使用 Jmeter 进行性能测试

目录

本文是属于个人技术手册一类的文档，是对 Jmeter 进行研究和使用之后，写下的方便记录和重新查找使用方法的文档，其中使用的参考标准为 HP 的 loadrunner，对于已经实现的方法和基本的使用情况，都有一定的介绍。本文从模块上分，可以分为：web 的录制回放执行、手动编辑 HTTP 请求、通过 JDBC 压数据库性能、通过 JAVA 请求压 class 性能等部分。具体细节上还有：jmeter 的参数化方法、集合点设置、检查点（断言）设置、关联设置、事务控制等内容。之后还要补充的有 LDAP，webservice，mail 等相关内容的描述，本文暂没涉及。

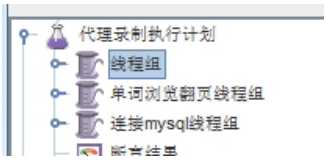
一、Web 的性能测试

性能测试工具的最大用途，就是模拟高并发，验证 web 系统的性能，所以先从 web 的性能测试开始说起。在 jmeter 中，有两种方法可以进行 web 性能测试的准备，一个是录制出模拟脚本，然后回放进行。另一个是直接写 http 请求到 web 服务器，这个方法可以直接发参数给 servlet，可以更有针对行的验证服务器的处理能力。接下来一个一个的介绍：

1. 录制脚本实现 web 性能测试

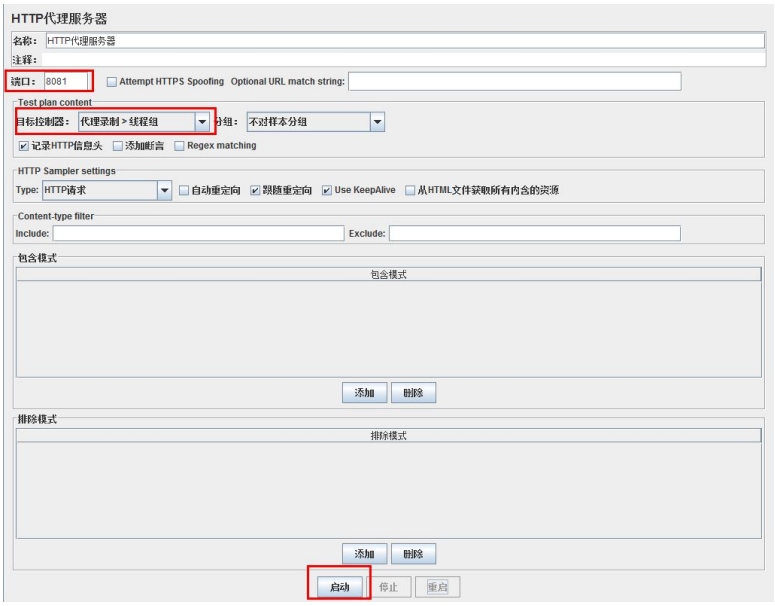
实际上在 jmeter 中，有两种办法可以实现录制功能，一种是借助第三方软件 bad boy，通过 bad boy 录制后，导出 jmx 文件被 jmeter 使用。另一种是通过本机的代理服务，让 jmeter 捕获代理端口上 http 的各种响应，自己实现录制效果。
出于减少对第三方软件的借助，这里只介绍通过代理，让 jmeter 自己实现录制的操作。

1．在 jmeter 中创建测试计划，并添加需要的线程组：



2．之后在“工作台”上添加一个“非测试元件”，选择“HTTP 代理服务器”

3．对 HTTP 代理服务器进行配置

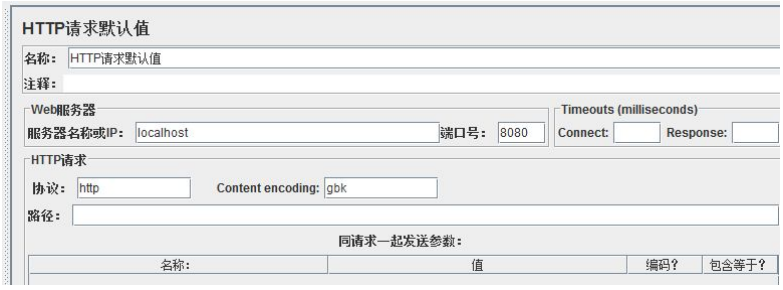


这里的端口，指的是浏览器配置的代理端口，目标控制器是指录制的内容存放在哪个线程组中。包含和排除模式，是指定录制时针对的内容，默认不选择则不生效。设置完后要启动这个功能才能生效。

