Оглавление

[Установка плагинов. 3](#_Toc505023592)

[Запись с рекордером 4](#_Toc505023593)

[Дефолтные настройки. 6](#_Toc505023594)

[HTTP Request Defaults 6](#_Toc505023595)

[HTTP Header Manager 6](#_Toc505023596)

[Глобальные переменные 7](#_Toc505023597)

[Тестовые данные 8](#_Toc505023598)

[Корреляция запросов (связывание запросов). 10](#_Toc505023599)

[Проверки (assertion) 10](#_Toc505023600)

[Управление потоком выполнения 12](#_Toc505023601)

Классификация проблем:

1. Медленная подсистема/функция
2. Узкое место (бутылочное горлышко)
3. Точка насыщения
4. Функциональный дефект
5. Дефект интерфейса

Ошибки первого и второго рода

1. Ложная тревога. Ошибка первого рода. Ложноположительный результат.
2. Пропуск дефекта. Ошибка второго порядка. Ложноотрицательный результат.

Сбор данных:

1. Генератор нагрузки
2. Операционная система
3. Приложения (сервера)
4. Распределенная система

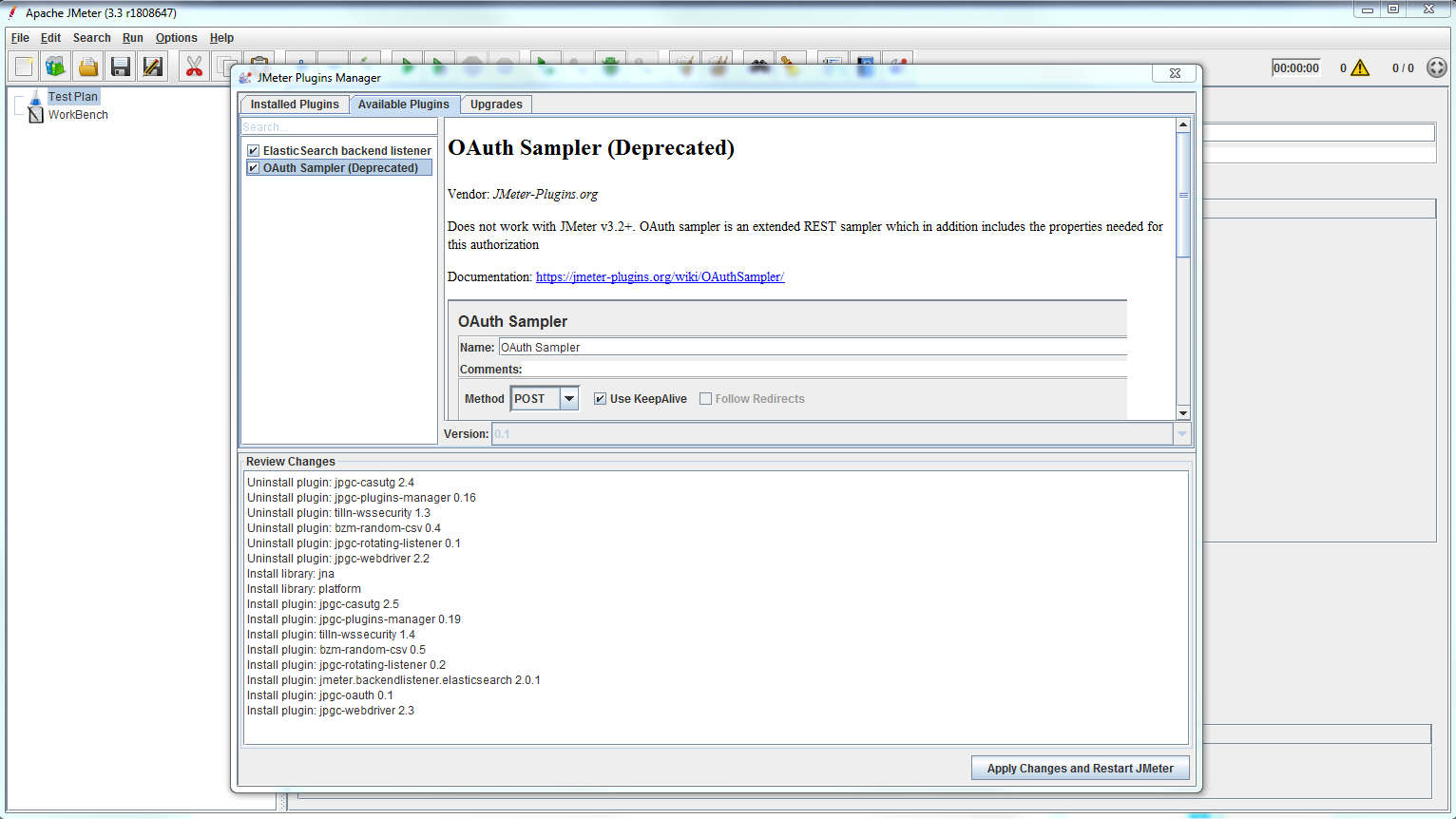
Анализ данных:

1. Время отклика, отказы
2. Использование ресурсов
3. В режиме реального времени
4. Среднее значение и аномалии
5. Динамические данные

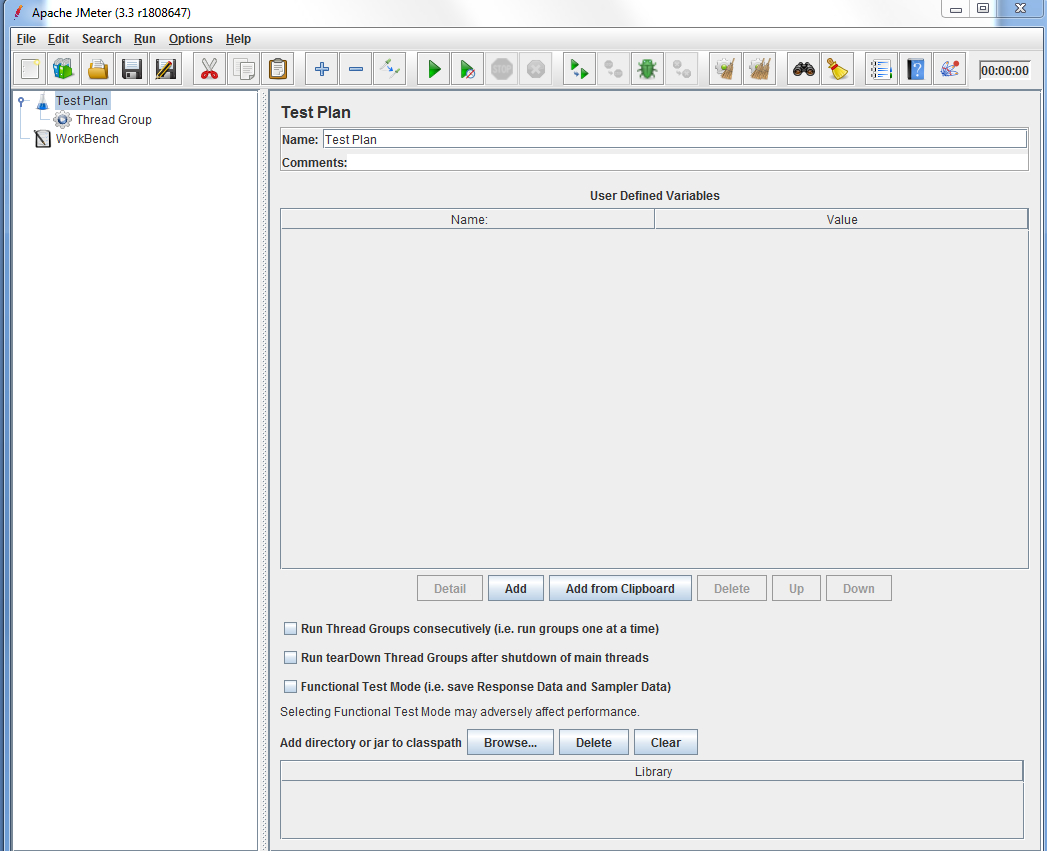
# Установка плагинов.

Сначала установить плагин менеджер <https://jmeter-plugins.org/install/Install/>

Открыть Options->Plugins Manager->Available Plugins нажать кнопку Apply Changes and Restart JMeter



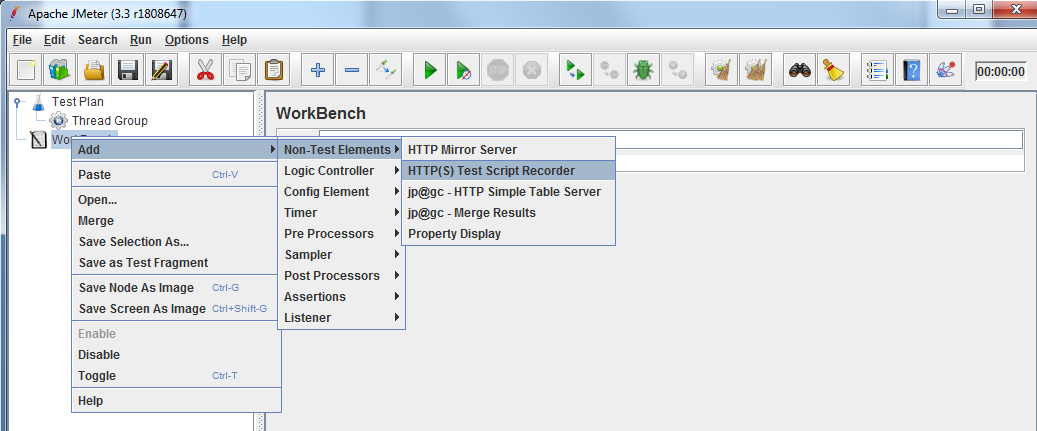
Test Plan-> чек-бокс Functional Test Model (i.e. save …) – используется для функционального тестирование, идет нагрузка 1 пользователя



WorkBench – временное рабочее пространство, данные не сохраняются в нем. Временное хранилище.

