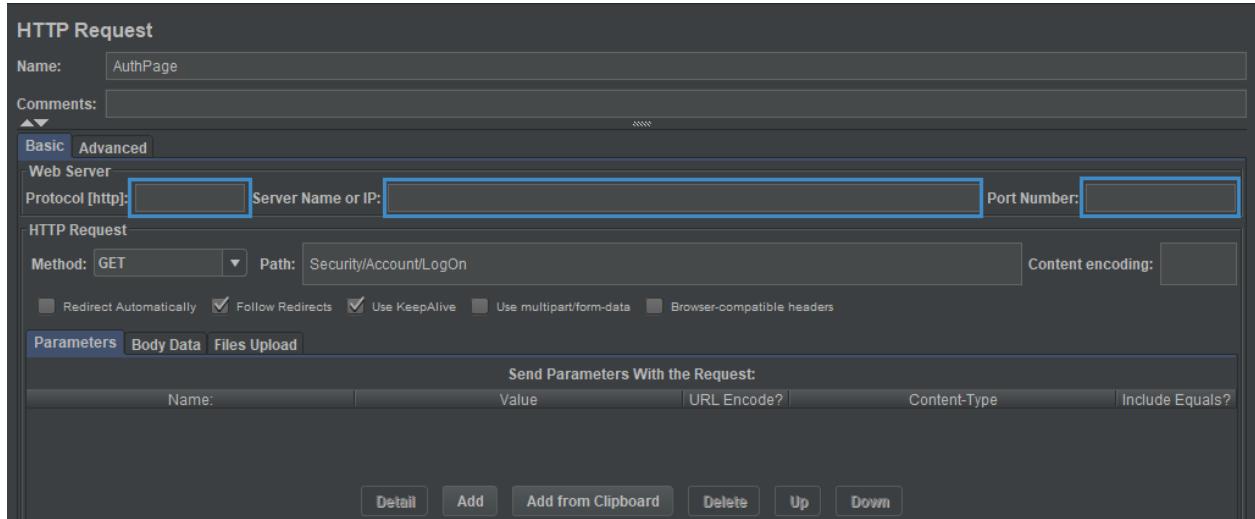


Рис. 150. HTTP Request. Поля "Protocol [http]", "Server Name or IP" и "Port Number"

Также из каждого HTTP запроса, совершающегося методом GET, удалим вложенный элемент HTTP Header Manager, поскольку он необходим только дляAjax-запросов, а также запросов, для которых необходимо выполнение редиректа, так как содержит в себе заголовок "Referer" (Рис. 151).

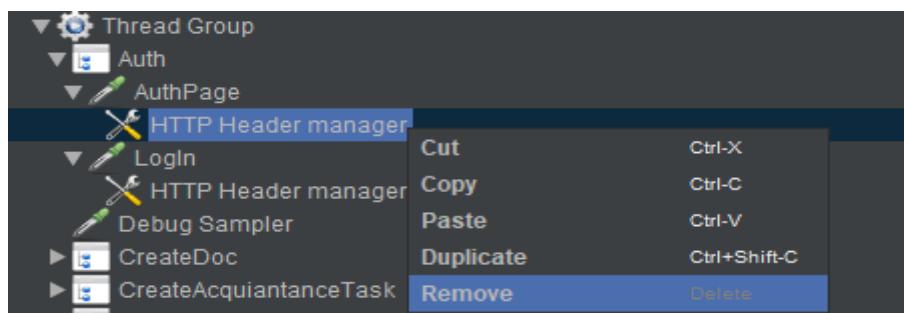


Рис. 151. Удаление вложенного элемента "HTTP Header Manager"

Многие данные на странице формируются динамически, в зависимости от контекста или логики выполняемой операции, поэтому использование одного и того же набора данных приведет к некорректной работе вашего тестового сценария. Поэтому главным этапом настройки тестового сценария является создание разных наборов данных для каждого виртуального пользователя или его **параметризация**.

Во время настройки определенной операции тестового плана желательно временно отключать следующие после нее группы запросов, чтобы не дожидаться их отработки в JMeter, так как ненастроенные они с большой вероятностью все равно не дадут нужного результата.

По мере окончания отладки операции и перехода к следующей, необходимый контроллер нужно будет активировать снова и так шаг за шагом, пока все этапы тестового плана не будут настроены. Поскольку мы начнем с этапа авторизации, деактивируем все контроллеры, кроме контроллера Auth. Для этого выделим их и выберем пункт контекстного меню **Disable** или используем сочетание клавиш **Ctrl + T**.

4.4.1. Авторизация

Перейдем в Post-запрос Login (Рис. 152). Удалим вложенный в него элемент HTTP Headers Manager, так как здесь он нам не нужен.

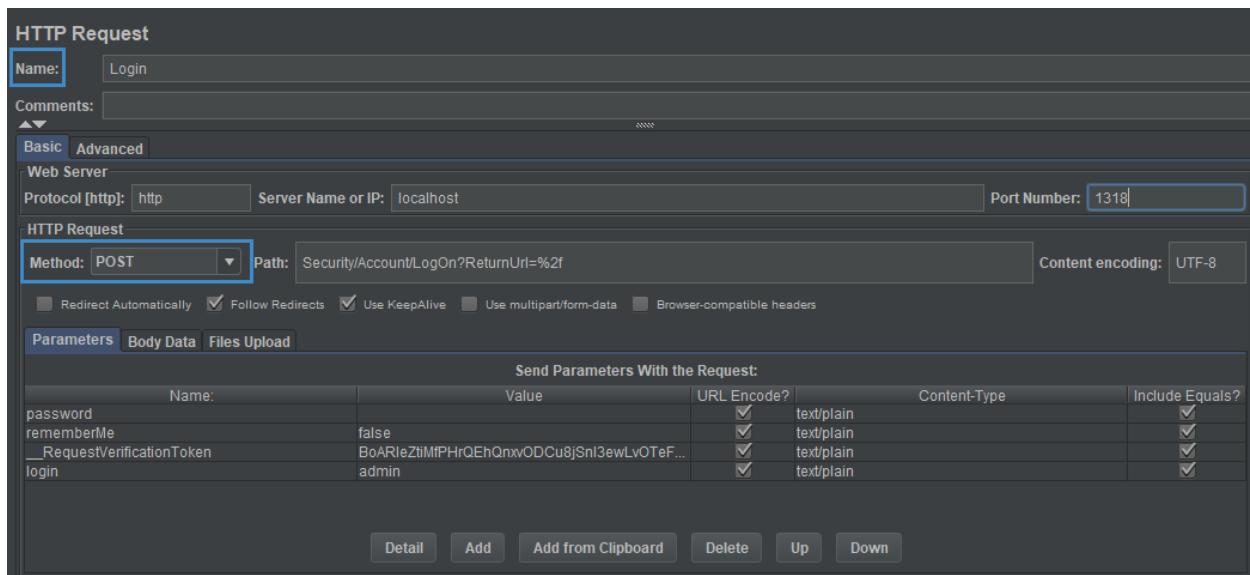


Рис. 152. Post-запрос "Login"

Ранее в наш тестовый план мы добавили элемент конфигурации CSV Data Set Config, в который мы загрузили CSV-файл с учетными данными участующего в teste пользователя и определили локальные переменные, в которые будут записаны нужные значения при старте теста. Чтобы использовать значения данных переменных, мы должны передать имя переменной в формате **\${имя переменной}** .

В нашем случае для параметра **password** в поле **Value** указываем переменную, хранящую пароль пользователя – **\${password}** , для параметра **login** указываем переменную, хранящую логин пользователя – **\${login}** .

Как мы видим, в данном запросе также передается параметр **_RequestVerificationToken** или [токен верификации запроса](#) , который динамически генерируется в коде запрашиваемой страницы и затем передается в теле связанного с ним запроса отправки формы этой страницы.

В нашем случае токен генерируется при получении страницы авторизации и передается в момент отправки пользовательских данных для входа в систему. Соответственно, необходимо извлечь со страницы авторизации значение данного токена. Для этого воспользуемся пост-обработчиком CSS/JQuery Extractor, в котором создадим запрос на языке CSS, описывающий данные, которые мы хотим получить. Разместим CSS/JQuery Extractor в качестве вложенного элемента GET-запроса AuthPage и для простоты восприятия переименуем его в Token Extractor (Рис. 153).

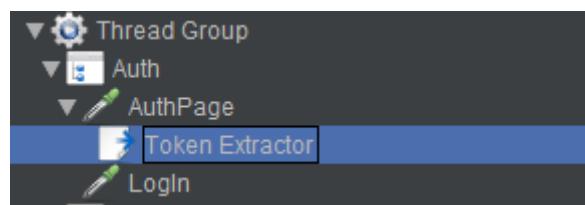


Рис. 153. Token Extractor

Для начала найдем нужный элемент на самой странице:

1. В браузере переходим на страницу авторизации ELMA.
2. Нажимаем на клавишу **F12**, чтобы вызвать консоль браузера.
3. На вкладке **Elements** (Google Chrome) нажимаем на клавиатуре сочетание клавиш **Ctrl + F** и в окне поиска элемента вводим его название, в нашем случае – **_RequestVerificationToken** (Рис. 154).

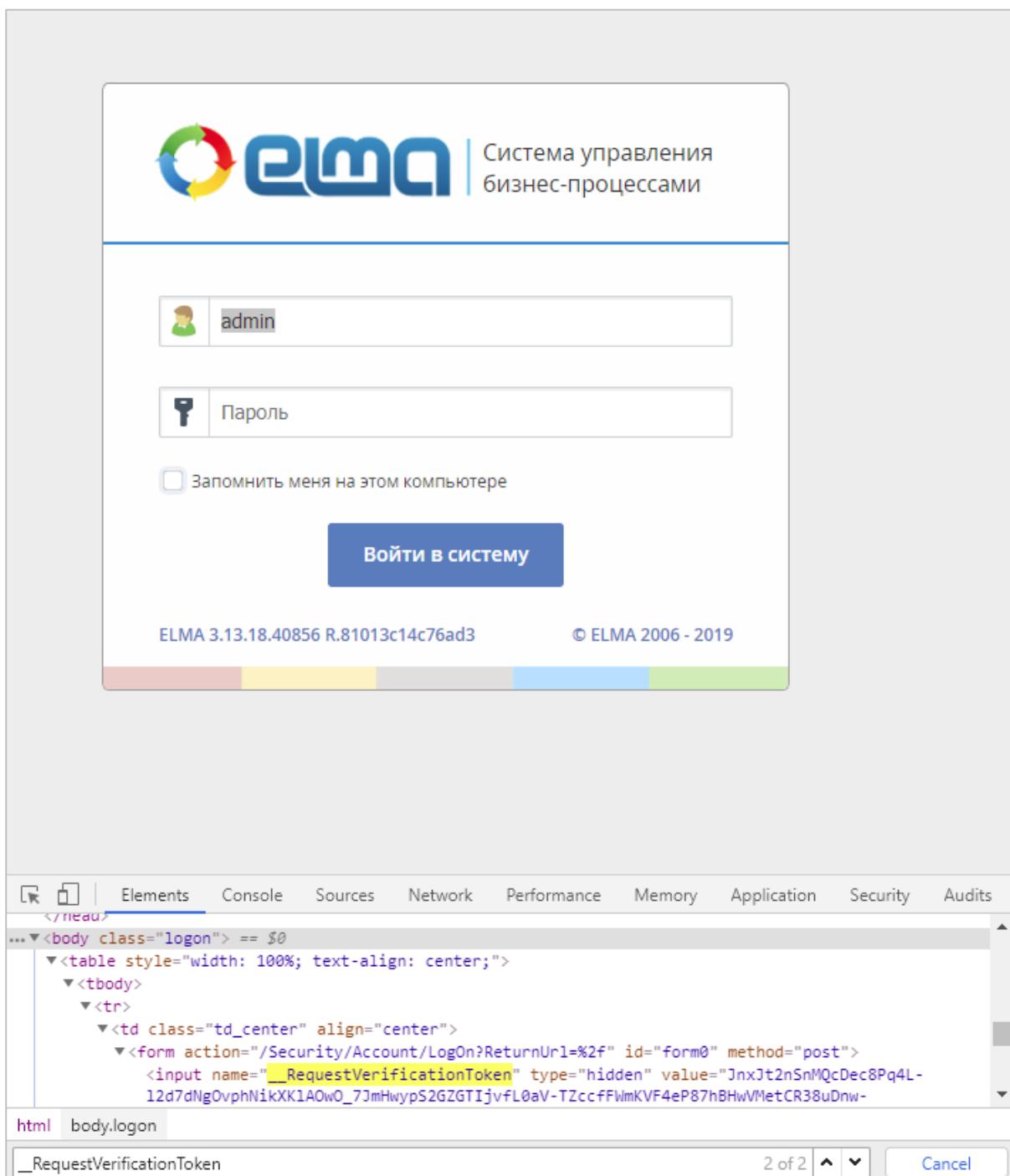


Рис. 154. Страница авторизации ELMA. Консоль браузера. Вкладка "Elements"

Как можно заметить, поиск выдал нам 2 результата. Однако нас интересует токен, который находится в атрибуте **name** тега **input**, а значит для его получения напишем соответствующий CSS-селектор – **input[name='RequestVerificationToken']**. Проверяем, что теперь поиск выдает единственный необходимый нам результат (Рис. 155), и переходим в Jmeter в элемент Token Extractor.

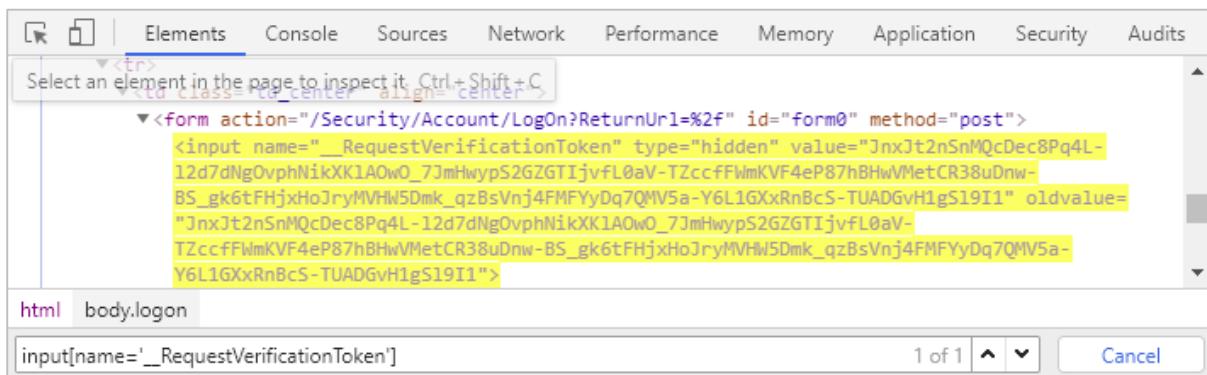


Рис. 155. Консоль браузера. Вкладка "Elements"

Настроим наш пост-обработчик следующим образом (Рис. 156):

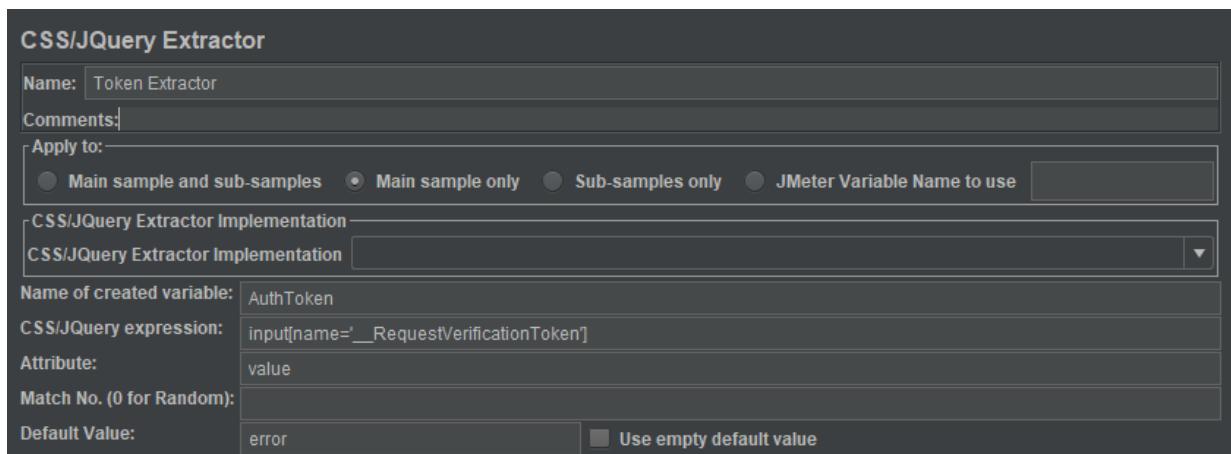


Рис. 156. Пост-обработчик "CSS/JQuery Extractor" для извлечения токена верификации

- в поле **Name** в качестве названия переменной Jmeter, в которой будет храниться значение извлеченного нами токена, укажем AuthToken;
- в поле **CSS/JQuery expression** впишем созданный нами запрос;
- в поле **Attribute** в качестве атрибута, значение которого необходимо извлечь, укажем атрибут **value**;
- в Поле **Default Value** в качестве значения по умолчанию создаваемой переменной укажем **error**.

После заполнения элемента переходим обратно в Post-запрос Login и для параметра **_RequestVerificationToken** в поле **Value** записываем **\${AuthToken}** .

Таким образом, после параметризации сэмплер будет выглядеть так (Рис. 157).

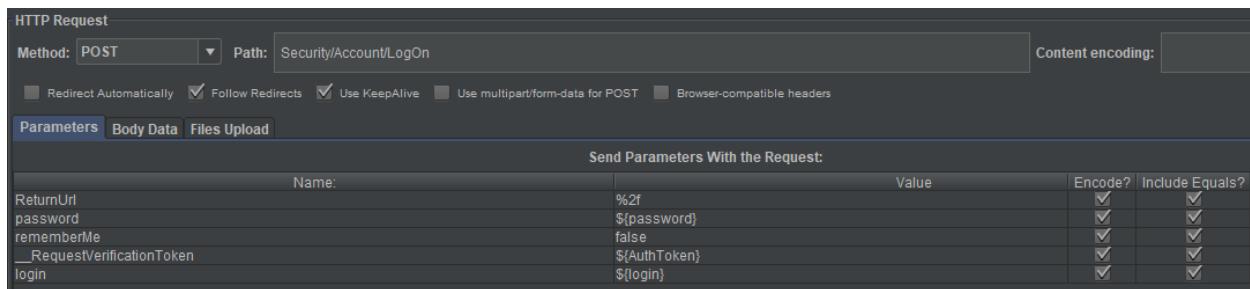


Рис. 157. Параметризация сэмплера "Login"

Для того чтобы отследить, что в ходе теста были присвоены корректные значения переменным JMeter, которые впоследствии будут переданы в качестве параметров сэмплеров, добавим элемент Debug Sampler. Разместим его в контроллере Auth после запроса Login (Рис. 158).

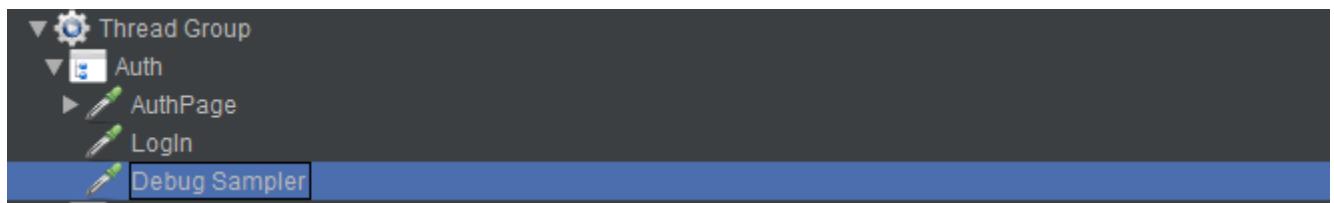


Рис. 158. Элемент "Debug Sampler"

В настройках элемента оставим только JMeter variables, остальные поставим в false (Рис. 159), поскольку нас интересует только значения самих переменных JMeter.



Рис. 159. Настройки Debug Sampler

Итак, первая группа запросов настроена, теперь необходимо запустить тест и проверить, что авторизация выполняется корректно.

Нажимаем на кнопку запуска теста и переходим в элемент View Results Tree (Рис. 160).

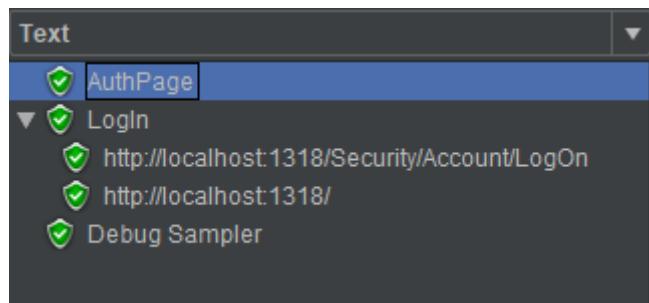


Рис. 160. Элемент "View Results Tree"

Все запросы выполнились без ошибок. Однако о том, что сама операция была выполнена корректно, говорить еще рано. Сначала перейдем в Debug Sampler и на вкладке **Response data** посмотрим результаты (Рис. 161).

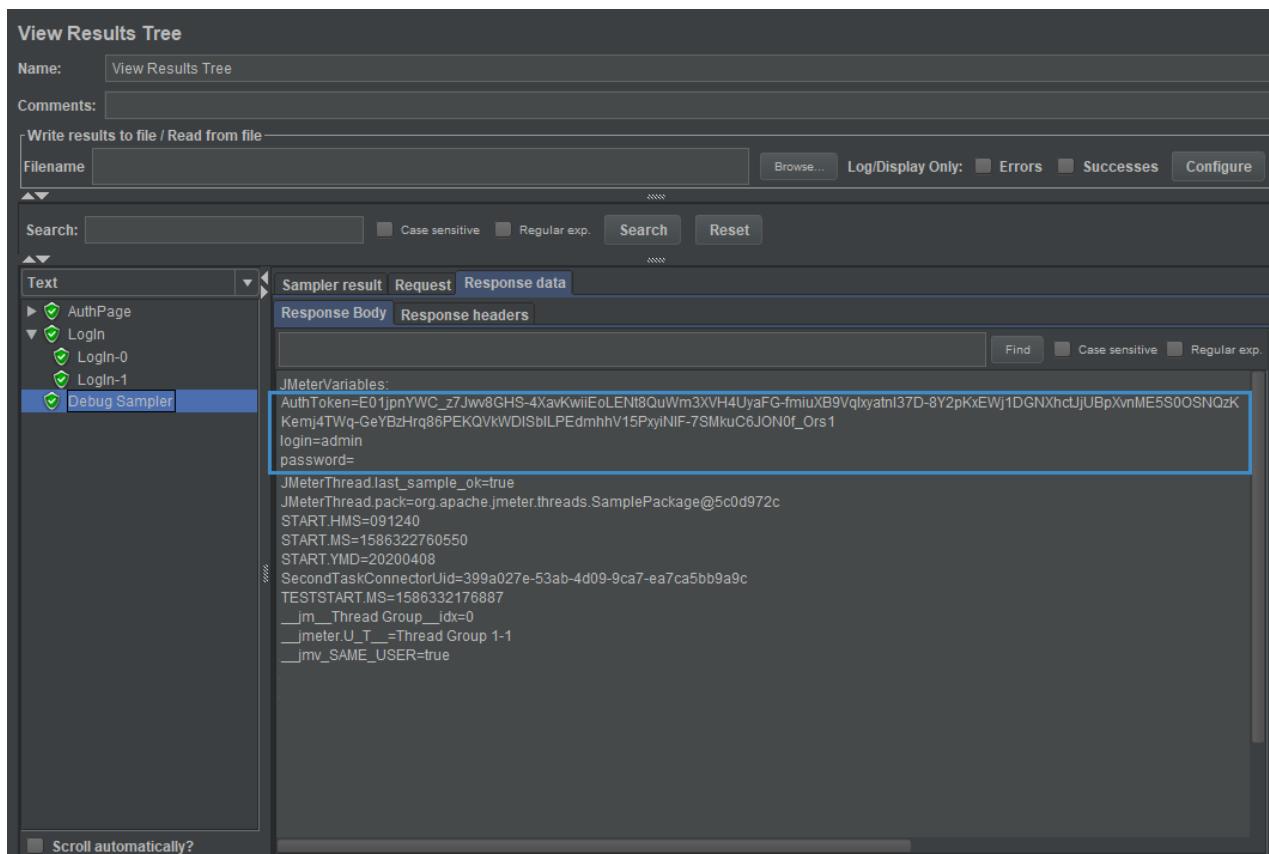


Рис. 161. Debug Sampler. Вкладка "Response data"

Как мы видим, значением переменной **AuthToken** не является **error**, а в переменные **login** и **password** записались нужные нам значения. Это означает, что в запросах также передались корректные значения.

Чтобы окончательно в этом убедиться, перейдем в каждый из выполненных сэмплеров и посмотрим, те ли данные вернул нам сервер.

Перейдем на вкладку **Response data** запроса AuthPage и в строку поиска вставим значение токена, полученного в переменной **AuthToken** (Рис. 162).