4．配置本机的代理环境

由于我就设置用我本机的 8081 端口作代理服务，故就按上图的方法设置，当然这个 IP 和地址要和 jmeter 中，http 代理服务器的填写方式一样。

5．设置线程组中的配置元件



添加一个 HTTP 默认请求，为后面所有的 HTTP REQUEST 配置通用的属性，这里的地址和端口号，是能够通过 HTTP 协议访问到 WEB 服务的属性。

6．开始通过代理录制脚本

因为当前本机的 HTTP 协议都会经过刚刚设置的地址和端口号，并且已经启动了 jmeter 的 HTTP 代理服务功能，因此现在所有的 HTTP 协议都会经过本机的 8081 端口并留下记录。

这里我只记录了一个页面的打开，在 HTTP 代理服务器的设置里，选择录制到哪个线程组中，访问的目录自动称为该 HTTP 请求的名称。

此时设置好线程数和执行时间，就可以进行回放了，这是最简单的通过本机代理的方式录制 web 脚本的方法。

补充：很多人可能更喜欢使用 BAD BOY 去进行录制，其实没什么区别，都是生成 jmx 文件，这里只介绍用代理去录制，目的只是想针对完全不借助第三方软件的办法，自己实现需要的功能。

但是有一个问题，因为设置了代理，所以本机上所有经过代理访问网络的 HTTP 协议，都会被记录在线程组中，这时需要根据实际情况，删除不必要的内容，同时如果可以的话，关掉当前系统上没用的网络访问也行。总之使用这个方法，最后需要自己过滤一下有用的内容。

2. 执行 HTTPS 的 web 服务性能测试（暂留）

（需要研究）

3. 直接编写 HTTP REQUEST 测试 WEB 系统及 servlet 的处理能力

其实上面的部分里，我们录制的内容就是 HTTP REQUEST，这里说直接编写，其实主要针对的是 web 中需要有参数操作的两种情况：

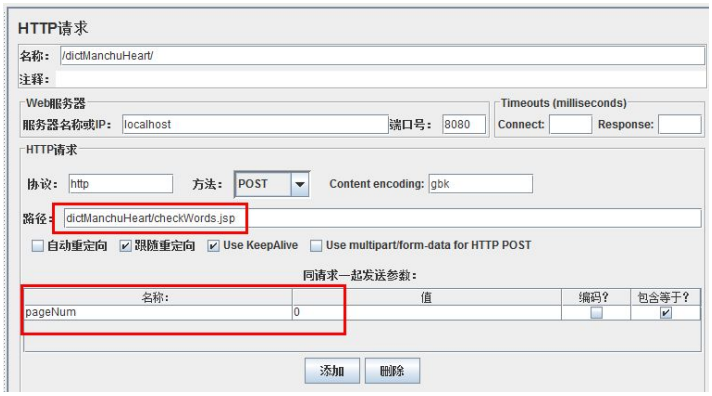
1）直接在地址栏里传递参数，在后面的 page 上进行解析；

“？”后面是传递的参数名称，0 为参数，在当前页上直接进行处理，不需要经过 servlet。

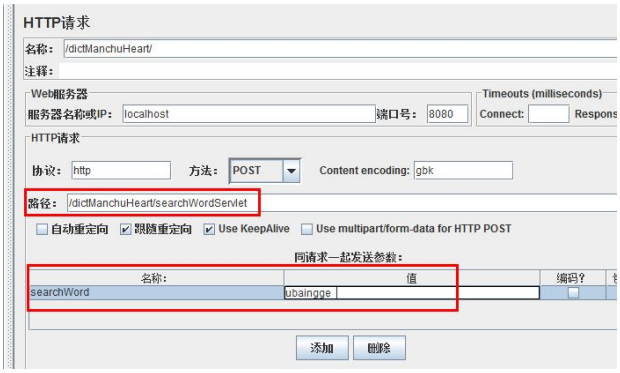
2）页面上的表单把变量提交到 servlet，处理在 servlet 中，然后跳转到指定页面；