Запись с рекордером

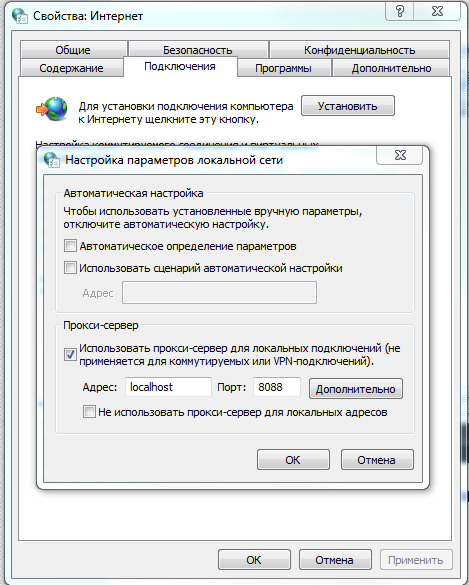
Рекордер работает как прокси сервер, прослойка между браузером и сервером.



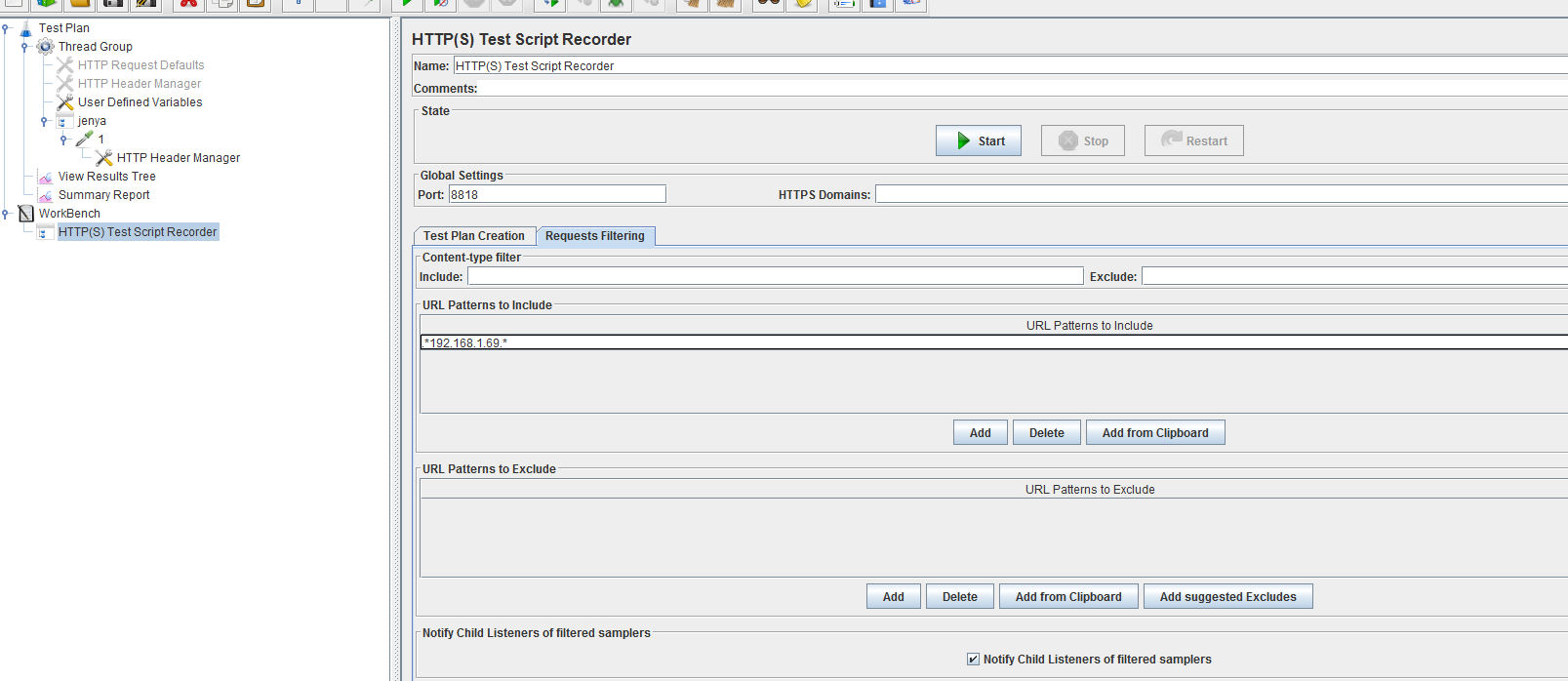
Сначала установить сертефикат как описано в <https://wiki.webmoney.ru/projects/webmoney/wiki/%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0_%D0%BA%D0%BE%D1%80%D0%BD%D0%B5%D0%B2%D0%BE%D0%B3%D0%BE_%D1%81%D0%B5%D1%80%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%82%D0%B0_%D0%B2_%D0%B1%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80%D0%B5_Mozilla_Firefox> firefox->Настройки -> Приватность и защита -> Просмотр сертефикатов -> Центры сертефикации -> Импортировать -> отметить все чек-боксы

Сертефикат действует 7 дней и лежит в папке apache-jmeter-3.3\bin\ ApacheJMeterTemporaryRootCA

Чтобы работало через прокси-сервер нужно в настройках браузера изменить параметры локальной сети.



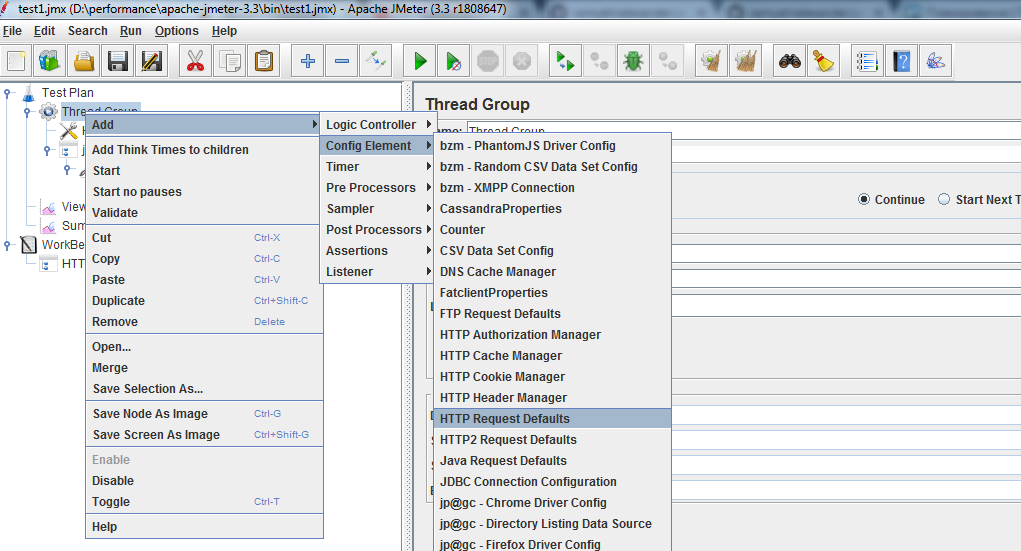
Чтобы шла запись только с определенного сайта нужно добавить url паттерн



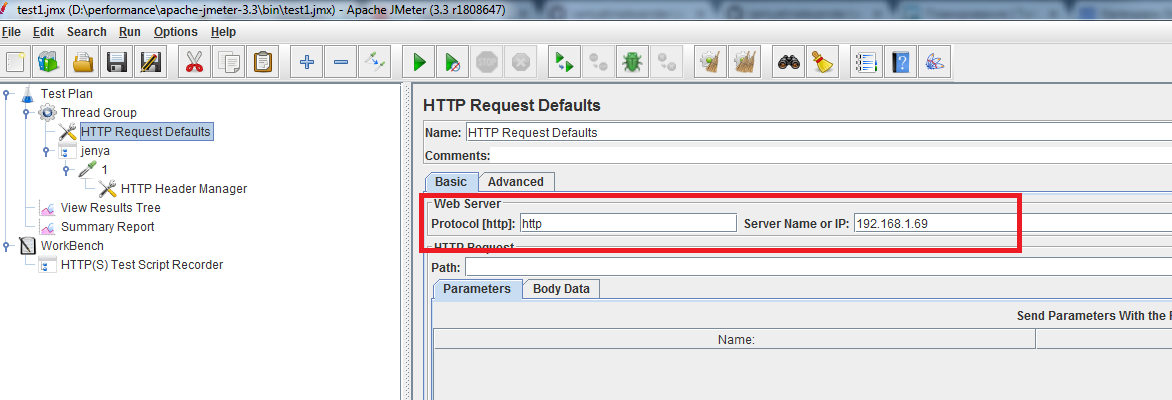
# Дефолтные настройки.

