淘淘商城

配置tomcat插件:

在maven工程,war工程 pom.xml文件下:

<build>

<plugins>

<plugin>

<groupId>org.apache.tomcat.maven</groupId><--组织-->

<artifactId>tomcat7-maven-plugin</artifactId>

<configuration>

<path>/</path><--访问路径-->

<port>8081</port><-- 端口号 -->

</configuration>

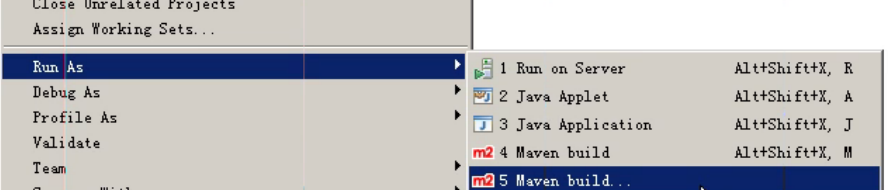
</plugin>

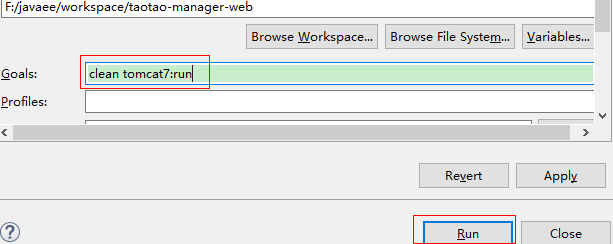
</plugins>

</build>

运行插件:

在maven项目上点击鼠标右键:





配置文件:

Dubbo

Dubbo 使用java开发,只能用于java项目之间的通信

Dubbo就是资源调度和治理中心的管理工具

使用rpc协议进行远程调用,直接使用socket通信.传输效率高,并且可以统计出系统之间的调用关系和调用次数.

流动计算架构

当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。

此时，用于提高机器利用率的 资源调度和治理中心(SOA) 是关键

Dubbo超时:1秒一次,三次抛异常

## 系统间通信

### 分析

由于淘淘商城是基于soa的架构，表现层和服务层是不同的工程。所以要实现商品列表查询需要两个系统之间进行通信。

如何实现远程通信？

1. Webservice：效率不高基于soap协议。项目中不推荐使用。
2. 使用restful形式的服务：http+json。很多项目中应用。如果服务太多，服务之间调用关系混乱，需要治疗服务。
3. 使用dubbo。使用rpc协议进行远程调用，直接使用socket通信。传输效率高，并且可以统计出系统之间的调用关系、调用次数。

### 什么是dubbo

随着互联网的发展，网站应用的规模不断扩大，常规的垂直应用架构已无法应对，分布式服务架构以及流动计算架构势在必行，亟需一个治理系统确保架构有条不紊的演进。



* **单一应用架构**
  + 当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。
  + 此时，用于简化增删改查工作量的 **数据访问框架(ORM)** 是关键。
* **垂直应用架构**
  + 当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。
  + 此时，用于加速前端页面开发的 **Web框架(MVC)** 是关键。
* **分布式服务架构**
  + 当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。
  + 此时，用于提高业务复用及整合的 **分布式服务框架(RPC)** 是关键。
* **流动计算架构**
  + 当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。
  + 此时，用于提高机器利用率的 **资源调度和治理中心(SOA)** 是关键。

Dubbo就是资源调度和治理中心的管理工具。

### Dubbo的架构



节点角色说明：

Provider: 暴露服务的服务提供方。

Consumer: 调用远程服务的服务消费方。

Registry: 服务注册与发现的注册中心。

Monitor: 统计服务的调用次调和调用时间的监控中心。

Container: 服务运行容器。

调用关系说明：

0. 服务容器负责启动，加载，运行服务提供者。

1. 服务提供者在启动时，向注册中心注册自己提供的服务。

2. 服务消费者在启动时，向注册中心订阅自己所需的服务。

3. 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。

4. 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。

5. 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

### 使用方法

Dubbo采用全Spring配置方式，透明化接入应用，对应用没有任何API侵入，只需用Spring加载Dubbo的配置即可，Dubbo基于Spring的Schema扩展进行加载。

**单一工程中spring的配置**

|  |
| --- |
| <bean id="xxxService" class="com.xxx.XxxServiceImpl" />  <bean id="xxxAction" class="com.xxx.XxxAction">  <property name="xxxService" ref="xxxService" />  </bean> |

**远程服务：**

在本地服务的基础上，只需做简单配置，即可完成远程化：

将上面的local.xml配置拆分成两份，将服务定义部分放在服务提供方remote-provider.xml，将服务引用部分放在服务消费方remote-consumer.xml。

并在提供方增加暴露服务配置<dubbo:service>，在消费方增加引用服务配置<dubbo:reference>。

发布服务：

|  |
| --- |
| <!-- 和本地服务一样实现远程服务 -->  <bean id="xxxService" class="com.xxx.XxxServiceImpl" />  <!-- 增加暴露远程服务配置 -->  <dubbo:service interface="com.xxx.XxxService" ref="xxxService" /> |

调用服务：

|  |
| --- |
| <!-- 增加引用远程服务配置 -->  <dubbo:reference id="xxxService" interface="com.xxx.XxxService" />  <!-- 和本地服务一样使用远程服务 -->  <bean id="xxxAction" class="com.xxx.XxxAction">  <property name="xxxService" ref="xxxService" />  </bean> |