这种情况在地址栏里看不到参数，中间需要 servlet 进行处理。

其实这两种情况实现起来都一样，需要编辑线程组下的那个 HTTP REQUEST：



第一种情况：因为参数都是在地址栏里传递的，所以直接可以知道该访问哪个 jsp 文件，同时知道该传哪个参数和哪个值，当然这个值我们后面可以进行参数化。

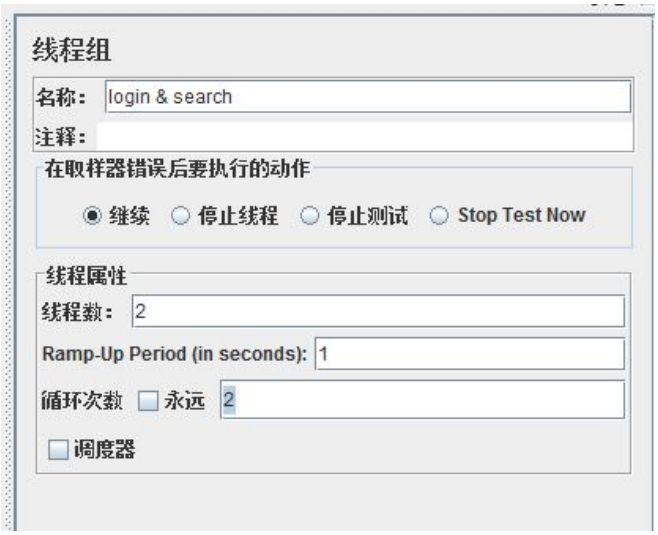


第二种情况：需要访问的是一个 servlet 文件，这个时候要从页面源码中了解，表单提交的参数名是什么，也就是这个“searchWord”，提交的值可以参数，然后直接压这个 servlet 文件，执行效果和上面一种方法一致。

4. 关于 session 问题的讨论和使用

Session 的概念大家都知道，是 web 服务器服务器分发给每个客户端的一个会话号，在一些 web 应用里，需要使用 session 作为用户身份判断的标志，在 LR 中，session 已经是自动关联的内容了。为了更真实的模拟用户的行为和服务器的处理过程，有时候要考虑 jmeter 在 session 方面的处理。

首先，Jmeter 的线程组的概念，实际上就是虚拟用户组，他跟 session 的相同与否没有直接的联系，更多的只是把做相似功能的用户进行分类，Jmeter 线程组里的用户，及可以获得相同的 session，也可以获得不同的 session。



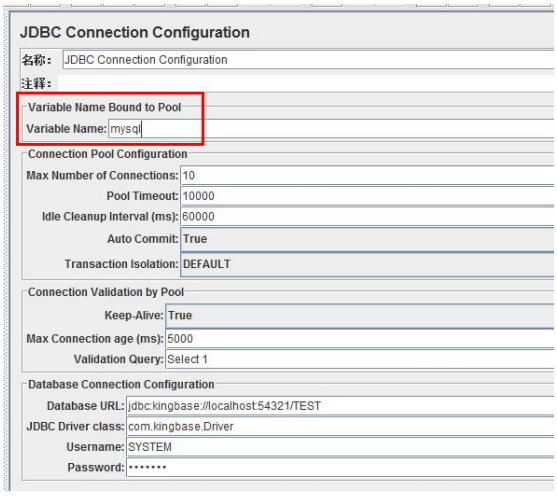
只有在编辑线程组的线程数的时候才能获得不同的 session，如上图，如果一个线程组安排多余 1 的线程数，那显然程序一开始执行就会得到不同的 session，但如果设置执行多次循环，却不能不断的获得新的 session，因为 session 的超时实际和 tomcat 的设置有关，根据时间设置，判断当前 session 是否超时，而是否能获得新的 sessionID。

