ソフトウェアテストシンポジウム2007東京

Apache JMeterで負荷試験をしよう！

水野浩典
コアテクノロジー部
コンサルティング・インテグレーション統括本部
日本ヒューレットパッカード株式会社

2007年1月30日

内容

- 負荷試験の重要性
- Apache JMeter
  - 概要、入門、実践編
- 負荷試験方法
  - プロジェクトにおける負荷試験の実例
  - 負荷試験方法のセオリー

負荷試験の重要性

- 必要な理由
  - 負荷試験は、単体テストや結合テストでは発見することが出来なかった問題を見つけるために**絶対に必要**
- 必要な理由 (その2)
  - 想定外の負荷が掛かることも考慮し、**想定外**の負荷試験を実施することも重要
- 本当の理由
  - システムのカットオーバー後の夜中や休日に、緊急に呼び出されないようにするため！

現実に発生している問題

- 負荷試験が、そもそも計画に入っていない。
- 納期に間に合わないので、機能試験はしたが負荷試験まで出来なかった。

負荷試験の重要性をプロジェクト内で認識し、実施するようにして下さい。

- 負荷試験を実施するには、商用製品が必要なようだが、予算的に買えない。

Freeで使用できるApache JMeterの使用を検討して下さい。

- 試験ツールの使い方が良く分からない。

今から、御説明致します。

- とりあえず、負荷試験をしてみたが、試験結果の評価がよく分からない？
- 以前に負荷試験をしてみたが、大変だったので出来ればやりたくない。

経験によるところも多いのですが、負荷試験の基本的な方法について御説明致します。

Apache JMeter概要

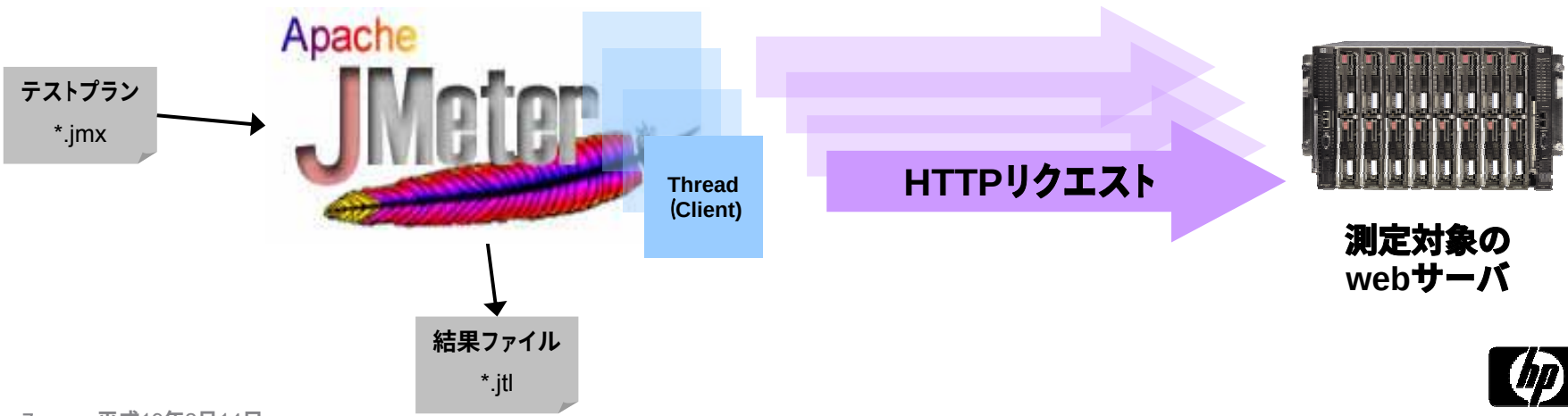
JMeterとは

JMeterはJakartaプロジェクトが開発しているパフォーマンス計測/負荷テストツール

- オープンソース、無償
- 100% pure JavaなのでWindows、Linuxなど、実行環境に依存しない
- GUIを用いて操作できる
- 大量のクライアントからのリクエストの生成/実行が可能
- HTTP(HTTPS)、データベース、FTP、WEBサービスなどの様々なプロトコルでの負荷試験やパフォーマンス検証が可能
- エラー発生の有無、期待したデータがレスポンスに含まれているかの確認なども簡単に行うことが可能
- Webブラウザ上で行った操作をテストのシナリオとして記録し、それをテスト・スクリプトとして繰り返し利用、簡単に編集可能

JMeterの動作概要

- 複数クライアントの実行
  - Javaのスレッドを用いて、複数クライアントの同時実行を実現
- 実行
  - JMeterにテストプランを設定して、これを実行する
- テストプラン
  - 手動で作成するか、プロキシ機能を用いて自動で作成
  - 生成するスレッド数（クライアント数）と実行回数を指定



Apache JMeter入門

JMeterのインストール

ファイルをダウンロードし、適当なディレクトリに展開するだけです。

1. JMeterのダウンロード
   Jakartaサイトの以下ページより最新のJMeterバイナリをダウンロード
   http://jakarta.apache.org/site/binindex.cgi#jmeter_binaries
   ※最新版は JMeter2.2です。
   ここではjakarta-jmeter-2.2.zipをダウンロード

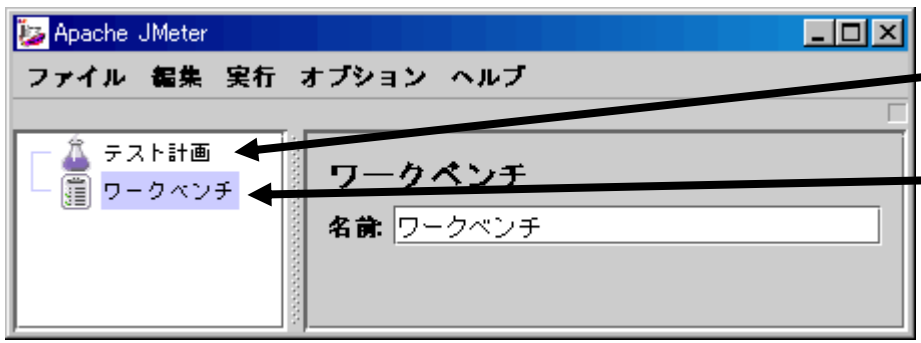
2. JMeterのインストール
   ダウンロードしたzip／tarファイルをJMeterをインストールしたいディレクトリに展開