Эти настройки поставить заранее, до запуска рекордера

## HTTP Request Defaults

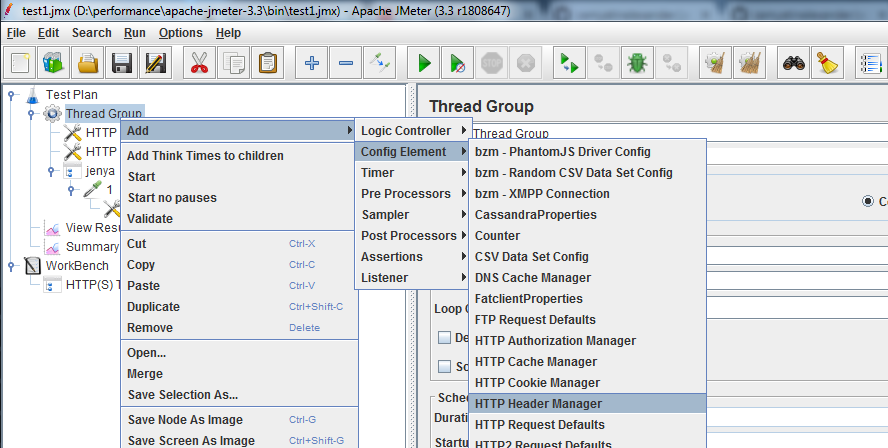


Прописать протокол и Server Name or IP

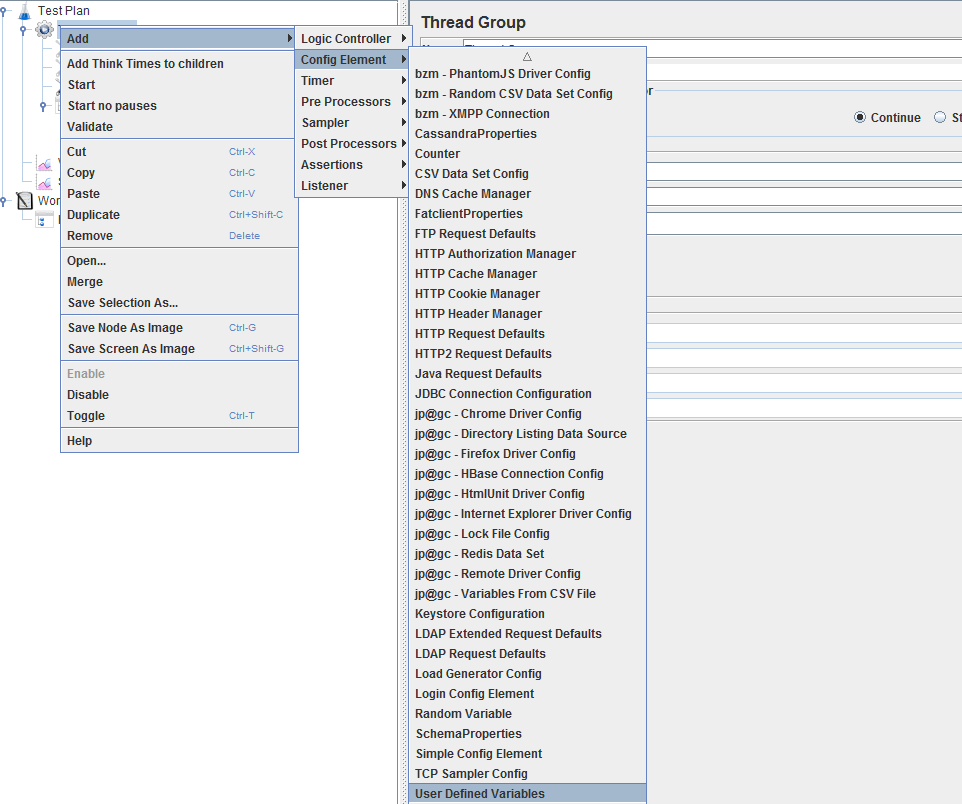


Тогда все запрос в которых не заполнены протокол и Server Name or IP будут отправляться на адрес прописанный в HTTP Request Defaults