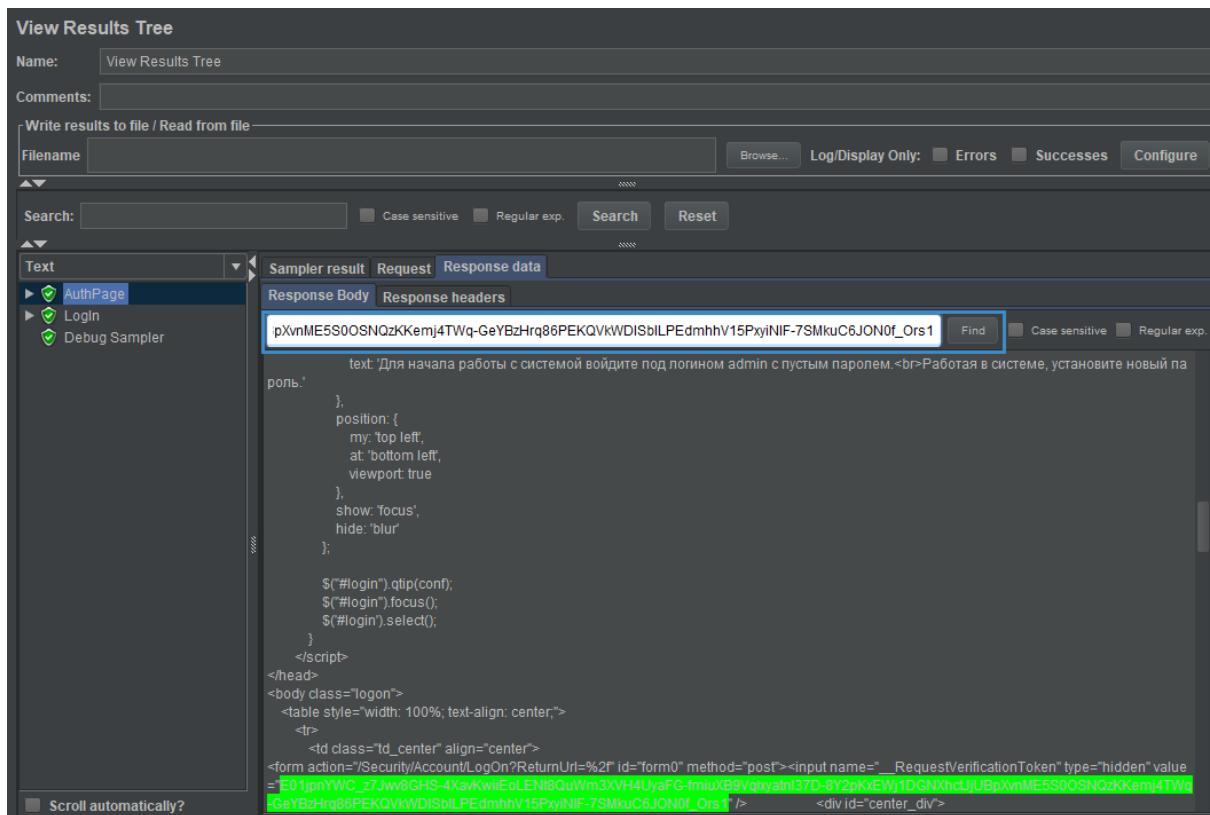


Рис. 162. Результаты выполнения сэмплера "AuthPage". Вкладка "Response data"

Как мы видим, токены полностью совпадают, а значит в теле Post-запроса Login должен был передаться корректный токен.

Перейдем на вкладку **Response data** запроса Login (Рис. 163). Воспользовавшись строкой поиска, в коде ответа, содержащем разметку главной страницы ELMA, мы обнаружили имя нашего пользователя, соответственно, пользователь вошел в систему.

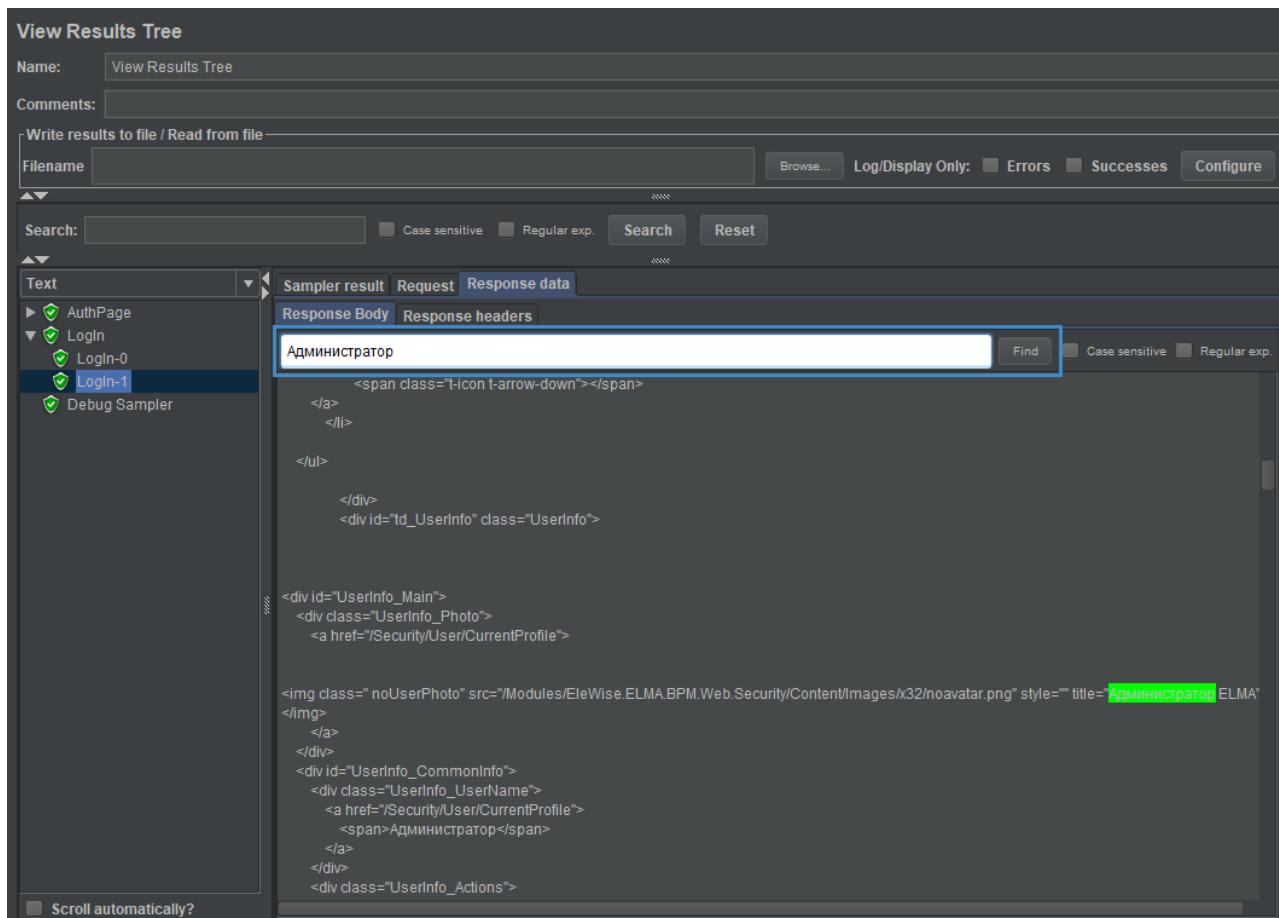


Рис. 163. Сопоставление имени пользователя в результатах выполнения сэмплера "Login". Вкладка "Response data"

После успешного выполнения операции авторизации переходим к отладке следующего этапа нашего сценария – создания документа.

4.4.2. Создание документа

Включаем контроллер CreateDoc (Рис. 164) и переходим в сэмплер GetDocForm (Рис. 165).

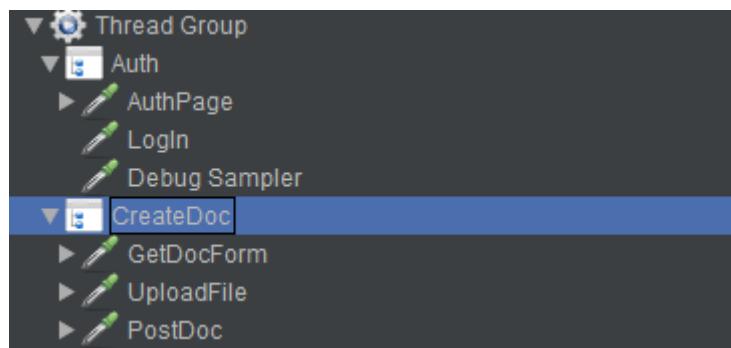


Рис. 164. Контроллер "CreateDoc"

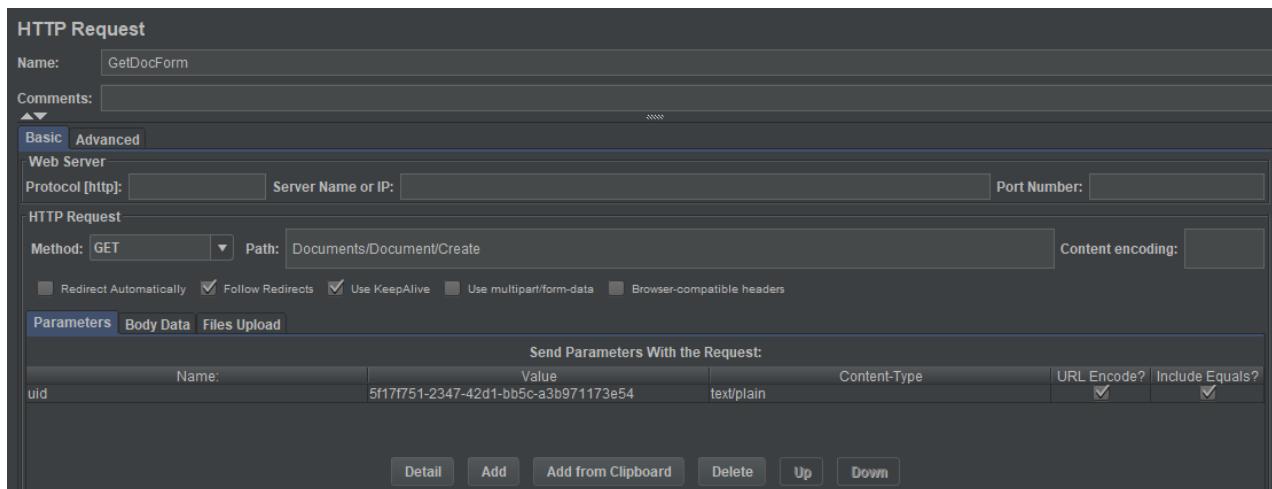


Рис. 165. Сэмплер "GetDocForm"

В строке данного запроса передается параметр **uid** для открытия формы создания конкретного документа. [Ранее](#) мы добавили элемент конфигурации Users Defined Variables, где создали переменную **docTypeUid**, хранящую в себе значение Uid нужного нам типа документа. Впишем **\${docTypeUid}** в качестве значения параметра **uid**.

Далее скопируем в сэмплер GetDocForm ранее созданный нами пост-обработчик Token Extractor из сэмплера AuthPage. Перейдем в настройки пост-обработчика и в поле **Name of created variable** укажем **DocToken**. Это необходимо для создания дополнительной переменной, которая будет хранить токен верификации, извлеченный в этот раз со страницы создания документа, что облегчит отладку в случае обнаружения проблем, связанных с передачей токена. В противном случае, значение переменной бы перезаписалось.

Перейдем в запрос UploadFile (Рис. 166), в котором мы загружаем версию для создаваемого нами документа. В тестовом сценарии мы будем генерировать каждый раз новую версию документа, содержание которой будет передаваться во вкладке **Body Data**.

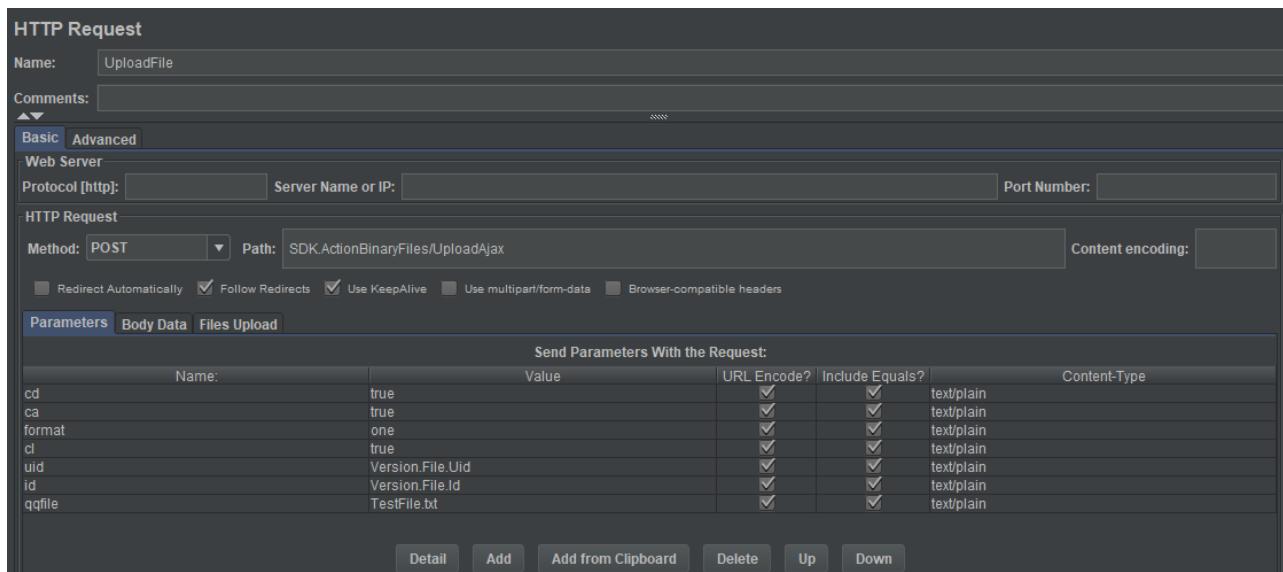


Рис. 166. Сэмплер "UploadFile"

Однако при нажатии на вкладку **Body Data** появляется следующее предупреждение (Рис. 167).

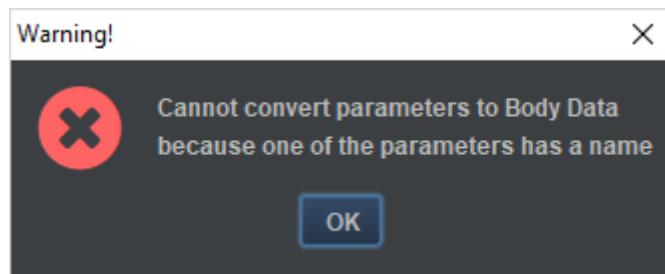


Рис. 167. Предупреждение при переходе на вкладку "Body Data"

Переход на данную вкладку невозможен, пока во вкладке **Parameters** имеются передаваемые запросом параметры на сервер. Для того чтобы вкладка **Body Data** оказалась доступна, и при этом не была нарушена структура запроса, необходимо передать имеющиеся во вкладке **Parameters** параметры в поле **Path** текущего запроса.

Передача параметров осуществляется с помощью символа "?", параметры разделяются между собой символом "&".

Таким образом, в поле **Path** после имеющейся строки "SDK.Action/BinaryFiles/UploadAjax" мы ставим "?" и начинаем прописывать необходимые данные в виде параметр=значение, при этом разделяя их между собой символом "&". Конечная строка поля **Path** будет выглядеть следующим образом (Рис. 168).

Path: SDK.Action/BinaryFiles/UploadAjax?cd=true&ca=true&cl=true&format=one&uid=Version.File.Uid&id=Version.File.Id&qqfile=TestFile.txt

Рис. 168. Поле "Path"

После этого очистим вкладку **Parameters**. Для этого выделите все имеющиеся на ней параметры и нажмите на кнопку **Delete**, которая находится на панели снизу (Рис. 169).

Name:	Value	URL Encode?	Include Equals?	Content-Type
cd	true	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	text/plain
ca	true	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	text/plain
format	one	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	text/plain
cl	true	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	text/plain
uid	Version.File.Uid	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	text/plain
id	Version.File.Id	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	text/plain
qqfile	TestFile.txt	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	text/plain

Detail Add Add from Clipboard Delete Up Down

Рис. 169. Очистка вкладки "Parameters". Кнопка "Delete"

Перейдем на вкладку **Body Data** и введем содержание версии документа (Рис. 170).

Protocol [http]: Server Name or IP: Port Number:

Method: POST Path: SDK.Action/BinaryFiles/UploadAjax?cd=true&ca=true&cl=true&format=one&uid=Version.File.Uid&id=Version.File.Id&qqfile=TestFile.txt Content encoding:

Redirect Automatically Follow Redirects Use KeepAlive Use multipart/form-data for POST Browser-compatible headers

Parameters Body Data Files Upload

I Some text

Рис. 170. Ввод содержания версии документа. Вкладка "Body Data"

После настройки сэмплера перейдем в сформированный внутри него элемент HTTP Header Manager (Рис. 171).

Name:	Value
Host	localhost:1318
Connection	keep-alive
Content-Length	28061
Origin	http://localhost:1318
X-File-Name	TasiCreate.jmx
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0...
Content-Type	application/octet-stream
X-Requested-With	XMLHttpRequest
IsLastXhr	true
_RequestVerificationToken	T4qla72r-7ZT-yAzKvkuMd05EPGCWA-THEgaiSKgE4CwlyEvfDb9oc8JwQO4gpvm01gPY2v42d...
Accept	*/*
Referer	http://localhost:1318/Documents/Document/Create?gridUniqueName=ContractStepsGrid699467...
Accept-Encoding	gzip, deflate, br
Accept-Language	ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7

Рис. 171. Элемент "HTTP Header Manager"

В самом начале составления тестового плана мы добавили элемент конфигурации [User Defined Variables](#), в котором определили переменные,

содержащие значения протокола передачи данных, имени и порта тестируемого сервера. Необходимо параметризовать каждый заголовок, строка которого содержит адрес сервера, используя созданные нами переменные, при этом сохранив имеющийся синтаксис. Например, строка "http://localhost:1318" превратится в **\${protocol}://\${server}:\${port}**.

В заголовок **_RequestVerificationToken** необходимо передать переменную **DocToken**, хранящую значение токена верификации со страницы создания документа, которую мы создаем в пост-обработчике Token Extractor.

В заголовке **Referer**, предназначенном для обратного перенаправления на страницу создания документа после добавления версии, имеется параметр **gridUniqueName**, значение которого формируется динамически. Для извлечения данного параметра воспользуемся пост-обработчиком CSS/JQuery Extractor. Для составления корректного запроса на языке CSS сначала найдем этот элемент в коде страницы:

- в браузере откроем страницу авторизации в системе и авторизуемся;
- перейдем на форму создания документа "Входящий договор".

Откроем консоль браузера и на вкладке **Elements** вызовем окно поиска элемента, куда введем "ContractStepsGrid". Нас интересует следующая часть кода страницы (Рис. 172).

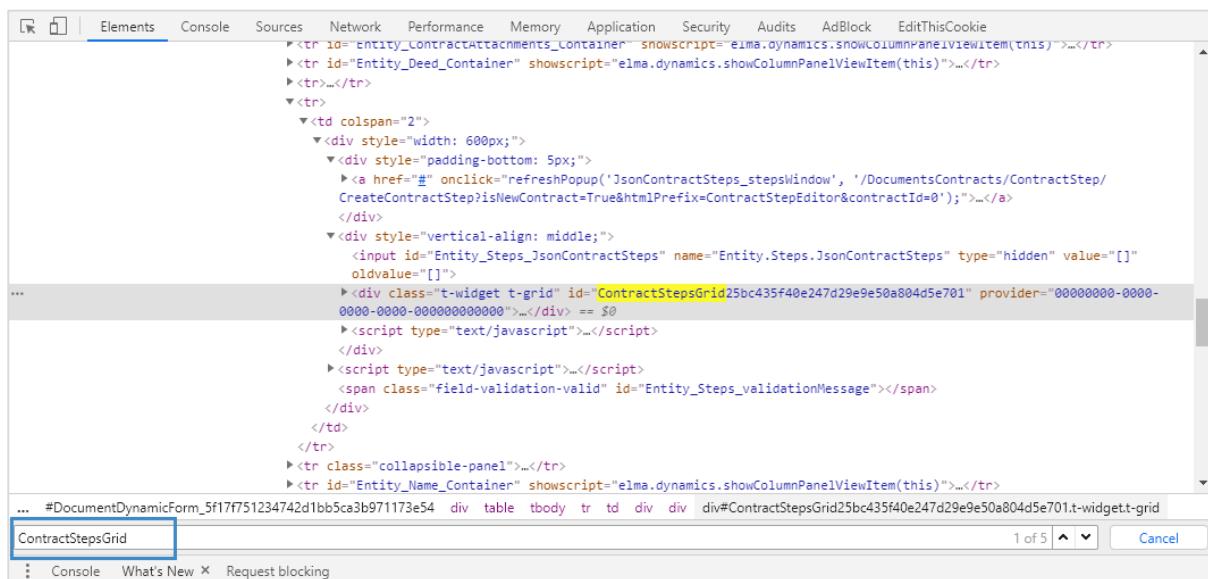


Рис. 172. Консоль браузера. Вкладка "Elements"

Когда нужный нам селектор составлен, добавим элемент CSS/JQuery Extractor в качестве вложенного элемента сэмплера GetDocForm со следующими настройками (Рис. 173).

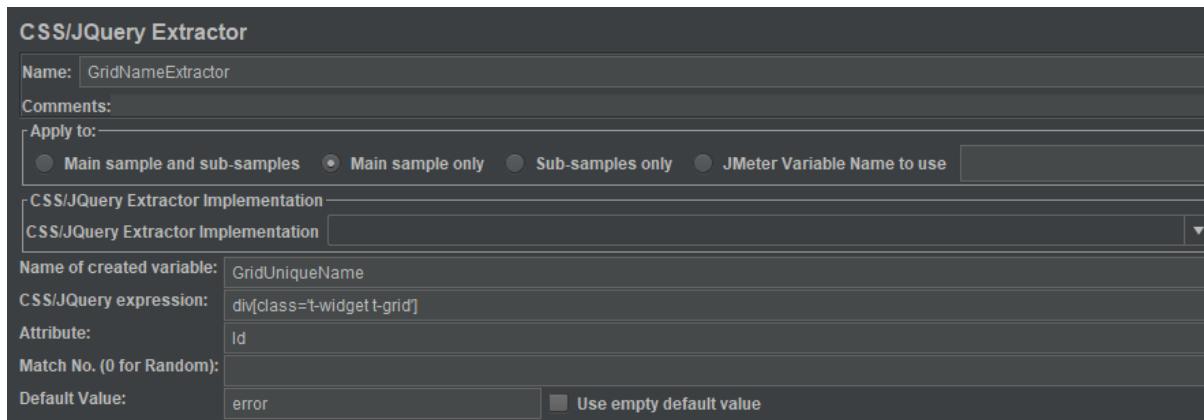


Рис. 173. Элемент "CSS/JQuery Extractor"

После параметризации элемент HTTP Header Manager будет выглядеть следующим образом (Рис. 174).

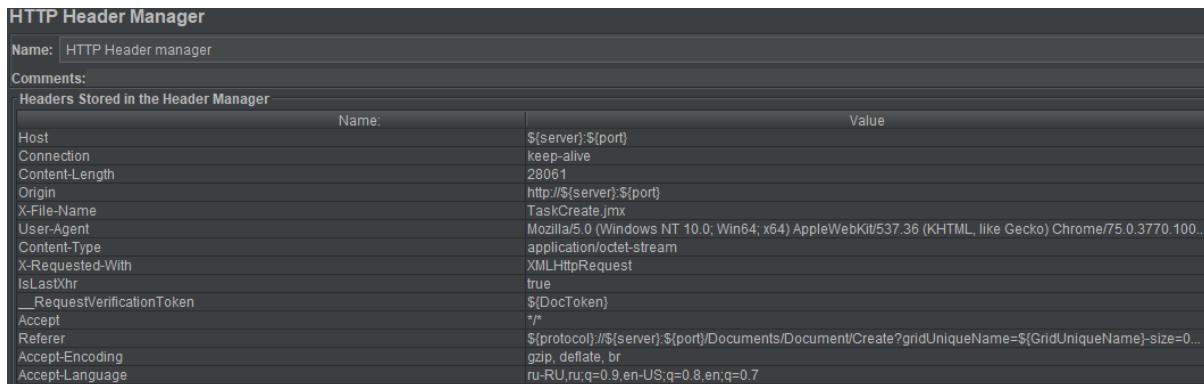


Рис. 174. Параметризованный элемент "HTTP Header Manager" сэмплера "UploadFile"

Перейдем к последнему запросу данной группы – PostDoc.