JMeterの起動

JMeterを展開してできたディレクトリにあるbinディレクトリのJMeter起動ファイル
jmeter.bat をダブルクリックして下さい。

jmeter.bat

正しく起動されていれば、コマンドプロンプトと以下の通りの画面が表示されます。



テスト計画
実際のテストシナリオを作成する場所

ワークベンチ
テストに使用しない要素を一時的保管
HTTPプロキシを動作させる場所

1．スレッドグループの作成

JMeterのテスト実行計画を作成する場合、まず**ユーザ（スレッドグループ）を作成**します。

テスト計画を選択して、右クリックします。
追加⇒スレッドグループを選択します。

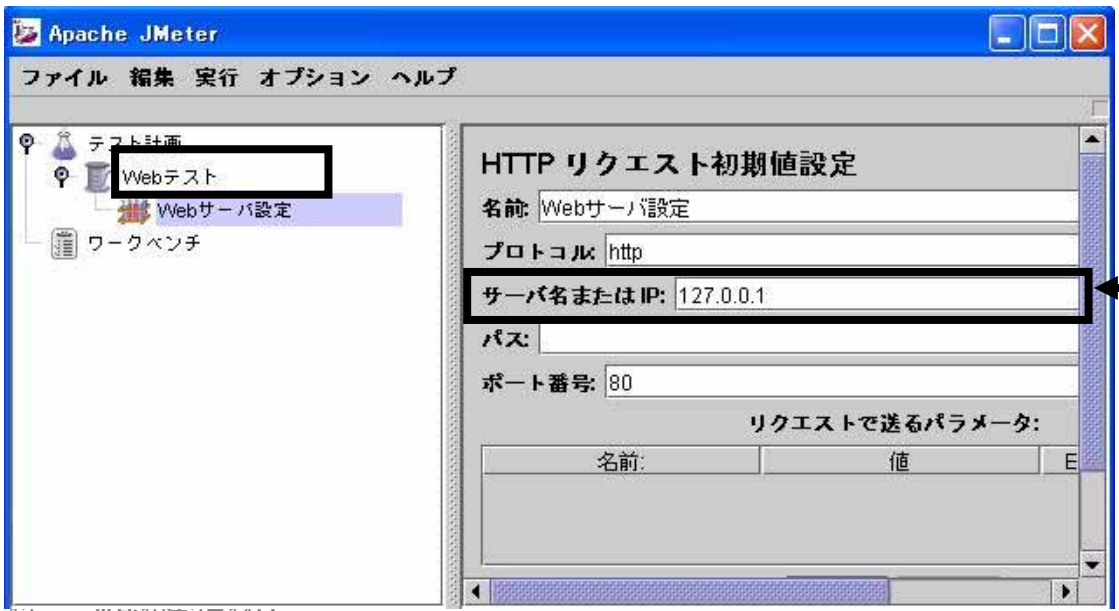
わかりやすい名前を付けます。

実施すべきテストプランに沿った
スレッド数（ユーザ数）等の設定をします。

2．デフォルトHTTPリクエストの作成

スレッド**共通のHTTPリクエストのデフォルト値**を設定エレメントの作成より設定します。
※その他　設定すべき設定エレメントがあれば設定します。

作成したスレッドグループを選択して、右クリックします。
追加⇒設定エレメント⇒HTTPリクエスト初期値設定を選択します。

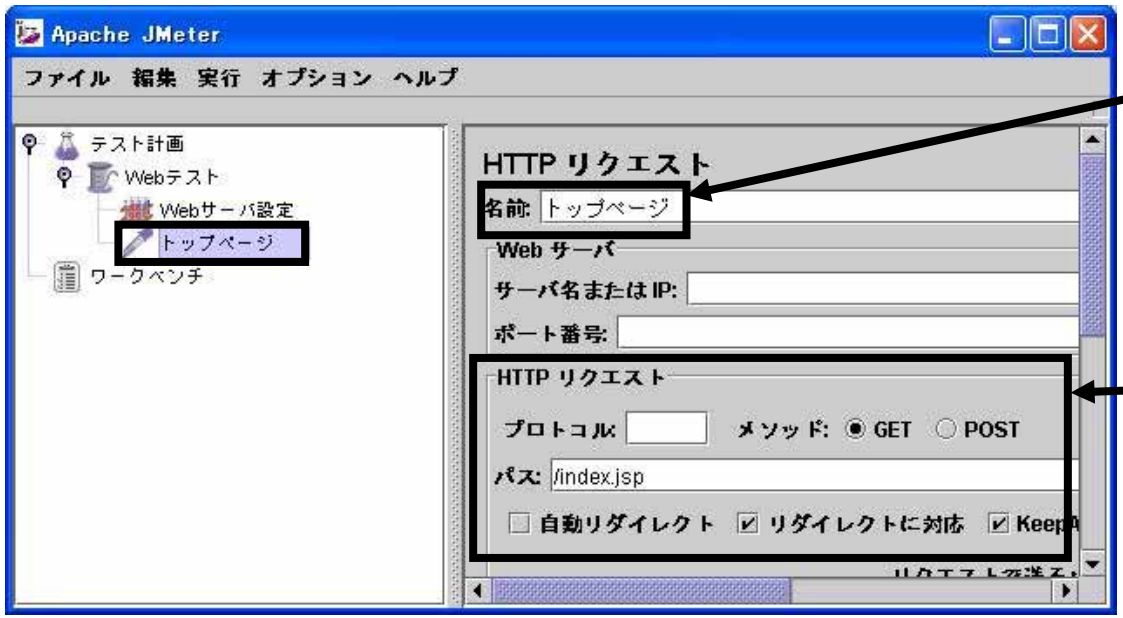


テスト対象のサーバ名/IPを定義します

3．サンプラーの作成

実際にHTTPリクエスト処理を行うものとして、サンプラーを作成します。

作成したスレッドグループを選択して、右クリックします。
追加⇒サンプラー⇒HTTPリクエストを選択します。



わかりやすい名前を付けます。

アクセス対象は、設定エレメントで設定してます。
ここでは、実際のHTTPリクエストのパス等のみ設定します。

4．リスナーの作成

テスト結果の表示と保存のためにリスナーを追加します。
※リスナーは、各々のテストプランにおいて必要なものを設定します。

作成したスレッドグループを選択して、右クリックします。
追加⇒リスナー⇒統計レポートを選択します。

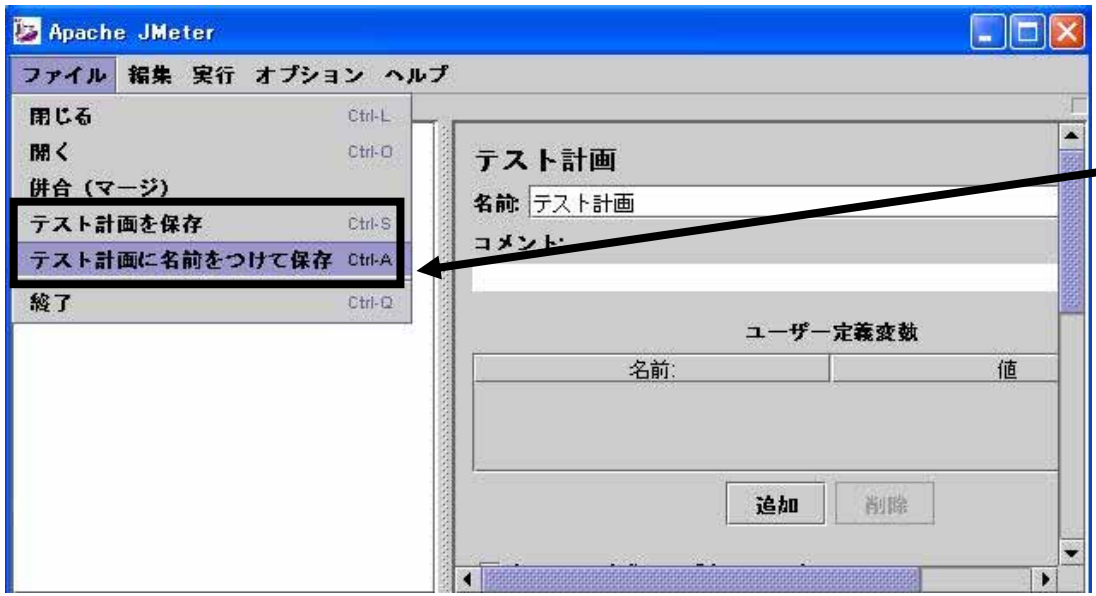
結果をファイルに保存できます。

サンプラーの処理結果がそれぞれの
リスナーに応じた形で動的に表示されます。

5．テストプランの保存

テストプランを実行する前に、**テストプランの保存**をします。

メニューの**ファイル⇒テスト計画を保存**
または、**ファイル⇒ テスト計画に名前をつけて保存**を選択します。

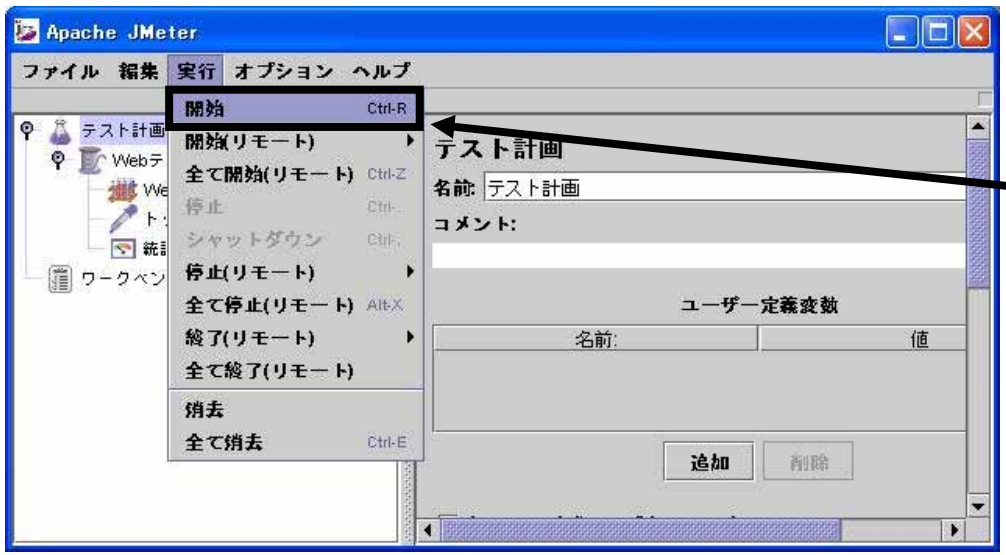


