Лабораторная работа №6

### Нагрузочное тестирование

*Цель работы:* научиться производить нагрузочное тестирование.

*Отчет по лабораторной работе:* отчет о проведении нагрузочного тестирования в Apache JMeter.

## Теоретическая часть

Современное программное обеспечение просто обязано бесперебойно работать под колоссальными нагрузками. Любого рода проблемы, связанные с плохой производительностью, могут стать причиной отказа клиентов от использования вашего ПО. В связи с этим, проведение качественного нагрузочного тестирования должно стать обязательным, для обеспечения стабильности работы ваших приложений.

**Нагрузочное тестирование** или **тестирование производительности** - это автоматизированное тестирование, имитирующее работу определенного количества бизнес-пользователей на каком-либо общем (разделяемом ими) ресурсе.

Начиная работу в области нагрузочного тестирования, следует четко понимать, что это не просто запись и прогон (Record and Playback) скриптов, а более сложный процесс:

* Во-первых, нагрузочное тестирование - это серьезная исследовательская и аналитическая работа;
* Во-вторых - это реальное автоматизированное тестирование, требующее серьезных навыков программирования, а также знания сетевых протоколов и различных серверов приложений и баз данных;
* В-третьих - существуют разные виды нагрузочного тестирования, ставящие перед собой разные цели.

#### Терминология

В качестве примера тестируемого ПО можно привести работу сотрудников современного банка, в котором все работают с одними и теми же программными приложениями, установленными на банковских серверах. Или использование программного приложения «веб-магазин», в данном случае посетителями, нагружающими сервера, будут пользователи интернета.

1. Виртуальный пользователь (Virtual User) - программный процесс, циклически выполняющий моделируемые операции
2. Итерация (Iteration) – это один повтор выполняемой в цикле операции
3. Интенсивность выполнения операции (Operation Intensity) - частота выполнения операции в единицу времени, в тестовом скрипте задается интервалом времени между итерациями
4. Нагрузка (Loading) - совокупное выполнение операций на общем ресурсе (тр./сек, хитов/сек)
5. Производительность (Performance) - количество выполняемых операций за период времени (N операций за M часов)
6. Масштабируемость приложения (Application Scalability) - пропорциональный рост производительности при увеличении нагрузки
7. Профиль нагрузки (Performance Profile) - это набор операций с заданными интенсивностями, полученный на основе сбора статистических данных либо определенный путем анализа требований к тестируемой системе
8. Нагрузочной точкой называется рассчитанное (либо заданное заказчиком) количество виртуальных пользователей в группах, выполняющих операции с определенными интенсивностями

Теперь рассмотрим, как эти сущности связаны между собой. Выразив интенсивность через интервал времени между итерациями, видим что рост интенсивности выполняемых операций это сокращение интервала времени. Рост нагрузки пропорционален росту интенсивности. Естественно также, что при увеличении интенсивности растет производительность. При этом увеличивается степень использования (загруженности) ресурсов. С какого-то момента рост производительности прекращается (а нагрузка может продолжать расти), происходит насыщение и затем деградация системы. В дополнение можно заметить что при тестировании изменение интенсивности операций может подчиняться какому либо закону (например, Пуассона) либо быть равномерным в течении всего теста.

#### Основные виды тестирования производительности

Рассмотрим основные виды нагрузочного тестирования, также задачи стоящие перед ними.

##### Тестирование производительности (Performance testing)

Задачей тестирования производительности является определение масштабируемости приложения под нагрузкой, при этом происходит:

* измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций
* определение количества пользователей, одновременно работающих с приложением
* определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций)
* исследование производительности на высоких, предельных, стрессовых нагрузках

##### Стрессовое тестирование (Stress Testing)

Стрессовое тестирование позволяет проверить, насколько приложение и система в целом работоспособны в условиях стресса и также оценить способность системы к регенерации, т.е. к возвращению к нормальному состоянию после прекращения воздействия стресса. Стрессом в данном контексте может быть повышение интенсивности выполнения операций до очень высоких значений или аварийное изменение конфигурации сервера. Также одной из задач при стрессовом тестировании может быть оценка деградации производительности, таким образом цели стрессового тестирования могут пересекаться с целями тестирования производительности.