В этот раз все параметры передаются во вкладке **Body Data** в виде текста, разделенного с помощью специального маркера границы **WebKitFormBoundary**, по которому в дальнейшем эти данные анализируются сервером (Рис. 175).

```

HTTP Request
Method: POST Path: Documents/Document/Create Content encoding:
    Redirect Automatically Follow Redirects Use KeepAlive Use multipart/form-data for POST Browser-compatible headers
Parameters Body Data Files Upload
1 -----WebKitFormBoundary5Fzr5pT7ecFnLqnq
2 Content-Disposition: form-data; name="__RequestVerificationToken"
3
4 IFNn367GX6d3auiyMq7g145LmjYqa0LUN8FL9PVexEAogZTAGwbnuJSu1yz5HDDmH4dFL6ykGRX48nnHiKyN-Qzq9HO_VrBFrI11rOUa7TbL-urD8RGk4-
5 P3mSIpxuXL754J0atpzclWu23AOV91uT_UPqVvLfSt2Q0vrspDs1
6 -----WebKitFormBoundary5Fzr5pT7ecFnLqnq
7 Content-Disposition: form-data; name="InitFolderId"
8
9 -----WebKitFormBoundary5Fzr5pT7ecFnLqnq
10 Content-Disposition: form-data; name="LinkedDocument"
11
12 -----WebKitFormBoundary5Fzr5pT7ecFnLqnq
13 Content-Disposition: form-data; name="ParentDocument"
14
15 -----WebKitFormBoundary5Fzr5pT7ecFnLqnq
16 Content-Disposition: form-data; name="Entity.TypeId"
17
18 Sf17f751-2347-42d1-bb5c-a3b971173e54
19 -----WebKitFormBoundary5Fzr5pT7ecFnLqnq
20 Content-Disposition: form-data; name="CurrentViewProvider"
21
22 00000000-0000-0000-000000000000
23 -----WebKitFormBoundary5Fzr5pT7ecFnLqnq
24 Content-Disposition: form-data; name="SaveToCurrentVersion"
25
26 False
27 -----WebKitFormBoundary5Fzr5pT7ecFnLqnq
28 Content-Disposition: form-data; name="PopupId"
29
30 -----WebKitFormBoundary5Fzr5pT7ecFnLqnq
31 Content-Disposition: form-data; name="Callback"
32
33 -----WebKitFormBoundary5Fzr5pT7ecFnLqnq
34 Content-Disposition: form-data; name="__DynamicFormSettings.ShowDescription"
35
36
37 -----WebKitFormBoundary5Fzr5pT7ecFnLqnq
38 Content-Disposition: form-data; name="Entity.__jsonState"
39
40

```

Рис. 175. Запрос "PostDoc". Вкладка "Body Data"

Для начала удалим из **Body Data** параметры с пустыми значениями и относящиеся к ним маркеры границы (Рис. 176), а также параметры и их значения, содержащие в своем имени "json" (Рис. 177 Рис. 176) или "validate" (Рис. 178).

```

-----WebKitFormBoundaryuyMvQGEHkB0Lbhuj6
Content-Disposition: form-data; name="ComplexExecutor_Workers_PrefixedValuePersonalGroupSelector-input"

```

Рис. 176. Параметры с пустыми значениями

```

-----WebKitFormBoundary5Fzr5pT7ecFnLqnq
Content-Disposition: form-data; name="Entity.__jsonState"

```

Рис. 177. Параметры и их значения, содержащие в имени "json"

```

-----WebKitFormBoundary5Fzr5pT7ecFnLqnq
Content-Disposition: form-data; name="Entity.MyLegalPerson_validate"

```

Рис. 178. Параметры и их значения, содержащие в имени "validate"

Разберем оставшиеся параметры запроса (Рис. 179, Рис. 180, Рис. 181):

```

1 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
2 Content-Disposition: form-data; name="__RequestVerificationToken"
3
4 IFNn367GX6d3auiyMq7gL45LmjYqal0LUN8FL9PVexEAogZTAGwbnuJSu1yz5HDDmH4dFL6ykGRX48nnHiKyN-Qzq9HO_vrBFrUa7TbL-
5 P3mSIpzUXL7S4J0atppzcWu23A0V91IuT_UPqYvLfSt2Q0vrspDs1
-----WebKitFormBoundarySFZr5pT7ecFnLqnq
6 Content-Disposition: form-data; name="Entity.TypeUid"
7
8 Sf17f751-2347-42d1-bb5c-a3b971173e54
9 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
10 Content-Disposition: form-data; name="CurrentViewProvider"
11
12 00000000-0000-0000-000000000000
13 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
14 Content-Disposition: form-data; name="SaveToCurrentVersion"
15
16 False
17 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
18 Content-Disposition: form-data; name="__DynamicFormSettings.FormId"
19
20 GlobalForm
21 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
22 Content-Disposition: form-data; name="__DynamicFormSettings.HtmlPrefix"
23
24 Entity
25 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
26 Content-Disposition: form-data; name="__DynamicFormSettings.DynamicFormsProviderUid"
27
28 e2dae6ba-2b80-4b71-a12d-f8c694dcc50f
29 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
30 Content-Disposition: form-data; name="__DynamicFormSettings.DynamicFormsProviderData"
31
32 Sf17f751-2347-42d1-bb5c-a3b971173e54
33 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
34 Content-Disposition: form-data; name="__DynamicFormSettings.ViewProviderUid"
35
36 76c8ee6e-f80c-4fd8-b085-312ffc97def4
37 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
38 Content-Disposition: form-data; name="Entity.ContractDate_date"
39
40 21.06.2019

```

Рис. 179. Параметры запроса "PostDoc"

- **_RequestVerificationToken** – извлекаем его значение из HTML-кода страницы создания документа;
- **Entity.TypeUid** – значение определено в переменной **User Defined Variables**;
- **CurrentViewProvider** – оставляем без изменения;
- **SaveToCurrentVersion** – оставляем без изменения;
- **_DynamicFormSettings.FormId** – извлекаем его значение из HTML-кода страницы создания документа;
- **_DynamicFormSettings.HtmlPrefix** – извлекаем его значение из HTML-кода страницы создания документа;

- **_DynamicFormSettings.DynamicFormsProviderUid** – извлекаем его значение из HTML-кода страницы создания документа;
- **_DynamicFormSettings.DynamicFormsProviderData** – извлекаем его значение из HTML-кода страницы создания документа;
- **_DynamicFormSettings.ViewProviderUid** – извлекаем его значение из HTML-кода страницы создания документа;
- **Entity.ContracDate_date** (Дата создания договора) – будем генерировать текущую дату;

```

41 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
42 Content-Disposition: form-data; name="Entity.ContractDate"
43
44 21.06.2019
45 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
46 Content-Disposition: form-data; name="Entity.ContractDate_NoOffset"
47
48 True
49
50 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
51 Content-Disposition: form-data; name="Entity.ContractPeriod_NoOffset"
52
53 True
54 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
55 Content-Disposition: form-data; name="Entity.Subject"
56
57 Тема договора
58 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
59 Content-Disposition: form-data; name="Entity.Name"
60
61 Входящий договор
62
63 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
64 Content-Disposition: form-data; name="FolderTreePopup710e2e80_ff15_45c0_a9a5_1c078b91314dfolder_name"
65
66 203
67
68 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
69 Content-Disposition: form-data; name="Entity.Folder.Id"
70
71 203
72 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
73 Content-Disposition: form-data; name="Profile.Id"
74
75 2

```

Рис. 180. Параметры запроса "PostDoc"

- **Entity.ContractDate** (Дата договора) – будем генерировать текущую дату;
- **Entity.ContractDate_NoOffset** – оставляем без изменения;
- **Entity.ContractPeriod_NoOffset** – оставляем без изменения;
- **Entity.Subject** (Тема договора) – будем генерировать случайную тему;
- **Entity.Name** (Название договора) – будем генерировать случайное название;

- **Folder_name** (Id папки, в которой будет создан документ) – определим значение в переменной **User Defined Variables**;
- **Entity.Folder.Id** (Id папки, в которой будет создан документ) – определим значение в переменной **User Defined Variables**;
- **Profile.Id** – извлекаем его значение из HTML-кода страницы создания документа;

```

76 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
77 Content-Disposition: form-data; name="UseFile"
78
79 True
80 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
81 Content-Disposition: form-data; name="Version.File.Uid"
82
83 5bf52b74-c570-44cf-aeb3-7db73a0da702
84 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
85 Content-Disposition: form-data; name="file"; filename=""
86 Content-Type: application/octet-stream
87
88 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
89 Content-Disposition: form-data; name="MakeVersionCurrent"
90
91 True
92 -----WebKitFormBoundarySFZr5pT7ecFnLqnq--
```

Рис. 181. Параметры запроса "PostDoc"

- **UseFile** – оставляем без изменения;
- **Version.File.Uid** (Uid версии документа) – извлекаем его значение из ответа на запрос UploadFile;
- **Content-Disposition/Content-Type** – оставляем без изменения;
- **MakeVersionCurrent** (флажок **Сделать текущей**) – оставляем без изменения.

После просмотра всех параметров можно выделить следующие группы действий.

1. Извлечение значения параметра из HTML-кода страницы.

RequestVerificationToken в нашем сценарии уже извлекается с помощью пост-обработчика CSS/JQuery Extractor, и его значение передается в переменную **DocToken**, соответственно, впишем **\${DocToken}** в качестве значения передаваемого токена верификации (Рис. 182).

```

1 -----WebKitFormBoundarySFZr5pT7ecFnLqnq
2 Content-Disposition: form-data; name="__RequestVerificationToken"
3
4 ${DocToken}
```

Рис. 182. Значение передаваемого токена верификации

Добавим 6 пост-обработчиков CSS/JQuery Extractor и разместим их внутри сэмплера GetDocForm для извлечения значения каждого из следующих параметров:

- [DynamicFormSettings.FormId](#);
- [DynamicFormSettings.HtmlPrefix](#);
- [DynamicFormSettings.DynamicFormsProviderUid](#);
- [DynamicFormSettings.DynamicFormsProviderData](#);
- [DynamicFormSettings.ViewProviderUid](#);
- [Profile.Id](#).

Принцип выполняемых действий по настройке пост-обработчиков схож с настройкой элемента при [извлечении токена верификации](#). Различие заключается в том, что в этот раз поиск будет осуществляться в HTML-коде страницы создания документа (Рис. 183).

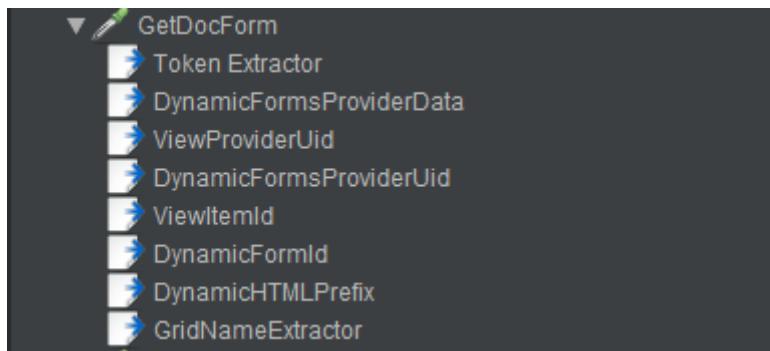


Рис. 183. Сэмплер "GetDocForm"

После настройки пост-обработчиков у нас получилось следующее:

- переменная **FormId** со значением параметра **_DynamicFormSettings.FormId**. Используемый CSS-селектор: `input[name='__DynamicFormSettings.FormId']`. Значение переменной по умолчанию – error;
- переменная **HtmlPrefix** со значением параметра **_DynamicFormSettings.HtmlPrefix**. Используемый CSS-селектор: `input[name='__DynamicFormSettings.HtmlPrefix']`. Значение переменной по умолчанию – error;
- переменная **ProviderUid** со значением параметра **_DynamicFormSettings.DynamicFormsProviderUid**. Используемый CSS-селектор: `input[name='__DynamicFormSettings.DynamicFormsProviderUid']`. Значение переменной по умолчанию – error;

- переменная **ProviderData** со значением параметра **_DynamicFormSettings.DynamicFormsProviderData**. Используемый CSS-селектор: `input[name='__DynamicFormSettings.DynamicFormsProviderData']`. Значение переменной по умолчанию – error;
- переменная **ViewProviderUid** со значением параметра **_DynamicFormSettings.ViewProviderUid**. Используемый CSS-селектор: `input[name='__DynamicFormSettings.ViewProviderUid']`. Значение переменной по умолчанию – error.
- переменная **Profile.Id** со значением параметра **Profile.Id**. Используемый CSS-селектор: `input[name='Profile.Id']`. Значение переменной по умолчанию – error.

Перейдем во вкладку **Body Data** запроса PostDoc и впишем в поле со значением каждого из разбираемых выше параметров соответствующую ему переменную в формате **\${имя переменной}**.

Итак, нам осталось извлечь значение параметра **Version.File.Uid**. Для начала необходимо увидеть текст ответа на запрос загрузки файла **UploadFile**, в котором это значение содержится:

- перейдите в Fiddler и убедитесь, что в строке состояния имеется надпись "Capturing";
- в браузере авторизуйтесь в системе ELMA и перейдите на страницу создания документа;
- на странице создания документа выполните загрузку файла-версии документа;
- в Fiddler в списке сессий найдите запрос, имеющий в своем URL строку "/SDK.Action/BinaryFiles/UploadAjax", и нажмите на него;
- перейдите в рабочую вкладку **Inspectors**, а затем в подвкладку ответа **TextView** (Рис. 184).

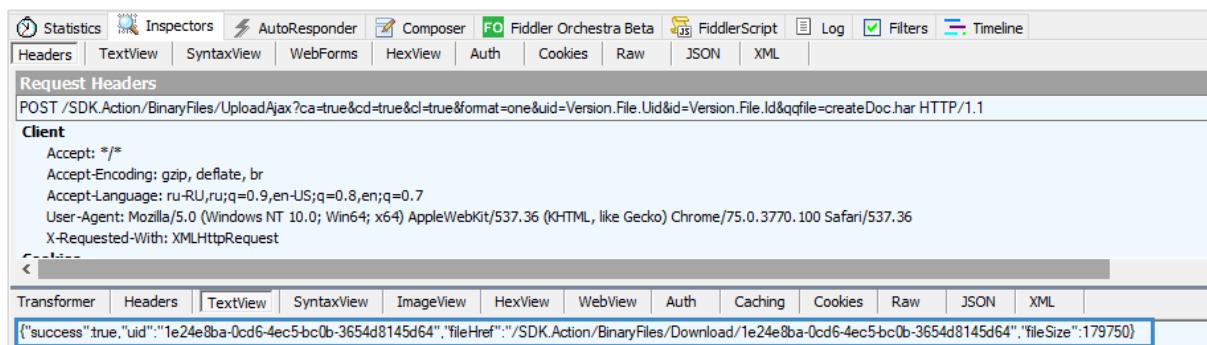


Рис. 184. Запрос. Вкладка "Inspectors". Подвкладка "TextView"

Необходимая нам строка выглядит следующим образом:

```
{"success":true,"uid":"1e24e8ba-0cd6-4ec5-bc0b-3654d8145d64","fileHref":"/SDK.Action/BinaryFiles/Download/1e24e8ba-0cd6-4ec5-bc0b-3654d8145d64","fileSize":179750}, где 1e24e8ba-0cd6-4ec5-bc0b-3654d8145d64 – Uid версии документа, который нам необходимо получить.
```

В этот раз извлекать данные мы будем с помощью регулярного выражения, используя пост-обработчик Regular Expression Extractor. Добавим Regular Expression Extractor в качестве вложенного элемента сэмплера UploadFile, поскольку нам необходимо извлечь информацию из ответа после обработки данного запроса сервером, и переименуем его в FileUid Extractor (Рис. 185)



Рис. 185. Сэмплер "UploadFile". Вложенный элемент "FileUid Extractor"

После этого выполним следующие действия:

- перейдем на сайт <https://regex101.com/> для создания и проверки регулярного выражения;
- в поле **Test String** введем полученную строку ответа, а в поле **Regular Expression** начнем по нему подбирать необходимое нам выражение (Рис. 186);

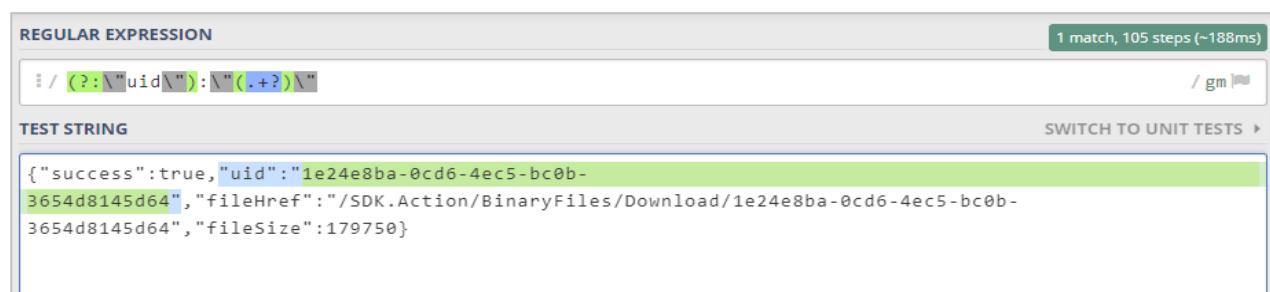


Рис. 186. Разбор регулярного выражения. Поля "Regular Expression" и "Test String"

- информацию о полученных совпадениях можно посмотреть в форме **Match Information** (Рис. 187).

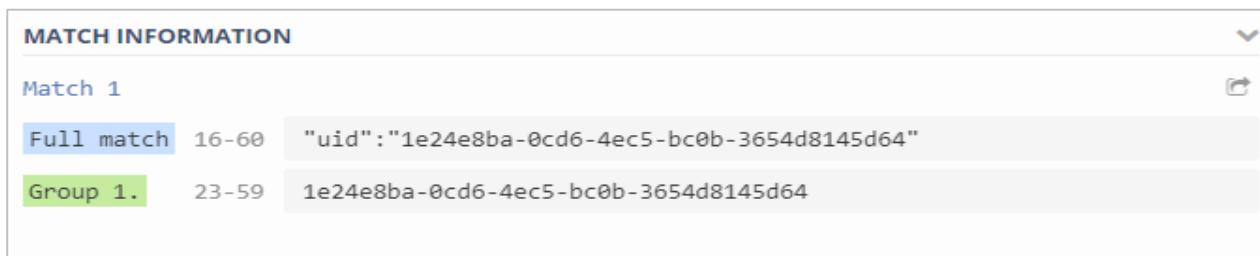


Рис. 187. Форма "Match Information"

После нахождения необходимого регулярного выражения настроим пост-обработчик следующим образом (Рис. 188):

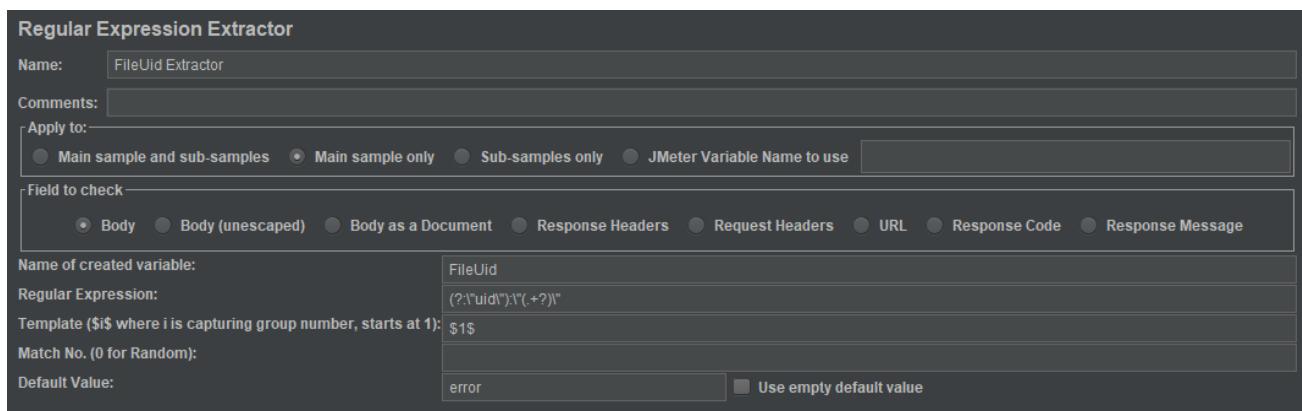


Рис. 188. Настройки пост-обработчика "Regular Expression Extractor"

- **FileUid** – название переменной Jmeter, в которой будет храниться значение извлеченного Uid;
- **(?:"uid\"):\"(.+?)\"** – регулярное выражение для извлечения Uid;
- **\$1\$** – группа совпадений, результат которой необходимо извлечь. В нашем случае – группа 1;
- **error** – значение переменной по умолчанию.

После настройки элемента переходим во вкладку **Body Data** запроса PostDoc и в качестве значения параметра **Version.File.Uid** указываем **\${FileUid}** .

2. Генерация значений переменных

В отправляемом нами запросе каждый раз будет задаваться разное значение для темы и названия договора, тем самым имитируя реальное поведение пользователей. Добиться этого нам поможет сценарий, написанный в элементе [JSR223PreProcessor](#). Выбор препроцессора в данном случае обусловлен тем, что генерация данных должна производиться непосредственно перед выполнением самого запроса.

Добавим JSR223PreProcessor в качестве вложенного элемента сэмплера PostDoc и переименуем его в GenerateRandomStrings (Рис. 189).

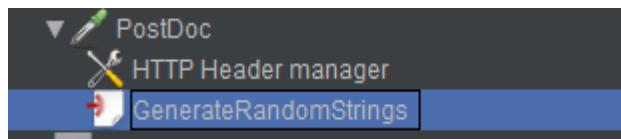


Рис. 189. Сэмплер "PostDoc". Вложенный элемент "GenerateRandomStrings"

Далее перейдем в настройки элемента. В качестве языка сценария выберем Groovy (Рис. 190).

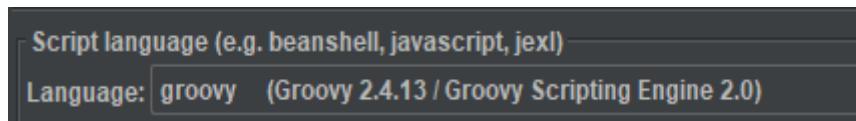


Рис. 190. Выбор языка сценария

В поле **Script** вводим следующий код:

```
import java.util.Random;
import java.lang.StringBuilder;
import java.text.SimpleDateFormat;

String generateRandom(int quantity, Random random)
{
    StringBuilder builder = new StringBuilder();
    String mCHAR = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
    for (int i = 0; i < quantity; i++)
    {
        int number = random.nextInt(mCHAR.length());
        char ch = mCHAR.charAt(number);
        builder.append(ch);
    }
    return builder.toString();
}
Random random = new Random();
String docName = "Document_" + generateRandom(4, random);
String subject = "Subject_" + generateRandom(6, random);
vars.put("docName", docName);
vars.put("subject", subject);
```

- **generateRandom** – функция для генерации строки из определенного числа латинских букв, в зависимости от переданного в нее значения этого числа;
- с помощью метода **vars.put** мы создаем переменную Jmeter и сохраняем в ней необходимое значение. К записанной переменной впоследствии можно будет обратиться в teste в формате **\${имя_переменной}** или в сценарии с помощью метода **vars.get ("имя переменной")** . Однако следует помнить, что методы **vars.put** и

vars.get поддерживают только переменные с типом строки (**String**). Сам объект **vars (Jmeter.Variables)** представляет собой хеш-таблицу (**HashMap**), в которую мы складываем переменные с типом **object** или **string**. Они являются изолированными и могут быть использованы только в рамках потока, в котором были созданы.

После настройки элемента переходим во вкладку **Body Data** запроса PostDoc и в качестве значения параметра **Entity.Subject** указываем **\${subject}** , а в качестве значения параметра **Entity.Name** указываем **\${docName}** .

В качестве значения параметров **Entity.ContracDate_date** и **Entity.ContractDate** нам необходимо передать текущую дату в формате **dd.MM.yyyy** . Для этого воспользуемся встроенной функцией JMeter **"\${__time(dd.MM.yyyy)}"**, которая будет возвращать текущую дату в указанном в скобках формате. Переходим во вкладку **Body Data** запроса PostDoc и впишем эту функцию в качестве значения данных параметров (Рис. 191).

```
-----WebKitFormBoundarySFZr5pT7ecFnLqnq
Content-Disposition: form-data; name="Entity.ContractDate_date"
${__time(dd.MM.yyyy,)}

-----WebKitFormBoundarySFZr5pT7ecFnLqnq
Content-Disposition: form-data; name="Entity.ContractDate"
${__time(dd.MM.yyyy,)}
```

Рис. 191. Встроенная функция JMeter в качестве значения параметров запроса "PostDoc".
Вкладка "Body Data"

3. Использование значений из объявленных переменных JMeter

Во время отладки теста нередко возникает необходимость записать значение определенного параметра запроса в переменную и использовать его в качестве константы для определенных элементов теста, как, например, в случае со значением Id папки, в которую будет помещаться создаваемый нами документ, передаваемым в параметрах **Folder_name** и **Entity.Folder.Id** текущего запроса.

Для этого перейдем в элемент конфигурации User Defined Variables и в нижней панели нажмем на кнопку **Add** (Рис. 192).

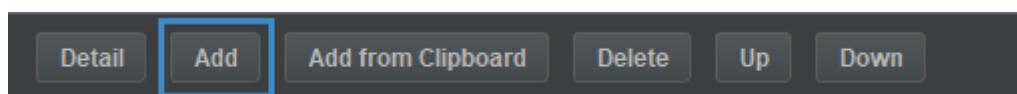


Рис. 192. Элемент конфигурации "User Defined Variables". Кнопка "Add"

В качестве имени переменной зададим "folderId", а ее значением укажем "203" – идентификатор папки **Мои документы** на тестируемом веб-сервере ELMA (Рис. 193).

User Defined Variables		
Name:	Value	Description
docTypeUid		Uid типа документа "Входящий договор"
protocol	http	Протокол
server	localhost	Имя сервера
port	1318	Порт
FolderId	203	Id папки документа

Рис. 193. Переменная в Users Defined Variables

После этого перейдем во вкладку **Body Data** запроса PostDoc и в качестве значения параметров **Folder_name** и **Entity.Folder.Id** укажем **\${folderId}**.

Итак, сэмплер PostDoc настроен. Теперь осталось параметризовать сформированный внутри него элемент HTTP Headers Manager. После выполнения тех же самых действий что и при параметризации HTTP Header Manager, расположенного внутри сэмплера Upload File, элемент стал выглядеть следующим образом (Рис. 194).