わかりやすい名前をつけて保存します。

6．テストプランの実行

テストプラン実行開始の準備が完了後、**テストプランの実行**を開始します。

メニューの**実行⇒開始**を選択、または、『Ctrl』+『R』を押します。

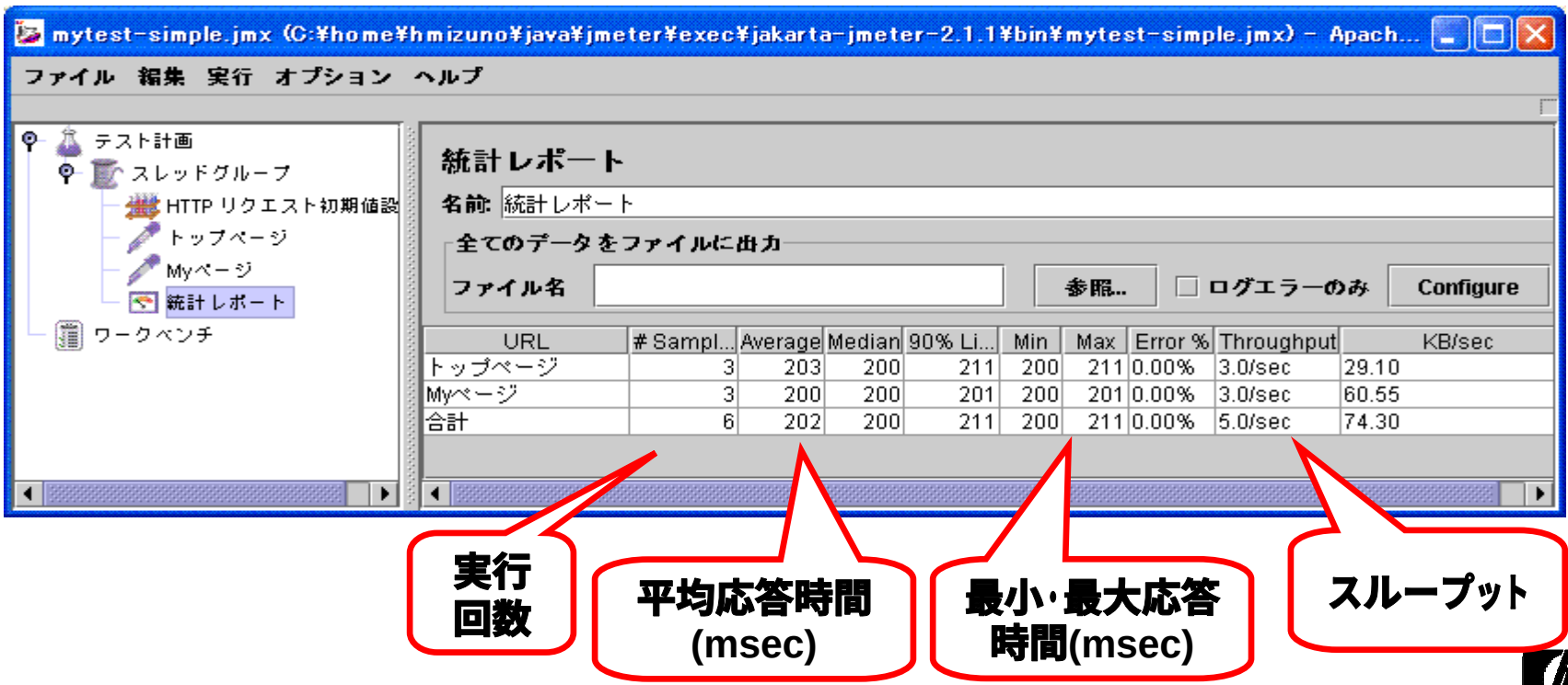


テストプランの実行後、テストが正常に行われているかどうかを確認するには、設定したリスナーを参照します。

7．実行結果の確認

テストが終了したらリスナーにおいて、テスト**実行結果を確認**します。

追加⇒リスナー⇒統計レポートを選択します。



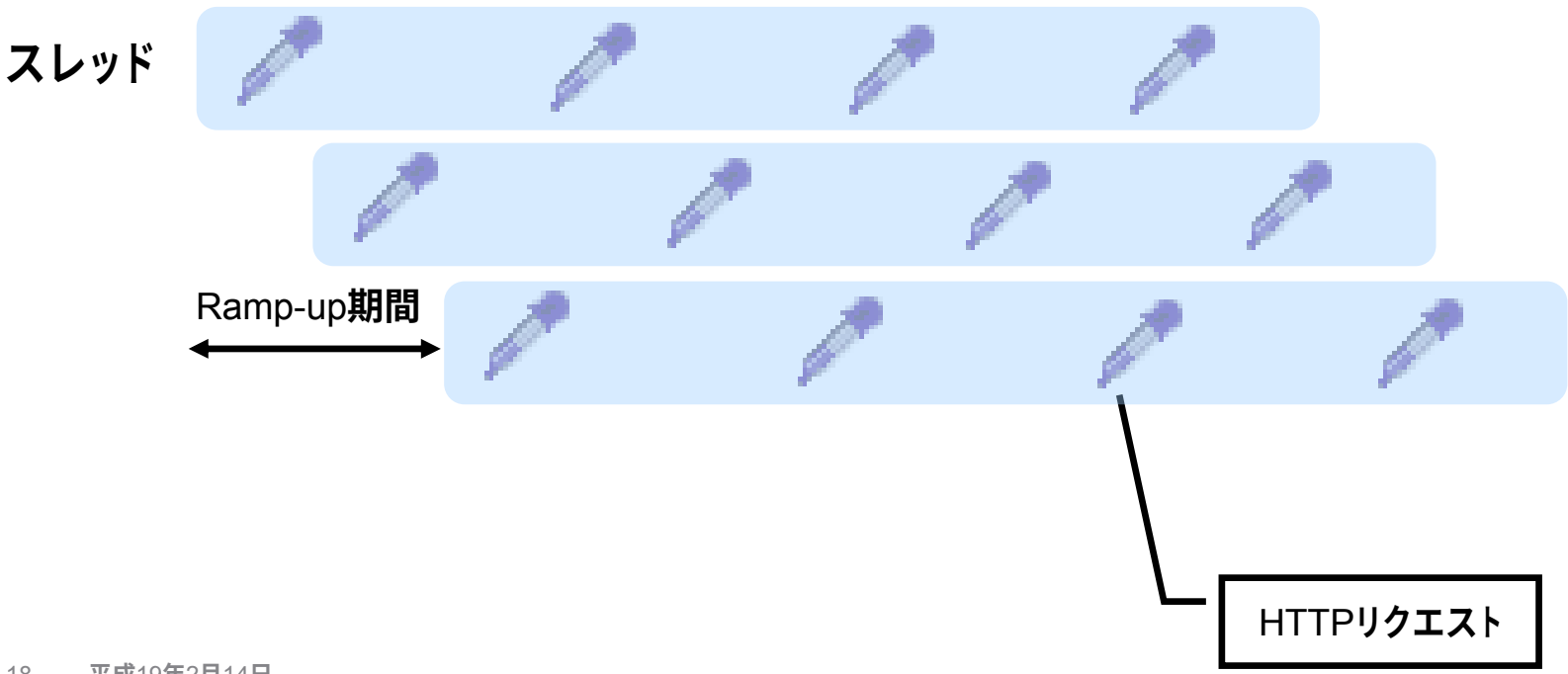
実行イメージ

テスト計画

スレッドグループ (3スレッド、4回)

リクエスト初期値設定

統計レポート



Apache JMeter実践編

内容

1. JMeterのチューニング
2. タイマ
3. テストプランの自動生成

JMeterのチューニング

JMeterで数多くのスレッド（クライアント）を実行すると、容量不足になり以下のようなエラーが発生することがあります。

```
ERROR - jmeter.threads.JMeterThread: Test failed!
        java.lang.OutOfMemoryError: Java heap space
```

JMeterのチューニングが必要です!!

- 起動スクリプト（$JMETER_HOME/bin/jmeter.bat）を編集します。
- Javaヒープの大きさを大きくして下さい。

テストとマシンのスペックに応じてチューニングします。

```
set HEAP=-Xms256m -Xmx256m
```

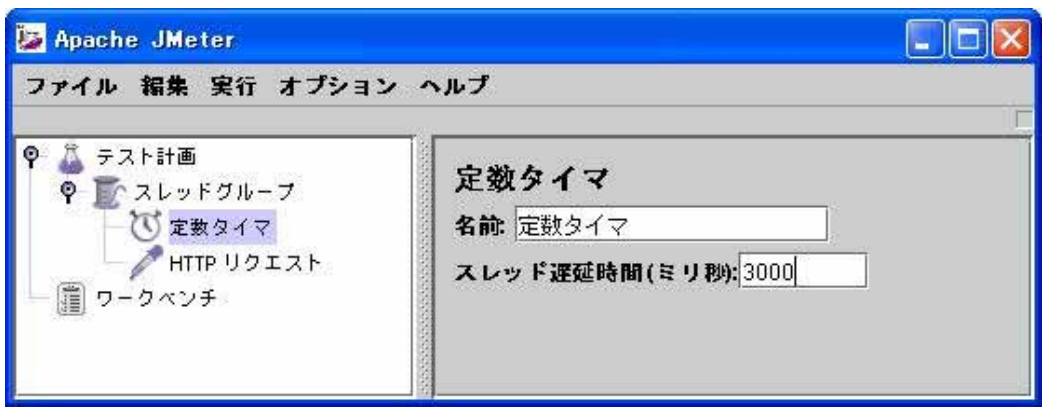
タイマ

- JMeterはリクエストを送信するとき、各リクエスを連続して送信します。
- タイマを利用することで、各リクエストの送信の間に間隔を入れることが出来ます。

左フレームの『スレッドグループ』を選択した後、右クリックまたは『編集』メニューか**追加⇒タイマ**を選択して、**『定数タイマ』**や**『定数スループットタイマ』**等を追加します。

「定数タイマ」は、待ち時間をミリ秒で指定します。

「定数スループットタイマ」は、1分間に送出するリクエスト数で指定します。
例) 60を設定すると1秒間に1つのリクエストが送出されます。



タイマを使用したときの実行イメージ

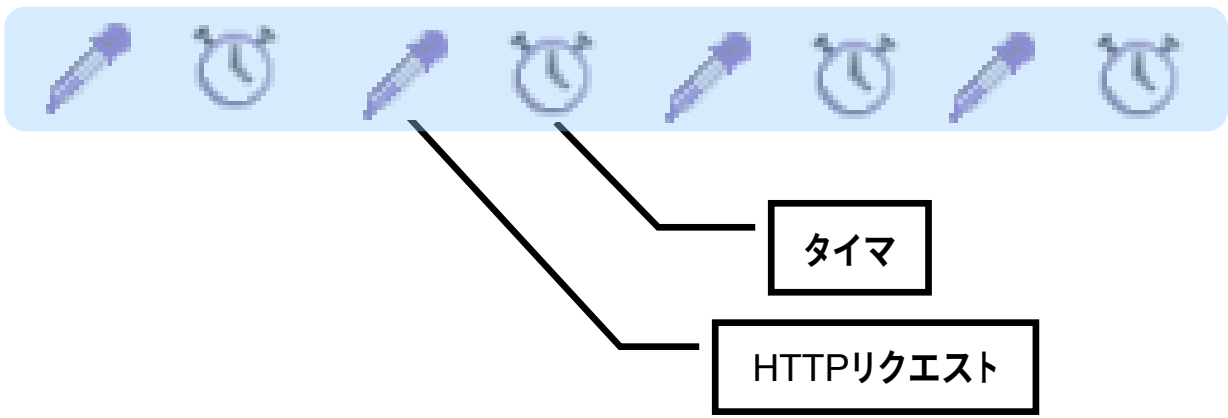
テスト計画

スレッドグループ (1スレッド、4回)