因此，在执行需要考虑 session 一致性的测试时（用户以登录的身份进行后续操作），首先要把一系列的操作放在一个线程组中，然后根据执行顺序，依靠录制或直接编写 http 请求，并且需要在第一个 http 请求之前，加上 cookie 管理器，（拖动 cookie 管理器图标到第一个 http 请求，选择添加在前。）避免回放时出现故障。

二、使用 JDBC 对数据库进行性能测试

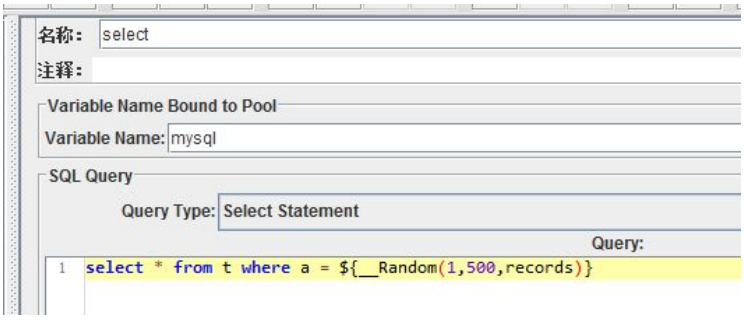
对数据库进行性能测试，无论执行何种操作（TPCC 也好，TPCH 也好），连接方式都是类似的，要不就选择创建数据源做 ODBC 连接，要不就选择用 java 的 jdbc 连接，当然 jmeter 本身包含 jdbc 连接选项，毋庸置疑的还是选择 JDBC 的办法。

1. 使用时，首先创建一个测试计划，并为这个测试导入对应的 jdbc 连接 jar 包：

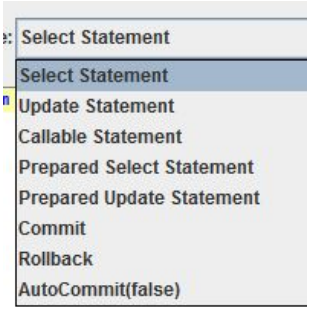


注意，要在这里给绑定的连接池定义一个名字，当然叫什么任意，但是后面的 jdbc 请求需要他。

根据需要，添加 jdbc 请求，下面用一个例子做简单介绍：



截图是一个 select 类型的请求，注意绑定的连接池名称一定要填写，否则访问会报错。
这里关于参数的使用，虽然网上很多的截图表名，可以使用“?”加下面配置的办法，但我用多个版本的 jmeter 都提示莫名其妙的错误，因此我改用通过函数助手的方法生成随机数，或使用“用户参数”的办法创建需要的参数了。
再一个需要注意的是查询类型这个下拉菜单，根据 SQL 不同而不同，基本上如果涉及表修改的，就要选择 UPDATE，没有修改的就用 SELECT 的。



其中最常用的是 select statement 和 update statement。这两种查询类型中，一次只能包含一句 SQL，并且结尾没有“;”

使用 Callable statement 时，一次可以包含多个 SQL，但非结尾的 SQL 要加“;”

其余的 Prepared、commit、Rollback、AutoCommit(false)是调用 jdbc 的内部方法，配合实际 SQL 决定使用，暂没有过多研究。

三、使用 Jmeter 的 java request 测试 java class 的性能方法

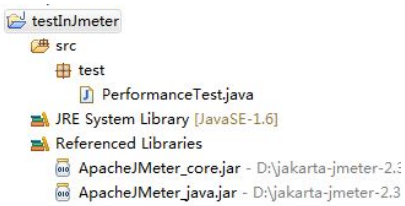
对比 Loadrunner 的 java vuser，在 jmeter 中，我们可以通过使用 java request 的方法，实现对普通 java 类的测试。

有的人说，可以利用这个功能进行白盒或灰盒测试，说实在的，白盒测试已经有 junit 的框架了，jmeter 无非就是可以直观的把需要的参数变量一次性的发送给 class，他在执行白盒测试时候的优势我倒没看出来，并且有些判断仅仅用 jmeter 的断言还不好解决，可能还得在 class 里自己写一些判断，增加了代码的复杂度，所以如果是我的话，我就老老实实用 jmeter 压性能，白盒的工作仍然交给 junit 去执行。