Name:	Value
Host	<code> \${server}:\${port}</code>
Connection	keep-alive
Content-Length	10173
Cache-Control	max-age=0
Origin	http://\${server}:\${port}
Upgrade-Insecure-Requests	1
Content-Type	multipart/form-data; boundary=----WebKitFormBoundarySFZr5pT7ecFnLqnq
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100...
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-...
Referer	<code> \${protocol}://\${server}:\${port}/Documents/Document/Create?gridUniqueName=\${GridUniqueName}-size=0...</code>
Accept-Encoding	gzip, deflate, br
Accept-Language	ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7

Рис. 194. Параметризованный элемент "HTTP Headers Manager" сэмплера "PostDoc"

Мы завершили настройку второй группы запросов, теперь необходимо запустить тест и проверить, что создание документа и его версии выполняется корректно.

Перед запуском теста добавим в конец контроллера CreateDoc элемент Debug Sampler для отслеживания значения переменных JMeter (Рис. 195).

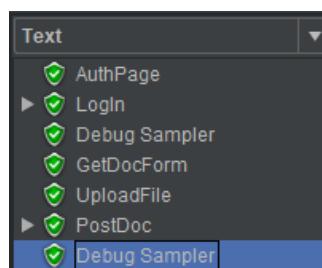


Рис. 195. Контроллер "CreateDoc". Элемент "Debug Sampler"



Нажимаем на кнопку запуска теста и переходим в элемент View Results Tree.

Все запросы выполнились без ошибок, но нам необходимо проверить, корректно ли выполнилась сама операция. Сначала перейдем в Debug Sampler и на вкладке **Response data** посмотрим результаты (Рис. 196).

```

Text Sampler result Request Response data
AuthToken=QMfhuAD26HFRZKFcvnMFN3uFekkR_H18do15l-VcTCBztBYLdhqkk7PnoSly0i3uH8L6RIU5daVo9aDnax1XDfn8ITgydNhPaPuaXMN6sx_JkCdm0-VptC3WtV5ve89
yE0rYXD5SHUfa2OX-GuSnazQt-xgsnPws001
DocToken=2YD8xvOBG_HpWP-B513gm3v32cV06a-N5NP8R6k8GLx-bl5PdICxGhAwDE5wnanvGDGpCoF8bz_Fpw3_1qgjH1HqP5j9x2CJeFkl4TsPnUEhc0vpoorNgAv6F
radmyT0FkUouPmdxtOPoR20WB6V_EHv5tSdC81
FileId=76fe06b7-752b-4106-8073-599e10d4b29f
FileId_g1
FileId_g0="uid":76fe06b7-752b-4106-8073-599e10d4b29f
FileId_g1=76fe06b7-752b-4106-8073-599e10d4b29f
GridUniqueName=ContractStepsGrid21e6bb9800847a492ac55fe79e807f
Id=1
JMeterThread.last_sample_ok=true
JMeterThread.pack=org.apache.jmeter.threads.SamplePackage@10a8b153
START.HMS=104432
START.MS=1561704272374
START.YMD=20190628
TESTSTART.MS=1561898446518
__jm__Thread Group__idx=0
__jm__USER_TOKEN__=Thread Group 1-1
docName=Document_rBsH
docTypeUid=5f17f751-2347-42d1-bb5c-a3b971173e54
folderId=203
login=admin
password=
port=1318
protocol=http
secDynamicFormsProviderData=5f17f751-2347-42d1-bb5c-a3b971173e54
secDynamicFormsProviderUid=e2daebba-2b80-4b71-a12d-f8c694dcc50f
secFormId=GlobalForm
secHTMLPrefix=Entity
secViewItemId=5f17f751-2347-42d1-bb5c-a3b971173e54:0
secViewProviderUid=76c8ee6e-f80c-4fd8-b085-312ff97def4
server=localhost
subject=Subject_cHrIg

```

Рис. 196. Debug Sampler. Вкладка "Response data"

Среди результатов ни одна из переменных, которые мы инициализируем в обработчиках запросов, не имеет значения **error** (установленного по умолчанию), а это значит, что использованные в них выражения извлекли совпадающие значения.

Чтобы окончательно убедиться в том, что сценарий выполнен успешно, откроем интересующие нас сэмплеры и посмотрим, те ли данные вернул нам сервер. Переходим на вкладку **Response data** запроса GetDocForm. Как можно заметить, в разметке, полученной в ответе страницы, имеется тег **title**, в котором в качестве создаваемого типа документа указан "Входящий договор". Это означает, что, отправив данный запрос, мы действительно оказались на форме создания нужного нам типа документа (Рис. 197).

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="ru-RU">
<head id="Head1" runat="server">
    <meta http-equiv="X-UA-Compatible" content="IE=Edge" />
    <meta content="text/html; charset=utf-8;" />
    <title>Новый документ &amp;lt;input type="checkbox" checked="" checked="" value="checked" /&gt; Входящий договор - ELMA</title>
    <link rel="icon" href="/Content/Images/favicon.ico" type="image/ico" />
    <link rel="shortcut icon" href="/Content/Images/favicon.ico" type="image/ico" />
    <!--[if (gte IE 9)|(IE)]><!-->
    <link href="/Content/ModernCheckBox.css" rel="stylesheet" type="text/css" />
    <!--<![endif]-->

    <link href="/Content/GetCssBCB45AD257DAD0639F155F91033D29B30085471C" rel="stylesheet" type="text/css" />
    <link href="/Content/Print.css" media="print" rel="stylesheet" type="text/css" />
```

Рис. 197. Результат выполнения сэмплера "GetDocForm". Вкладка "Response data".

Сопоставление создаваемого типа документа

Далее перейдем на вкладку **Response data** запроса UploadFile (Рис. 198). Мы видим, что сервер вернул нам статус **"success":true**. Это говорит о том, что загрузка файла-версии прошла успешно.

```
{"success":true,"uid":"78fe06b7-752b-4106-8073-599e10d4b29f","fileHref":"/SDK.Action/BinaryFiles/Download/78fe06b7-752b-4106-8073-599e10d4b29f","fileSize":9}
```

Рис. 198. Результат выполнения запроса "UploadFile". Вкладка "Response data". Анализ ответа сервера

И, наконец, перейдем в запрос PostDoc. В нем дублируется информация последнего выполненного из двух имеющихся в нем подзапросов – редиректа на страницу создания документа, запрашиваемую уже методом GET. Нам нужен первый подзапрос, который выполнялся методом Post и представляет собой непосредственно отправку формы создания документа. Перейдем в его вкладку **Response data**. В полученных данных ответа есть ссылка на карточку созданного в запросе документа, в конце которой находится Id созданного запросом документа (Рис. 199).

```
<html><head><title>Object moved</title></head><body>
<h2>Object moved to <a href="/Documents/Document/View/1222">here</a></h2>
</body></html>
```

Рис. 199. Ссылка на карточку созданного в запросе документа. Вкладка "Response data"

В браузере перейдем по данной ссылке, чтобы убедиться, что документ был создан корректно. Перейдя в карточку документа, мы видим, что все необходимые атрибуты документа были переданы корректно (Рис. 200).

Рис. 200. Карточка созданного в запросе документа

Примечание: при переходе в карточку созданного запросом документа может быть отображена страница с ошибкой (Рис. 201).

Рис. 201. Ошибка при переходе в карточку созданного запросом документа

При отображении ошибки необходимо выполнить следующее:

- раскрыть текст ошибки и найти объект, который ее вызывает;
- перейти на вкладку **Request** элемента View Results Tree, чтобы увидеть параметры, отправленные POST-запросом, настройкой которых мы занимались во вкладке **Body Data** (Рис. 202);

```

Text
AuthPage
Login
Debug Sampler
GetDocForm
GetDocFile
Uploadfile
PostDoc
http://localhost:1318/Documents/DocumentCreate
POST http://localhost:1318/Documents/DocumentCreate
POST data:
-----WebKitFormBoundarySFzr5pT7eFnLqng
Content-Disposition: form-data; name="__RequestVerificationToken"
OJuJTP6shJUCVsChbzEoIbf-SGVchUeC2wKPAUz7-p4xbQRImojWFDrvUcU0KYOCq0vhLy04qhwnuj2Sm18u1QWPu9ONt93py8SwGnTnZwYcbV4Cd2LzUsEa5qeMC91XgVJAN6hAvCFvLwMCyx6GyDslHe1or1AtbKhxRs1
-----WebKitFormBoundarySFzr5pT7eFnLqng
Content-Disposition: form-data; name="Entity.typeUid"
5f17751-2347-42d1-bb5c-a3b971173e54
-----WebKitFormBoundarySFzr5pT7eFnLqng
Content-Disposition: form-data; name="CurrentViewProvider"
00000000-0000-0000-0000-000000000000
-----WebKitFormBoundarySFzr5pT7eFnLqng
Content-Disposition: form-data; name="SaveToCurrentVersion"
False
-----WebKitFormBoundarySFzr5pT7eFnLqng
Content-Disposition: form-data; name="__DynamicFormSettings.FormId"
GlobalForm
-----WebKitFormBoundarySFzr5pT7eFnLqng
Content-Disposition: form-data; name="__DynamicFormSettings.HtmlPrefix"
Entity
-----WebKitFormBoundarySFzr5pT7eFnLqng
Content-Disposition: form-data; name="__DynamicFormSettings.DynamicFormsProviderUid"
e2dae6ba-2b80-4b71-a12d-f8c694dc50f
-----WebKitFormBoundarySFzr5pT7eFnLqng
Content-Disposition: form-data; name="__DynamicFormSettings.DynamicFormsProviderData"
5f17751-2347-42d1-bb5c-a3b971173e54

```

Рис. 202. Параметры, отправленные в POST-запросе. Элемент "View Results Tree". Вкладка "Request"

- в списке параметров найти тот, который указан в тексте ошибки, и проверить его значение. Если значение оказалось некорректным, проанализировать причины его появления.

После успешного выполнения операции создания документа переходим к отладке следующего этапа нашего сценария – отправки документа на ознакомление.

4.4.3. Отправка на ознакомление

Включаем контроллер CreateAcquaintanceTask (Рис. 203) и переходим в запрос GetSendToAcqForm (Рис. 204).

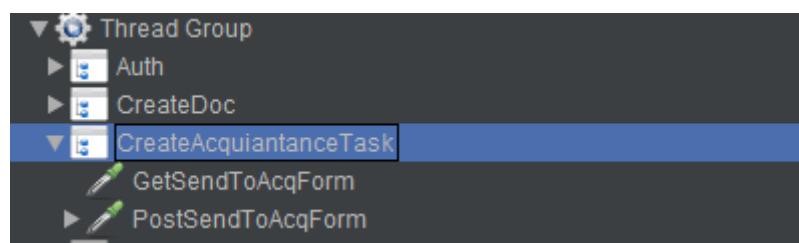


Рис. 203. Контроллер "CreateAcquaintanceTask"

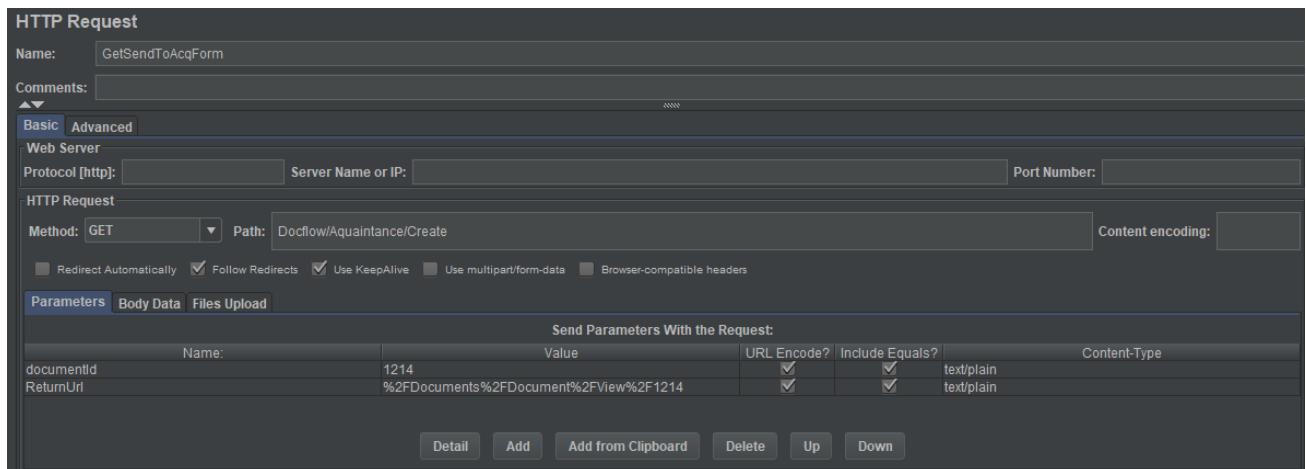


Рис. 204. Сэмплер "GetSendToAcqForm"

В строке запроса передается параметр **documentId** для отправки на ознакомление конкретного документа. После успешного выполнения запроса PostDoc происходит редирект на страницу созданного документа по ссылке:

```
<html><head><title>Object moved</title></head><body>
<h2>Object moved to <a href="/Documents/Document/View/1222">here</a>. </h2>
</body> </html>
```

Для дальнейшего выполнения сценария нам необходимо извлекать данный идентификатор из URL страницы и передавать его в качестве параметра запроса GetSendToAcqForm. Для этого в сэмплер PostDoc добавим пост-обработчик Regular Expression Extractor и переименуем его в DocumentId Extractor. Выполнив действия, описанные в разделе создание документа, мы нашли необходимое регулярное выражение:

```
\V(Documents\Document\View\)(\d+)
```

Далее настраиваем пост-обработчик (Рис. 205).

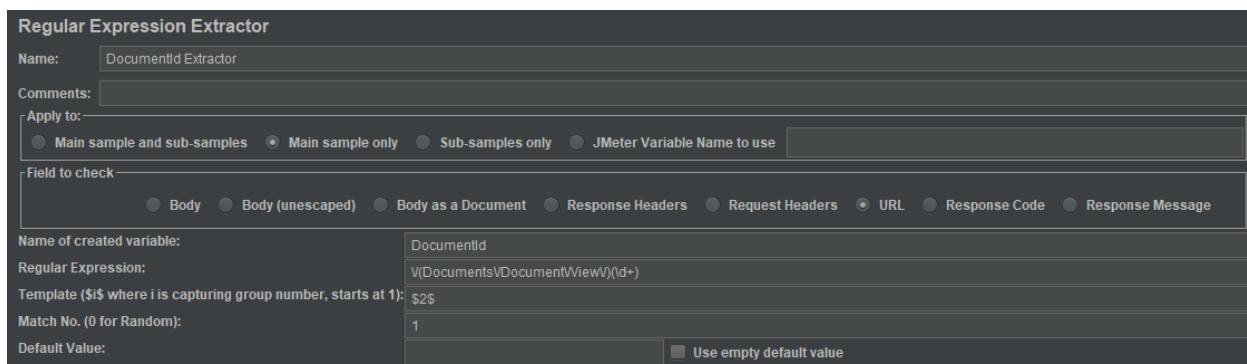


Рис. 205. Настойки пост-обработчика "DocumentId Extractor"

После настройки элемента переходим во вкладку **Parameters** запроса GetSendToAcqForm и в качестве значения параметра **documentId** указываем **\${DocumentId}** .

Скопируем созданный нами ранее пост-обработчик Token Extractor, который используется для извлечения токена верификации из HTML-кода запрашиваемой страницы, и вставим его внутрь сэмплера, изменив названием создаваемой в нем переменной на AcqToken (Рис. 206).

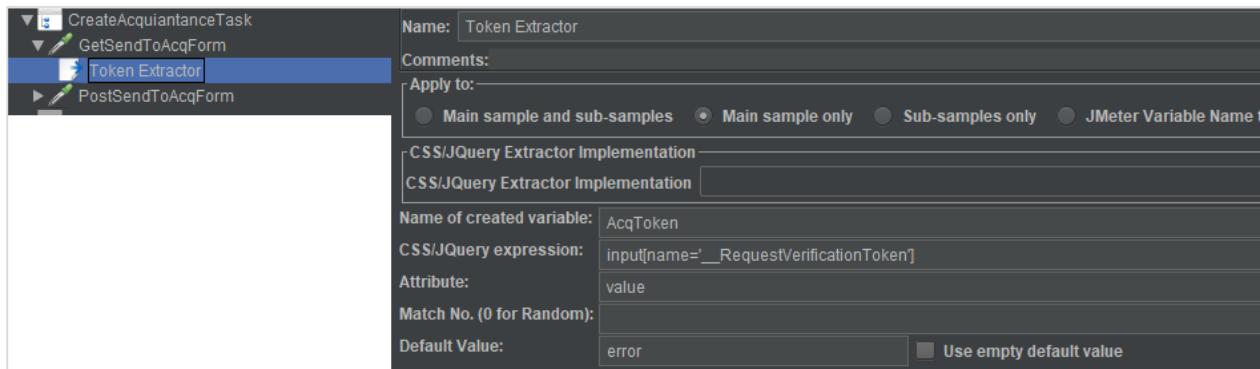


Рис. 206. Переменная "AcqToken" в пост-обработчике "Token Extractor"

Далее перейдем в POST-запрос отправки на ознакомление – PostSendToAcqForm.

Как и в случае с сэмплером [PostDoc](#), параметры данного запроса передаются во вкладке **Body Data** в виде текста, разделенного маркером границы **WebKitFormBoundary**. Удаляем из **Body Data** параметры с пустыми значениями вместе с относящимися к ним маркерами границы, а также параметры и их значения, содержащие в своем имени "json" или "Validate".

Рассмотрим оставшиеся параметры запроса (Рис. 207, Рис. 208, Рис. 209).

```

1 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
2 Content-Disposition: form-data; name="__RequestVerificationToken"
3
4 IFNn367GX6d3auiyMq7gL45LmjYqal0LUN8FL9PVexEAogZTAGwbnuJSu1yz5HDDmH4dFL6ykGRX48nnHiKyN-Qzq9H0_vrBFRI11r0Ua7TbL-
5 P3mSIpzUXL7S4J0atpzzcWu23AOV91IuT_UPqYvLfSt2Q0vrspDs1
6 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
7 Content-Disposition: form-data; name="ReturnUrl"
8
9 /Documents/Document/View/1214
10 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
11 Content-Disposition: form-data; name="SubTaskModel.SubTaskTypeUid"
12 00000000-0000-0000-0000-000000000000
13
14 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
15 Content-Disposition: form-data; name="SubTaskModel.ParentId"
16
17 0
18 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
19 Content-Disposition: form-data; name="SubTaskModel.CopySubject"
20
21 False
22 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
23 Content-Disposition: form-data; name="SubTaskModel.CopyPriority"
24
25 False
26 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
27 Content-Disposition: form-data; name="SubTaskModel.CopyDates"
28
29 False
30 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
31 Content-Disposition: form-data; name="SubTaskModel.CopyDescription"
32
33 False
34 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
35 Content-Disposition: form-data; name="SubTaskModel.AppendCommentsToDescription"
36
37 False
38 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
39 Content-Disposition: form-data; name="SubTaskModel.CopyAttachments"

```

Рис. 207. Параметры POST-запроса отправки на ознакомление

- **_RequestVerificationToken** – извлекаем его значение из HTML-кода страницы отправки документа на ознакомление;
- **ReturnUrl** – конце ссылки укажем переменную, хранящую в себе идентификатор документа;
- **SubTaskModel.SubTaskTypeUid** – оставляем без изменения;
- **SubTaskModel.ParentId** – оставляем без изменения;
- **SubTaskModel.CopySubject** – оставляем без изменения;
- **SubTaskModel.CopyPriority** – оставляем без изменения;
- **SubTaskModel.CopyDates** – оставляем без изменения;
- **SubTaskModel.CopyDescription** – оставляем без изменения;
- **SubTaskModel.AppendCommentsToDescription** – оставляем без изменения;

- **SubTaskModel.CopyAttachments** – оставляем без изменения;

```

40 False
41 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
42 Content-Disposition: form-data; name="SubTaskModel.CopyCrmEntity"
43
44 False
45 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
46 Content-Disposition: form-data; name="SubTaskModel.CopyDocuments"
47
48 False
49 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
50 Content-Disposition: form-data; name="SubTaskModel.GroupId"
51
52 0
53 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
54 Content-Disposition: form-data; name="SubTaskModel.RequireDigitalSignature"
55
56 False
57 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
58 Content-Disposition: form-data; name="SubTaskModel.CopyResolution"
59
60 False
61 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
62 Content-Disposition: form-data; name="SubTaskModel.CopyResult"
63
64 False
65 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
66 Content-Disposition: form-data; name="SubTaskModel.CopyPeriods"
67
68 False
69 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
70 Content-Disposition: form-data; name="SubTaskModel.CopyProject"
71
72 False
73 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
74 Content-Disposition: form-data; name="Entity.Subject"
75
76 Входящий договор с № от 21.06.2019
77
78
79

```

Рис. 208. Параметры POST-запроса отправки на ознакомление

- **SubTaskModel.CopyCrmEntity** – оставляем без изменения;
- **SubTaskModel.CopyDocuments** – оставляем без изменения;
- **SubTaskModel.GroupId** – оставляем без изменения;
- **SubTaskModel.RequireDigitalSignature** – оставляем без изменения;
- **SubTaskModel.CopyResolution** – оставляем без изменения;
- **SubTaskModel.CopyResult** – оставляем без изменения;
- **SubTaskModel.CopyPeriods** – оставляем без изменения;
- **SubTaskModel.CopyProject** – оставляем без изменения;
- **Entity.Subject** – укажем переменную, хранящую в себе название договора;

```

80 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
81 Content-Disposition: form-data; name="ComplexExecutor.Workers[0].PrefixedValue"
82
83 User.1
84 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
85 Content-Disposition: form-data; name="CheckReplacement"
86
87 True
88 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
89 Content-Disposition: form-data; name="Entity.ExecutionDate_date"
90
91 22.06.2019
92 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
93 Content-Disposition: form-data; name="Entity.ExecutionDateDamper"
94
95 false
96 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
97 Content-Disposition: form-data; name="Entity.ExecutionDate"
98
99 22.06.2019 23:59:50
100 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
101 Content-Disposition: form-data; name="ShowPlanWorkLog"
102
103 False
104 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
105 Content-Disposition: form-data; name="Entity.NotifyMe"
106
107 False
108 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
109 Content-Disposition: form-data; name="Entity.Items[0].Version.Id"
110
111 810
112 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6
113 Content-Disposition: form-data; name="Entity.Items[0].Document.Id"
114
115 1214
116 -----WebKitFormBoundaryuyMvQGEHkB0LbhU6 --

```

Рис. 209. Параметры POST-запроса отправки на ознакомление

- **ComplexExecutor.Workers[0].PrefixedValue** – укажем Id исполнителя задачи;
- **CheckReplacement** – оставляем без изменения;
- **Entity.ExecutionDate_date** – будем генерировать дату;
- **Entity.ExecutionDateDamper** – оставляем без изменения;
- **Entity.ExecutionDate** – будем генерировать дату и время;
- **ShowPlanWorkLog** – оставляем без изменения;
- **Entity.NotifyMe** – оставляем без изменения;
- **Entity.Items[0].Version.Id** – будем извлекать Id версии документа из HTML-кода страницы отправки на ознакомление;
- **Entity.Items[0].Document.Id** – укажем переменную, хранящую в себе Id документа.

После просмотра передаваемых параметров можно выделить следующие группы действий:

1. Извлечение значения параметра из HTML-кода страницы

[Ранее](#) в тестовый план мы добавили **Token Extractor** для извлечения **RequestVerificationToken**, значение которого передается в переменную

AcqToken. Соответственно, впишем **\${AcqToken}** в качестве значения передаваемого токена верификации (Рис. 210).