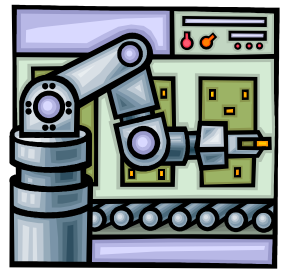
リクエスト初期値設定

統計レポート

スレッド

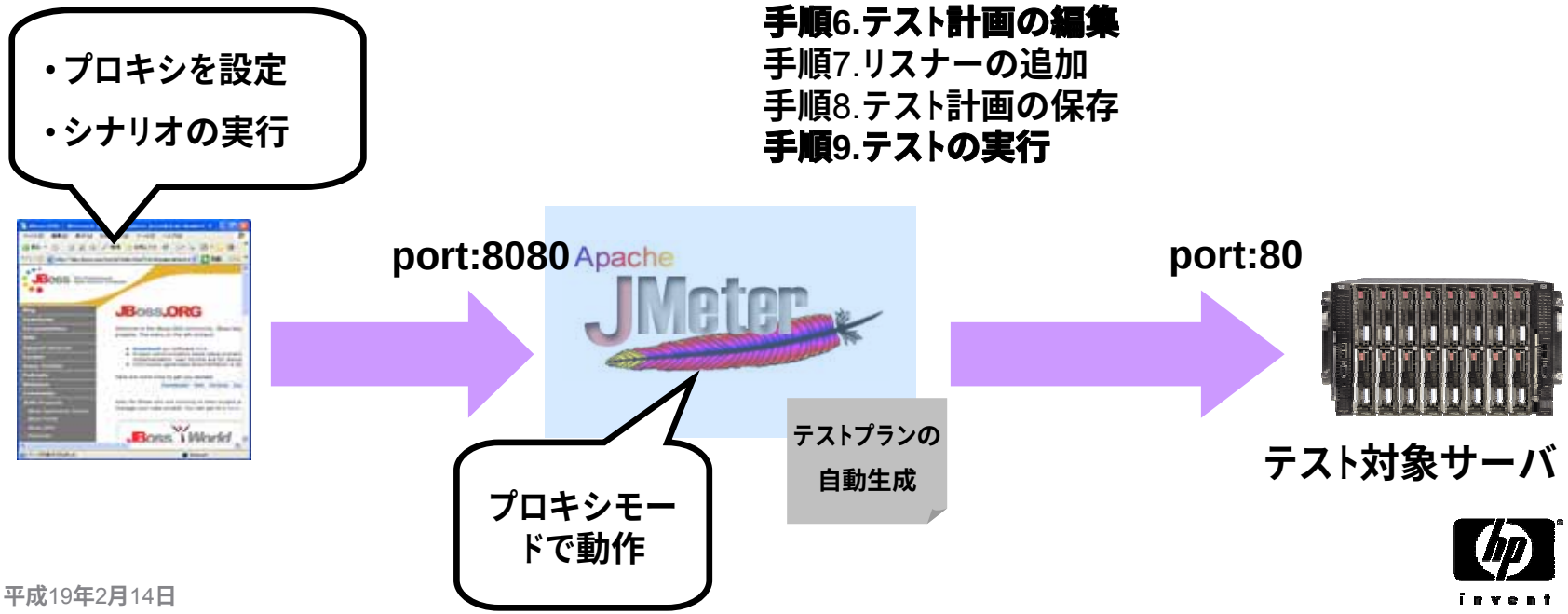


テストプランの自動生成



JMeterには、ブラウザから行った操作をテスト計画に自動的に変換してくれるHTTPプロキシサーバ機能があります。

手順1.スレッドグループの作成
手順2.HTTPリクエスト初期値設定エレメントの追加
手順3.HTTPプロキシサーバエレメントの追加
手順4. Webブラウザのプロキシサーバを設定
手順5.シナリオの実行 (テスト計画の記録)
手順6.テスト計画の編集
手順7.リスナーの追加
手順8.テスト計画の保存
手順9.テストの実行



負荷試験

負荷試験概要

負荷試験を実施する時のフローの概要

1.事前調査（想定負荷、試験環境、試験項目）

↓

2.環境の準備（HW、SW、試験データ）

↓

3.試験プラン作成と実施

↓

4.結果の分析と確認

ここで性能が出ないと大変です！

プロジェクトにおける負荷試験の実例

- Aさんは、下記のようなシステム用件で負荷試験をすることになりました。
  - 応答時間は2秒以内
  - 最大同時ユーザー数は100人
  - 試験シナリオ
    - 商品購入シナリオ（ログイン、商品検索、カートに商品を入れる、購入、ログアウト）
- 試験環境を構築し、JMeterのシナリオを準備しました。

試験を実施するにあたりAさんは以下のように考えました

- 最大同時ユーザが１００人なので、JMeterは１００スレッドで試験しよう。
- あまり多く実行すると怖いから、とりあえず1スレッドにつき1シナリオ実行しよう！
- そして、全てのリクエストの応答時間が2秒以内であればシステム要件を満たすことになる。
- これで僕の仕事も終了だ！

試験開始！

結果

応答時間がとんでもない値になってしまいました。
キャッシュが原因かなぁ！？　もう一回実施しよう！
あれ？　変わらないなぁ？
あと、もう一回実施する。　うんん……変わらないなぁ！

やっぱ、フリーのJMeterはだめだなぁ！

待ってください！
JMeterは
悪くありません。

なぜ、このようなことになって
しまったのでしょうか？

Aさんが実施した試験

- Aさんはタイマーを使用せずに、Rump-up時間の設定も行いませんでした。
- どのような試験を実施したことになるのでしょうか？
  - 100人が同時に、商品購入のシナリオを1回実施
  - その上100人が、各操作を必死で休み無く実行
- これってどんな負荷？
  - 人気商品を特定の時刻に売り出した時のような負荷

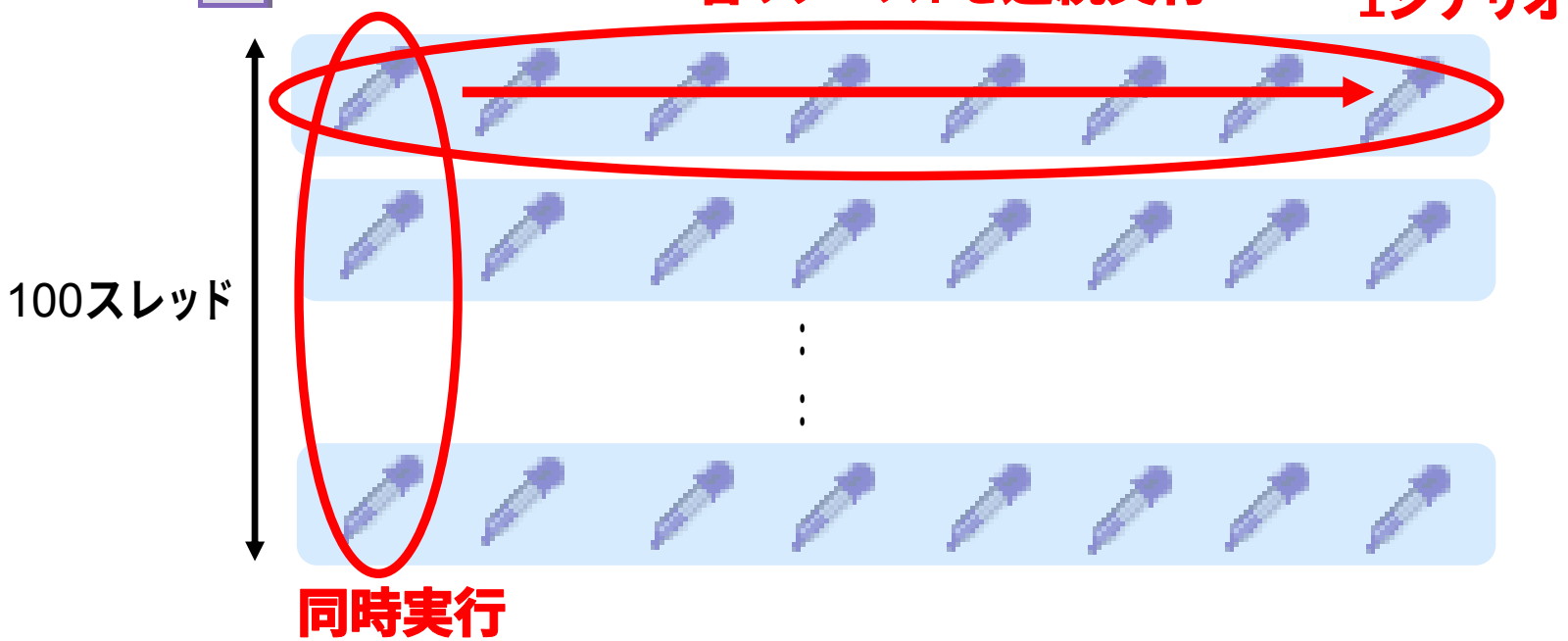
Aさんが実施した試験の概要

テスト計画

スレッドグループ

リクエスト初期値設定

統計レポート



Aさんの試験の改善方法

テスト計画

スレッドグループ

リクエスト初期値設定

統計レポート

スレッド

Ramp-up期間

負荷試験方法のセオリー

ところで. . .

実際の試験では、何スレッドで試験をすれば良いのでしょうか？

シナリオは何回実施すれば良いのでしょうか?