##### Объемное тестирование (Volume Testing)

Задачей объемного тестирования является получение оценки производительности при увеличении объемов данных в базе данных приложения, при этом происходит:

* измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций
* может производиться определение количества пользователей, одновременно работающих с приложением

##### 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.

#### Цели нагрузочного тестирования

Основными целями нагрузочного тестирования являются:

* Оценка производительности и работоспособности приложения на этапе разработки и передачи в эксплуатацию
* Оценка производительности и работоспособности приложения на этапе выпуска новых релизов, патч-сетов
* Оптимизация производительности приложения, включая настройки серверов и оптимизацию кода
* Подбор соответствующей для данного приложения аппаратной (программной платформы) и конфигурации сервера

В рамках одной цели могут использоваться разные виды тестов производительности и нагрузки, например, для первой, второй и третьей цели нужно производить как тестирование производительности так и тестирование стабильности. Но при планировании нагрузочного тестирования логичнее все же отталкиваться от технических целей (а не коммерческих, перечисленных выше), которые достигаются в результате тестирования и классифицировать тесты по ним:

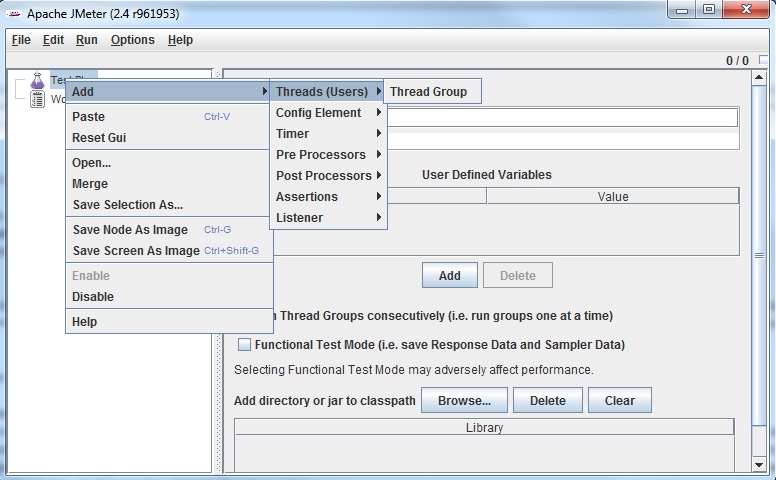
1. Если интересует исследование производительности приложения, а именно времена отклика для операций на разных нагрузках в довольно широких диапазонах, включая стрессовые нагрузки то это все-таки тестирование производительности (Performance Testing)
2. Если целью является понимание насколько приложение устойчиво в режиме длительного использования (исключение утечек памяти, некорректных конфигурационных настроек и т.д.) то проводится долгий нагрузочный тест - это тестирование стабильности (Stability Testing). При этом анализ времен отклика может иметь место, но не быть первым приоритетом, главное чтобы система "не упала".
3. Стресс-тестирование (Stress Testing) имеет своей целью проверить, возвращается ли система после запредельной нагрузки (и как скоро) к нормальному режиму, также целями стрессового тестирования могут быть проверки поведения системы в случаях когда, один из серверов приложения в пуле перестаёт работать, аварийно изменилась аппаратная конфигурации сервера базы данных и т.д. Отметим также, что при стрессовом тестировании проверяется не производительность системы, а её способность к регенерации после сверх нагрузки.