```

1 -----WebKitFormBoundaryuyMvQGEHkB0Lbhuj6
2 Content-Disposition: form-data; name="__RequestVerificationToken"
3
4 ${AcqToken}

```

Рис. 210. Значение передаваемого токена верификации

В качестве значения параметра **Entity.Items[0].Version.Id** необходимо указать Id версии отправляемого на ознакомление документа. Извлечь ее можно со страницы отправки на ознакомление с помощью пост-обработчика CSS/JQuery Extractor. Добавим CSS/JQuery Extractor в качестве вложенного элемента сэмплера GetSendToAcqForm и переименуем его в VersionId Extractor. Перед тем как настроить обработчик, найдем нужный элемент на самой странице:

- в браузере выполняем авторизацию в системе ELMA;
- переходим в карточку любого документа в системе и через меню кнопки отправляем его на ознакомление;
- переходим на страницу отправки документа на ознакомление;

Примечание: отправляемый на ознакомление документ должен иметь версию;

- открываем консоль браузера и переходим на вкладку **Elements**;
- в окне поиска элемента вводим "Version.Id" (Рис. 211).

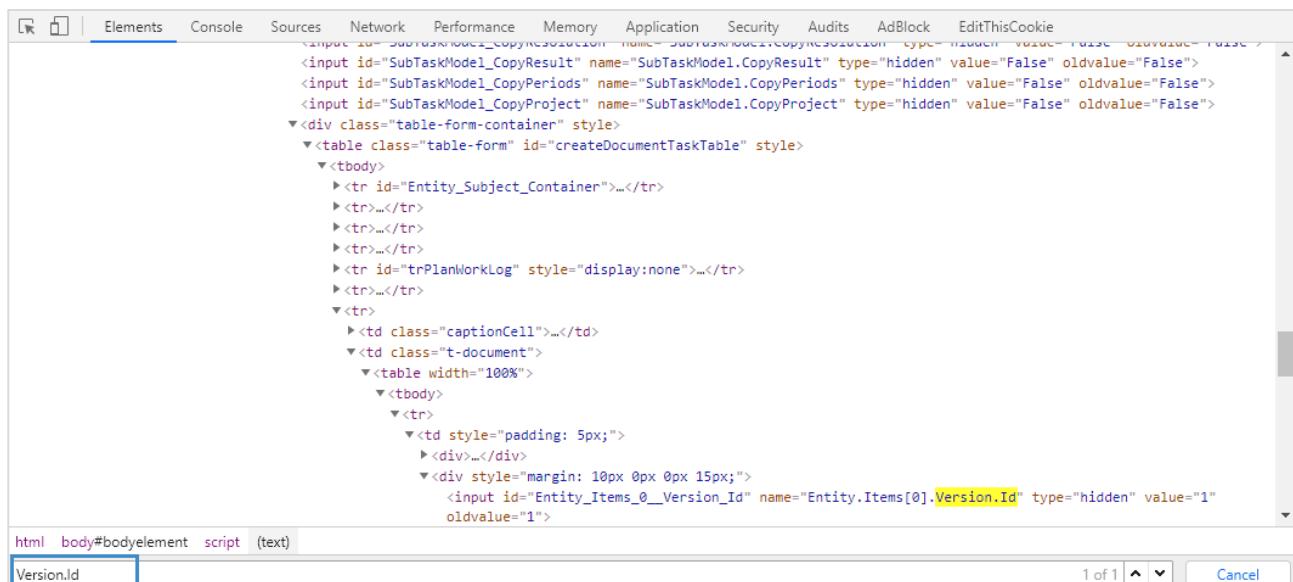


Рис. 211. Поиск элемента. Консоль браузера. Вкладка "Elements"

Искомый нами элемент подсветился. Для того чтобы получить Id версии, нам необходимо извлечь значение атрибута **value**. Для этого настроим CSS/JQuery Extractor следующим образом (Рис. 212).

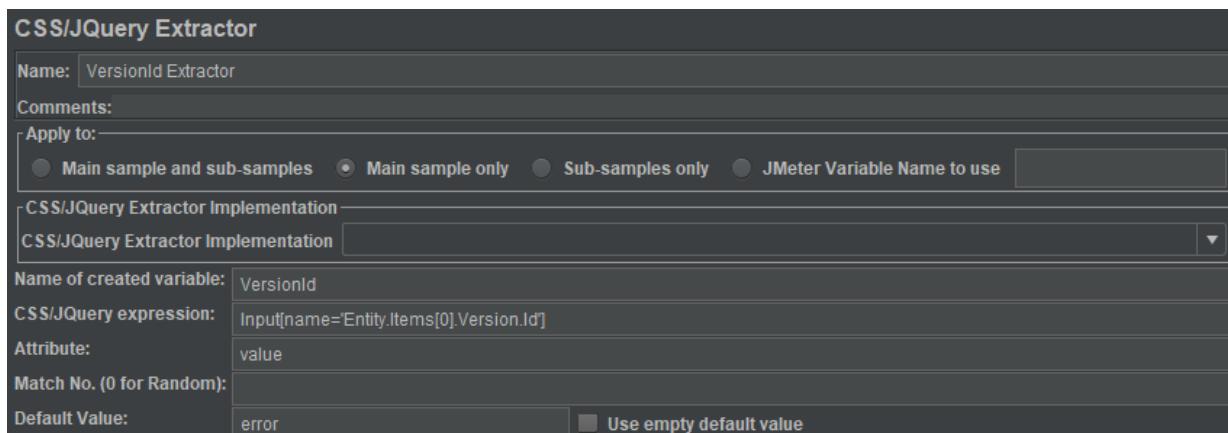


Рис. 212. Настройка CSS/JQuery Extractor

Настроив обработчик, переходим во вкладку **Body Data** запроса PostSendToAcqForm и в качестве значения параметра **Entity.Items[0].Version.Id** указываем **\${VersionId}**.

2. Генерация значений переменных

Согласно нашему плану, сроком ознакомления устанавливается следующий после создания задачи день. Соответственно, в нашем сценарии мы смоделируем данное поведение, вычисляя срок ознакомления по формуле: **текущая дата + 1 день**. Для этого будем использовать сценарий, написанный в элементе [JSR223PreProcessor](#). Выбор препроцессора в данном случае обусловлен тем, что генерация данных должна производиться непосредственно перед выполнением самого запроса. Добавим JSR223PreProcessor в качестве вложенного элемента сэмплера PostSendToAcqForm и переименуем его в GenerateExecutionDate.

Далее перейдем в настройки элемента. В качестве языка сценария выберем Groovy. В поле **Script** введем следующий код:

```
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

SimpleDateFormat sdf = new SimpleDateFormat ("dd.MM.yyyy");
SimpleDateFormat sdf2 = new SimpleDateFormat ("dd.MM.yyyy H:mm:ss");
Calendar cal = Calendar.getInstance();
cal.add(Calendar.DATE, 1);
vars.put("executionDate",sdf.format(cal.getTime()).toString());
vars.put("executionTime", sdf2.format(cal.getTime()).toString());
```

С помощью метода **vars.put** мы создаем переменную Jmeter, к которой впоследствии можно будет обратиться в teste, и сохраняем в ней необходимое значение.

После настройки элемента переходим во вкладку **Body Data** запроса PostSendToAcqForm и в качестве значения параметра **Entity.ExecutionDate_date** указываем **\${executionDate}**, а в качестве значения параметра **Entity.ExecutionDate** устанавливаем **\${executionTime}**.

3. Использование значений из объявленных переменных JMeter

На протяжении всего тестового сценария извлекаемые с помощью пост и пред-обработчиков запросов значения можно записывать в переменные Jmeter и использовать эти переменные в последующих после их инициализации элементах. Id документа, значение которого необходимо передать в параметры **ReturnUrl** и **Entity.Items[0].Document.Id**, мы извлекаем из URL получаемой после редиректа страницы и записываем в переменную **DocumentId**, название документа для параметра **Entity.Subject** хранится в переменной **docName**, которая была инициализирована в пост-процессоре **GenerateRandomStrings**, а Id исполнителя для параметра **ComplexExecutor.Workers[0].PrefixedValue** мы возьмем из переменной Id, инициализируемой в CSV Data Set Config.

Таким образом, после параметризации значения параметров **ReturnUrl**, **Entity.Subject**, **Entity.Items[0].Document.Id** и **ComplexExecutor.Workers[0].PrefixedValue** будут выглядеть следующим образом (Рис. 213).

```
-----WebKitFormBoundaryuyMvQGEHkB0Lbhui6
Content-Disposition: form-data; name="ReturnUrl"
/Products/Document/View/${DocumentId}
-----WebKitFormBoundaryuyMvQGEHkB0Lbhui6
Content-Disposition: form-data; name="Entity.Subject"
${docName}

-----WebKitFormBoundaryuyMvQGEHkB0Lbhui6
Content-Disposition: form-data; name="ComplexExecutor.Workers[0].PrefixedValue"
User.${Id}
-----WebKitFormBoundaryuyMvQGEHkB0Lbhui6
Content-Disposition: form-data; name="Entity.Items[0].Document.Id"
${DocumentId}
```

Рис. 213. Значения параметров

После завершения настройки сэмплера PostSendToAcqForm параметризуем заголовки в сформированном внутри него элементе HTTP Headers Manager (Рис. 214).

Headers Stored in the Header Manager	
Name:	Value
Host	<code> \${server}:\${port}</code>
Connection	keep-alive
Content-Length	5949
Cache-Control	max-age=0
Origin	<code> \${protocol}://\${server}:\${port}</code>
Upgrade-Insecure-Requests	1
Content-Type	multipart/form-data; boundary=----WebKitFormBoundaryuyMvQGEHkB0Lbhuh6
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.37...
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/so...
Referer	<code> \${protocol}://\${server}:\${port}/Docflow/Acquaintance/Create?documentId=\${DocumentId}&ReturnUrl=...</code>
Accept-Encoding	gzip, deflate, br
Accept-Language	ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7

Рис. 214. Заголовки запроса в элементе "HTTP Headers Manager"

Настройка третьей группы запросов завершена, теперь необходимо запустить тест и проверить, что отправка документа на ознакомление выполняется корректно. Перед запуском теста добавим в конец контроллера CreateAcquaintanceTask элемент Debug Sampler для отслеживания значения переменных JMeter.



Нажимаем на кнопку запуска теста и переходим в элемент View Results Tree (Рис. 215).

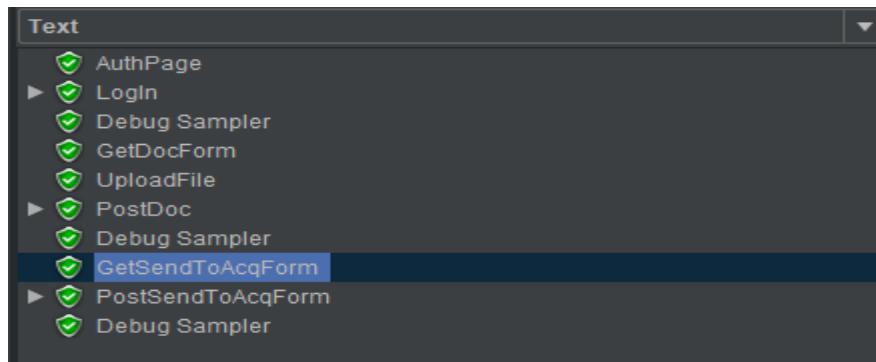


Рис. 215. GetSendToAcqForm

Все запросы выполнились без ошибок. Проверим, корректно ли выполнилась сама операция отправки на ознакомление. Для начала перейдем во вкладку **Response Data** элемента Debug Sampler (Рис. 216).

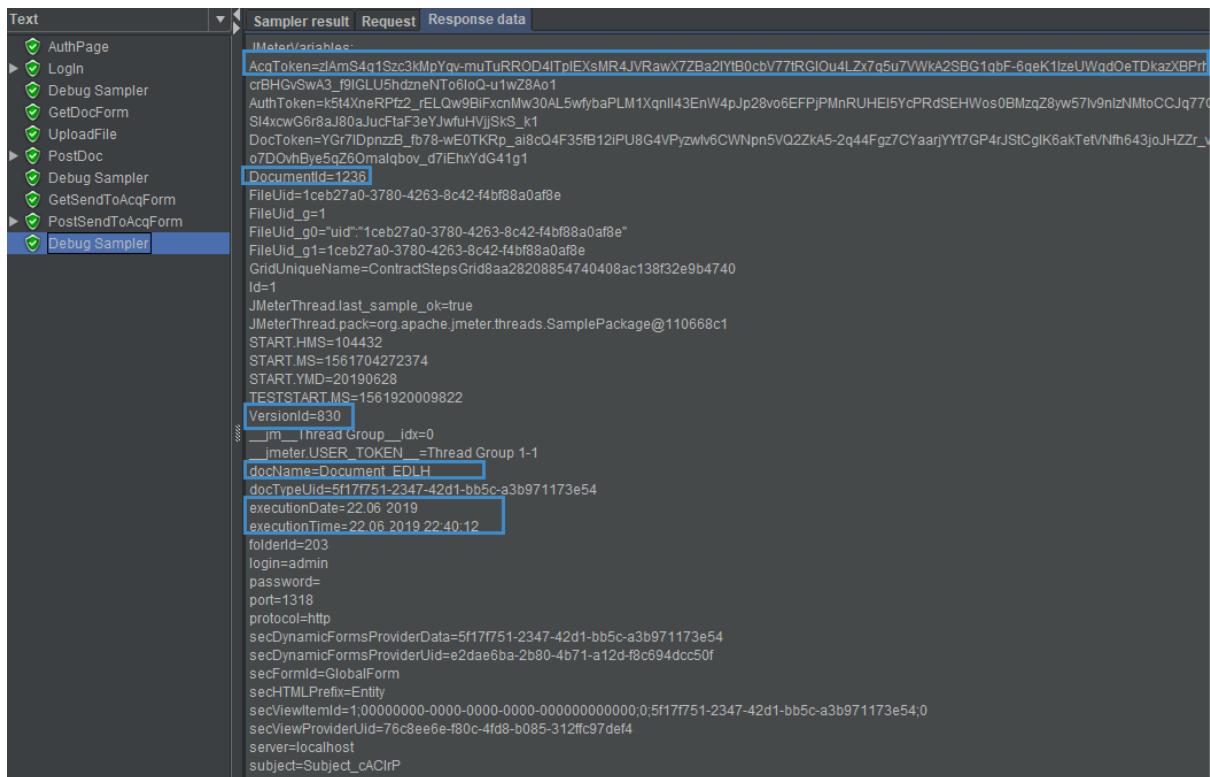


Рис. 216. Элемент "Debug Sampler". Вкладка "Response Data"

В результатах Debug Sampler участвующим в операции переменным были присвоены корректные значения. Чтобы окончательно убедиться в успешном выполнении операции, откроем сэмплеры GetSendToAcqForm и PostSendToAcqForm и посмотрим полученные от сервера данные. Перейдем на вкладку **Response data** запроса GetSendToAcqForm и в этот раз вместо текста выберем режим отображения **HTML**. Несмотря на то, что в данном режиме страница будет отображаться без подгруженных стилей, того, что там имеется, достаточно, чтобы понять, что происходило на странице в тот момент времени (Рис. 217).

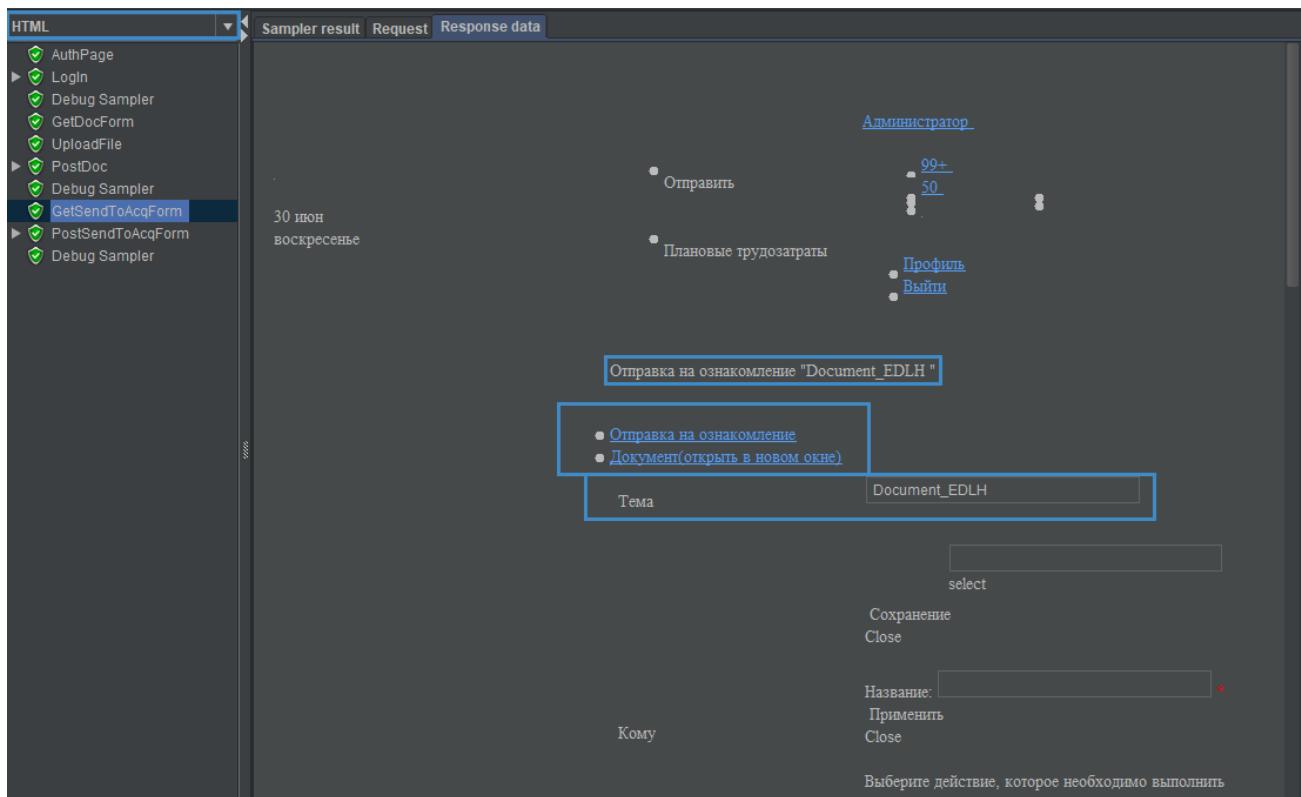


Рис. 217. Отображение результатов выполнения сэмплера "GetSendToAcqForm" в режиме HTML. Вкладка "Response data"

Из-за наличия специфических элементов разметки страницы, можно с уверенностью сказать, что мы находимся на странице отправки документа на ознакомление, а в теме задачи указано название документа, сформированное в текущем teste, чему свидетельствует значение переменной **docName** в [результатах](#) Debug Sampler.

Переключим режим отображения обратно на текст и перейдем в запрос PostSendAcq. В нем дублируется информация последнего выполненного из двух имеющихся в нем подзапросов – редиректа на страницу создания документа, запрашиваемую уже методом GET. Рассмотрим первый подзапрос, который выполнялся методом Post и представляет собой непосредственно отправку формы создания документа. На вкладке **Request** (Рис. 218) можно увидеть строку отправленного POST-запроса, а также его параметры, некоторые из которых мы параметризовали на вкладке **Body Data** в Jmeter.

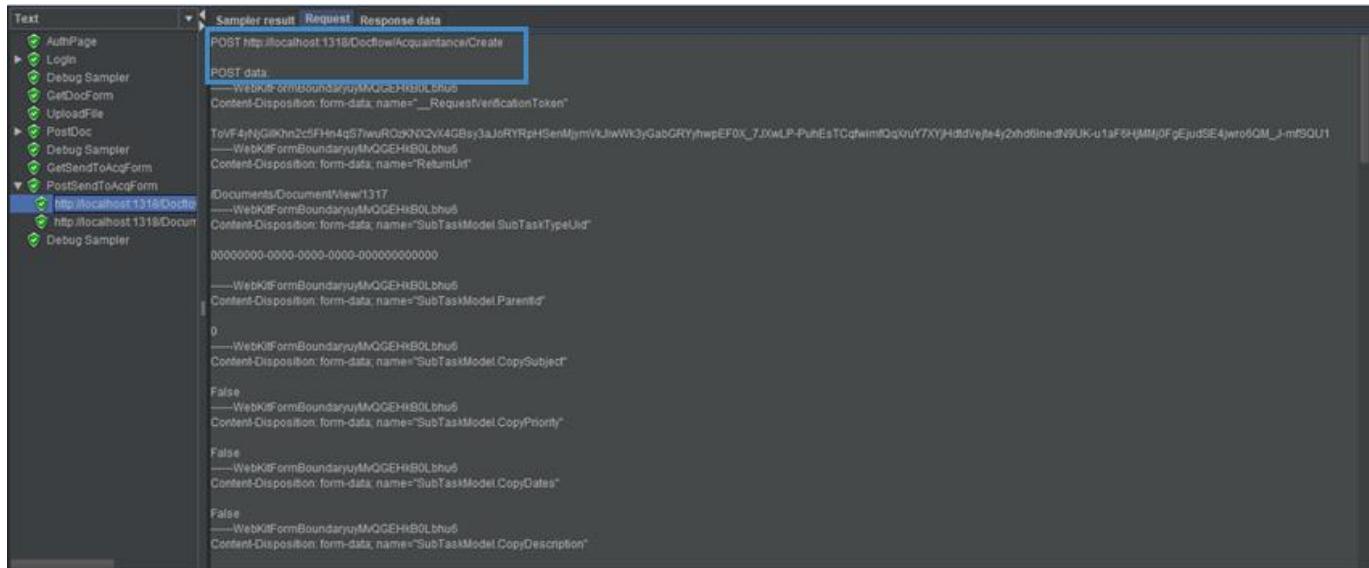


Рис. 218. Подзапрос сэмплера PostSendAcq. Вкладка "Request"

Перейдем во вкладку **Request** сэмплера PostDoc (Рис. 219) и скопируем оттуда ссылку на созданный в запросе документ.

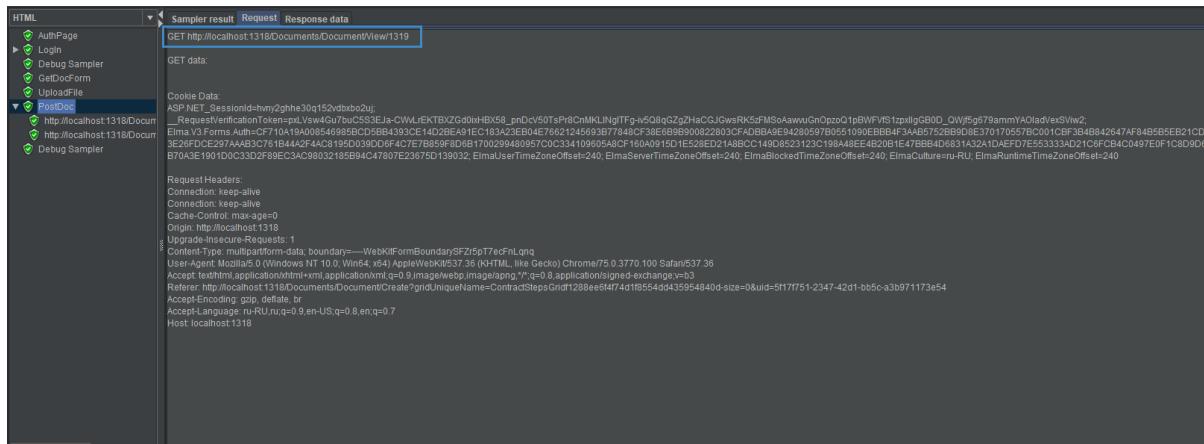


Рис. 219. Сэмплер "PostDoc". Вкладка "Request"

Вставим данную ссылку в строку браузера для перехода в карточку документа, чтобы убедиться, что на ней отображается связанная с ним задача ознакомления.

На вкладке **Задачи** (Рис. 220) имеется созданная нами задача ознакомления, в которую были корректно переданы все необходимые атрибуты.

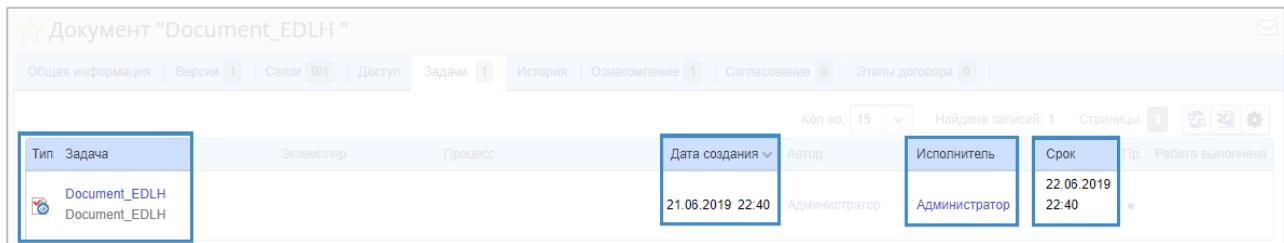


Рис. 220. Карточка документа. Вкладка "Задачи"

Таким образом, третья группа запросов была выполнена успешно. Теперь переходим к отладке следующего этапа нашего сценария – ознакомления с документом.

4.4.4. Ознакомление с документом

Включаем контроллер ExecuteAcqTask (Рис. 221) и переходим в сэмплер GetAcqForm (Рис. 222).

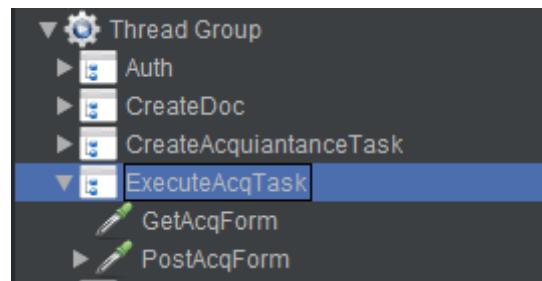


Рис. 221. Контроллер "ExecuteAcqTask"

Рис. 222. Сэмплер "GetAcqForm"

Цифры в строке запроса – это Id задачи ознакомления. Поскольку в тестовом сценарии мы будем создавать каждый раз новую задачу ознакомления, значение идентификатора данной задачи для последующего получения ее формы нам необходимо извлекать из разметки страницы

текущих задач по документу и передавать в строку запроса GetAcqForm. Для начала запишем запрос на получение страницы текущих задач по документу:

1. В браузере выполняем авторизацию в системе ELMA.
2. Переходим в карточку созданного документа.
3. Открываем консоль браузера и переходим на вкладку **Elements** (**Инспектор** в Mozilla Firefox).
4. Наводим инструмент исследования элементов на вкладку **Задачи** и нажимаем левую кнопку мыши, чтобы его зафиксировать (Рис. 223).

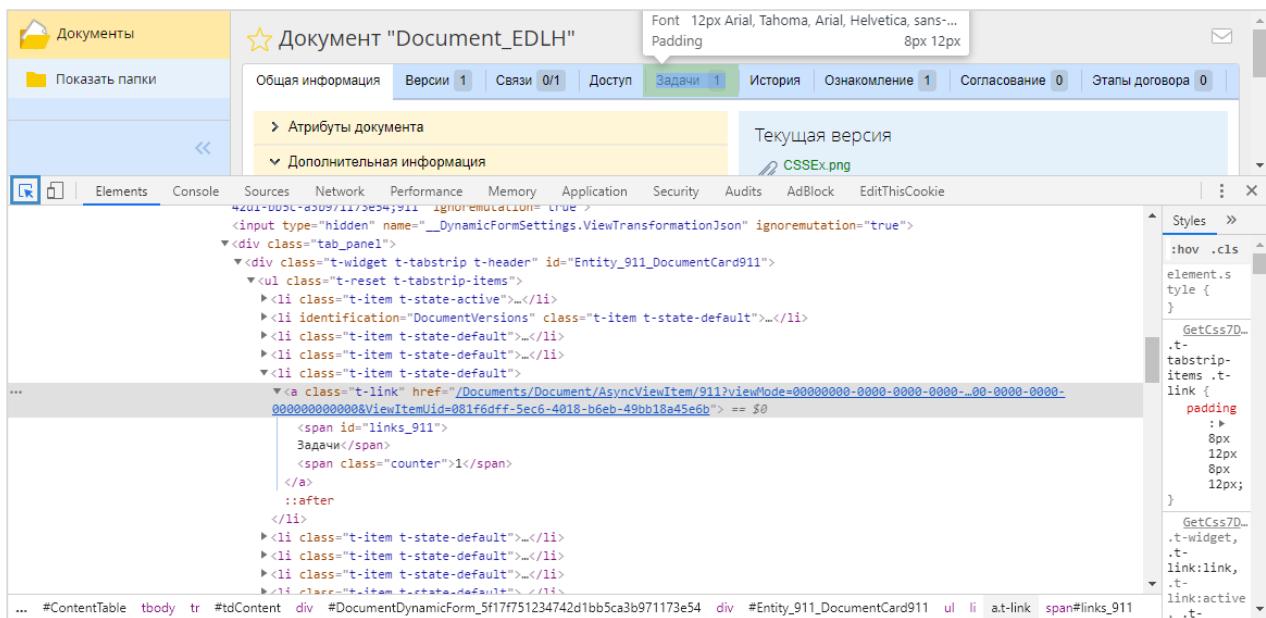


Рис. 223. Карточка документа. Консоль браузера. Вкладка "Elements"

Выделенная строка HTML-кода и есть непосредственно ссылка на страницу текущих задач по документу (Рис. 224).