## HTTP Header Manager

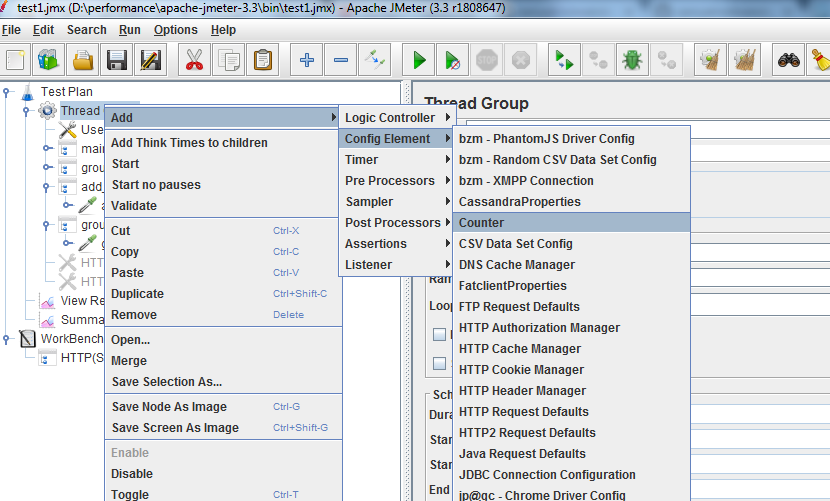


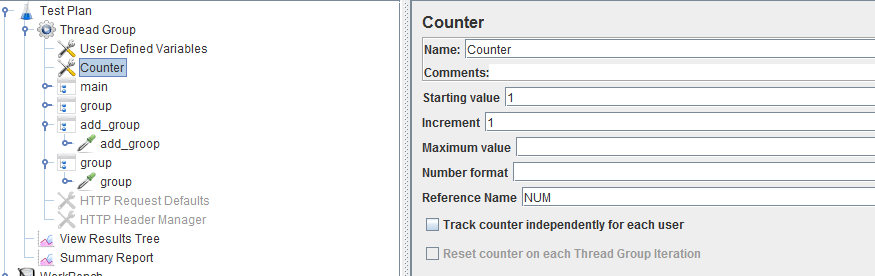
## Глобальные переменные



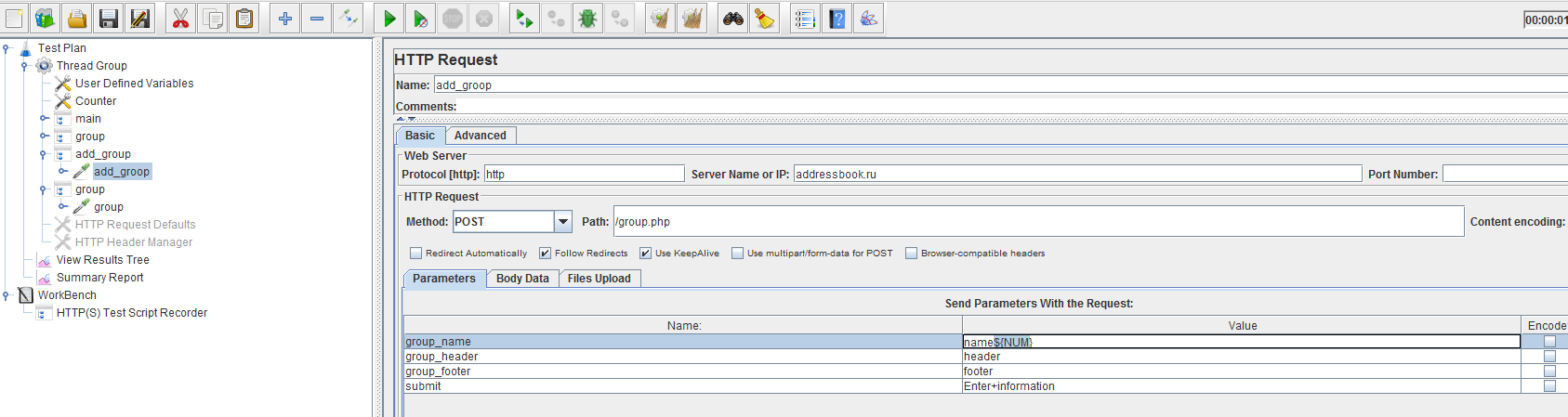
# Тестовые данные

Счетчик – меняет значение при каждом прохождении

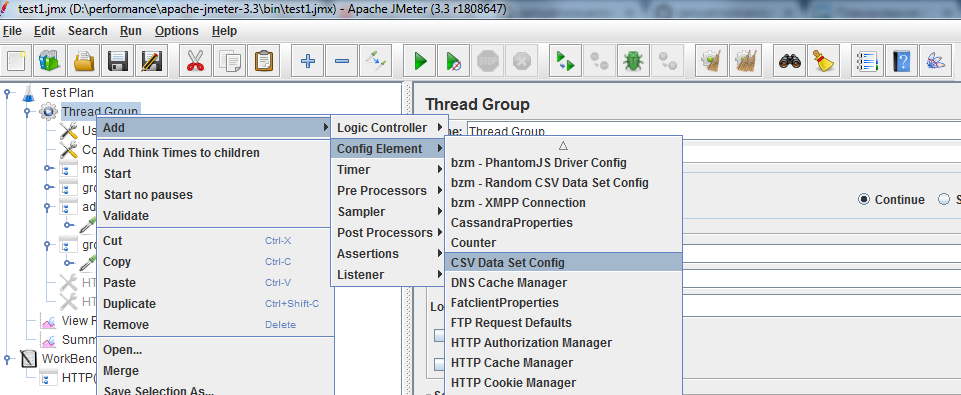


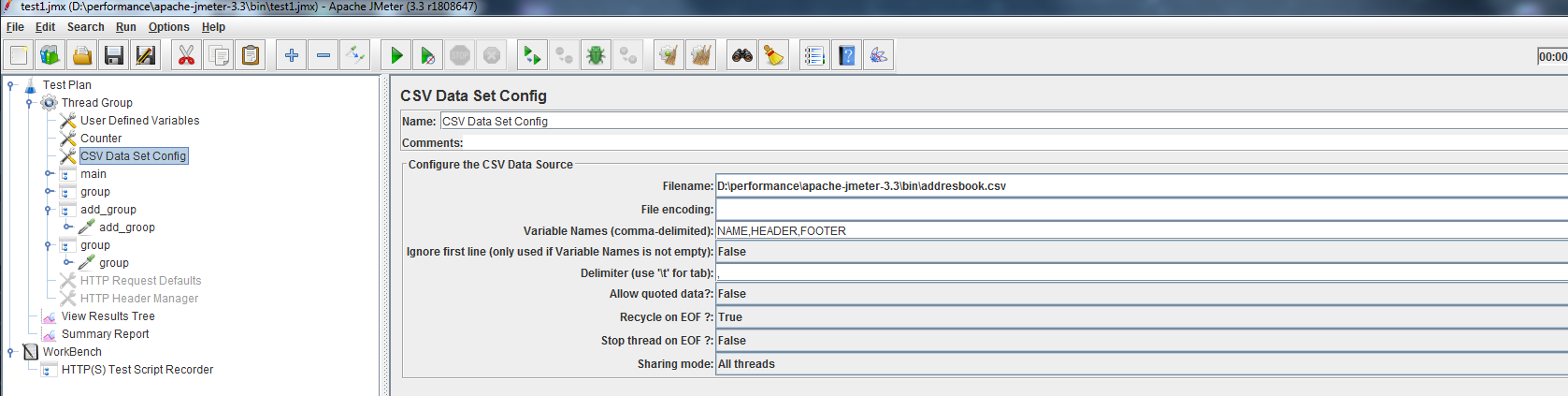


Использование

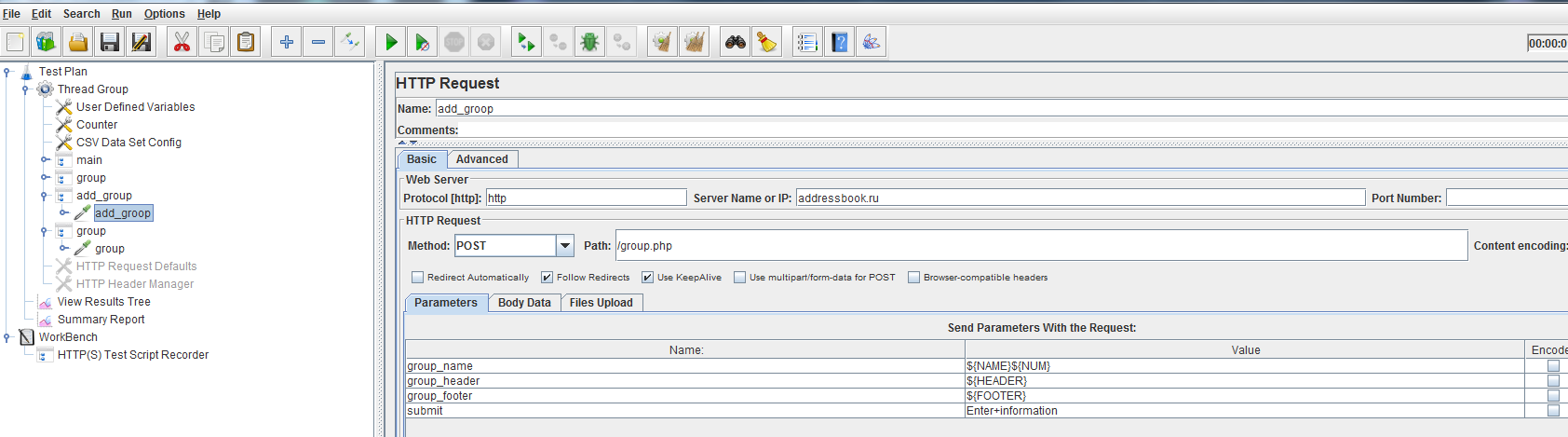


Набор значений





Использование



Рандомные значения:

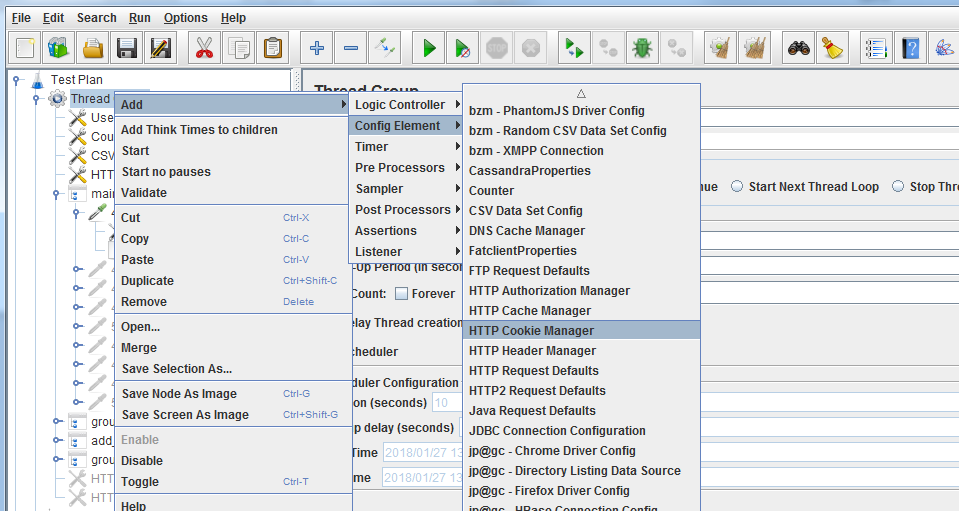
\_\_RandomString(20,qwertyuiopasdf) – строка из 20 символов из этого списка

\_\_RandomString(${\_\_Random(1,30), qwertyuiopasdf)} – строка длиной от 1 до 30 символов из этого списка символов

# Корреляция запросов (связывание запросов).

Чтобы брать параметр из response нужно написать регулярное выражение **Regular Expression Extractor**

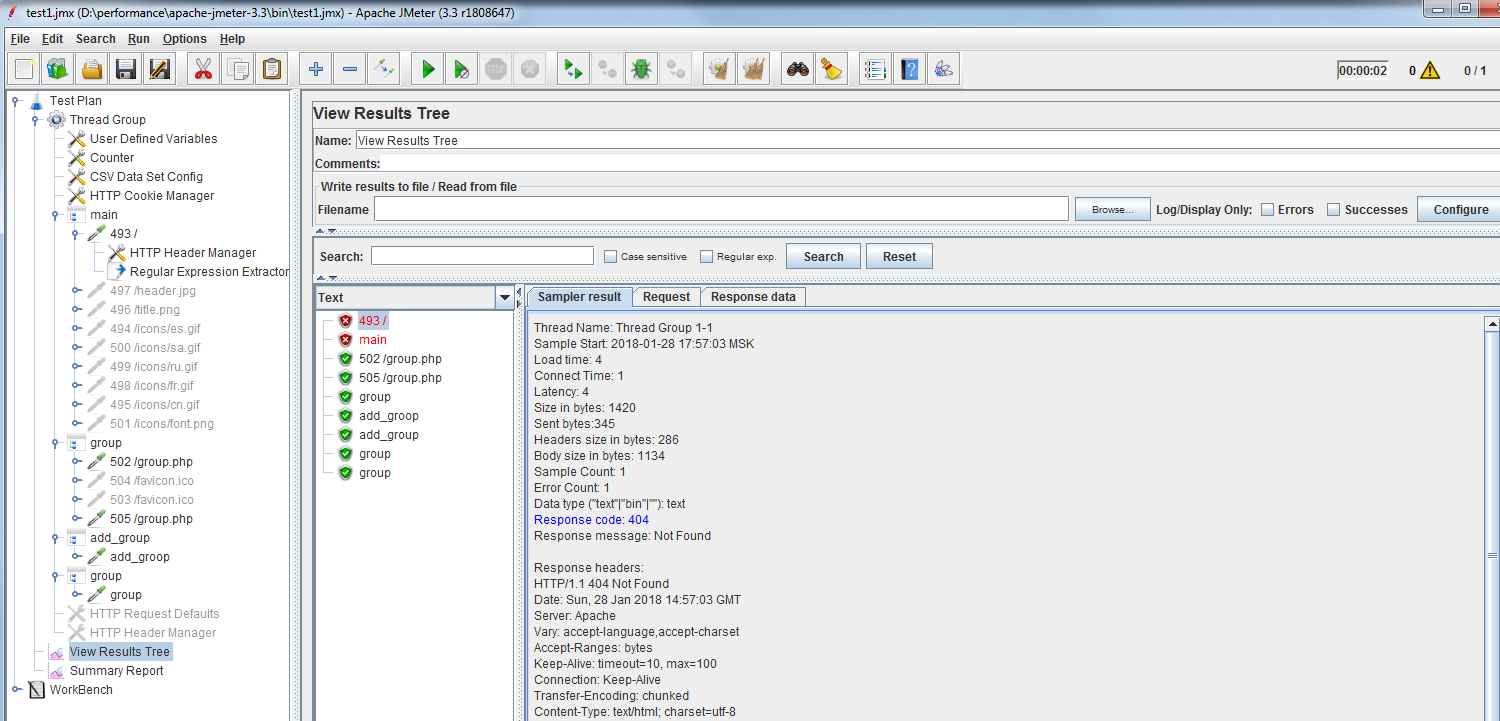
Важно добавлять HTTP Cookie Manager и HTTP Cache Manager, они нужны почти всегда.



