

# Ускоряем Apache.JMeter

«Ты можешь быстрее. Предела нет. Знай: ты можешь! Будь уверен.»  
(Морфеус, «Матрица»).

## О себе

### Вячеслав Смирнов

Эксперт по тестированию в Райффайзенбанк

Занимаюсь тестированием производительности

Читаю почту:

✉ [owasp@yandex.ru](mailto:owasp@yandex.ru)

Публикую код:

🔗 <https://github.com/polarnik/>

## В этой презентации

- Когда нужна оптимизация
- Известные рекомендации
- Производительность стандартных компонентов
  - результаты исследования

# Варианты подачи нагрузки из JMeter



# Варианты подачи нагрузки из JMeter

По частоте использования выделяю (субъективно) следующие варианты подачи нагрузки из `Apache.JMeter` :

- Встроенные компоненты
- Плагины
- Библиотеки
  - `JSR223 Sampler` и дополнительные библиотеки
  - `JUnit Sampler` и дополнительные библиотеки
- Внешние приложения
  - через `OS Process Sampler`
- Внешние системы
  - по отношению к которым `Apache.JMeter` выступает, как контрольный центр

# Варианты подачи нагрузки из JMeter

Прежде всего поговорим о производительности встроенных компонентов и популярных плагинов для `Apache.JMeter` :

- **Встроенные компоненты**
- **Плагины**
- **Библиотеки**
  - `JSR223 Sampler` и дополнительные библиотеки
  - `JUnit Sampler` и дополнительные библиотеки
- **Внешние приложения**
  - через `OS Process Sampler`
- **Внешние системы**
  - по отношению к которым `Apache.JMeter` выступает, как контрольный центр

## С чего всё начинается

- недовес:
  - нужно 1000 сценариев в минуту
  - видим 800 сценариев в минуту
- перегруз:
  - нужно уложиться в 4 ГБайта ОЗУ
  - ВИДИМ `java.lang.OutOfMemoryError: Java heap space`

И есть скрипт на `Apache.JMeter` , и, возможно, причина в нём.

# Некоторые причины недовеса и перегруза

- Профиль нагрузки:
  - рассчитан неверно
- Ограничения:
  - выставленные ограничения не позволяют использовать доступные системные ресурсы
- Ошибки:
  - ошибки компонентов и скрипта не позволяют загрузить систему
- Неоптимальное использование компонентов:
  - неоптимальные настройки `Apache.JMeter` или неоптимальное использование компонентов



# Некоторые причины недовеса и перегруза

- Профиль нагрузки:
  - ошибка расчёта профиля нагрузки
  - ошибка выставления шага нагрузки
- Ограничения:
  - сработали ограничения JVM
  - сработали ограничения cgroups
- Ошибки:
  - ошибки `Apache.JMeter`
  - ошибки разработки скрипта
- Неоптимальное использование компонентов:
  - недостаточно оперативной памяти, **можно изменить скрипт**
  - недостаточно ресурсов процессора, **можно изменить скрипт**
  - недостаточно соединений, **можно изменить скрипт**

# Что будем делать?

## Известные рекомендации

- Будет корректно рассчитывать профиль нагрузки
- Понимая ограничения JVM, cgroups, системы
- Обходя и не допуская ошибок

## Неизвестные рекомендации

- Исследуем
  - потребление памяти компонентами `Apache.JMeter`
  - потребление ресурсов процессора компонентами
  - общее влияние компонентов `Apache.JMeter` на производительность
- Выберем подходы для типовых задач

# Когда нужна оптимизация

Оптимизация оправдана, когда:

- Применены известные (очевидные) рекомендации
  - Профиль нагрузки известен и рассчитан корректно
  - Известны и настроены ограничения JVM, cgroups, системы
  - Скрипт написан без ошибок (без функциональных ошибок)

Сначала нужно делать то, что нужно делать с начала  
(капитан)

## Профиль нагрузки

- Закрытая модель нагрузки (RPS и потоки ограничены сверху)
- Шаг нагрузки рассчитан верно, потоки рассчитаны верно
- Зависающие запросы прерываются
- Ошибочные тесты прерываются

## Ограничения JVM и системы

- операционная система имеет ограничения и настройки
  - в операционной системе выполняются процессы
- для процессов есть ограничения и настройки
  - java-машина выполняет Apache.JMeter и его компоненты
  - некоторые компоненты Apache.JMeter запускают новые процессы
- 
-

# Ошибка расчёта профиля нагрузки

## Thread Group и одноразовые пользователи без таймеров

Дано:

- нужна интенсивность 16 в секунду

Ошибиться можно с математикой или пониманием

- Thread Properties
  - Number of Threads (users): 16000
  - Ramp-Up Period (in seconds): 1000

## **Ошибка выставления шага нагрузки**

## **Известные рекомендации**



## **Производительность стандартных компонентов**

## Постпроцессоры

- Regular Expression Extractor
- CSS Selector Extractor (was: CSS/JQuery Extractor )
- XPath2 Extractor
- XPath Extractor
- Result Status Action Handler
- BeanShell PostProcessor
- JSR223 PostProcessor
- JDBC PostProcessor
- JSON Extractor
- Boundary Extractor
- Debug PostProcessor

[http://jmeter.apache.org/usermanual/component\\_reference.html](http://jmeter.apache.org/usermanual/component_reference.html)

## Постпроцессоры для извлечения значений из ответа

### Для HTML

- Regular Expression Extractor
- CSS Selector Extractor (was: CSS/JQuery Extractor )
- XPath2 Extractor
- XPath Extractor
- BeanShell PostProcessor
- JSR223 PostProcessor
- Boundary Extractor

## Постпроцессоры для извлечения значений из ответа

### Для XML

- Regular Expression Extractor
- XPath2 Extractor
- XPath Extractor
- BeanShell PostProcessor
- JSR223 PostProcessor
- Boundary Extractor

## Постпроцессоры для извлечения значений из ответа

### Для JSON

- Regular Expression Extractor
- BeanShell PostProcessor
- JSR223 PostProcessor
- Boundary Extractor









## **Известные проблемы производительности**



## **Решение долгих и больших задач**

## Отправка больших HTTP-запросов

## Получение больших HTTP-ответов