```
<a class="t-link" href="/Documents/Document/AsyncViewItem/911?viewMode=00000000-0000-0000-0000-000000000000&ViewItemUid=081f6dff-5ec6-4018-b6eb-49bb18a45e6b"> = $0
```

Рис. 224. Ссылка на страницу текущих задач по документу

5. Переходим по этой ссылке, предварительно запустив прокси в элементе HTTP(s) Test Script Recorder или включив запись трафика в Fiddler. Как можно заметить, в разметке страницы имеется ссылка на интересующую нас задачу (Рис. 225).

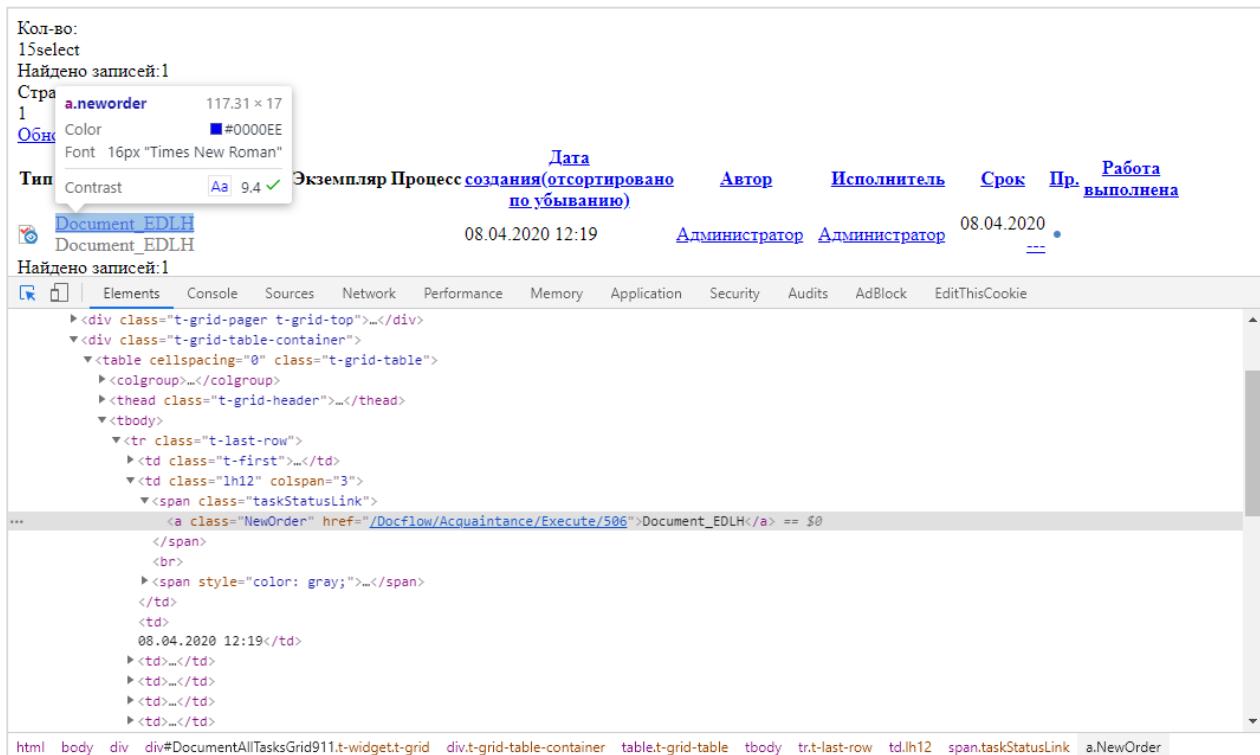


Рис. 225. Ссылка на задачу

Нас интересует запрос, имеющий "GET /Documents/Document/AsyncViewItem" в своем URL. Переходим в записанный в JMeter сэмплер (при использовании Fiddler после записи запроса необходимо воспользоваться процедурой его [импорта](#) в Jmeter) и переименовываем его в GetCurrentTask.

В строке запроса в поле **Path** необходимо будет параметризовать Id документа, страницу текущих задач по которому нам необходимо получить. Значение идентификатора документа в нашем сценарии извлекается пост-обработчиком [DocumentId Extractor](#) и записывается в переменную **DocumentId** (Рис. 226).

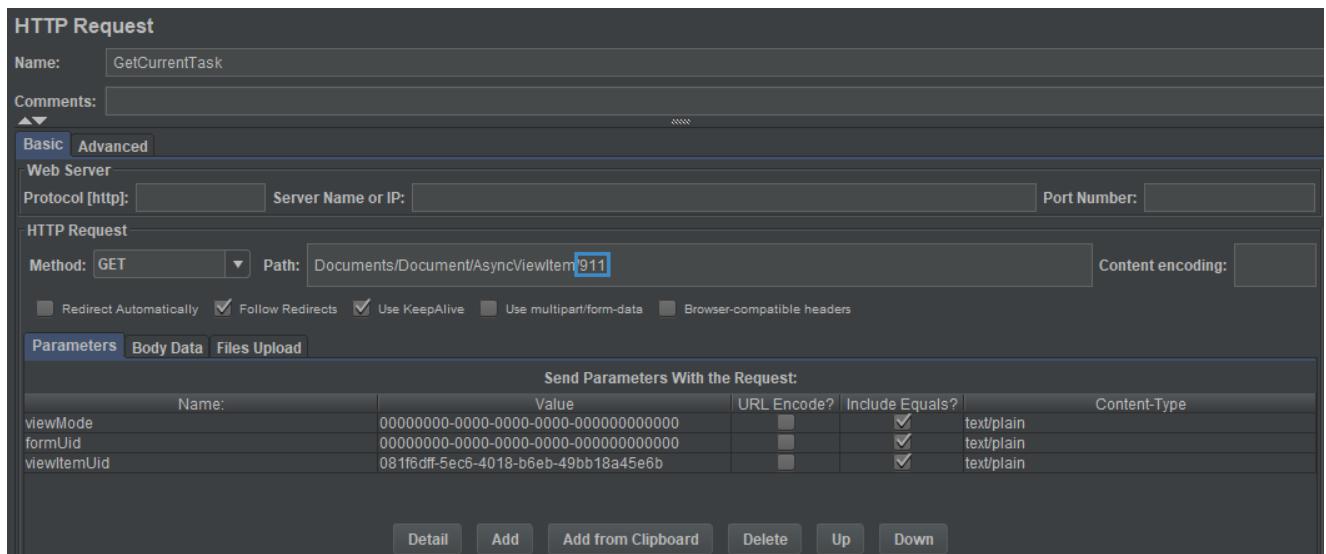


Рис. 226. Сэмплер "GetCurrentTask". Параметризация Id документа в поле "Path".

Впишем переменную **DocumentId** в поле **Path** (Рис. 227).

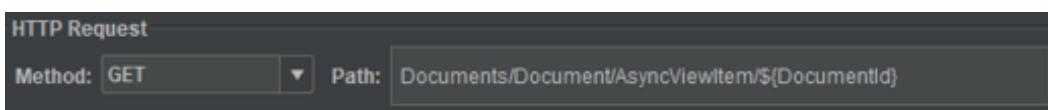


Рис. 227. Поле "Path". Переменная "DocumentId"

Передаваемые во вкладке **Parameters** параметры оставим без изменений. Также удалим вложенный элемент сэмплера HTTP Request Manager, поскольку здесь он нам не нужен.

Добавим пост-обработчик Regular Expression Extractor в качестве вложенного элемента сэмплера для извлечения Id задачи ознакомления и переименуем его в TaskId Extractor. Выполнив описанные ранее [действия](#), мы нашли следующее регулярное выражение для извлечения Id из строки ``:

```
\V(Docflow\Acquaintance\Execute\)(\d+)
```

Далее настраиваем пост-обработчик (Рис. 228).

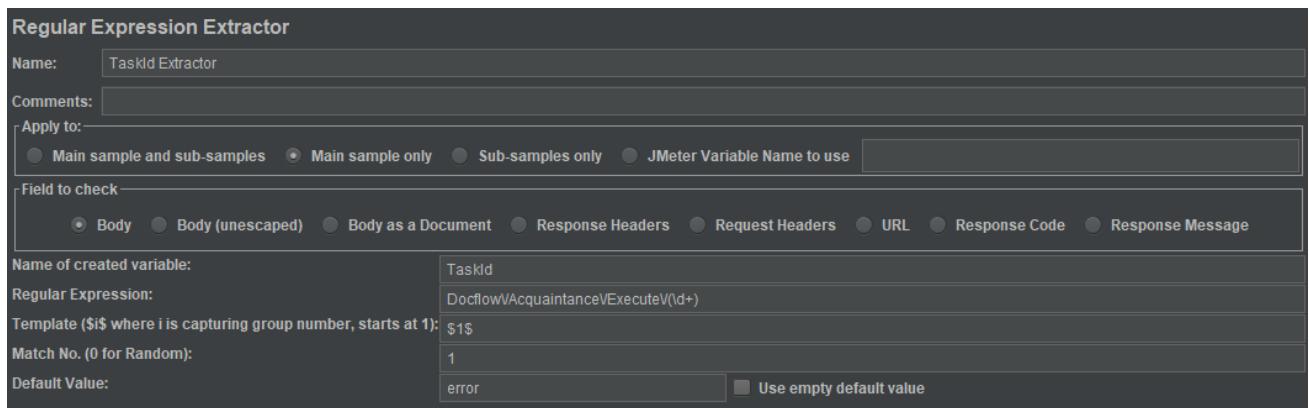


Рис. 228. Пост-обработчик "Regular Expression Extractor"

При высокой нагрузке на систему в момент отправки JMeter запроса на получение страницы текущих задач документа, данные на ней могут не успеть обновиться, и, соответственно, обработчик не найдет элемент по указанному в нем выражению и установит **error** в качестве значения переменной **TaskId**, что нарушит выполнение тестового сценария. Чтобы предупредить такой ход развития событий, воспользуемся элементом While Controller, который будет переотправлять запрос на получение страницы текущих задач по документу до тех пор, пока обработчик [TaskId Extractor](#) не извлечет корректный идентификатор задачи из ее разметки.

Расположим While Controller между контроллерами CreateAcquaintanceTask и ExecuteAcqTask и поместим в него сэмплер [GetCurrentTask](#) (Рис. 229).

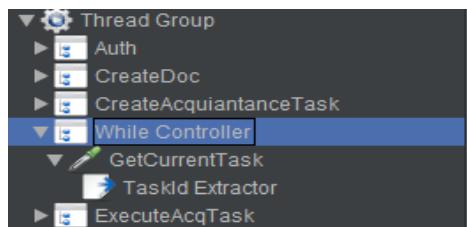


Рис. 229. While Controller

В нашем teste число повторений, размещенных в While Controller элементов, будет 5 раз. Добиться этого поможет элемент конфигурации [Counter](#). Добавим его в While Controller и настроим следующим образом (Рис. 230).

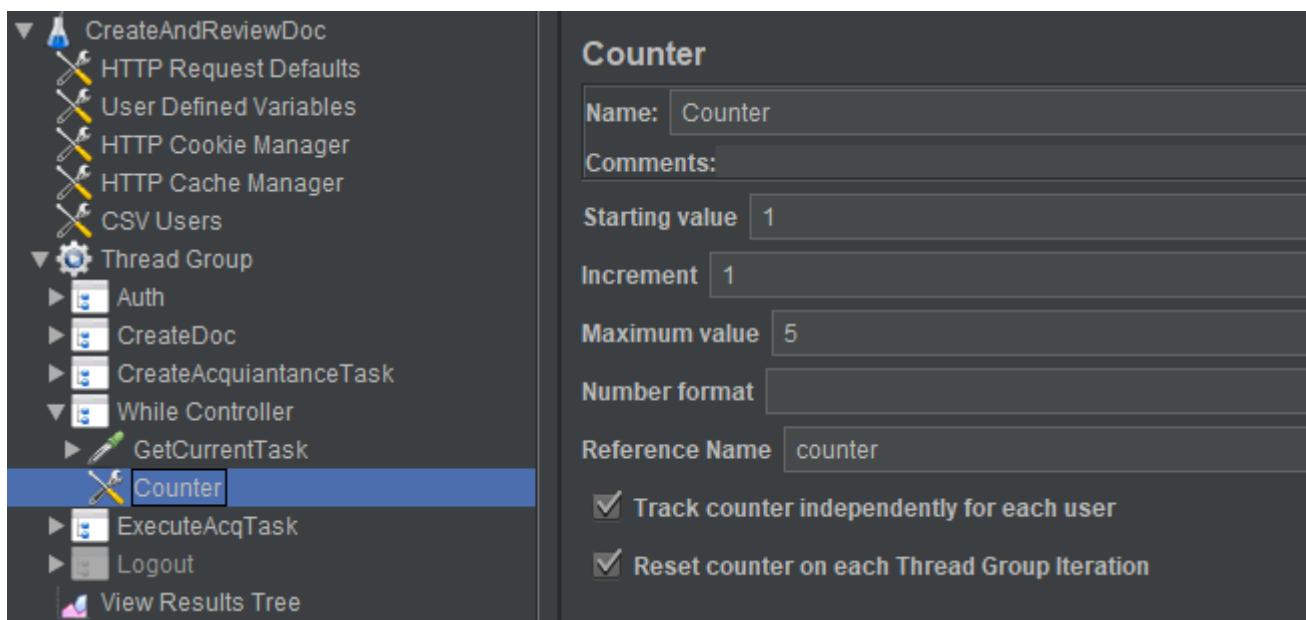


Рис. 230. Настройка элемента "Counter"

- в поле **Starting value** в качестве начального значения числа укажем 1;
- в поле **Increment** укажем увеличение на 1;
- в поле **Maximum value** укажем 5, как максимальное значение counter;
- в поле **Reference Name** в качестве переменной, в которую будет записываться числовое значение укажем "counter".

Также в настройках элемента установим два флагка, чтобы Counter высчитывался независимо для каждого виртуального пользователя и его значение очищалось при старте каждой новой итерации тестирования.

Далее перейдем в элемент User Defined Variables и добавим переменную с точно таким же названием, которое мы указали в настройках Counter, в нашем случае – "counter" и установим ей значение "0". Также добавим переменную с названием **TaskId** и установим ей значение "error".

Перейдем в While Controller и установим условие, при котором будет повторяться выполнение расположенных внутри контроллера элементов. В противном случае, тест пойдет дальше:

```
 ${__javaScript("${TaskId}" == "error" && ${counter} < 5,)}
```

После того как все настроено, перейдем в сэмплер GetAcqForm и в поле **Path** в качестве Id задачи ознакомления впишем **\${TaskId}** (Рис. 231).

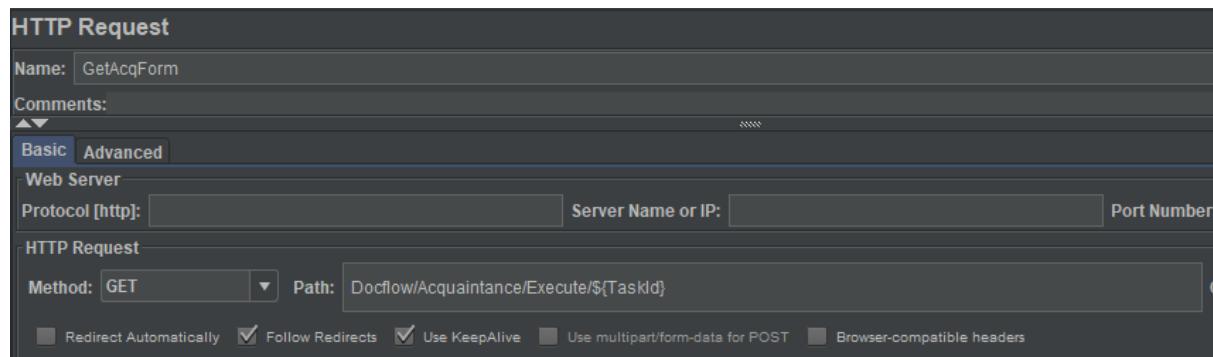


Рис. 231. Сэмплер "GetAcqForm"

Скопируем созданный нами ранее пост-обработчик Token Extractor, который используется для извлечения токена верификации из HTML-кода запрашиваемой страницы, и вставим его внутрь сэмплера, изменив названием создаваемой в нем переменной на **TaskToken**.

Далее перейдем в POST-запрос выполнения задачи ознакомления – PostAcqForm (Рис. 232).

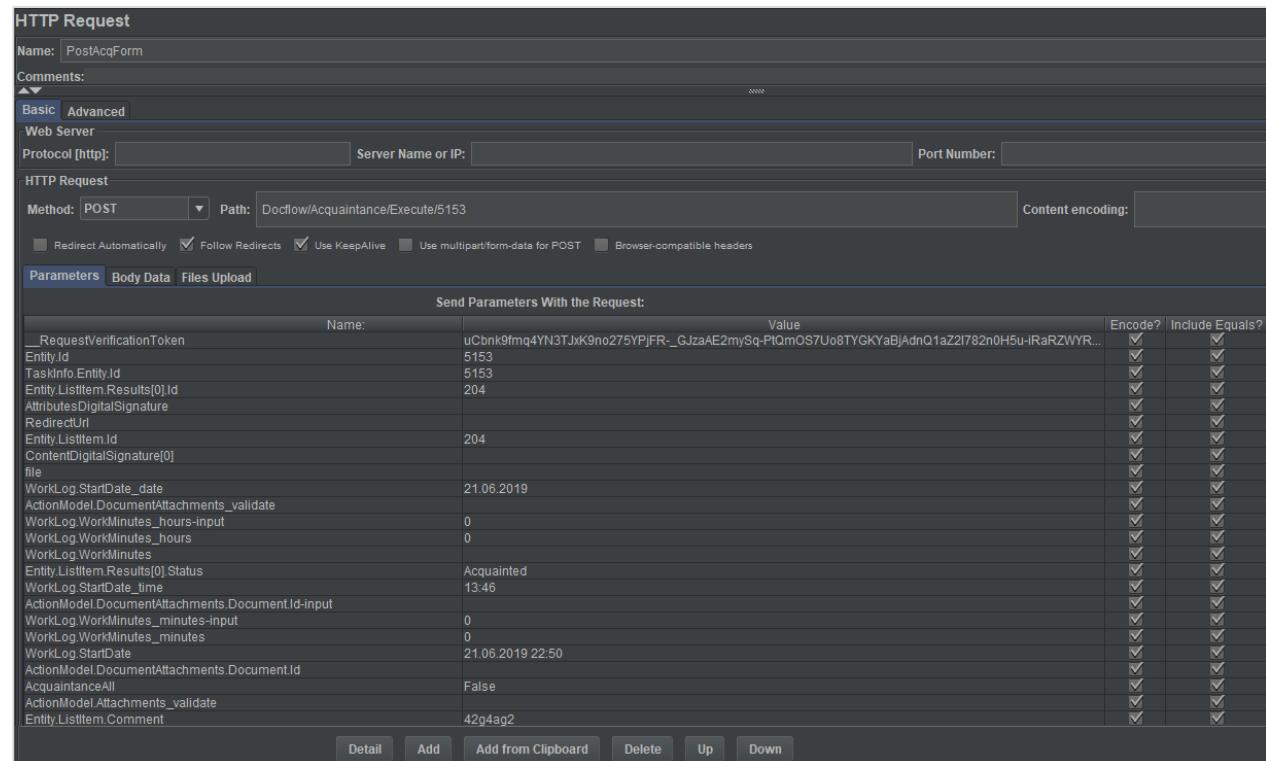


Рис. 232. Сэмплер "PostAcqForm"

Аналогично запросу GetAcqForm в поле **Path** установим переменную **\${TaskId}** в качестве значения Id задачи ознакомления.

В этот раз параметры запроса передаются во вкладке **Parameters** и представлены в виде таблицы. Удаляем с помощью расположенной ниже

кнопки **Delete** параметры с пустыми значениями, а также параметры, содержащие в своем имени "json" или "Validate".

Рассмотрим оставшиеся параметры запроса (Рис. 233).

Name:	Value
_RequestVerificationToken	uCbnk9fmq4YN3TJxK9no275YPjFR-_GJzaAE2mySq
Entity.Id	5153
TaskInfo.Entity.Id	5153
Entity.ListItem.Results[0].Id	204
Entity.ListItem.Id	204
WorkLog.StartDate_date	21.06.2019
WorkLog.WorkMinutes_hours-input	0
WorkLog.WorkMinutes_hours	0
Entity.ListItem.Results[0].Status	Acquainted
WorkLog.StartDate_time	13:46
WorkLog.WorkMinutes_minutes-input	0
WorkLog.WorkMinutes_minutes	0
WorkLog.StartDate	21.06.2019 22:50
AcquaintanceAll	False
Entity.ListItem.Comment	42g4ag2

Рис. 233. Параметры запроса "PostAcqForm"

- **_RequestVerificationToken** – укажем переменную, хранящую в себе токен верификации;
- **Entity.Id** – укажем переменную, хранящую в себе идентификатор задачи ознакомления;
- **TaskInfo.Entity.Id** – укажем переменную, хранящую в себе идентификатор задачи ознакомления;
- **Entity.ListItem.Results[0].Id** – будем извлекать значение из HTML-кода страницы задачи ознакомления;
- **Entity.ListItem.Id** – будем извлекать значение из HTML-кода страницы задачи ознакомления;
- **WorkLog.StartDate_date** – будем генерировать текущую дату;
- **WorkLog.WorkMinutes_hours-input** – оставляем без изменения;
- **WorkLog.WorkMinutes_hours** – оставляем без изменения;
- **Entity.ListItem.Results[0].Status** – Acquainted;
- **WorkLog.StartDate_time** – будем генерировать текущее время;
- **WorkLog.WorkMinutes_minutes-input** – оставляем без изменения;
- **WorkLog.WorkMinutes_minutes** – оставляем без изменения;
- **WorkLog.StartDate** – будем генерировать текущую дату и время;
- **AcquaintanceAll** – оставляем без изменения;
- **Entity.ListItem.Comment** – будем генерировать случайный комментарий.

После просмотра всех параметров выделяем следующие группы действий:

1. Извлечение значения параметра из HTML-кода страницы.

RequestVerificationToken, **Entity.Id** и **TaskInfo.Entity.Id** в нашем сценарии уже извлекаются с помощью пост-обработчиков запросов, и их значения передаются в переменные **TaskToken** и **TaskId** соответственно. Впишем **\${DocToken}** в качестве значения передаваемого токена верификации, а **\${TaskId}** в качестве значения для параметров **EntityId** и **TaskInfo.Entity.Id** (Рис. 234).

Name:	Value
__RequestVerificationToken	\${TaskToken}
Entity.Id	\${TaskId}
TaskInfo.Entity.Id	\${TaskId}

Рис. 234. Параметры RequestVerificationToken, Entity.Id и TaskInfo.Entity.Id

В качестве значения параметров **Entity.ListItem.Results[0].Id** и **Entity.ListItem.Id** необходимо указать **ListItemId**, который можно извлечь со страницы задачи ознакомления с помощью пост-обработчика CSS/JQuery Extractor. Добавим CSS/JQuery Extractor в качестве вложенного элемента сэмплера GetAcqForm и переименуем его в ListItemId Extractor. Перед тем как настроить обработчик, найдем нужный элемент на самой странице:

- в браузере выполняем авторизацию в системе ELMA;
- переходим в карточку документа и в верхнем меню через меню кнопки **Отправка** отправляем документ на ознакомление;
- переходим в задачу ознакомления;
- открываем консоль браузера и переходим на вкладку **Elements**;
- в окне поиска элемента вводим "Entity.ListItem.Id" (Рис. 235).



Рис. 235. Поиск элемента "Entity.ListItem.Id". Консоль браузера. Вкладка "Elements"

Извлечем значение атрибута **value** найденного элемента. Для этого настроим CSS/JQuery Extractor следующим образом (Рис. 236).

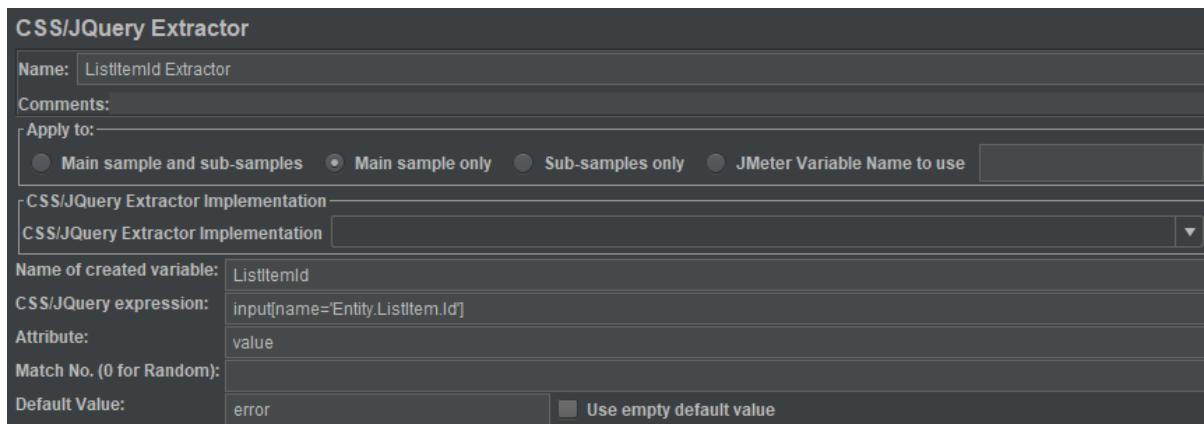


Рис. 236. Настройки CSS/JQuery Extractor

Настроив обработчик, переходим во вкладку **Parameters** запроса PostAcqForm и в качестве значения параметров **Entity.ListItem.Results[0].Id** и **Entity.ListItem.Id** указываем **\${ListItemId}**.

2. Генерация значений переменных.

В отправляемом нами запросе каждый раз будет генерироваться случайный комментарий ознакомления, который отправляется в параметре запроса **Entity.ListItem.Comment**, тем самым имитируя реальное поведение пользователей. Добиться этого нам поможет сценарий, написанный в элементе [JSR223PreProcessor](#). Выбор препроцессора в данном случае обусловлен тем, что генерация данных должна производиться непосредственно перед выполнением самого запроса.

Добавим JSR223PreProcessor в качестве вложенного элемента сэмплера PostAcqForm и переименуем его в GenerateComment. Далее перейдем в настройки элемента. В качестве языка сценария выберем Groovy. В поле **Script** введем следующий код:

```
import java.util.Random;
import java.lang.StringBuilder;
import java.text.SimpleDateFormat;
String generateRandom(int quantity, Random random)
{
    StringBuilder builder = new StringBuilder();
    String mCHAR = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
    for (int i = 0; i < quantity; i++)
    {
        int number = random.nextInt(mCHAR.length());
        char ch = mCHAR.charAt(number);
        builder.append(ch);
    }
    return builder.toString();
}
Random random = new Random();
```

```
String comment = "Comment_" + generateRandom(6, random);
vars.put("comment", comment);
```

- **generateRandom** – функция для генерации строки из определенного числа латинских букв, в зависимости от переданного в нее значения этого числа;
- с помощью метода **vars.put** мы создаем переменную Jmeter, к которой впоследствии можно будет обратиться в teste, и сохраняем в ней необходимое значение.

После настройки элемента переходим во вкладку **Parameters** запроса PostAcqForm и в качестве значения параметра **Entity.ListItem.Comment** указываем **\${comment}** .

В параметре **WorkLog.StartDate_date** нам необходимо передать текущую дату в формате **dd.MM.yyyy**. Для этого воспользуемся встроенной функцией JMeter **\${__time(dd.MM.yyyy)}** , которая будет возвращать текущую дату в указанном в скобках формате. С помощью функции **\${__time(HH:mm)}** зададим текущее время для параметра **WorkLog.StartDate_time**, а также текущую дату и время для параметра **WorkLog.StartDate** с помощью функции **\${__time(dd.MM.yyyy HH:mm)}** . Переидем во вкладку **Parameters** запроса PostAcqForm и впишем функции в качестве значения данных параметров (Рис. 237).

WorkLog.StartDate_date	 \${__time(dd.MM.yyyy)}
WorkLog.StartDate_time	 \${__time(HH:mm)}
WorkLog.StartDate	 \${__time(dd.MM.yyyy HH:mm)}

Рис. 237. Функции в качестве значения параметров

После завершения настройки сэмплера PostAcqForm параметризуем заголовки в сформированном внутри него элементе HTTP Headers Manager (Рис. 238).



Рис. 238. Элемент "HTTP Headers Manager" в сэмплере "PostAcqForm"

Настройка последней группы запросов завершена. Теперь необходимо запустить тест и проверить, что выполнение задачи ознакомления выполняется корректно. Перед запуском теста добавим в конец контроллера ExecuteAcqTask элемент Debug Sampler. Нажимаем на



кнопку запуска теста и переходим в элемент View Results Tree (Рис. 239).

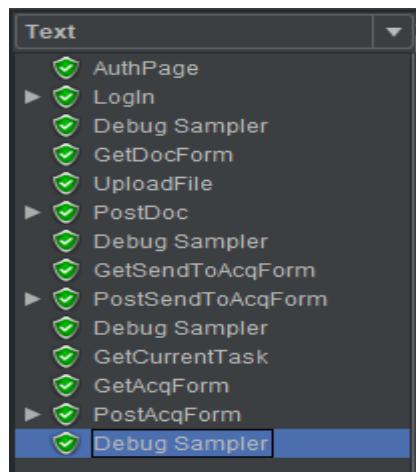


Рис. 239. Элемент "Debug Sampler"

Все запросы выполнились без ошибок. Проверим, корректно ли выполнилась сама задача ознакомления. Как и при отладке предыдущих операций, начинаем с перехода во вкладку **Response Data** элемента Debug Sampler (Рис. 240).

Рис. 240. Элемент "Debug Sampler". Вкладка "Response Data"

Здесь нас главным образом интересует значение переменной **TaskId**. В нее записалось значение 5370. Это говорит о том, что заданное в обработчике выражение нашло нужный элемент на странице. В этот раз при отладке сценария для документа сгенерировалось название "Document_qBzw", которое можно увидеть в качестве значения переменной **docName**. От него мы будем отталкиваться при анализе выполненных операций уже на веб-сервере ELMA.

Далее во View Results Tree перейдем на вкладку **Request** сэмпера GetAcqForm (Рис. 241) и скопируем оттуда ссылку на задачу ознакомления.

Рис. 241. Ссылка на задачу ознакомления. Сэмплер "GetAcqForm". Вкладка "Request"

Вставив ссылку в строку браузера, переходим на страницу задачи (Рис. 242).

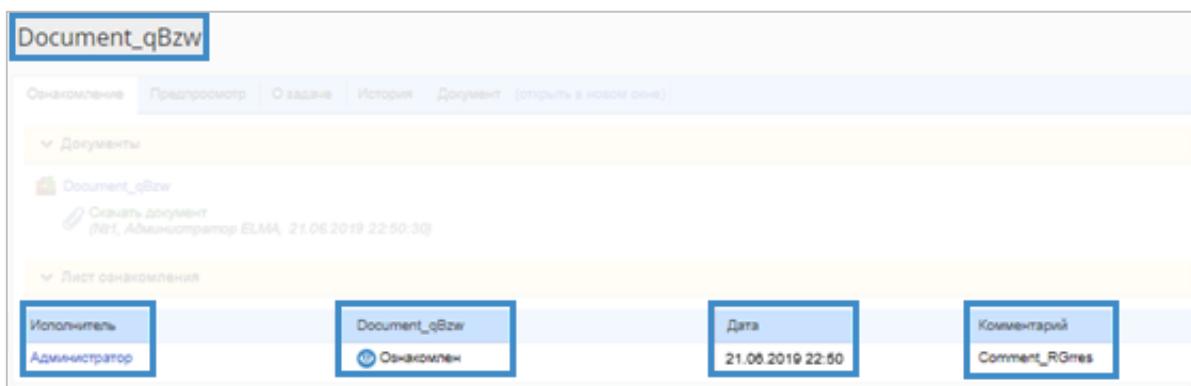


Рис. 242. Задача ознакомления

На форме задачи присутствует сгенерированный запросом PostDoc документ, название которого мы отметили [при просмотре результатов Debug Sampler](#), а также переданы корректные значения других переменных. На вкладке **Ознакомление** самого документа также отображается статус "Все ознакомлены" (Рис. 243).

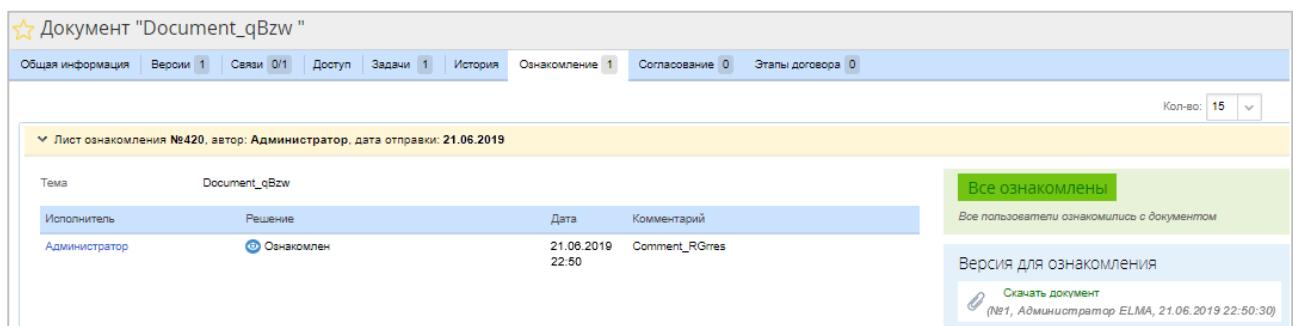


Рис. 243. Документ. Вкладка "Ознакомление"

Завершающим этапом нашего тестового сценария будет логаут текущего пользователя, чтобы в системе не накапливалось большое количество активных сессий.

4.4.5. Логаут

Включаем контроллер Logout (Рис. 244).

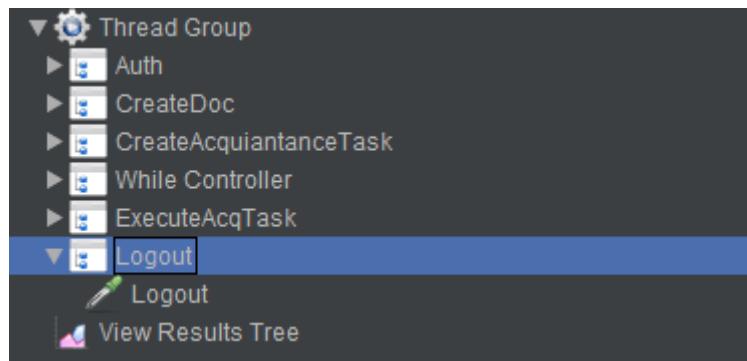


Рис. 244. Контроллер "Logout"

И переходим в сэмплер Logout (Рис. 245).

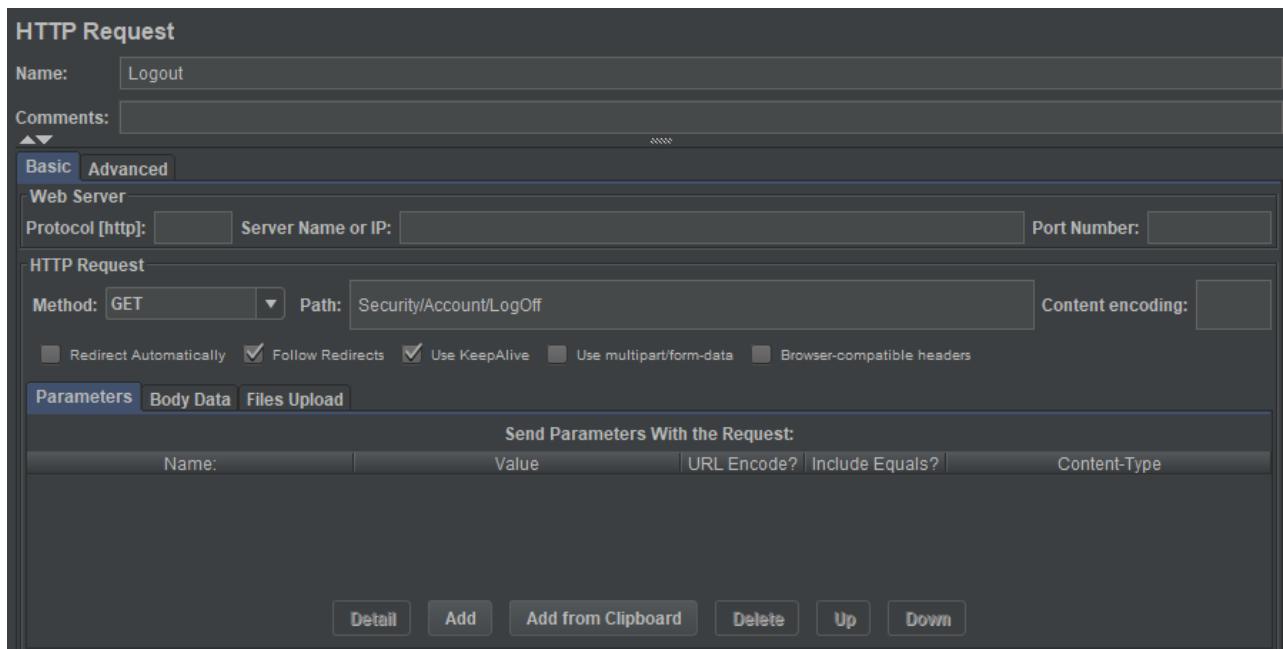


Рис. 245. Сэмплер "Logout"

Настройка данного сэмплера не требуется, поскольку сам запрос представляет из себя получение страницы .../**Security/Account/LogOff** без передачи каких-либо параметров, после чего информация о пользовательской сессии будет очищена.

Проведем проверочный запуск нашего сценария целиком. Требуется нажать на кнопку запуска теста и перейти в элемент View Results Tree (Рис. 246).

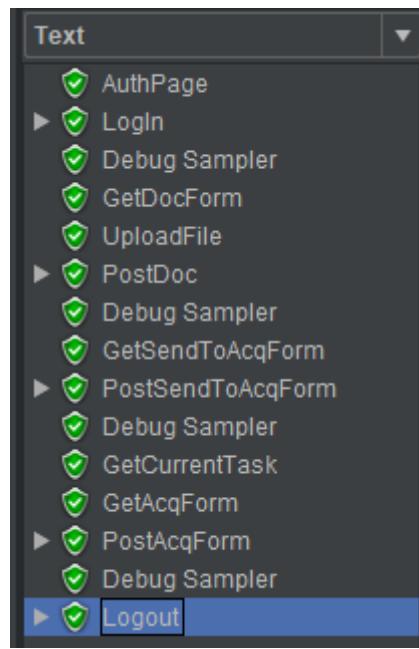


Рис. 246. Результаты выполнения тест плана во "View Results Tree"

Проанализировав результаты во View Results Tree, подобно тому как мы это делали при отладке предыдущих операций, мы пришли к выводу о том, что тестовый сценарий был выполнен успешно. Таким образом, теперь мы готовы приступить уже к самому нагрузочному тестированию.

Глава 5. Проведение нагрузочного тестирования

Опираясь на наш тестовый случай, мы можем составить соответствующий профиль нагрузки, разделив нагрузочное тестирование на 2 итерации.

5.1. Первая итерация тестирования

В рамках первой итерации будет определено максимальное число одновременно работающих пользователей в системе, чтобы выявить, по достижении какого числа пользователей будет начинаться деградация производительности. Исходя из того, что в системе в среднем одновременно работает 30 бизнес-пользователей, имеет смысл начать с данного фиксированного значения пользователей и каждые 30 минут следует увеличивать число пользователей на 10 вплоть до 100. Далее, проанализировав график активных виртуальных пользователей, определим, когда время отклика начинает превышать допустимые значения.

Переходим в Jmeter. Для начала отключим в нашем тестовом плане листенер View Results Tree, а также все элементы Debug Sampler, поскольку они необходимы только при отладке. Внутри каждого контроллера перед выполнением запроса методом POST разместим элемент Uniform Random Timer (Рис. 247), чтобы задать время на обдумывание пользователем, избежав тем самым режим стресс-теста.

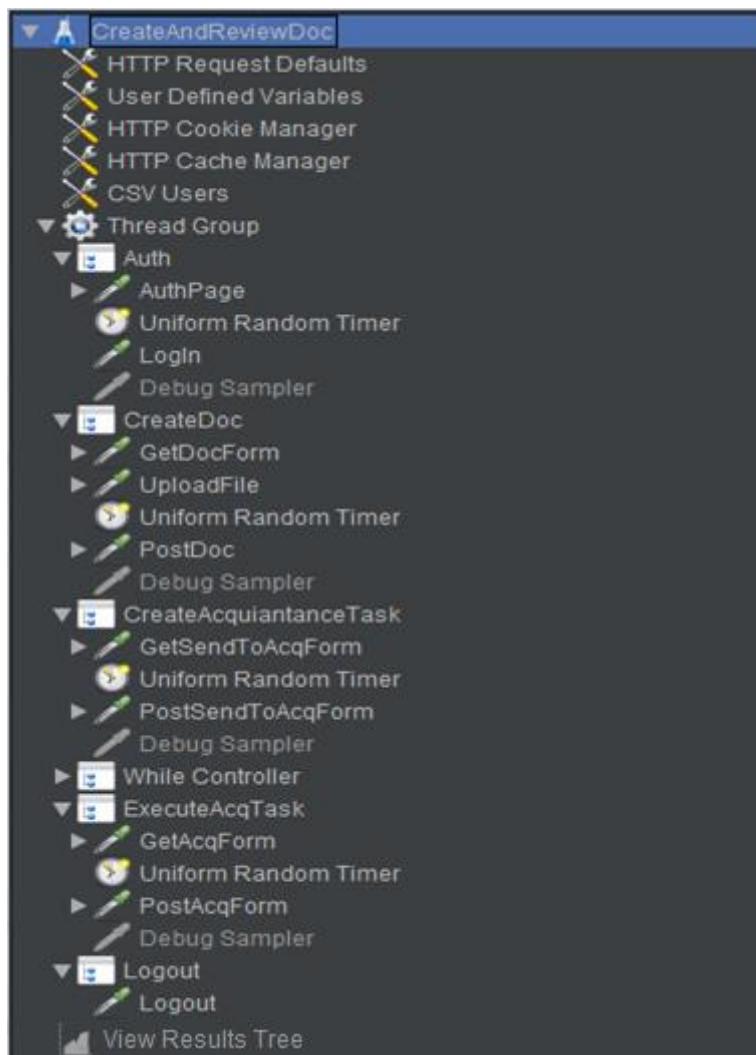


Рис. 247. Элемент "Uniform Random Timer"

При выборе времени на обдумывание следует опираться на реальное время, которое пользователь затрачивает до выполнения следующей операции. Как правило, в этом поле присутствует значение от 3 до 60 секунд. Перед выполнением сэмплера LogIn настроим таймер следующим образом (Рис. 248).

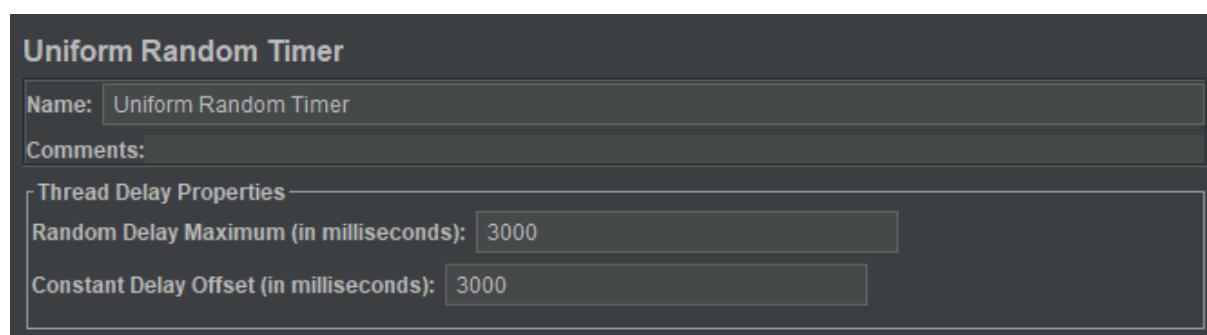


Рис. 248. Настройки Uniform Random Timer

- **Random Delay Maximum (in milliseconds)** – установим 3000 мс в качестве максимальной случайной задержки. При срабатывании таймера будет выбрано случайное время от 0 до 3 секунд включительно;
- **Constant Delay Offset (in milliseconds)** – установим 3000 мс в качестве фиксированного времени задержки, которое будет добавлено к случайному времени.

При данных настройках время на обдумывание перед выполнением сэмплера LogIn будет от 3 до 6 секунд.

Перед выполнением сэмплеров PostDoc, PostSendToAcqForm и PostAcqForm установим время на обдумывание от 6 до 12 секунд (Рис. 249).

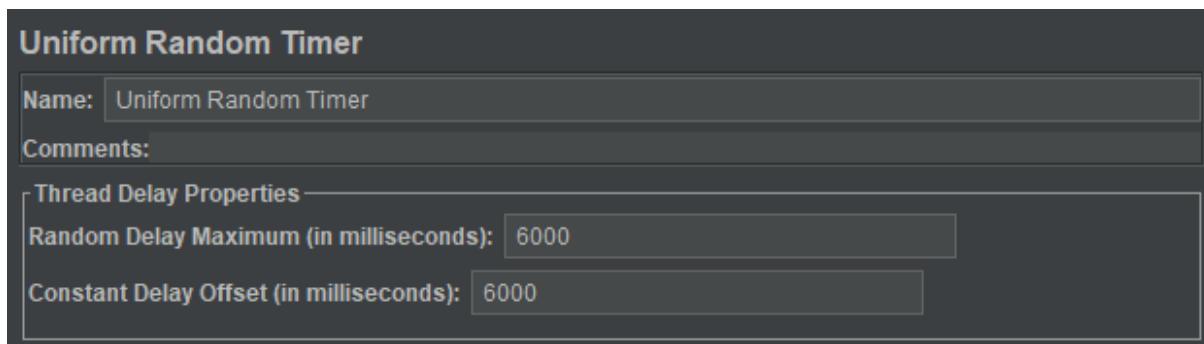


Рис. 249. Настройки Uniform Random Timer

Примечание: при формировании отчетами данных о времени выполнения сэмплеров, затраченное на обработку таймеров время не учитывается.

Далее перейдем к настройке элемента Thread Group (Рис. 250).

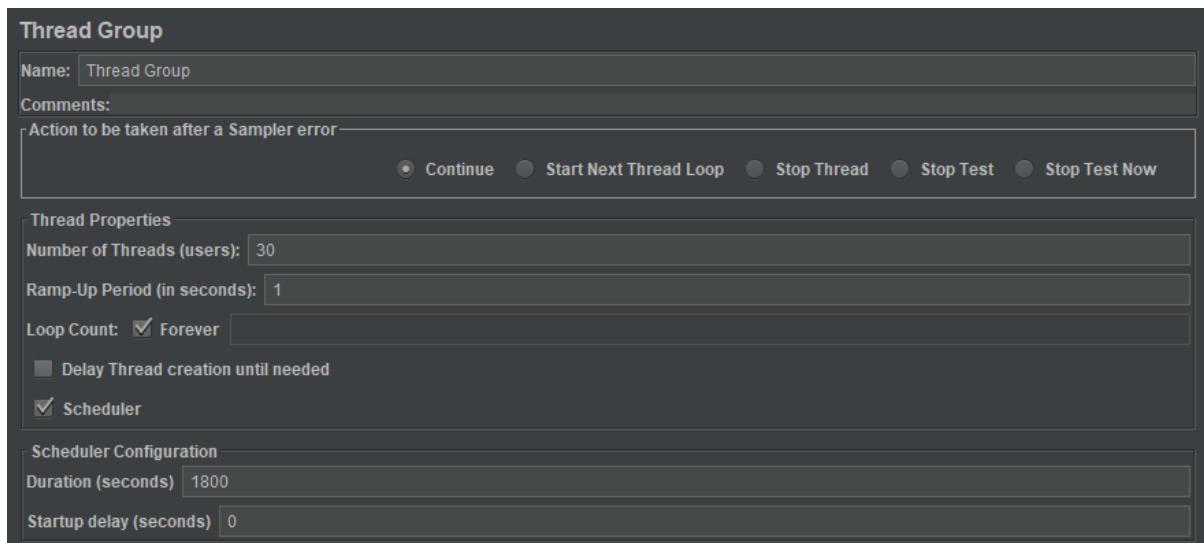


Рис. 250. Настройки элемента "Thread Group"

- в качестве начального числа виртуальных пользователей установим значение 30;
- **Ramp-Up Period** оставим без изменений, чтобы все виртуальные пользователи сразу же прогрузились Jmeter;
- в поле **Loop Count** установим флажок **Forever**;
- Включим **Scheduler**. В поле **Duration** введем значение 1800 секунд (30 минут) – время, в течение которого будет выполняться тест.

После завершения теста для 30 виртуальных пользователей мы перенастроим **Thread Group**, изменив число потоков на 40, и так каждый раз, до тех пор, пока не будет замечена деградация производительности. Отследить ее нам помогут отчеты [Aggregate Report](#) и [DashBoard Report](#), в которых после прохождения теста мы сможем проанализировать информацию о [метриках производительности](#).

Для начала разместим в конец тестового плана листенер Aggregate Report, а также листенеры Simple Data Writer и Summary Report, которые необходимы для формирования DashBoard Report (Рис. 251). Его полная настройка рассмотрена в главе [5.3](#).

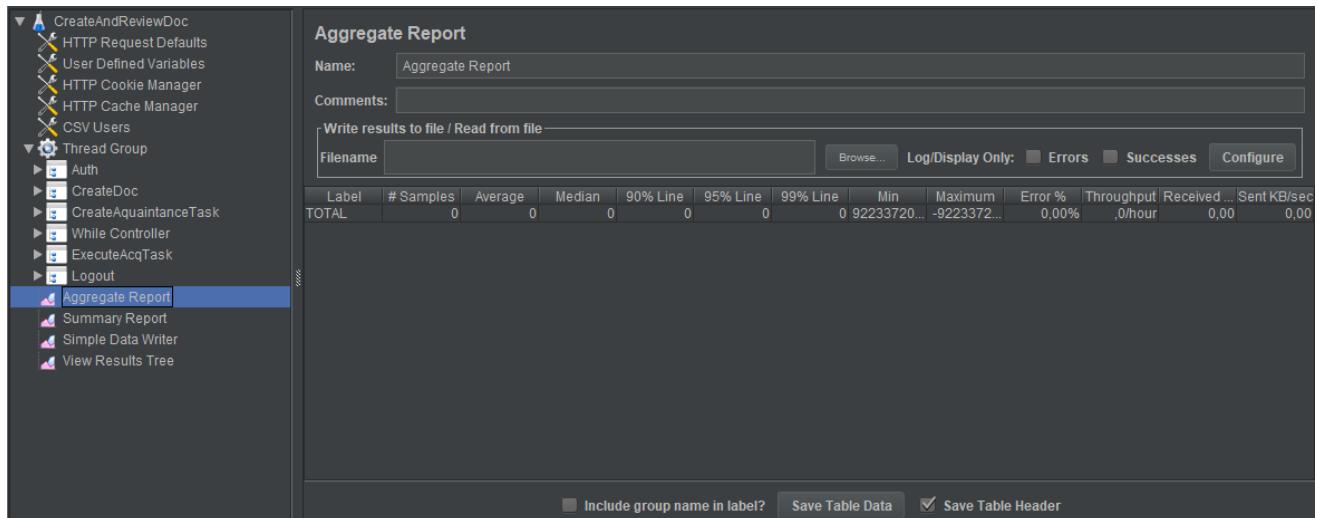


Рис. 251. Листенер "Aggregate Report"

Теперь все готово для проведения нагрузочного тестирования.

Нажимаем на кнопку запуска теста .

[Проанализировав результаты](#) прохождения первой итерации, мы пришли к выводу о том, что после 1 часа производительность системы начинает деградировать. Сопоставив необходимые данные, мы выявили, что при количестве в 50 пользователей время отклика начинает превышать

допустимое значение. Однако на протяжении всего теста, продолжительность которого составляет 4 часа, не было зафиксировано таймаутов, что свидетельствует о возможности приложения выдерживать нагрузку до 100 одновременно работающих пользователей с увеличением времени отклика.

5.2. Вторая итерация тестирования

По [итогам](#) предыдущего теста проводится вторая итерация с условиями одновременной работы 50 пользователей, чтобы убедиться в том, что система будет выдерживать нагрузку в 50 виртуальных пользователей без деградации в течение 1 часа.

Переходим в Jmeter и настраиваем элемент Thread Group следующим образом (Рис. 252).

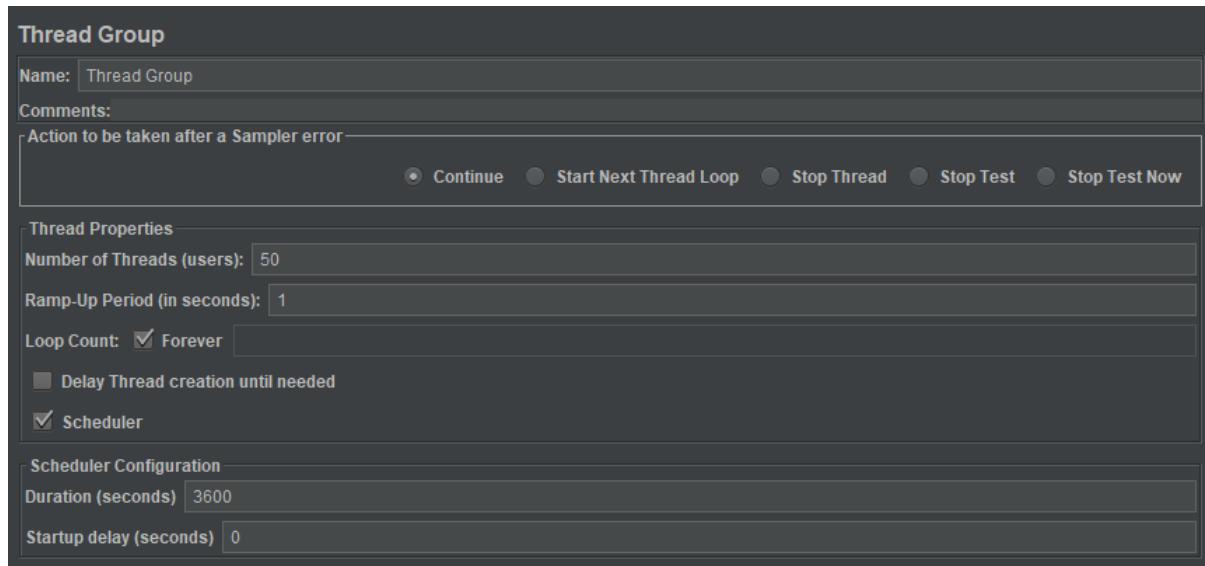


Рис. 252. Настройки элемента "Thread Group"

Остальные настройки тестового сценария оставляем без изменений.

Нажимаем на кнопку запуска теста .

После завершения второй итерации тестирования переходим к [анализу ее результатов](#).

5.3. Настройка DashBoard Report

Одним из наиболее показательных отчетов производительности в Jmeter является встроенное в него модульное расширение **DashBoard Report**, которое, будучи сгенерированным, включает в себя большинство стандартных графических метрик. Его принцип работы заключается в чтении и обработке данных из файла формата .CSV, записываемых листенером Summary Report, для последующей генерации отчета. Поскольку данный отчет является расширением Jmeter, для его генерации необходимо выполнить следующие настройки.

Примечание: данные шаги актуальны для Jmeter версии 3.0 и выше.

1. Перейти в каталог с установленным Jmeter.
2. В папке **bin** найти файлы **reportgenerator.properties** и **user.properties**.
3. Скопировать содержимое файла **reportgenerator.properties** в файл **user.properties**.
4. Сохранить файл.
5. Из папки **bin** запустить Jmeter и открыть имеющийся тестовый сценарий или создать новый.
6. В конец сценария добавить листенеры [Simple Data Writer](#) и [Summary Report](#). Необходимо нажать на кнопку **Configure** и убедиться, что в каждом из данных листенеров установлены следующие флагги (Рис. 253).

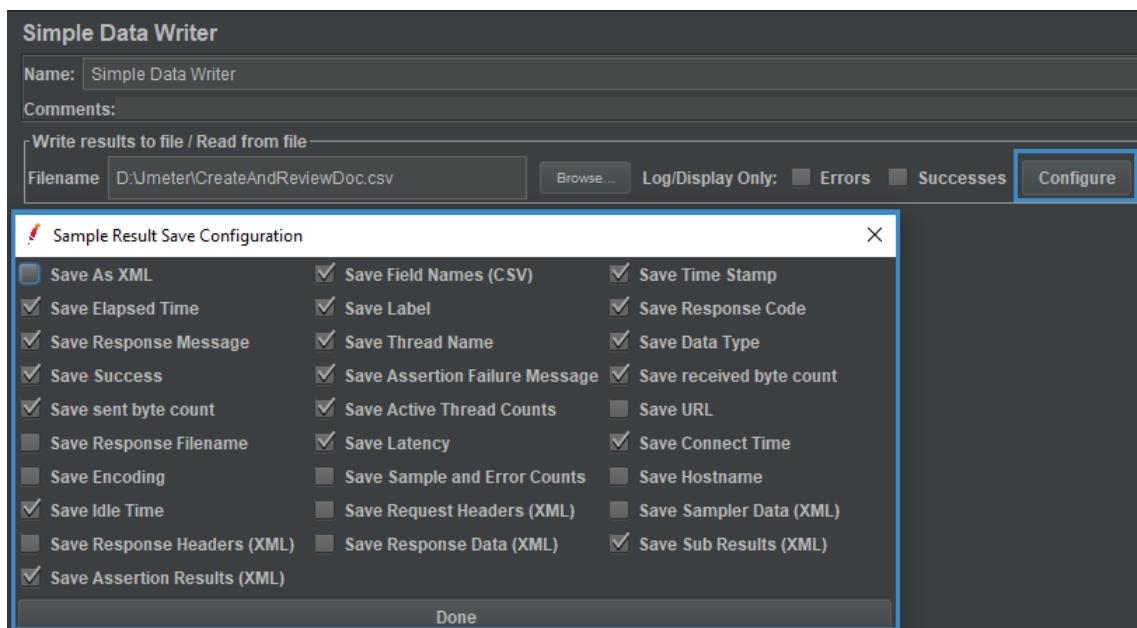


Рис. 253. Листенер "Simple Data Writer". Кнопка "Configure"