#### Нагрузочное тестирование с помощью Apache JMeter

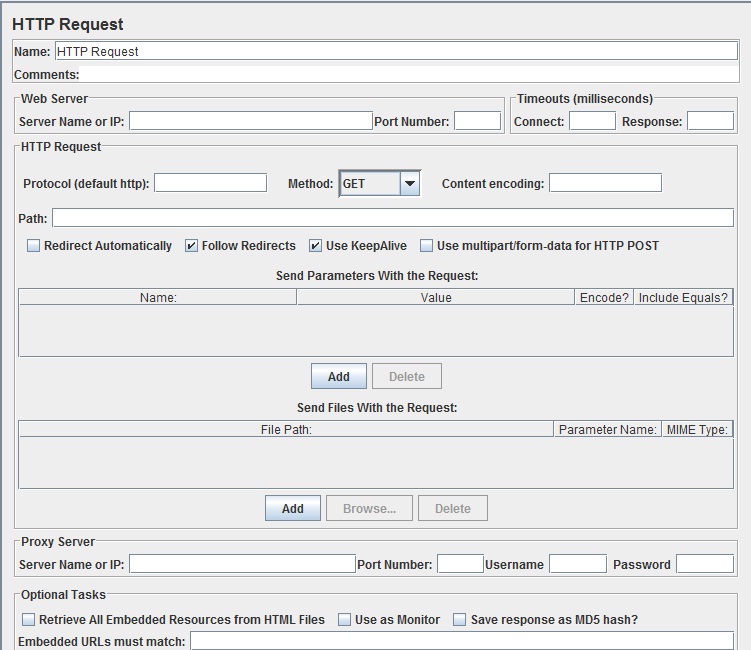
Для любого программного приложения, предназначенного для массового обслуживания пользователей, необходимо проводить нагрузочное тестирование на предмет его надежности и отказоустойчивости. А так как любой web-сайт — это по своей сути система массового обслуживания, то проверка его на отказоустойчивость всегда является неотъемлемой частью разработки.

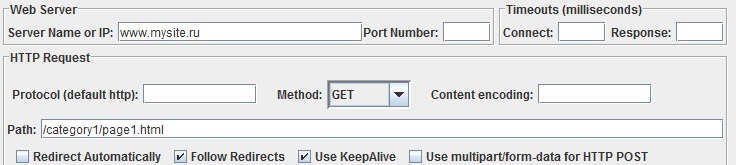
Ниже приведено, каким образом можно осуществить нагрузочное тестирование сайта в JMeter.

1. Запускаем JMeter.

2. Создаем Thread group:  


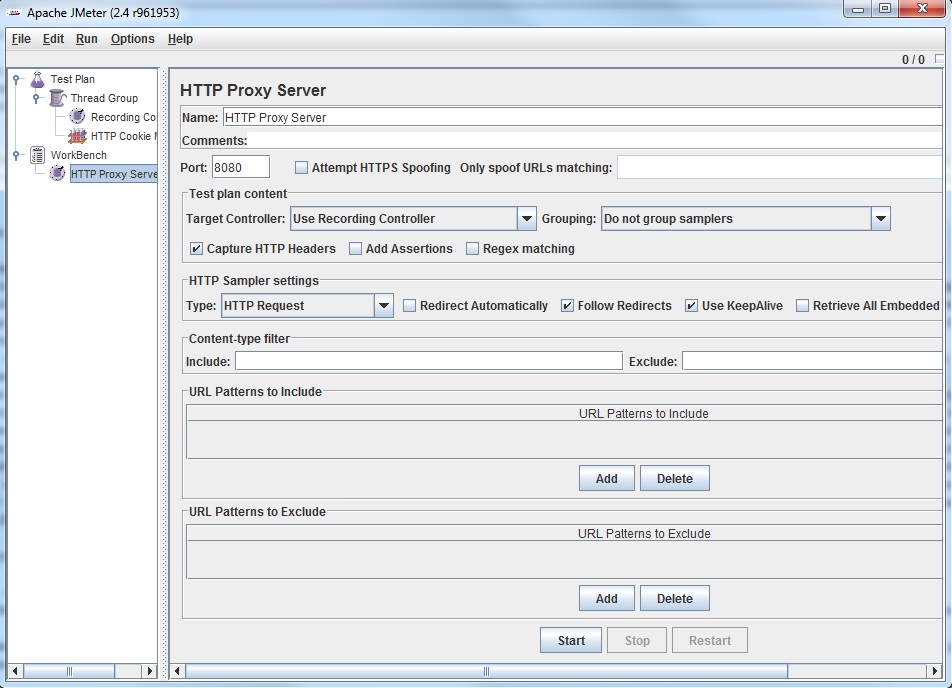
Это основной рабочий процесс, в котором мы будем записывать сценарии, добавлять различную логику и элементы управления.