# Проверки (assertion)

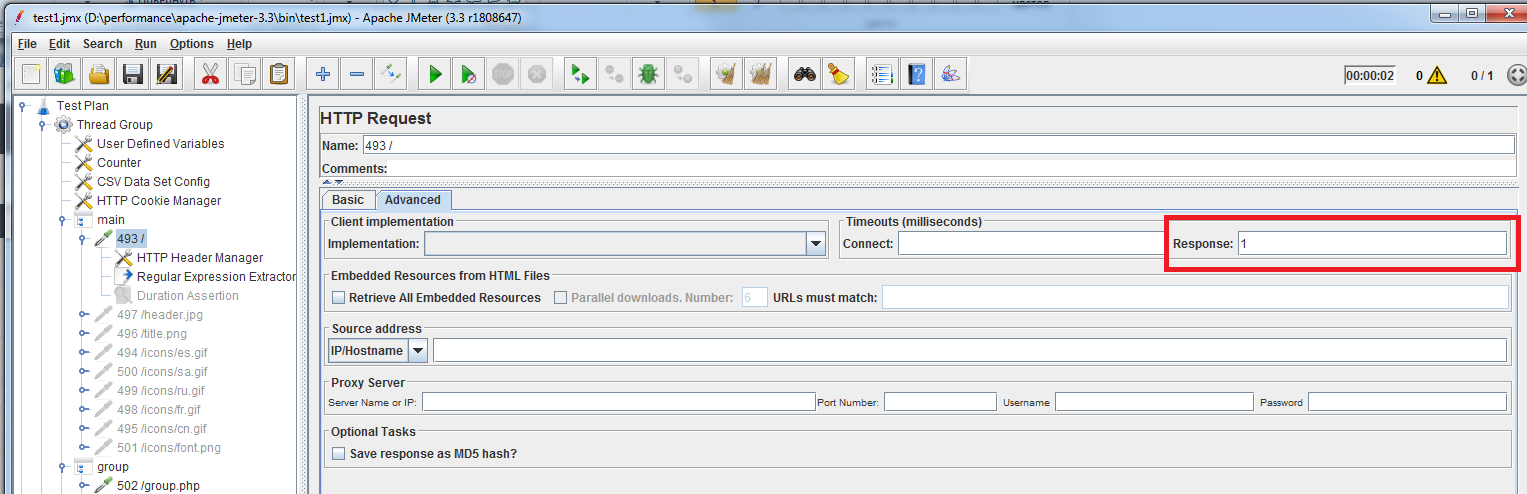
1. Код ответа

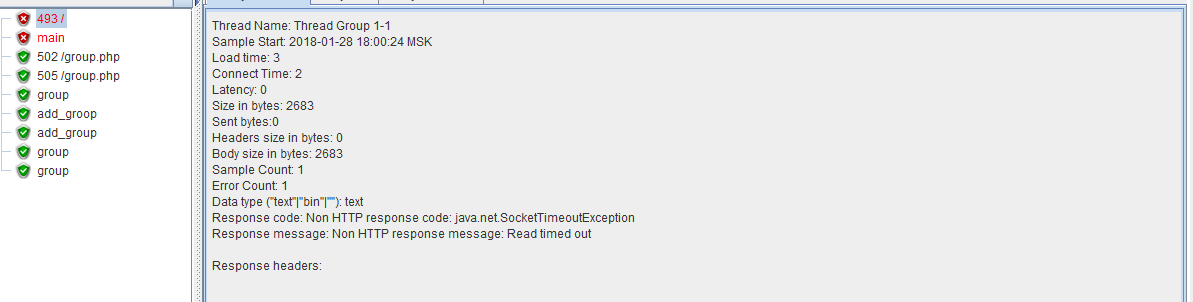
Будет видно в листенере



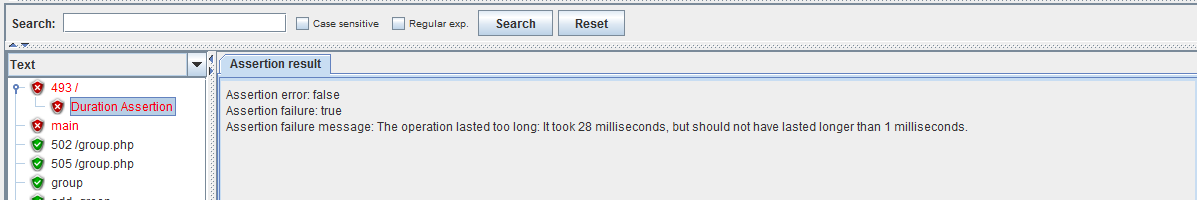
1. Таймауты

Запрос должен выполниться за время указанное в поле Response (в миллисекундах) при этом jmeter не ждет ответа, происходит разрыв соединения. Некая защита от зависания.