区别于使用 JDBC 测试数据库的性能，这里需要在线程组下添加一个 java 请求（java request），可以根据实际情况适当决定是否需要添加“java 请求默认值”。这是使用 jmeter 测试 class 性能的基础。

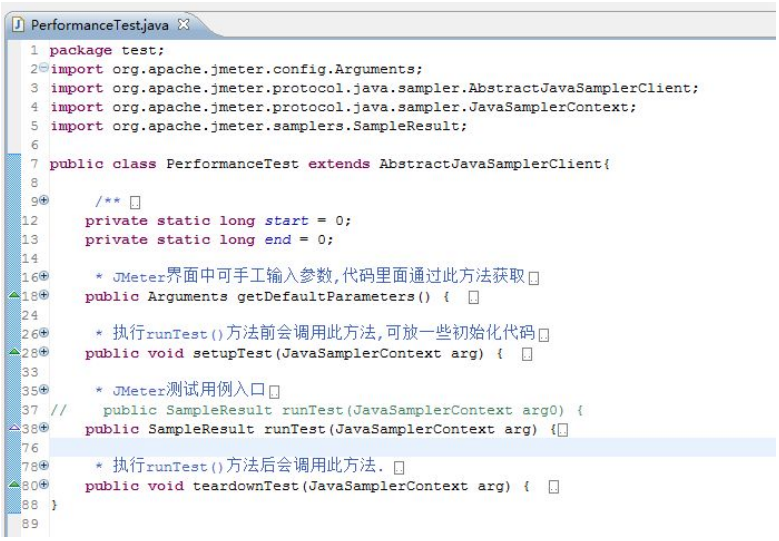
接下来是准备被测的 class 文件，和使用 junit 框架可以执行单元测试一样，使用 jmeter 的框架可以帮助我们编写测试 class 性能的脚本。

首先在 eclipse 等 IDE 中建立一个 java 工程，



通过导入外部库的办法，把 jmeter 目录里，Lib 中 ext 目录中的 core 和 java 两个 jar 包导进工程。

接着在 src 下根据使用情况创建包和类，以上图为例，PerformanceTest.java 里包含要测试性能的内容。



PerformanceTest.java

```
1  package test;
2  import org.apache.jmeter.config.Arguments;
3  import org.apache.jmeter.protocol.java.sampler.AbstractJavaSamplerClient;
4  import org.apache.jmeter.protocol.java.sampler.JavaSamplerContext;
5  import org.apache.jmeter.samplers.SampleResult;
6
7  public class PerformanceTest extends AbstractJavaSamplerClient{
8
9      /**
12     private static long start = 0;
13     private static long end = 0;
14
16     * JMeter界面中可手工输入参数,代码里面通过此方法获取
18     public Arguments getDefaultParameters() {
24
26     * 执行runTest()方法前会调用此方法,可放一些初始化代码
28     public void setupTest(JavaSamplerContext arg) {
33
35     * JMeter测试用例入口
37 //    public SampleResult runTest(JavaSamplerContext arg0) {
38     public SampleResult runTest(JavaSamplerContext arg) {
76
78     * 执行runTest()方法后会调用此方法.
80     public void teardownTest(JavaSamplerContext arg) {
88 }
89
```

如图，首先在 java 中引入框架需要的包：

```
import org.apache.jmeter.config.Arguments;

import org.apache.jmeter.protocol.java.sampler.AbstractJavaSamplerClient;

import org.apache.jmeter.protocol.java.sampler.JavaSamplerContext;

import org.apache.jmeter.samplers.SampleResult;
```