Теперь нужно создать собственно сценарий теста, т.е. набор различных действий для создания нагрузки на сайт. Можно создавать сценарий вручную. Это очень просто, для этого нужно добавить N-ое количество элементов HTTP Request, которые добавляются так Thread Group -> Add -> Sampler -> HTTP Request. Появляется окно настроек, представленное на рисунке ниже:  
  


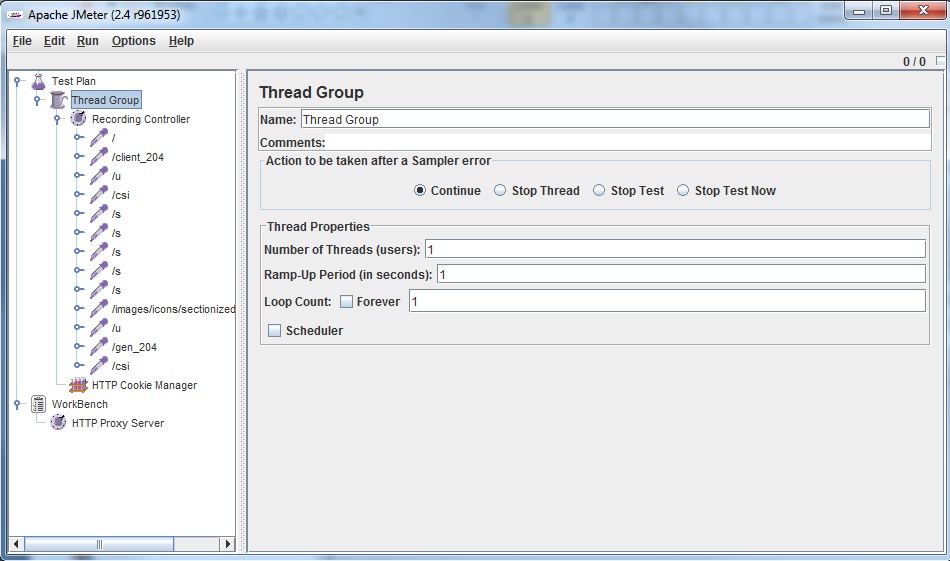
В соответствующие поля устанавливаем адрес сайта, порт (если нужно), путь к странице. Например, так:  
  


В JMeter есть возможность автоматизации действий, он называется «запись тестов через проксирование». Т.е. мы будем выполнять любые действия через браузер, и при этом все необходимые элементы HTTP Request будут создаваться без нашего участия.