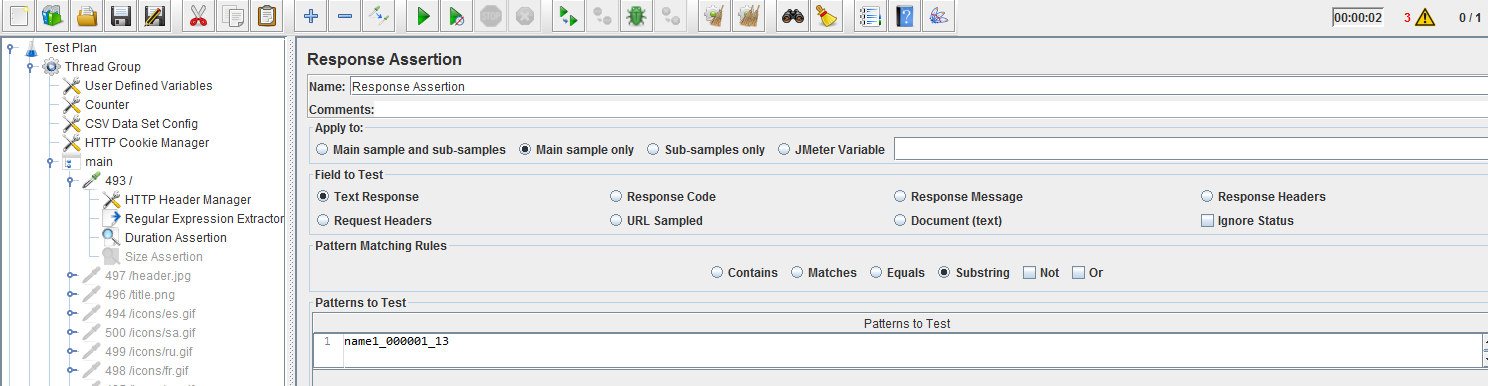




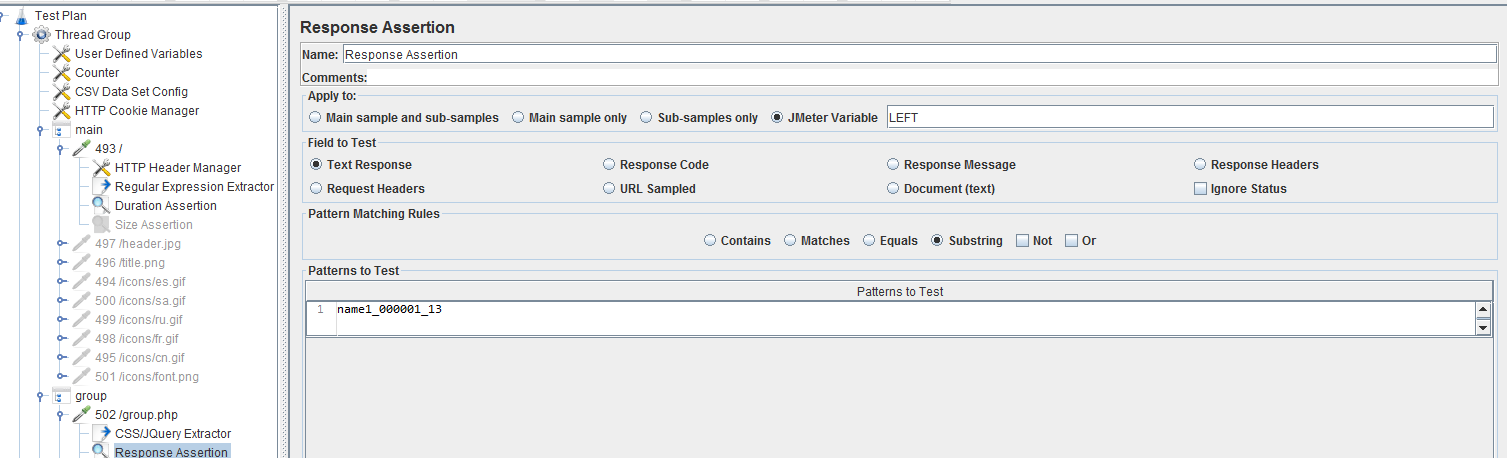
Duration Assertion, ждет ответа и показывает за сколько пришел ответ и сколько он ожидал. Нужен для отслеживания.



1. Размер ответа Size Assertion. Применяется для поиска больших страниц. Некоторые приложения при ошибке могут не вернуть страницу с большим размером.
2. Содержимое ответа Response Assertion



Можно добавить CSS/JQuery Extractor который ищет элемент на странице и присвоить переменной, переменную использовать Response Assertion



1. Что-то сложное. Использовать Bean Shell Assertion, BSF Assertion, JSR223 Assertion

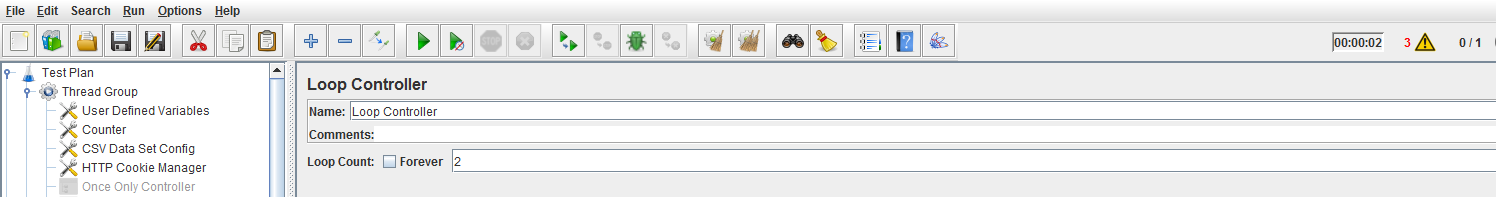
# Управление потоком выполнения

1. Один раз в начале сценария

Once Only Controller – кейс внутри этого элемента выполняется 1 раз.

1. Несколько раз подряд

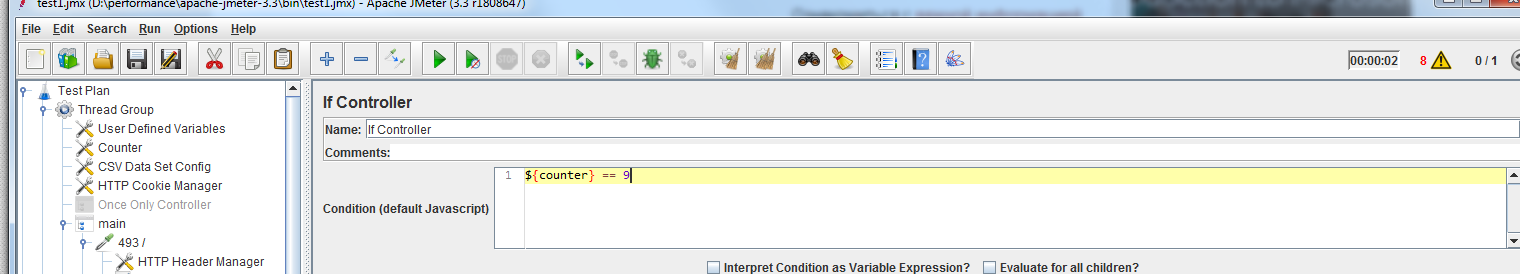
Loop Controller выполнится столько раз, сколько указано в поле Loop Count



While Controller будет выполняться пока переменная не примет значение false

Debug Sampler – показывает значения всех переменных

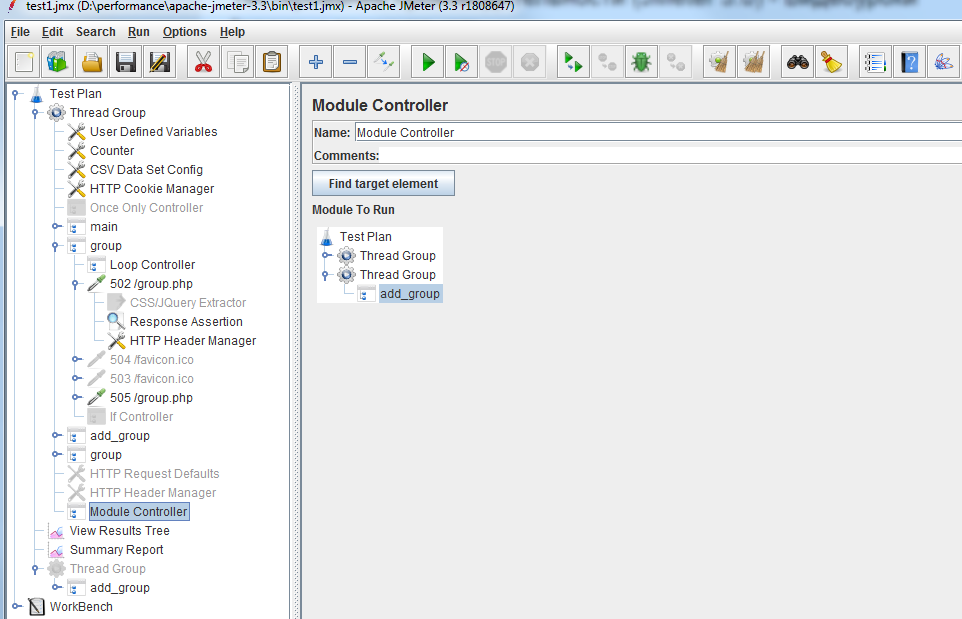
1. Условия выполнения. If Controller



1. Повторное использование

Использовать контроллер jp@gc - Parameterized Controller

Module Controller. Добавить новый Thread Group и задизэйблить его. Внутри поместить операции которые часто используются. И потом добавить Module Controller куда надо



# Протоколы взаимодействия

1. HTTP[S]
2. WebSocket
3. Remote API (SOAP, Rest)
4. Базы данных
5. SMTP, POP3, IMAP
6. WebDriver
7. TCP, UDP
8. Java API

# Как формулировать требования к производительности

**Характеристики качества:**

1. Время отклика (response time)
2. Пропускная способность (throughput)
3. Способность к распараллеливанию (concurrency)
4. Доступность (availability, uptime)
5. Потребление ресурсов (resoutce utilization)
6. Потенциальная мощность (capacity)

**Эволюция требования:**

* Среднее время отклика
* Дисперсия
* Максимальное время отклика
* Доля превышений максимум не более 0,1%
* Доля отказов не более 0,001%
* Отсутствие деградации (сохранение постоянного значения показателя в течении длительного времени)

# Цели тестирования и профили нагрузки

**Цели тестирования:**

* Сравнить две версии системы
* Найти причину проблемы с производительностью
* Оценить потенциальные возможности
* Получить подтверждение, что все хорошо

**Проектирование тестов:**

* Выяснить цель
* Описываем требования и условия
* Формулируем проверяемую гипотезу
* Описываем профиль нагрузки
* Делаем сценарии, которые его реализуют

**Как выбирать транзакции:**

* Критически важные
* Связанные с рисками
* Требуемые заказчиком
* Часто используемые
* Ресурсоемкие

**Профиль нагрузки:**

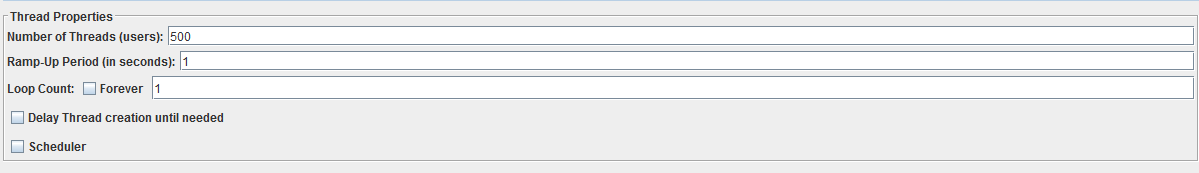
1. Количество виртуальных пользователей
2. Количество сценариев
3. Количество транзакция/запросов
4. Задержки между запросами
5. Задержки между транзакциями

**Виды тестирования:**

* Нагрузочное – постоянная нагрузка, 20-80% от «максимума»
* Стрессовое – непрерывно возрастающая нагрузка
* Спайк тестирование (пиковая нагрузка) – нагрузка с «пиками». Подается постоянная нагрузка, потом не на долго увеличивается, потом опять понижается.
* Тестирование стабильности – нагрузочное тестирование с постоянной нагрузкой в течении длительного времени.