よほど自信がある場合以外は、以下のように小さな負荷から試験を実施して下さい。

1. 想定負荷の検討
2. 基礎データの取得
3. 基礎試験 (低負荷試験)
4. 想定負荷の負荷試験
5. 高負荷試験

1．想定負荷の検討

- 実際のプロジェクトなどでは、性能の目標値が無かったり、不明確であることが多々あります。
  - 実例
    - 最大ユーザー数は100人、応答時間は、2秒以内　→　同時ユーザ数は？
    - ユーザー数は100人、秒間50個の処理　→　50個の単位は？
- 最低、確認すべき項目
  - ユーザー数はシステムに登録される人数？それとも同時にシステムにアクセスする人数？
    - JMeterで試験をする場合のスレッド数とは、同時アクセス数に対応します。
  - スループットの単位はHTTPリクエスト？　それともシナリオ単位？
    - JMeterでの試験結果の集計は、HTTPリクエスト単位で行われます。
  - 応答時間の目標値

2．基礎データの取得

- 1ユーザがシナリオを1回だけ実行した時のリクエストの応答時間を測定します。
  1. 1つのブラウザからシナリオを実行した時の応答時間の測定
  2. JMeterから1スレッド、1回のシナリオ、リクエストの間隔1秒で実施し、各リクエストの応答時間を計測
- 何が分かるの？
  - システムにおける最高の応答時間が分かります。
  - 1の試験の結果が、応答時間の要件を満たしていない場合
    - これ以上試験をする必要はありません。早速、遅い原因を調査して下さい。
  - 2の試験の結果が、1の試験結果と大きく異なっている場合は、作成したシナリオの内容やJMeterの設定などを確認してください。

3．基礎試験 (低負荷試験)

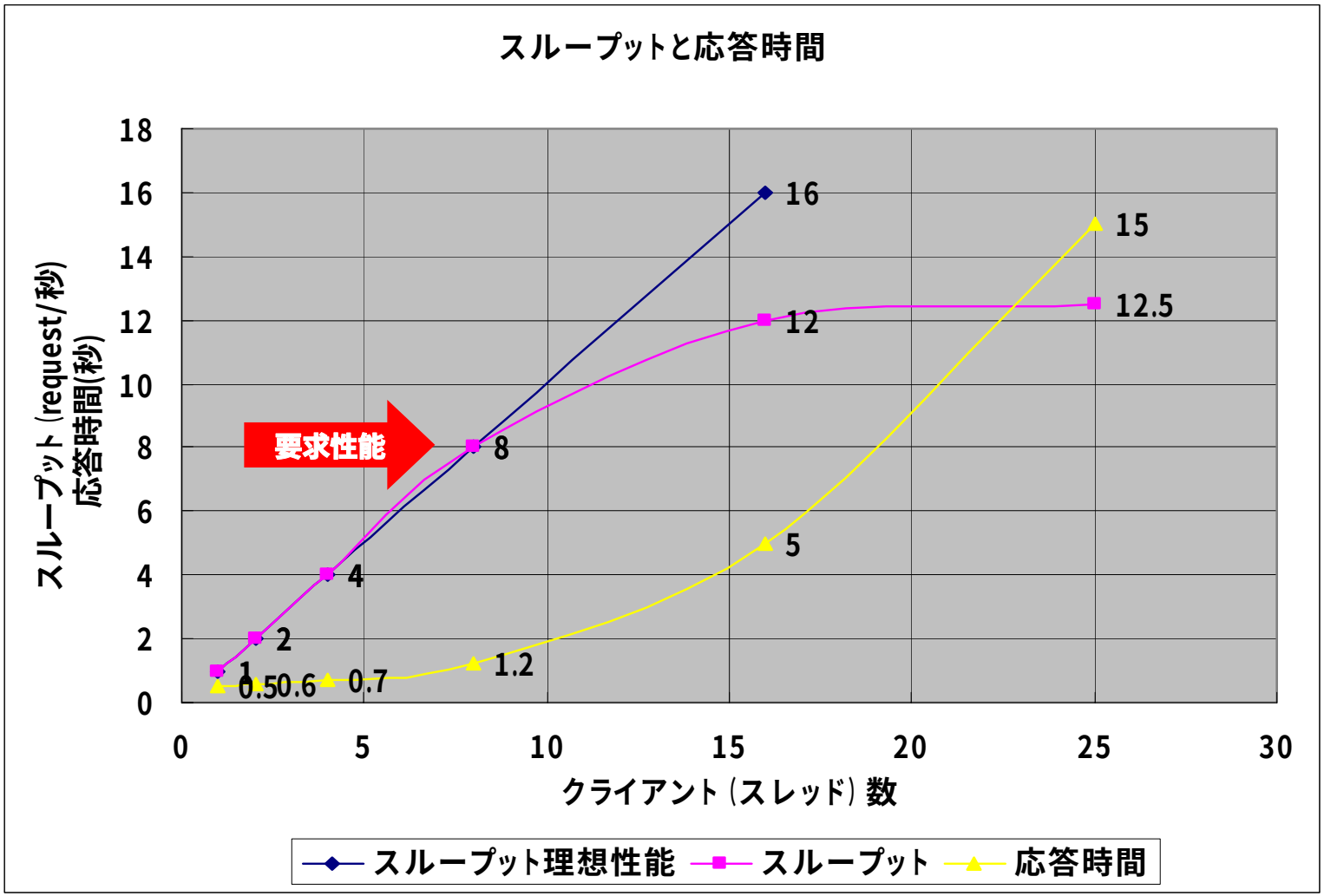
- いよいよ負荷をかけていきます。あせる気持ちを抑えて以下の順番で試験を実施しましょう。

- **試験A：1多重複数回** (1スレッド、100シナリオ)
  - 応答時間が、だんだん悪くなるようなことはありませんか？
  - スループットを記録して下さい。
- **試験B：2多重複数回** (2スレッド、100シナリオ)
  - 平均応答時間は、どの程度悪くなりましたか？
  - スループットは、試験Aの2倍ぐらいの値になっていますか？
- **試験C：4多重複数回** (4スレッド、100シナリオ)
  - 平均応答時間は、どの程度悪くなりましたか？
  - スループットは、試験Aの4倍ぐらいの値になっていますか？

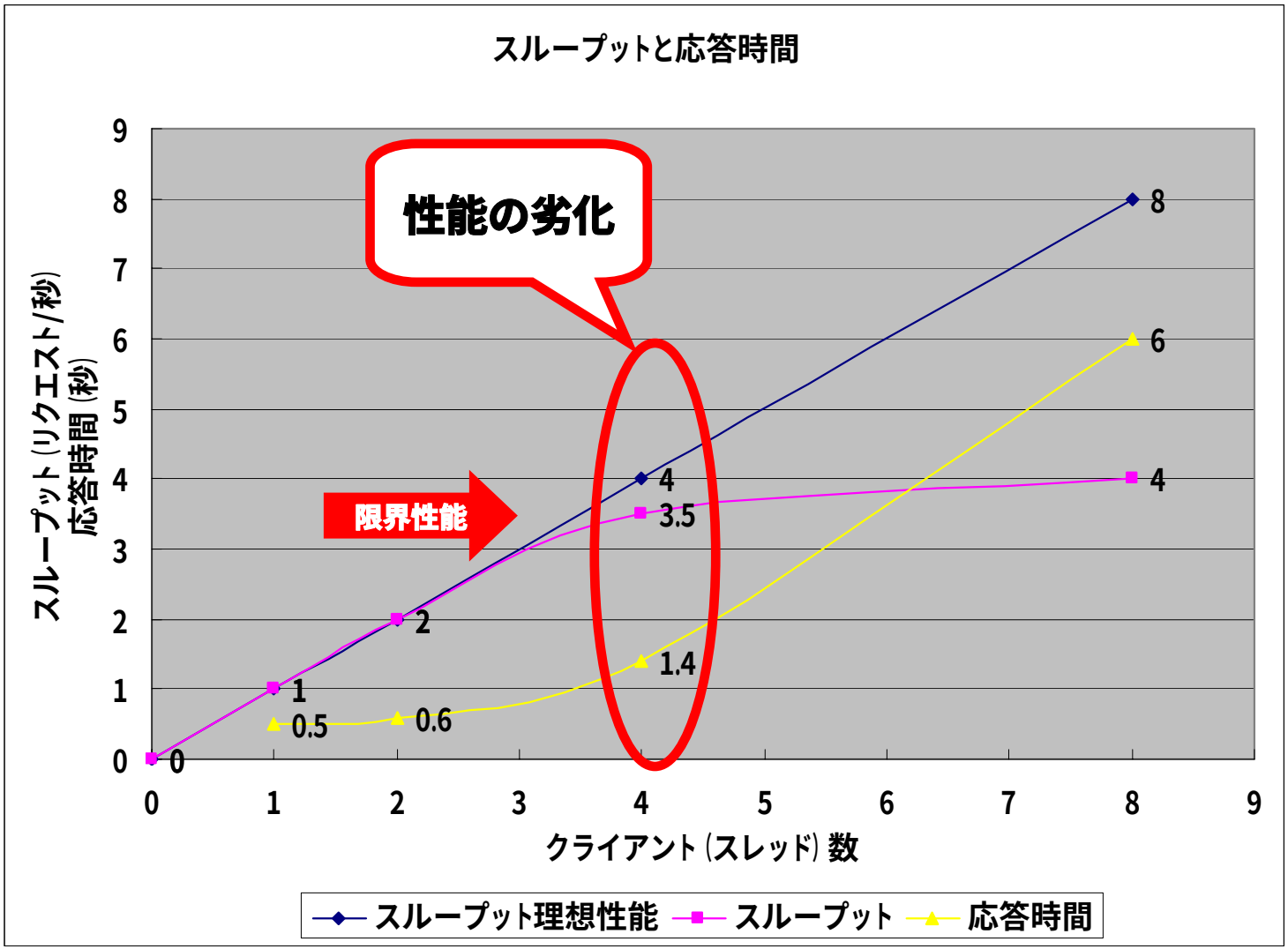
基礎試験って？

- 疑問？
  - 試験A、試験B、試験Cを実施しましたが、これらの試験は何のために行ったのでしょうか？
  - 次にどのような試験をすれば良いのでしょうか？
- 基礎試験を実施した意味
  - システムの限界性能に対して、試験Aから試験Cによる負荷がどの程度の負荷になっているかを見るために実施しています。
- 分析方法
  - システムが限界性能に達したときに、スループットの値や応答時間の値が一般的にどのように変化するかを知ることによって、試験Aから試験Cの負荷が限界性能に対してどの程度の負荷であるかを予想することが出来ます。

典型的な負荷試験の結果



試験結果の例


スループットと応答時間
性能の劣化
限界性能
スループット(リクエスト/秒)
応答時間(秒)
クライアント(スレッド)数
0
1
2
3
4
5
6
7
8
9
0
1
2
4
3.5
8
6
4
1
0.5
0.6
1.4
スループット理想性能
スループット
応答時間