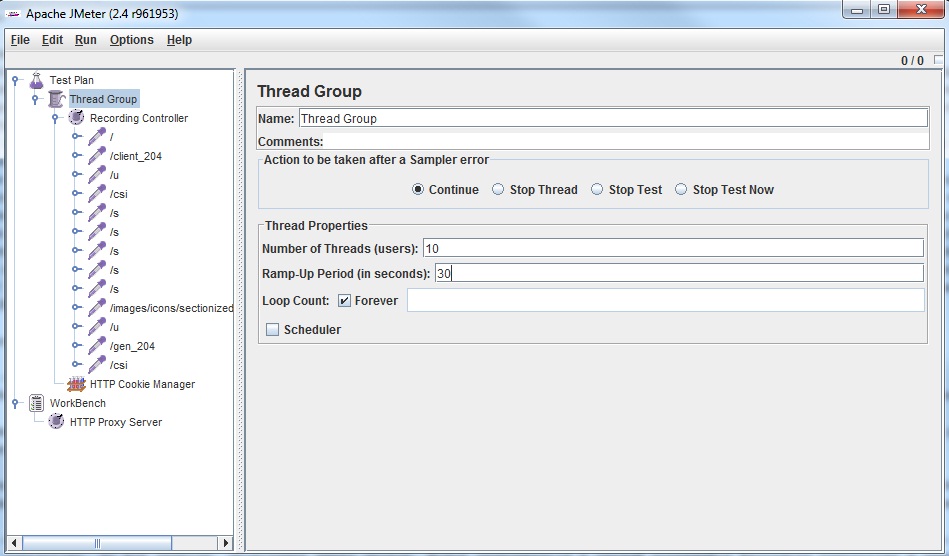
3. Добавляем Recording Controller (Thread Group -> Add -> Logic Controller -> Recording Controller). В данный элемент будут сохраняться все наши действия, которые мы будет делать в браузере.

4. Добавляем элемент HTTP Proxy Server. Добавлять его надо в раздел WorkBench (WorkBench -> Add -> Non-Test Elements -> HTTP Proxy Server), так как непосредственно в ходе тестирования этот элемент не будет принимать участия. Он нам нужен только, чтобы создать сценарии тестов. Здесь мы видим множество настроек данного элемента:  
  


Здесь достаточно только изменить номер порта прокси-сервера, если порт по умолчанию 8080 у вас уже занят, например, можно поставить 8089. Если оставить в графе Target Controller значение Use Recording Controller, то все запросы, проходящие через прокси, будут записываться в первый попавшийся Recording Controller в нашем тест-плане. Но так как на данный момент он там всего один, то нас этот вариант устроит. Далее рекомендую обратить внимание на настройку фильтрации. Эти настройки предоставляют широкие возможности по фильтрации запросов.

5. В настройках браузера нужно указать адрес прокси и порт. Убедиться, что браузер ходит в интернет именно через нее, для этого перейти на любой сайт в интернете, при этом страница не должна загрузиться. Далее осталось в окне с настройками HTTP Proxy Server нажать на кнопку Start, запустив тем самым прокси. После этого через браузер, открываем страницу тестируемого сайта, логинимся, выполняем различные действия, посещаем разных страницы, при этом сценарий уже начнет сохраняться в элемент Recording Controller. Когда вы поймете, что записанный вами сценарий уже достаточен, останавливаете прокси.  
Получили, примерно, следующую картину:  
  


Названия элементов могут оказаться мало о чем говорящими. Поэтому чтобы в дальнейшем было просто наблюдать за ходом теста, эти элементы имеет смысл переименовать.

6. В настройках элемента Thread Group выберите количество потоков, число итераций, время прогрева. Например, я установил такие настройки:  
  


Это значит, что тестирование будет проходить в 10 потоков, будет длиться бесконечно пока, его принудительно не остановят. Время прогрева я поставил 30 сек. Это значит, что потоки будут равномерно стартовать в течении 30 секунд, т.е. каждые три секунды, будет запускаться новый поток.

7. Чтобы наблюдать результаты тестов, а также следить за ходом выполнения, нужно добавить несколько элементов мониторинга. Я обычно добавляю такие: View Results Tree, View Results in Table, Graph Results, Summary Report.

**Graph Results** отобразит результат в виде графика:  


Значения предоставлены в миллисекундах.

Data — время отклика каждой отдельной единицы данных, т.е. каждого проверенного url.

Average — усредненное время отклика, объективный график изменения нагрузки.