其次要使当前的 class 继承 `AbstractJavaSamplerClient` 这个方法

后面的代码分为四个部分：

1 getDefaultParameters：用来接收图形界面传来的参数；

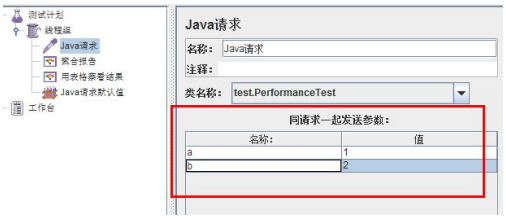
2 setupTest：runTest 运行前执行，可存放一下准备代码；

3 teardownTest：runTest 运行后执行；

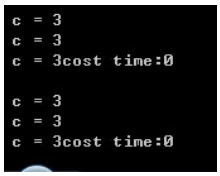
4 runTest：测试的关键，这里面存放的是要检查性能的方法：

```
//    public SampleResult runTest(JavaSamplerContext arg0) {
    public SampleResult runTest(JavaSamplerContext arg) {

            SampleResult sr = new SampleResult();

            try {

                    // Start
                    sr.sampleStart();

                    /**
                     * Start~End内的代码会被JMeter
                     * 纳入计算吞吐量的范围内,为了使
                     * 性能结果合理,无关代码不必放此
                     */

                    // TODO
                    int a = Integer.valueOf(arg.getParameter("a"));
                    int b = Integer.valueOf(arg.getParameter("b"));
                    int c = a + b;
                    System.out.print("\n c = "+c);

                    /**
                     * True/False可按测试逻辑传值
                     * JMeter会对失败次数做出统计
                     */
                    sr.setSuccessful(true);

                    // End
                    sr.sampleEnd();

            } catch (Exception e) {

                    e.printStackTrace();
            }

            return sr;
```

这里需要注意的是参数的传递，截图中整型变量 a 和 b 来自图形工具，使用 arg.getParameter 的方法可以取到值。



参数名称要相互对应。



执行的时候，根据 class 中的代码，可以将执行结果或特定信息打印到后台日志。

全部代码：

```
package test;

import org.apache.jmeter.config.Arguments;

import org.apache.jmeter.protocol.java.sampler.AbstractJavaSamplerClient;

import org.apache.jmeter.protocol.java.sampler.JavaSamplerContext;

import org.apache.jmeter.samplers.SampleResult;


public class PerformanceTest extends AbstractJavaSamplerClient{

    /**

     *
```

```
     */
    private static long start = 0;
    private static long end = 0;

    /**
     * JMeter界面中可手工输入参数,代码里面通过此方法获取
     */
    public Arguments getDefaultParameters() {

        Arguments args = new Arguments();

        return args;
    }

    /**
     * 执行runTest()方法前会调用此方法,可放一些初始化代码
     */
    public void setupTest(JavaSamplerContext arg) {

        // 开始时间
        start = System.currentTimeMillis();
    }

    /**
     * JMeter测试用例入口
     */
//    public SampleResult runTest(JavaSamplerContext arg0) {
    public SampleResult runTest(JavaSamplerContext arg) {

        SampleResult sr = new SampleResult();

        try {

            // Start
            sr.sampleStart();

            /**
             * Start~End内的代码会被JMeter
             * 纳入计算吞吐量的范围内,为了使
             * 性能结果合理,无关代码不必放此
             */

            // TODO
            int a = Integer.valueOf(arg.getParameter("a"));
            int b = Integer.valueOf(arg.getParameter("b"));
```

```
            int c = a + b;
            System.out.print("\n c = "+c);

             /**
              * True/False可按测试逻辑传值
              * JMeter会对失败次数做出统计
              */
             sr.setSuccessful(true);

             // End
             sr.sampleEnd();

        } catch (Exception e) {

             e.printStackTrace();
        }

        return sr;

}

    /**
     * 执行runTest()方法后会调用此方法.
     */
    public void teardownTest(JavaSamplerContext arg) {

        // 结束时间
        end = System.currentTimeMillis();

        // 总体耗时
        System.err.println("cost time:" + (end - start) / 1000);
    }
}
```