負荷試験のコツ

- シナリオの作成
  - 各リクエストの間には間隔を入れる
  - 各スレッドの開始時間をずらす
- 基礎データの取得
  - 最初に、ブラウザから実行する
  - 1スレッドでシナリオを1回だけ実行して、各リクエストの応答時間が要件を満たしていることを確認する
- 基礎試験の実施
  - 1スレッド複数回の実行
  - 2スレッド、4スレッドと、徐々に負荷をかける
- 実施している負荷試験が、限界性能に対してどの程度の負荷を与えているかを把握しながら試験を実施して下さい。

負荷試験のコツ：その2

早めの負荷試験のすすめ

- 出来れば、プロトタイプの作成時や、開発時から負荷試験を実施することをお勧めします。
- 単体機能の負荷試験も可能です。

まとめ

- 負荷試験の重要性
- Apache JMeter
  - 概要、入門、実践編
- 負荷試験方法
  - プロジェクトにおける負荷試験の実例
  - 負荷試験方法のセオリー

付録

参考URL

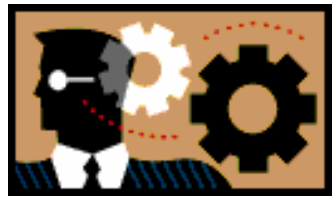
- URL
  - Apache JMeter
    http://jakarta.apache.org/jmeter/
  - Apache Jakaruta Project : JMeter User Manual
    http://jakarta.apache.org/jmeter/usermanual/index.html
  - Apache Jakaruta Project : JMeter ユーザマニュアル（日本語訳） http://cgi0.biwa.ne.jp/~yabuta/study/jmeter/usermanual/
- 書籍
  - 月刊JavaWorld (ジャバ ワールド) 2005年12月号

関連URL

- 日本ヒューレットパッカード・オープンソース＆Linux

  http://www.hp.com/jp/linux/

エレメント：スレッドグループ



- クライアントの設定
- 全てのテストプランはこれから始まります。
  - この下にサンプラーやリクエストといったエレメントを追加していくことで様々な設定を行います。

名前：テストケース名

サンプラーエラー後のアクション：
サンプラー実行中にエラーが発生した際のアクション
続行（デフォルト）/スレッド停止/テスト停止

スレッド数：（デフォルト1）
JMeterが生成するクライアント数（アクセスユーザ数）

Ramp-Up　期間（秒）：（デフォルト1）
全てのスレッドが起動するまでの時間

ループ回数：（デフォルト1）
各スレッドにて実行するテストケースの回数

スケジューラ：チェックを入れると時間指定などの
詳細な設定が可能(デフォルトOFF)

プロキシサーバの使用

- ファイヤーウォールやプロキシサーバ越しにテストする場合
- ファイヤーウォールやプロキシサーバの名前やポート情報をJMeterの起動コマンドに与えて起動します。

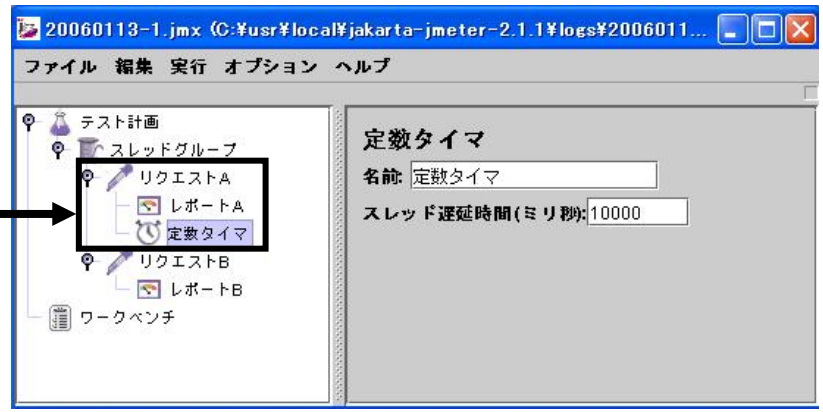
```
jmeter.bat –H [プロキシサーバのホスト名] –P [ポート]
                                    –u [ユーザ名] –p [パスワード]
```

タイマ

タイマを『スレッドグループ』に付け加えるか、あるいは『HTTPリクエスト』等に付け加えるかによって、そのタイマの影響の範囲が異なります。

『HTTPリクエスト』に『定数タイマ』をつけた例

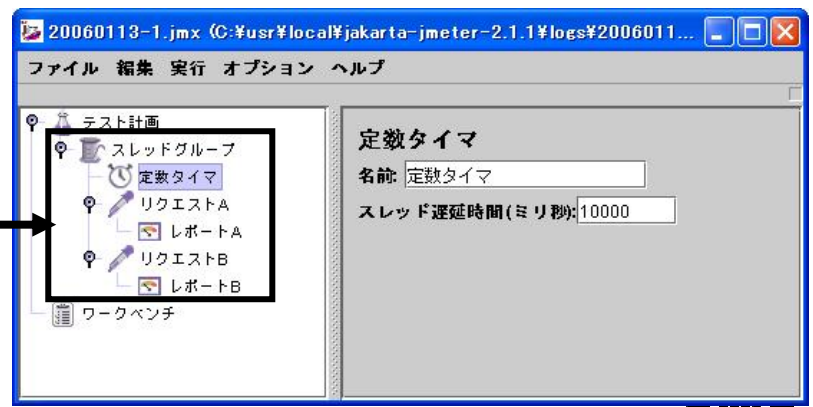
この場合、タイマが適用されるのは、『リクエストA』のみです。
『リクエストB』には、タイマは適用されません。
したがって、右の例の場合は、以下のように動作します。
実行⇒10秒待機⇒リクエストA⇒リクエストB



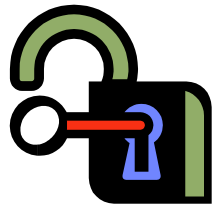
『スレッドグループ』に『定数タイマ』をつけた例

この場合は、『スレッドグループ』以下の全てのサンプラー(この例では、『リクエストA』と『リクエストB』)との間で、タイマによって指定された分だけ待機時間が発生します。

したがって、右の例の場合は、以下のように動作します。
実行⇒ 10秒待機⇒リクエストA⇒10秒待機⇒リクエストB



HTTP認証マネージャ



HTTP認証マネージャは、HTTP認証(Basic認証)を利用できます。

追加⇒設定エレメント⇒HTTP認証マネージャを選択します。

例えば、http://localhost/admin/にHTTPのBasic認証がかけられているとします。また、その認証IDとパスワードがそれぞれ“admin”と“admin”だとします。
その場合は、上記のような設定すれば、HTTPのBasic認証をパスすることができます。

※ 「パスワード」の部分は自動的に伏字になります

動的リクエスト

3.動的リクエストの利用

事前に登録しておいた**複数のユーザをスレッド起動毎に動的に切り替えたい場合**に、ユーザ変数を利用すると大変便利です。
1人のユーザではなく、複数のリクエストパラメータとしてユーザを用いて負荷試験をする際に使用します。

パラメータ情報を記述した外部ファイルを読み込ませ、動的にユーザパラメータを切り替える方法を使います。
手順としては、**ユーザパラメータ**と**外部ファイル**の利用となります。

この手順は、JMeterのFAQで紹介されています。詳しくは、下記URL先をご参考下さい。

JMeterFAQ
⇒ How do I use external data files to in my Test scripts?
http://wiki.apache.org/jakarta-jmeter/JMeterFAQ#head-1680863678257fbcb85bd97351860eb0049f19ae

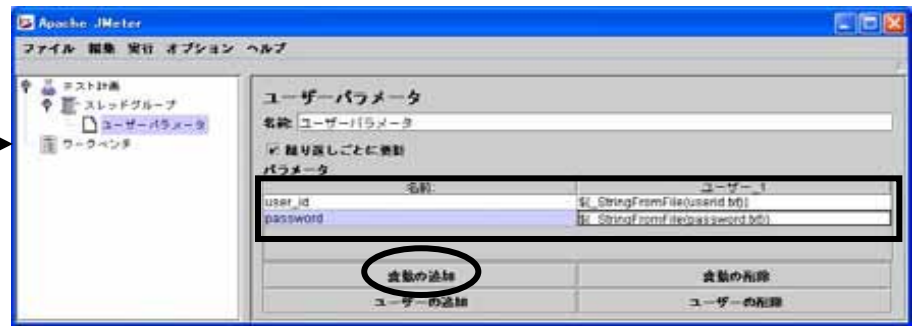
動的リクエスト

手順1.JMeterの起動を起動して、スレッドグループを作成します。

手順2.スレッドグループを選択し、前処理⇒ユーザパラメータを追加します。
『変数の追加』を選択して、変数を設定します。

記述形式：${_StringFromFile(ファイル名)}

名前	ユーザ_1
user_id	${_StringFromFile(userid.txt)}
password	${_StringFromFile(password.txt)}



動的リクエスト

手順3.スレッドグループを選択し、サンプラー⇒ HTTP リクエストを追加します。
リクエストで送るパラメータの『追加』を選択して、変数を設定します。

記述形式：**${名前}**
※名前は、手順2のユーザパラメータの名前

名前	値
user_id	${user_id}
password	${password}

動的リクエスト

手順4. $JMETER_HOME/bin以下にパラメータを記述したファイルを配置します。

※ $JMETER_HOMEとは、JMeterをインストールしたフォルダを示します。

※ ファイルの中身は、テキスト形式で以下のような記述となります。日本語もOKです。

```
---user_id.txt----
taro
jiro
saburo
```

```
---password.txt----
pass1
pass2
pass3
```

手順5.テストを実行します。

ファイルの内容は、スレッド起動毎に上から1行づつ読み込み、最後まで行くと最初のパラメータを再度読み込みます。

※上の例では、

START⇒1回目：taro/pass1

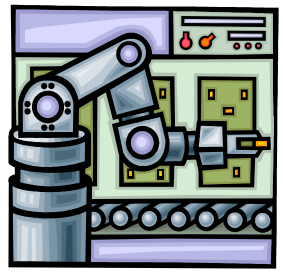
⇒2回目：jiro/pass2

⇒3回目：saburo/pass3

⇒4回目：taro/pass1(最初のパラメータ)