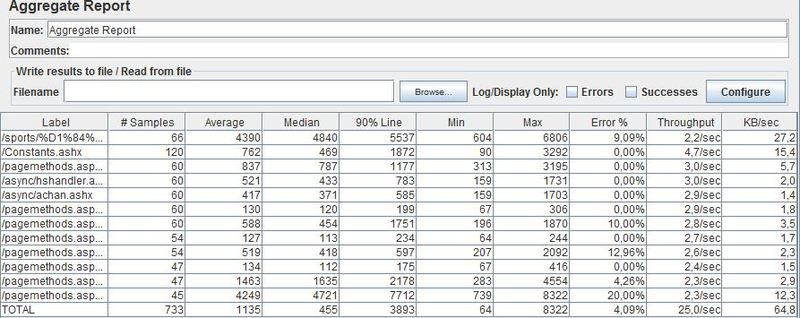
Median — значение медианы.

Deviation — погрешность, стандартное отклонение.

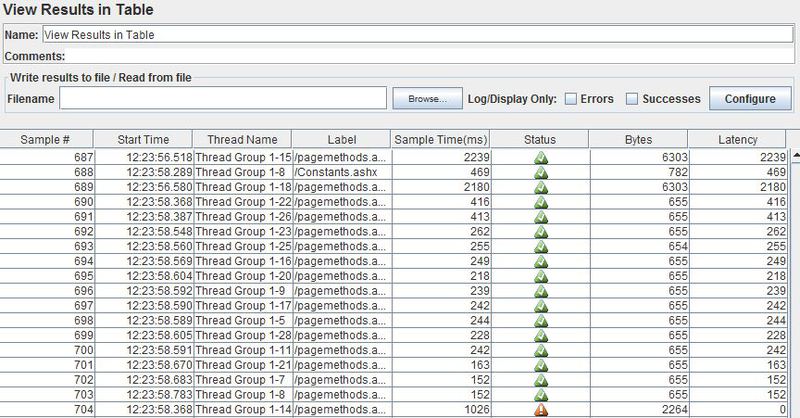
Throughput — пропускная способность выполняемых запросов.

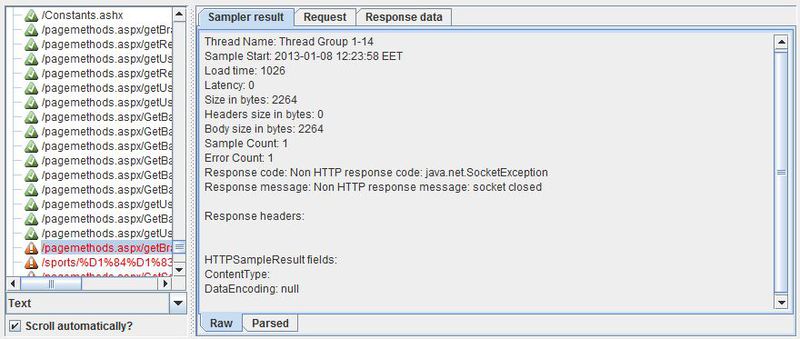
Для работы достаточно значений Average и Throughput, которые отобразят нагрузку на веб-сервер и пропускную способность запросов. По графику выше видно, что время отклика примерно 200мс и не растет, то есть, сервер нормально выдерживает нагрузку в 3 виртуальных пользователя. А вот что получится, если их будет 30:  
  


Время отклика значительно увеличивается, чем выше пропускная способность запросов (меньше миллисекунд), тем больше времени требуется серверу для обработки.

**Aggregate Report** отобразит статистику по каждому индивидуальному url отдельно.  
  


В столбце Average видим среднее время отклика, чем оно больше — тем больше нагрузка на данный url.

**View Result in Table** отобразит результат в виде таблицы, здесь указано время, а также статус (успешно/не успешно).  
  


В нижней строке ошибка. Почему именно она возникла, можно проверить в**View Result Tree**. Зайдем туда и найдем эту строку. Теперь видим причину ошибки.  


#### Задание

Осуществить нагрузочное тестирование сайтов по вариантам из лабораторной работы №1, с использованием тест-кейсов лабораторной работы №2. Предоставить отчет о нагрузочном тестировании, проанализировать графики и показатели производительности.