7. В Simple Data Writer и Summary Report нажать на кнопку **Browse** и выбрать папку, куда будет записан файл с тестовыми данными, а также указать для него имя (расположение и название файла должны совпадать для обоих листенеров).

Примечание: не нужно создавать .CSV файл до запуска теста. Во время прогона теста он будет создан автоматически.

8. Запустить тестовый сценарий и дождаться его завершения.

9. В командной строке необходимо выполнить следующую команду:

Jmeter -g D:\Jmeter\CreateAndReviewDoc.csv -o D:Report\ReportD , в которой:

- **D:\Jmeter\CreateAndReviewDoc.csv** – путь до файла из п.7;
- **D:Report\ReportD** – папка, в которой будет сгенерирован HTML файл с отчетом.

10. Перейти в **D:Report\ReportD**, открыть **index.html** и проверить результаты теста.

5.4. Результаты тестирования производительности

На основании требований к результатам тестирования, которые, как правило, ожидаются от заказчика, по результатам тестирования должен быть сформирован вывод о том, справляется ли система или приложение с ожидаемым уровнем нагрузки, а также о факторах, ограничивающих дальнейший рост производительности. Проводится подробный анализ метрик производительности для выявления ограничителей производительности, чтобы определить конкретные причины их возникновения и выработать рекомендации по их устранению.

5.4.1. Ключевые метрики производительности

№	Метрика производительности	Описание	Средства диагностики
1.	Потребление ресурсов центрального процессора, %.	Метрика, показывающая, сколько времени из заданного определенного интервала было потрачено процессором на вычисления для выбранного процесса.	Счетчики производительности Windows.
2.	Потребление оперативной памяти, Мб.	Метрика, показывающая количество памяти, использованной приложением.	Счетчики производительности Windows.
3.	Потребление сетевых ресурсов, r/w кб/сек, io wait.	Данная метрика не связана непосредственно с производительностью приложения, однако ее показатели могут указывать на пределы производительности системы в целом.	Счетчики производительности Windows.
4.	Работа с дисковой подсистемой, r/w кб/сек, io wait.	Работа с дисковой подсистемой может значительно влиять на производительность системы, поэтому сбор статистики по	Счетчики производительности Windows.

№	Метрика производительности	Описание	Средства диагностики
		работе с диском может помогать выявлять узкие места в этой области.	
5.	Время выполнения запроса с детализацией по типам запросов, мс.	Данное время может быть измерено как на серверной стороне, как показатель времени, которое требуется серверной части для обработки запроса, так и на клиентской, как показатель полного времени, которое требуется на сериализацию/десериализацию, пересылку и обработку запроса.	Отчет о производительности ELMA, отчет Jmeter.
6.	Количество запросов в секунду.	Показатель масштабирования, характеризующий пропускную способность системы.	Отчет о производительности ELMA, отчет Jmeter.
7.	Число и тип ошибок.	Метрика, используемая для сбора и категоризации ошибок, возникающих в ходе теста.	Отчет о производительности ELMA, отчет Jmeter.

Как видно из таблицы, приведенной выше, для отслеживания данных показателей могут использоваться как стандартные утилиты, входящие во все современные редакции ОС Windows, так и стандартные средства диагностики системы ELMA. В самом Jmeter существует широкий спектр разнообразных "Листенеров" (Listeners) для сбора и отображения метрик производительности. Ниже будут рассмотрены добавленные ранее в тестовый план отчеты и то, как их анализировать.

В первой итерации нагрузочного тестирования было выполнено несколько тридцатиминутных тестов с увеличением числа виртуальных пользователей. После завершения каждого теста генерировался **DashBoard Report** путем ввода в командную строку следующей команды: **Jmeter -g D:\Jmeter\CreateAndReviewDoc.csv -o D:Report\ReportD**. После этого страницу сгенерированного отчета можно было просмотреть. Для этого требовалось открыть в папке **D:Report\ReportD** файл **index.html**. Спустя

полтора часа после завершения третьего тридцатиминутного теста на странице отчета мы обратили внимание на следующие метрики:

- **Response Times Over Time** – график, на котором отображено среднее время отклика в миллисекундах для каждого запроса, а также время прохождения тестового сценария (Рис. 254).

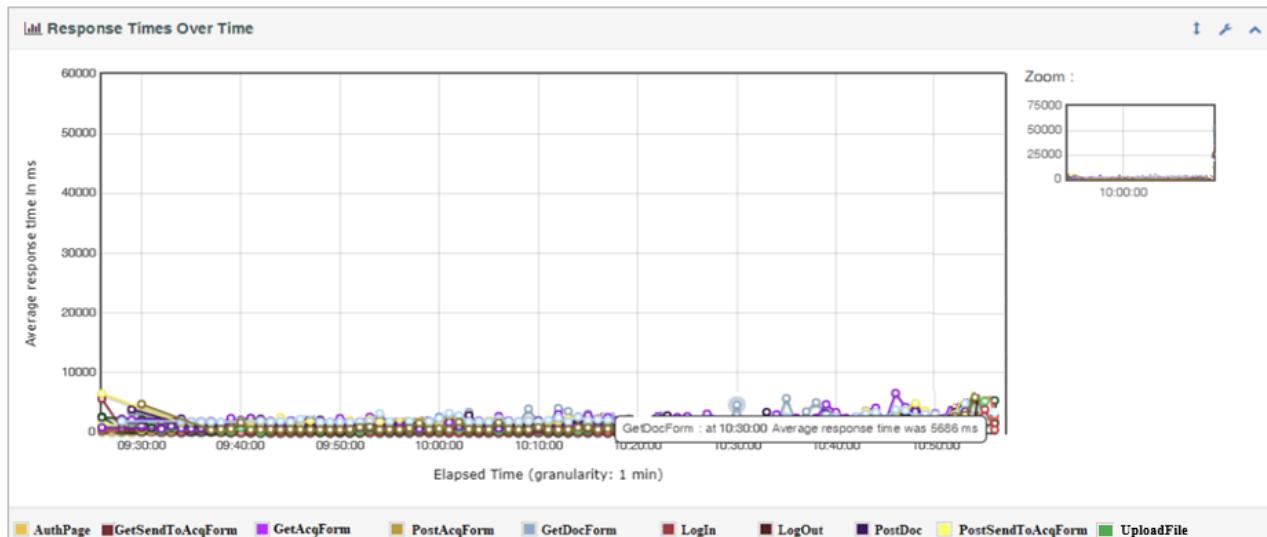


Рис. 254. График "Response Times Over Time"

- **Active Threads Over Time** – график, показывающий число активных виртуальных пользователей в течение прохождения теста (Рис. 255).

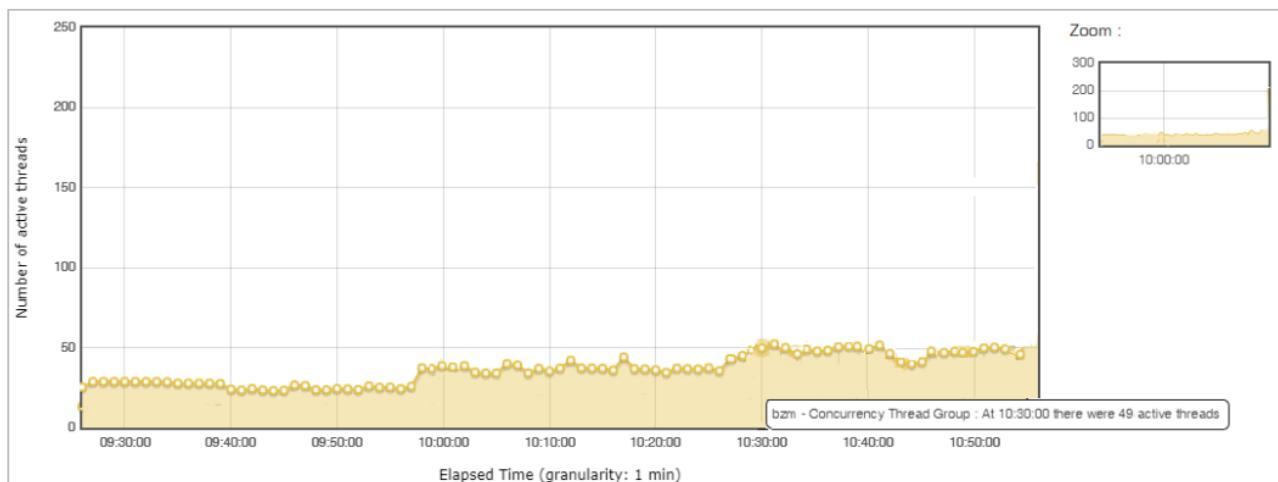


Рис. 255. График "Active Threads Over Time"

Как видно на графике **Response Times Over Time**, спустя ровно час после запуска теста, в 10:30 отклик на запрос GetDocForm (получение формы создания документа) был получен спустя 5,6 секунды и далее, как показано на графике, все последующие запросы выполнялись приблизительно с тем же интервалом (и даже более высоким).

Как мы знаем, данный тест начинался с фиксированного значения в 30 виртуальных пользователей, и каждые полчаса их число увеличивалось на 10. Таким образом, через час после запуска теста их число должно было составлять 50, что и показано на графике **Active Threads Over Time**. Сопоставив данные этих двух графиков, мы делаем вывод, что время откликов начинает превышать значение 5 секунд при одновременной работе 50 пользователей.

Важно учитывать, что среднее значение времени отклика на запрос не является наиболее показательной метрикой. Несколько запросов, выполняющихся существенно дольше остальных, дадут очень большой вклад при расчете среднего значения. Гораздо важнее анализировать **перцентиль (Line)** 90% / 95% или 99% – время, за которое отклик получает соответствующая доля запросов. Таким образом, перцентиль 90% – это значение, для которого 90% выборки имеет меньшее время отклика. Перейдем в Jmeter и посмотрим результаты прохождения второй итерации тестирования с 50 виртуальными пользователями в листенере Aggregate Report (Рис. 256).

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	Received KB	Sent KB/s
AuthPage	501	6077	6118	9361	11404	12938	22	14203	0,00%	0,39517	4,15	0,45
Login	500	6513	6567	9501	11435	13170	205	15154	0,00%	0,40031	21,45	0,97
GetDocForm	499	9189	9142	12253	13638	17007	504	19943	0,00%	0,39213	47,05	0,53
UploadFile	498	7646	7246	9120	11310	13442	647	15232	0,00%	0,38556	37,82	3,67
PostDoc	498	10748	10380	14980	17605	21801	647	23626	0,00%	0,38371	37,33	3,57
GetSendToAcqForm	497	6046	5394	9382	12487	16354	283	16794	0,00%	0,3699	28,82	0,51
PostSendToAcqForm	497	10046	10086	12932	13894	17146	421	18556	0,00%	0,37371	40,33	3,41
GetAcqForm	495	3566	3454	5455	6550	11137	0	11699	0,00%	0,3672	6,46	0,7
PostAcqForm	493	6824	6990	9222	10714	13572	0	15311	0,00%	0,36053	32,42	0,51
LogOut	491	6588	6294	8977	10598	14638	35	15719	0,00%	0,35494	3,71	0,73
TOTAL	4969	7634	7243	12034	13668	18167	0	23626	0,00%	3,72302	256,31	12,76

Рис. 256. Результаты прохождения второй итерации тестирования

В полученном нами отчете замечена деградация производительности системы, которая выражается в увеличенном времени отклика запроса. Желтым цветом выделено среднее время выполнения каждого запроса, которое составляет свыше 6 секунд, за исключением запроса получения формы задачи отправки на ознакомление (**GetAcqForm**), который выполнялся в среднем за 3,5 секунды. Например, проанализировав перцентили (**Line**), мы видим, что подавляющее большинство операций по созданию документов (99%) выполнялись за 52,2 секунды (сумма времени выполнения стадий создания документа), а именно: 17 секунд на выполнение запроса по получению формы (**GetDocForm**), 13,4 секунды потребовалось для загрузки файла-версии (**UploadFile**) и 21,8 секунды для отправки формы (**PostDoc**).

Таким же образом, опираясь на имеющиеся метрики производительности, можно проанализировать остальные запросы и сделать выводы о производительности системы в целом. К плюсам можно отнести то, что в течение всего теста не было зафиксировано ни одного таймаута, что говорит о способности системы выдерживать нагрузку в 50 одновременно работающих бизнес-пользователей.

Таким образом, по результатам проведенных нагрузочных тестов системы, была получена информацию об оптимальном числе одновременно работающих бизнес-пользователей, а также выявлены параметры системы, при которых наблюдается деградация ее производительности. Данная информация критически важна для ее дальнейшего конфигурирования.

Глава 6. Полезные ресурсы

Помимо данного руководства, посвященного проведению тестирования производительности веб-приложений с помощью Apache Jmeter на примере системы ELMA, существуют и другие издания, в которых описываются основные возможности приложений системы ELMA:

- [Краткое руководство по Платформе ELMA BPM;](#)
- [Краткое руководство по Платформе ELMA BPM версии 4.0.0 RC;](#)
- [Краткое руководство по внутреннему порталу ELMA;](#)
- [Краткое руководство по приложению ELMA ECM+;](#)
- [Краткое руководство по приложению ELMA CRM+;](#)
- [Краткое руководство по приложению ELMA Проекты+;](#)
- [Краткое руководство по приложению ELMA KPI;](#)
- [Краткое руководство администратора ELMA;](#)
- [Регламент поддержки production-среды заказчика;](#)
- [Краткое руководство по настройке работы системы ELMA на высоких нагрузках;](#)
- [Краткое руководство по интеграции системы ELMA с внешними системами;](#)
- [Краткое руководство по интеграции системы ELMA с системой «1С:Предприятие»;](#)
- [Краткое руководство по генерации документов по шаблону в системе ELMA.](#)

Данные руководства знакомят читателя с ключевыми особенностями системы, подробное и исчерпывающее описание функционала системы ELMA содержится в справке, которая входит в поставку системы, а также всегда доступна в сети Интернет: <https://www.elma-bpm.ru/KB/help>.

Справочные материалы по каждому приложению разбиты на три категории: для пользователя, для внедрения и для администратора, что позволяет быстро найти нужную информацию.

Общее описание приложений и условия их приобретения доступны на **сайте ELMA**: <https://www.elma-bpm.ru>. Также на данном сайте всегда можно обратиться в компанию ELMA с помощью кнопки **Задать вопрос**, расположенной на главной странице сайта.

Ключевые возможности приложений и основные способы их использования продемонстрированы в **on-line демоверсии** <https://www.elma-bpm.ru/download>. Если же Вы хотите подробнее изучить

какое-либо из приложений, по этой же ссылке доступно скачивание демоверсии с такими же настройками, как и в on-line версии.

Система ELMA постоянно развивается, и на базе Платформы и приложений разрабатываются компоненты, предназначенные для решения различных более узких и конкретных задач. Со списком и условиями приобретения таких готовых решений Вы можете ознакомиться в **ELMA Store**: <https://store.elma-bpm.ru>.

При разработке собственных решений полезными окажутся материалы **Базы знаний ELMA**: <https://www.elma-bpm.ru/KB>.

Если же при работе в системе возникли вопросы технического характера, можно обратиться на сайт технической поддержки ELMA: <http://support.elma-bpm.ru>.

Для получений консультаций по системе ELMA или по сотрудничеству с компанией ELMA позвоните нам:

- Ижевск: +7 (3412) 93-66-93
- Москва: +7 (499) 921-02-87
- Казань: +7 (843) 567-17-69
- Киров: +7 (8332) 22-13-11
- Киев: +7 909 050-10-06
- Алматы: +7 (727) 313-15-04