⇒・・・・・・・・・・・・・・・・・・・・・

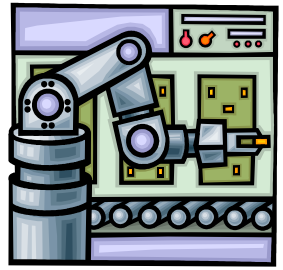
テストプランの自動生成



手順1.JMeterの起動を起動して、スレッドグループを作成します。

まず「テスト計画」の中にスレッドグループを作成します。
「テスト計画」を右クリックし、**追加→スレッドグループ** を選択します。
「テスト計画」の下に「スレッドグループ」が追加されます。

テストプランの自動生成

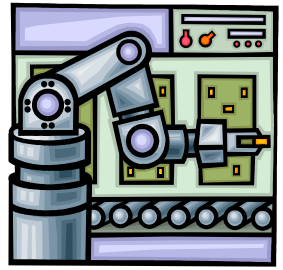


手順2. HTTPリクエスト初期値設定エレメント

共通設定がある場合は、スレッドグループにあらかじめ追加しておきます。

スレッドグループ上で右クリックした後、 追加→設定エレメント より
HTTPリクエスト初期値設定を追加します。

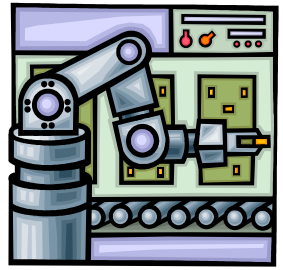
テストプランの自動生成



手順3. HTTPプロキシサーバエレメント

ワークベンチ上で右クリックした後、追加→Non Testエレメントより
HTTPプロキシサーバを追加します。

テストプランの自動生成



手順3. HTTPプロキシサーバエレメント

□ ポートの指定

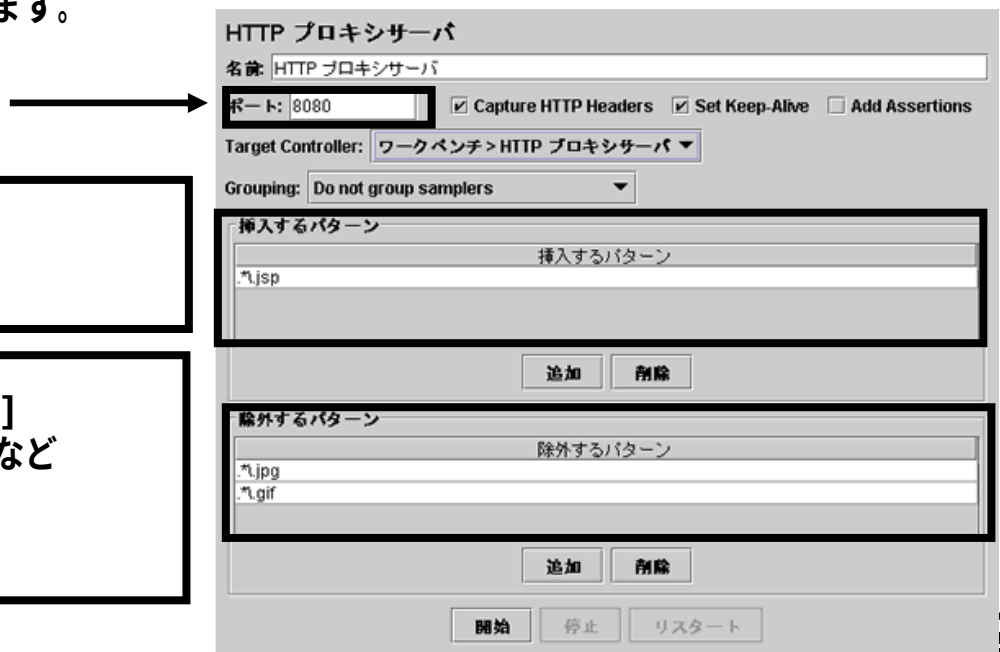
JMeterがプロキシサーバとしてリクエストを受け付けるポート番号を指定します。(デフォルトは8080番)

□ テスト計画に挿入するパターンと、除外するパターンの指定

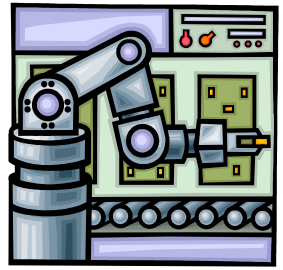
テスト計画に必要なURLと、除外したいURLのパターンを指定します。
URLの指定は、Perl形式の正規表現を用います。

[テスト計画に挿入するURLパターンの例]
***index¥.jsp**

[テスト計画から除外するURLパターンの例]
画像ファイル ：*¥.gif や *¥.jpg や *¥.png など
CSSファイル ：*¥.css
Javaスクリプト ：*¥.js



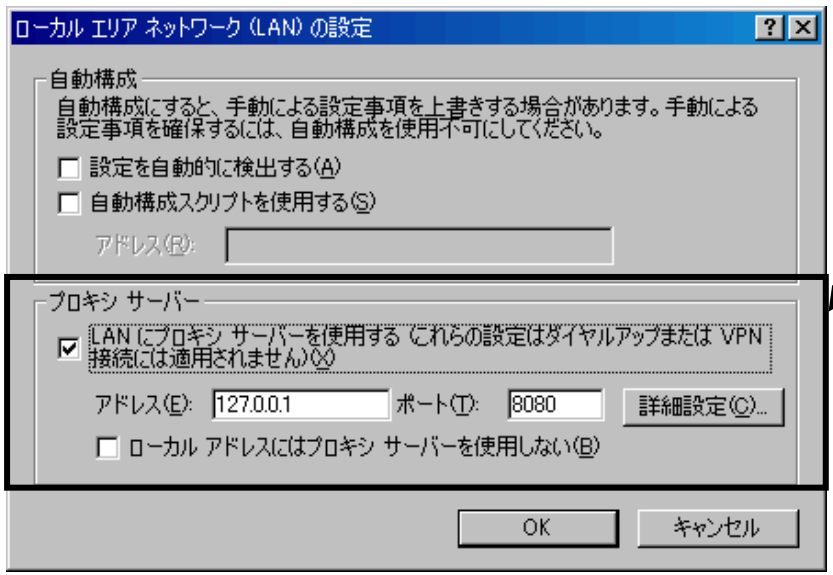
テストプランの自動生成



手順4.Webブラウザのプロキシサーバの設定をします。

JMeterがインストールされているマシンのブラウザのプロキシサーバの設定をします。
例えば、Internet Explorerであれば、**ツール→インターネットオプション→設定→LAN**の設定より行えます。
入力欄「ポート」にはHTTPプロキシサーバ詳細設定画面で入力したのと同じポート番号を指定してください。

[Internet Explorerのプロキシ設定]

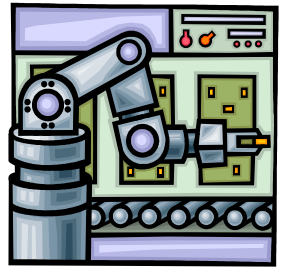


■LANにプロキシサーバを使用するにチェックを入れます。

■アドレスは、自分自身のアドレス (127.0.0.1)

■ポートは、HTTPプロキシサーバで設定した値を入れます。
※ここでは、8080番

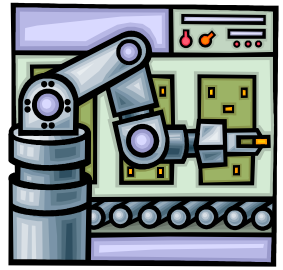
テストプランの自動生成



手順5.テスト計画の記録

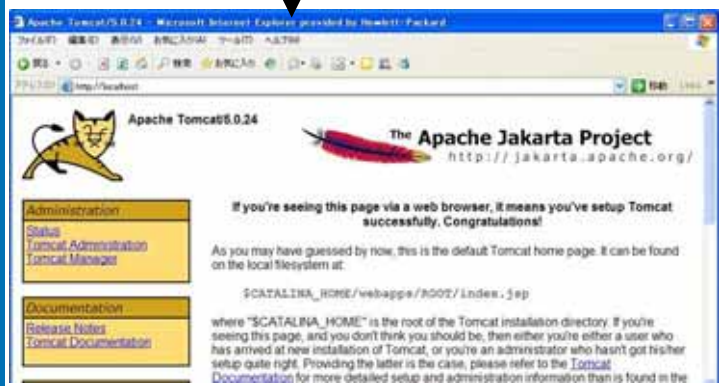
HTTPプロキシサーバ画面の「**開始**」ボタンを押すと、JMeterがプロキシサーバとして動作し始めます。