四、在 Jmeter 的场景中使用集合点

使用“固定定时器”模拟集合点效果

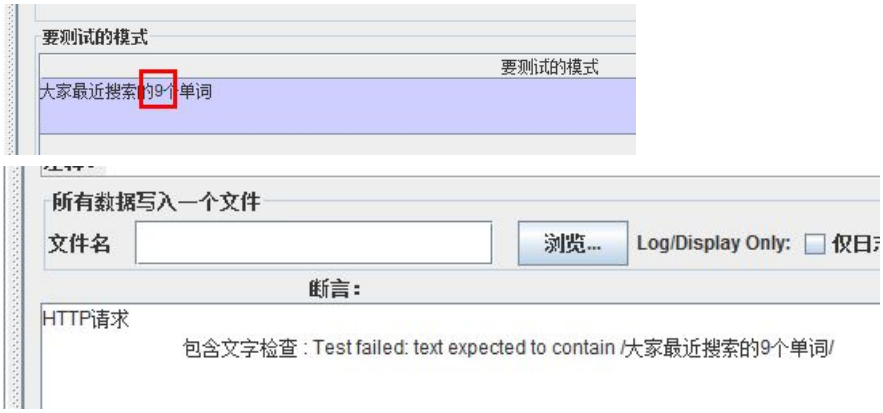
五、在 Jmeter 的场景中使用检查点和断言

最常用的是“响应断言”，加在要检查的响应信息下面。

要检查的页面和内容

在 Jmeter 上添加了对应的内容

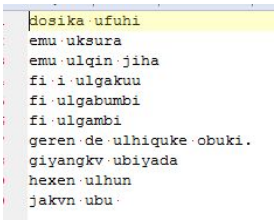
断言成功：不提示有错误

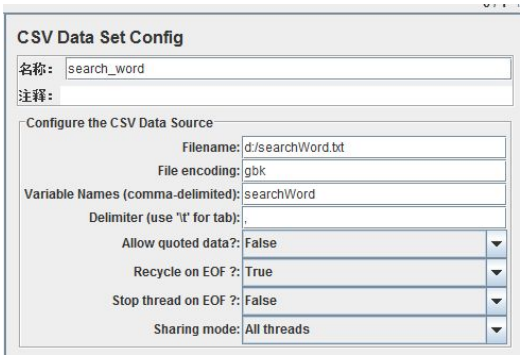


断言失败：提示文本域找不到

六、使用 Jmeter 加载变量的第一种方法：使用函数助手

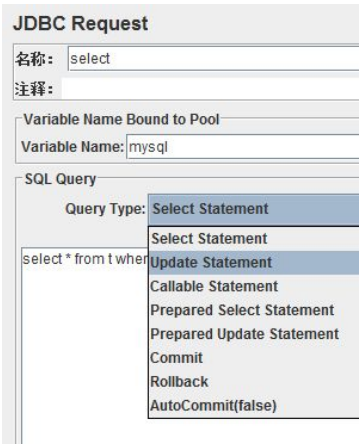
七、使用 Jmeter 加载变量的第二种方法：CSV 文件

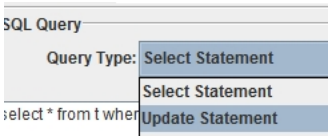




八、关于 JDBC 的 QUERY TYPE 的讨论

在 jmeter 的 jdbc 请求中，可以选择不同的 Query Type,
根据不同的选择，可以创建复杂的查询场景。





JDBC 最基本的两个操作，查询和更新。

select：ResultSet rs = stmt.executeQuery(sql) ;

update： int rows = stmt.executeUpdate("INSERT INTO ORDERS (ISBN, CUSTOMERID) VALUES(195123018, 'BILLING')", Statement.RETURN_GENERATED_KEYS);

JDBC 的两个预编译语句，执行效果和前面两个相同，但是相对更安全，避免攻击操作。

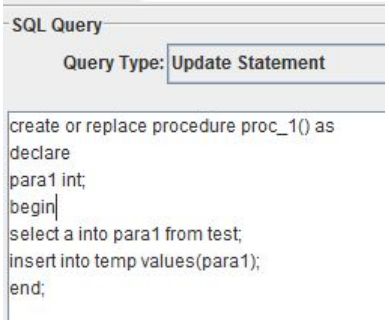
prepared select：ResultSet rs = PreparedStatement pstmt .executeQuery(sql) ;

prepared update： PreparedStatement pstmt = conn.prepareStatement("UPDATE EMPLOYEES SET SALARY = ? WHERE ID = ?");

create table test(a int);

insert into test values(1);

create table temp(b int);




Variable Name: mysql
SQL Query
Query Type: Callable Statement
call proc_1();