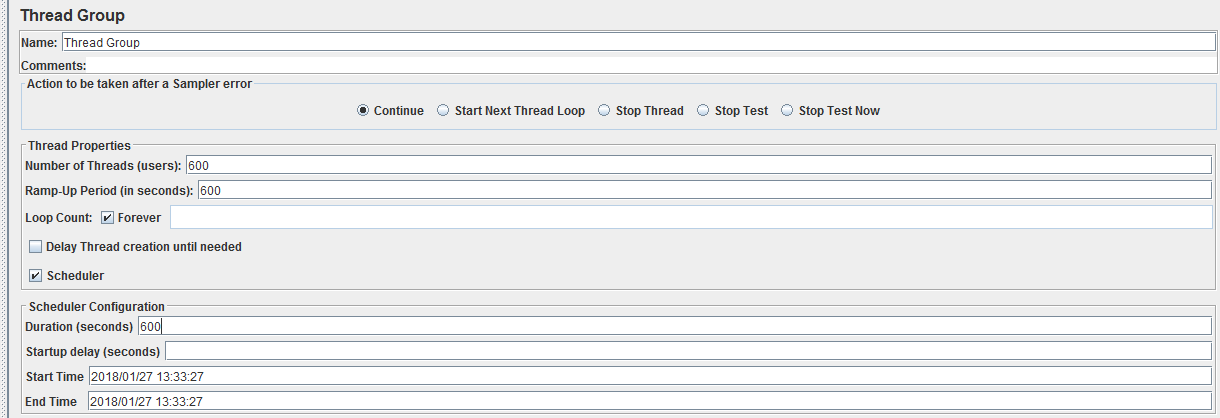
# Реализация профилей нагрузки

Считается что можно запустить 500-600 потоков на одной машине.

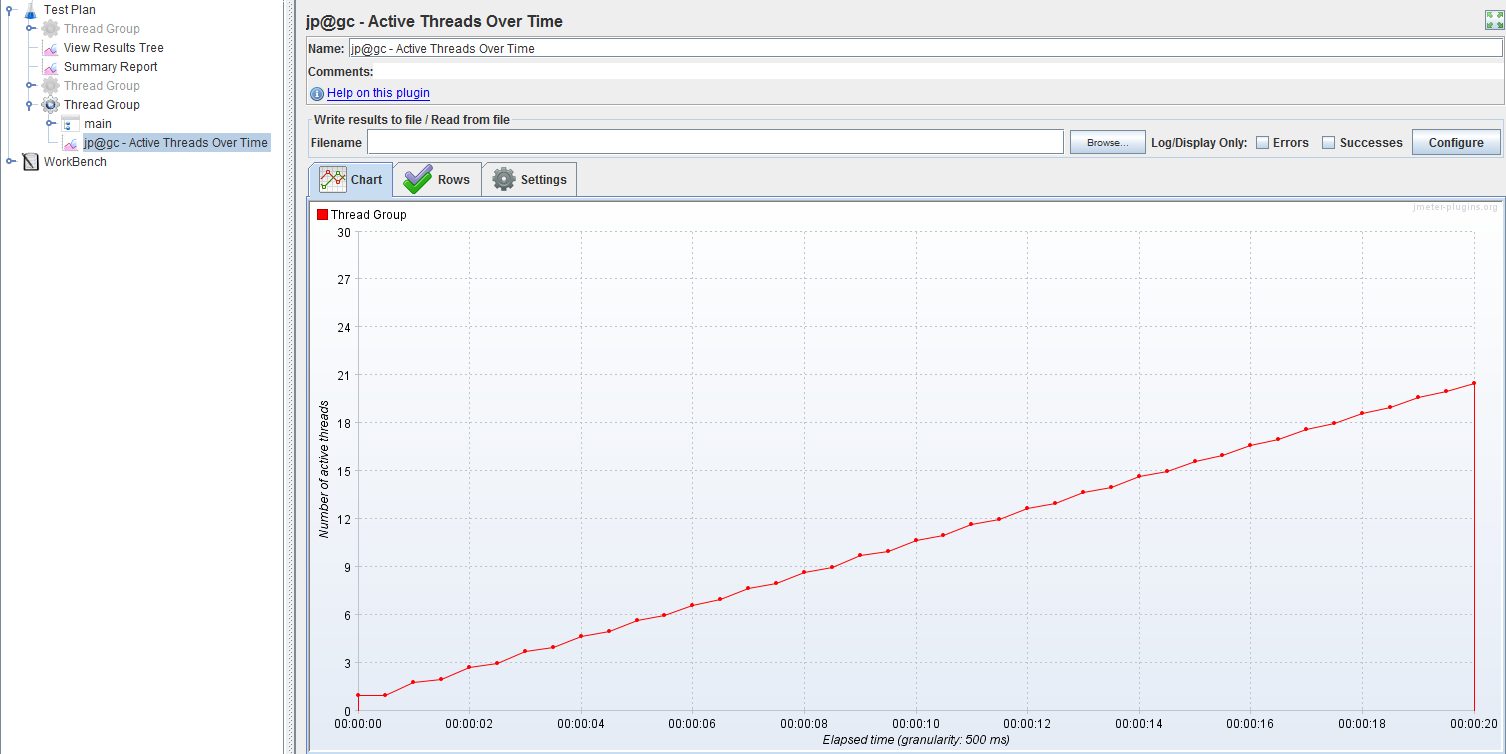


Constant Throughput Timer – подбирает нужные задержки и может обеспечить нужную пропускную способность (количество запросов в еденицу времени). Именно запросов, а не транзакций.

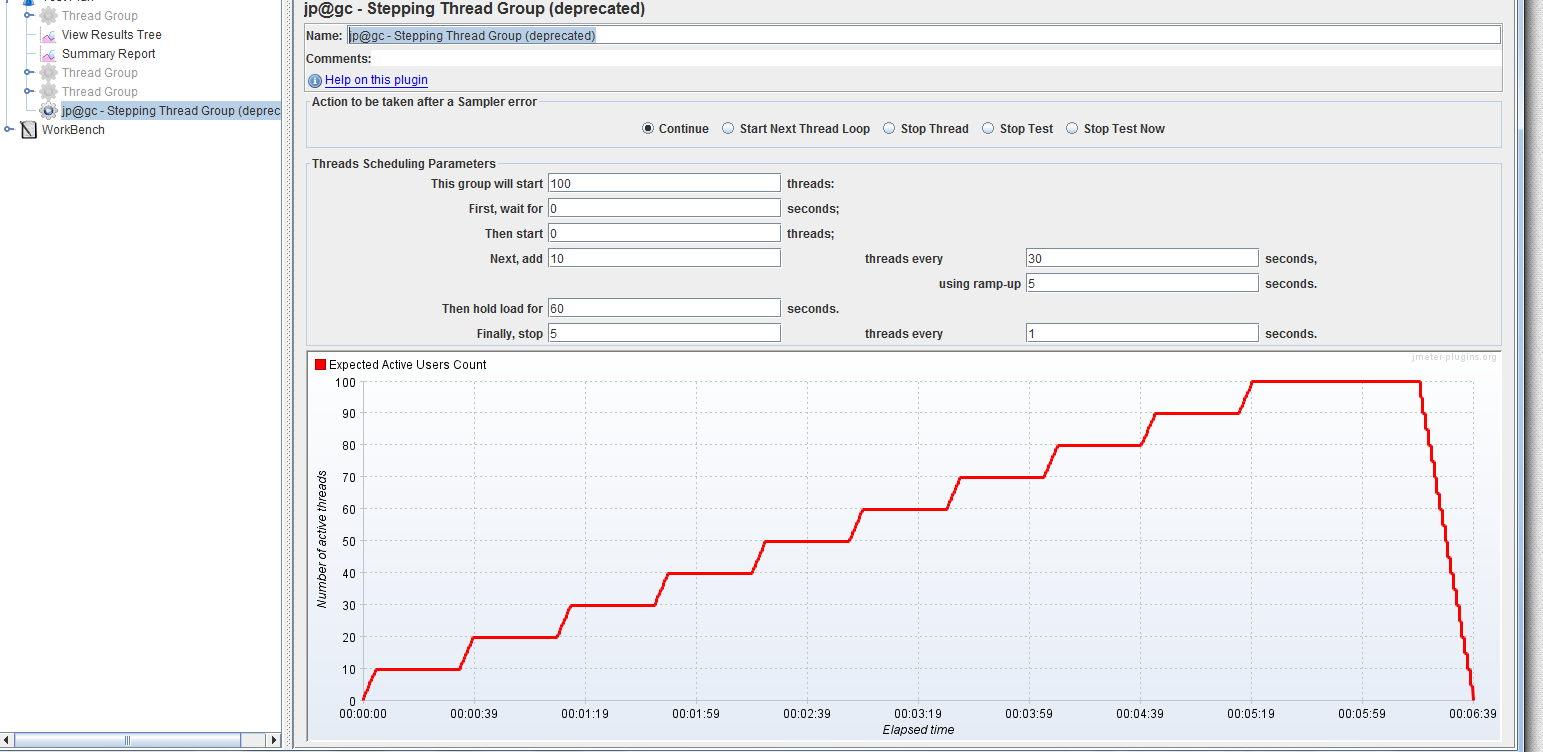
Непрерывная возрастающая нагрузка: чтобы определить пик/точку насыщения указать большое количество виртуальных пользователей и длительное время разогрева.