テストプランの自動生成



手順5.テスト計画の記録

手順4にてプロキシの設定を行ったブラウザにて、
実際にテスト対象ページを閲覧、操作します。

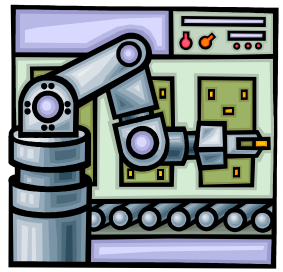


注意点

1.URLにミスがあったり余分なページを閲覧してしまった場合は、
手動で修正や削除が可能です。

2. Webブラウザによる巡回が終わったら、JMeterの
「HTTPプロキシサーバ」で [停止] ボタンをクリックして記録を終了します。
もし、停止する前にブラウザを閉じてしまうとその動作も記録されてしまいます。

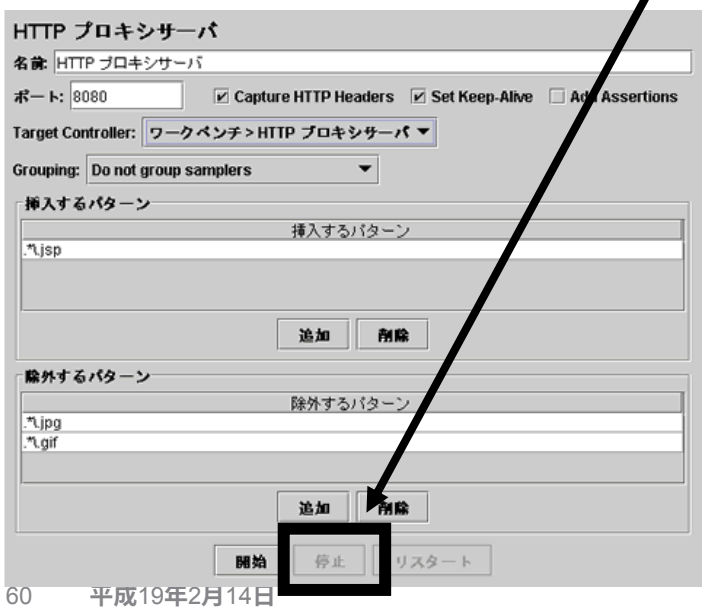
テストプランの自動生成

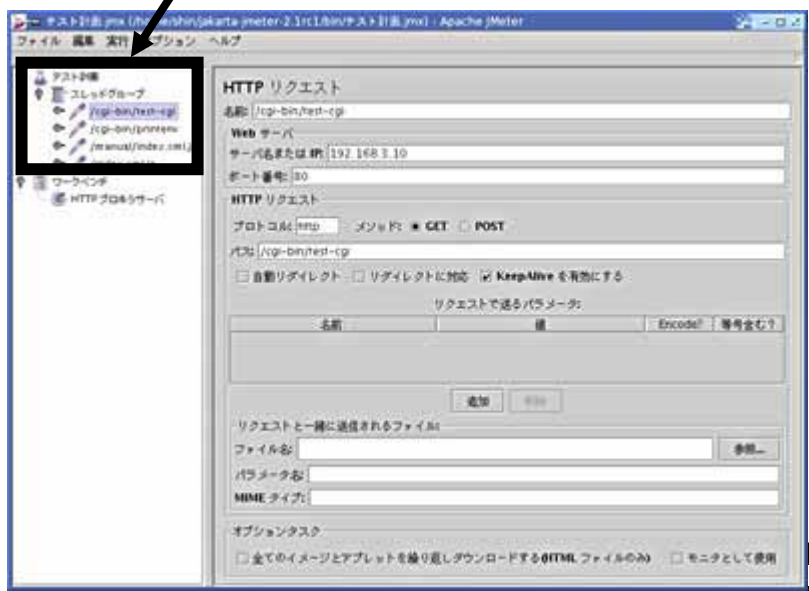


手順5.テスト計画の記録

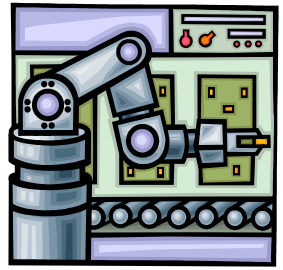
□ 『プロキシサーバ』を停止させ、テスト計画が記録されていることを確認します。

『プロキシサーバ』の起動後から「停止」ボタンを押し『プロキシサーバ』を停止します。
テスト計画の『スレッドグループ』以下に結果が組み込まれているのが確認できます。



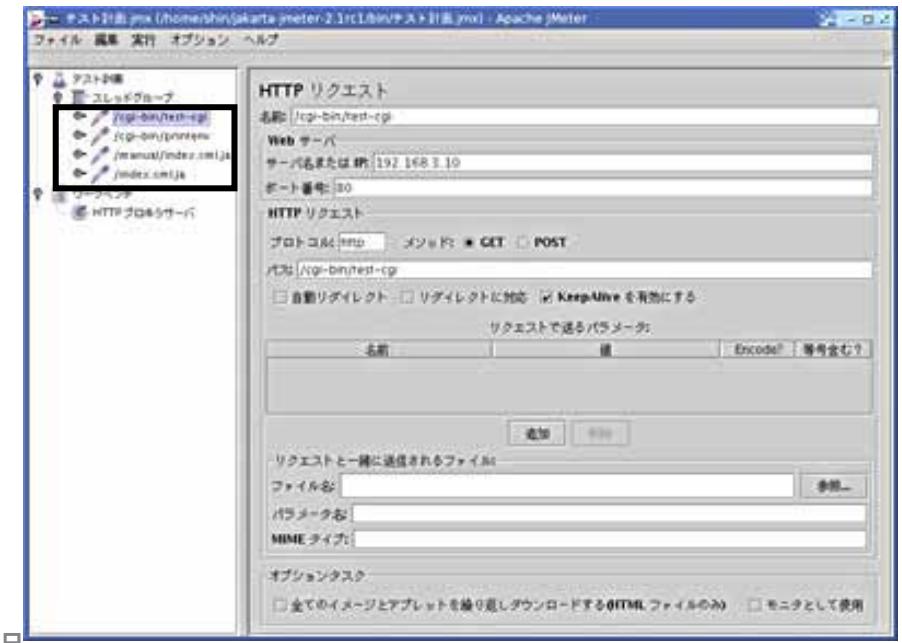


テストプランの自動生成

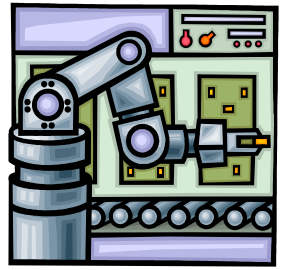


手順6. テスト計画の編集

自動的に追加された「スレッドグループ」下の「HTTPリクエスト」に余分なものがある場合は、そのリクエストを右クリックして [削除] を選択します。
リクエストをクリックすれば、URLの修正などが行えます。CGIなどの動的コンテンツで、サーバに渡すパラメータがある場合はここで指定を行います。



テストプランの自動生成

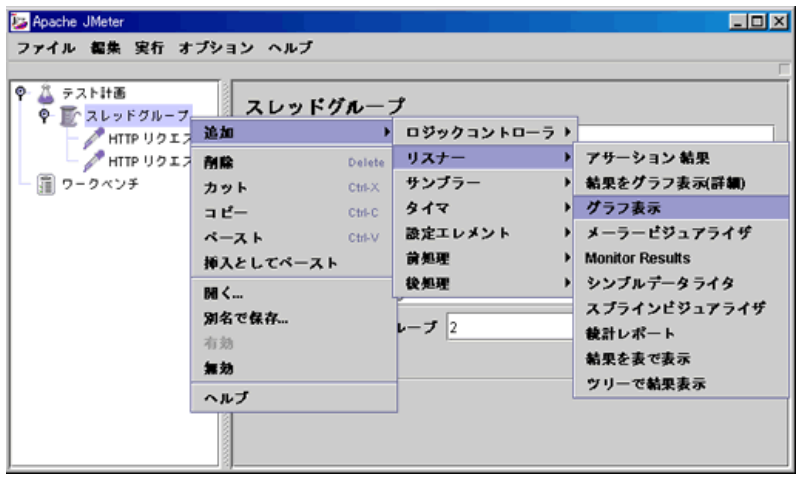


手順7. リスナーの追加

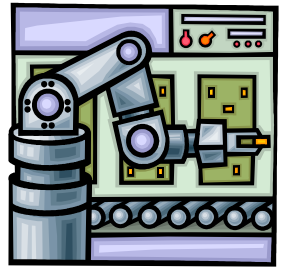
試験結果を見るために、必要なリスナーを追加します。ここでは、[グラフ表示] と [統計レポート] のリスナーを追加しました。

手順8. テスト計画の保存

作成したテスト計画を保存します。

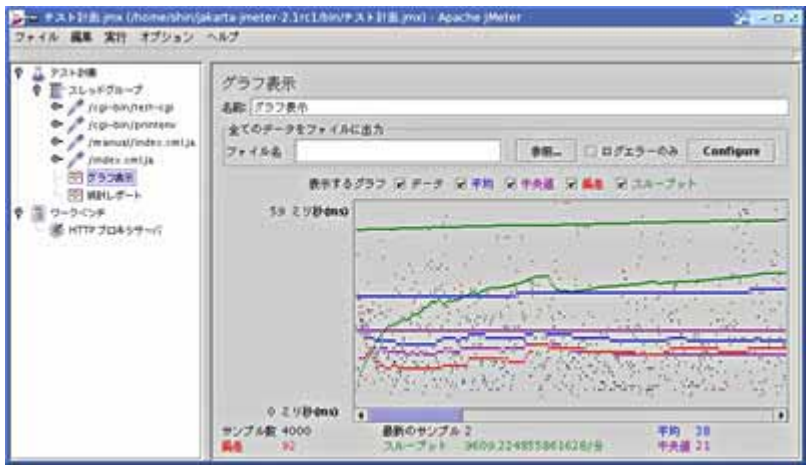


テストプランの自動生成

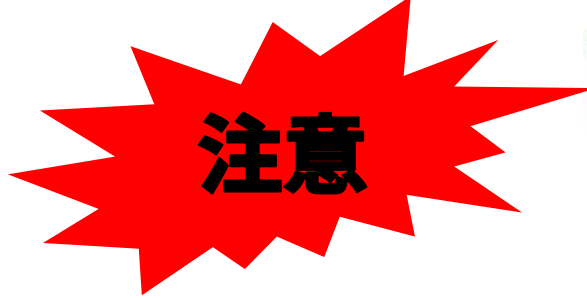


手順9. テストの実行

「スレッドグループ」の [スレッド数] や [ループ回数] などの値を調整してから実行します。
また、実行結果をクリアするには「実行」メニューの「全て消去」を選択します。



試験を実施する前に



- 各試験の前には以下のような事前試験を実施して下さい。

- 試験前の事前処理
  - 測定するシナリオを複数回実行する
- 理由
  - システムが保持しているキャッシュに値を入れたり、Javaのコードのコンパイル動作を事前に実施するため。
- 注意
  - 試験時にかける負荷に合わせた事前処理をすることをお勧めします。


invent