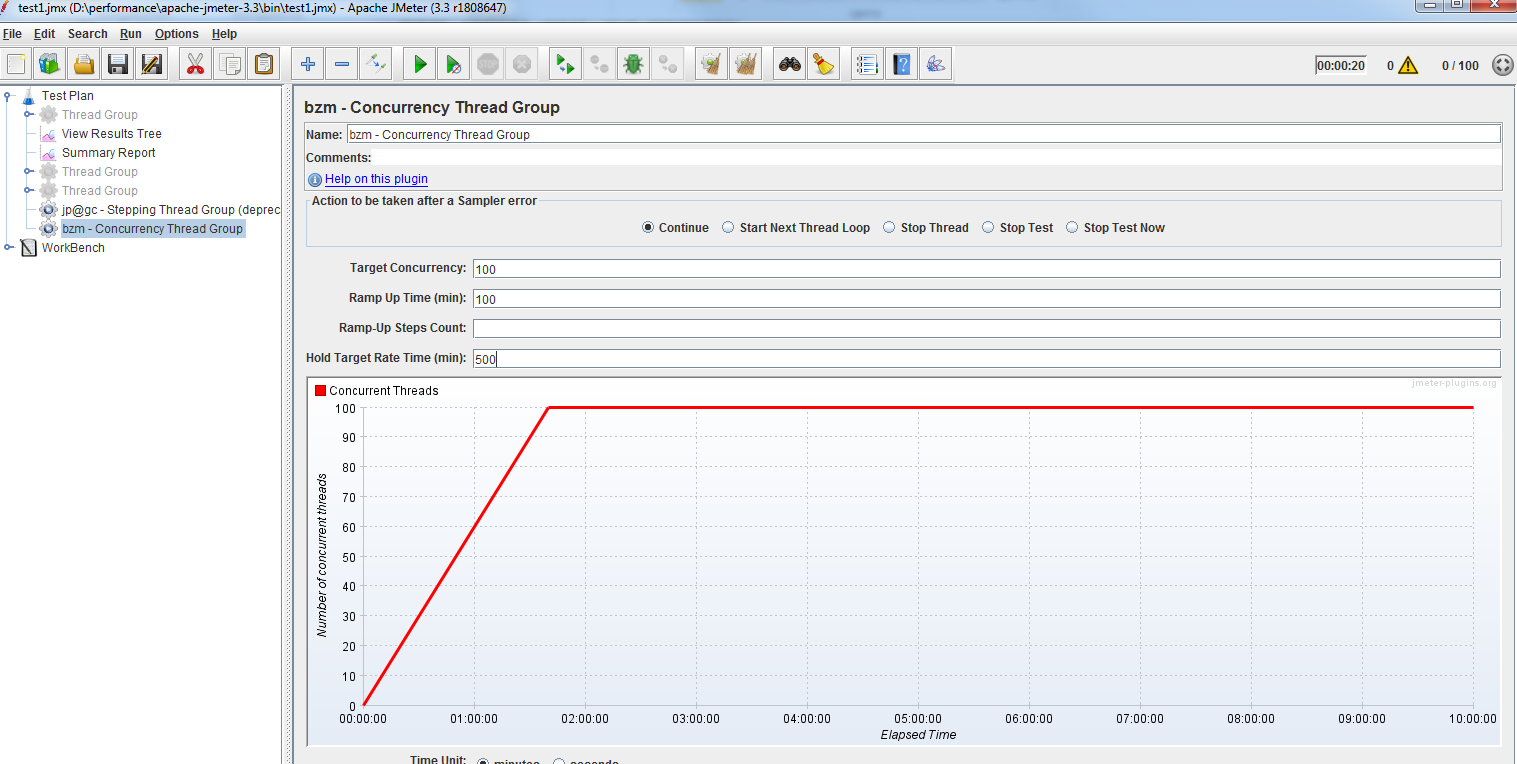
Чтобы посмотреть количество потоков использовать плагин jp@gc - Active Threads Over Time



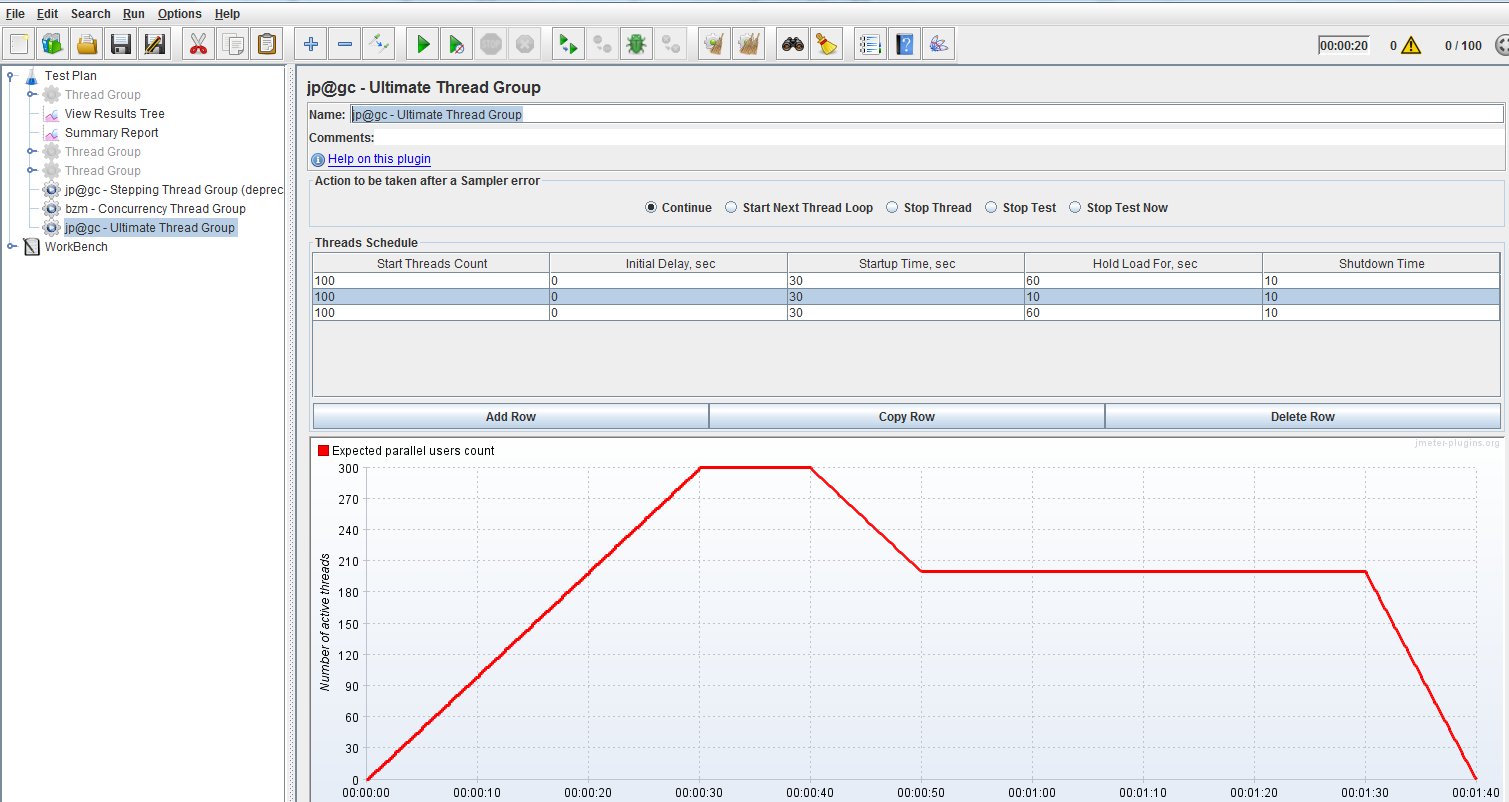
jp@gc - Stepping Thread Group (deprecated)



Схож с bzm - Concurrency Thread Group



Для настраивания сложных профилей нагрузки jp@gc - Ultimate Thread Group



# Задержки

Задержки, зачем они нужны:

* Формирование профиля нагрузки
* Управление частотой запросов
* Пользователи делают паузы:
  + Кеширование в браузере
  + Кеширование на сервере/прокси

Таймер действует на ту группу куда он поставлен.

Таймер срабатывает до запроса. Чтобы тайме сработал после запроса принято добавлять Test Action (Duration = 0) и внутри него таймер