### 注册中心

注册中心负责服务地址的注册与查找，相当于目录服务，服务提供者和消费者只在启动时与注册中心交互，注册中心不转发请求，压力较小。使用dubbo-2.3.3以上版本，建议使用zookeeper注册中心。

Zookeeper是Apacahe Hadoop的子项目，是一个树型的目录服务，支持变更推送，适合作为Dubbo服务的注册中心，工业强度较高，可用于生产环境，并推荐使用

**Zookeeper的安装：**

第一步：安装jdk

第二步：解压缩zookeeper压缩包

第三步：将conf文件夹下zoo\_sample.cfg复制一份，改名为zoo.cfg

第四步：修改配置dataDir属性，指定一个真实目录

第五步：

启动zookeeper：bin/zkServer.sh start

关闭zookeeper：bin/zkServer.sh stop

查看zookeeper状态：bin/zkServer.sh status

注意要关闭linux的防火墙。

### FastDFS 服务器

FastDFS是用c语言编写的一款开源的分布式文件系统。FastDFS为互联网量身定制，充分考虑了冗余备份、负载均衡、线性扩容等机制，并注重高可用、高性能等指标，使用FastDFS很容易搭建一套高性能的文件服务器集群提供文件上传、下载等服务。

因为中央仓库中没有FastDFS的jar包所以要自己手动安装到本地仓库中

Redis集群搭建

首先linux中已安装一个redis(在usr/local/目录下)

1) 在usr/local目录下创建一个redis-cluster目录:mkder redis-cluster

2) 再复制redis是把快照文件删除



3) 要搭建集群需要把:

(表示为集群文件)打开,并修改集群端口号

[root@itcast redis-cluster]# vi redis04/bin/redis.conf (因为在一台虚拟机下搭建的伪集群)

3) 将复制redis/bin/ 复制到redis-cluster/中:



复制6个

4) 修改集群端口号

[root@itcast redis-cluster]# vi redis04/bin/redis.conf

创建批处理文件启动所有集群:

Jetty和tomcat类似

在linux下安装solr

准备:因为solr是java项目所以需要安装java环境;要发布到tomcat中所以也需要安装tomcat环境;

1) 将solr安装包上传到linux中,并解压: tar –zxvf 压缩文件

2) 将 dist目录下的: solr-4.10.3.war 发布到tomcat中

3) 将 solr目录下的example/lib/ext下的所有jar包拷到tomcat工程下的lib

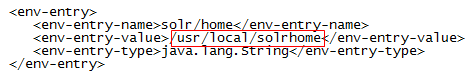


4) 复制solrhome:

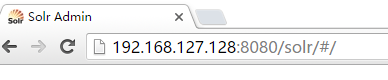


5) 修改项目中的web.xml文件:





验证是否配置成功:



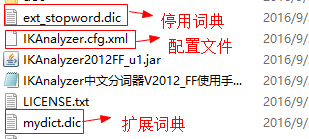
配置中文解析器

1) 复制到linux

2) 将:



3) 将



移动到solr项目下的classes目录下

修改solrhome中的conf总的schema.xml:

添加:

<fieldType name="text\_ik" class="solr.TextField">

<analyzer class="org.wltea.analyzer.lucene.IKAnalyzer"/>

</fieldType>

<field name="item\_title" type="text\_ik" indexed="true" stored="true"/>

<field name="item\_sell\_point" type="text\_ik" indexed="true" stored="true"/>

<field name="item\_price" type="long" indexed="true" stored="true"/>

<field name="item\_image" type="string" indexed="false" stored="true" />

<field name="item\_category\_name" type="string" indexed="true" stored="true" />

<field name="item\_desc" type="text\_ik" indexed="true" stored="false" />

<field name="item\_keywords" type="text\_ik" indexed="true" stored="false" multiValued="true"/>

<copyField source="item\_title" dest="item\_keywords"/>

<copyField source="item\_sell\_point" dest="item\_keywords"/>

<copyField source="item\_category\_name" dest="item\_keywords"/>

<copyField source="item\_desc" dest="item\_keywords"/>

activeMQ

消息队列

Queue方式

只有一个消费者,消费后就不存在了

一对一,点到点的方式

默认是持久化的

Topic广播

可以有多个消费者消费

不是持久化的;默认不缓存不会在服务端存储

基础代码:

发送消息:

//1.创建一个连接工厂对象ConnectionFactory对象。需要指定mq服务的ip及端口

ConnectionFactory connectionFactory = **new** ActiveMQConnectionFactory("tcp://192.168.25.168:61616");

//2.使用ConnectionFactory创建一个连接Connection对象

Connection connection = connectionFactory.createConnection();

//3.开启连接。调用Connection对象的start方法

connection.start();

//4.使用Connection对象创建一个Session对象

//第一个参数是是否开启事务，一般不使用事务。保证数据的最终一致，可以使用消息队列实现。

//如果第一个参数为true，第二个参数自动忽略。如果不开启事务false，第二个参数为消息的应答模式。一般自动应答就可以。

Session session = connection.createSession(**false**, Session.***AUTO\_ACKNOWLEDGE***);

//5.使用Session对象创建一个Destination对象，两种形式queue、topic。现在应该使用queue

//参数就是消息队列的名称

Queue queue = session.createQueue("test-queue");

//6.使用Session对象创建一个Producer对象

MessageProducer producer = session.createProducer(queue);

//7.创建一个TextMessage对象

/\*TextMessage textMessage = new ActiveMQTextMessage();

textMessage.setText("hello activemq");\*/

TextMessage textMessage = session.createTextMessage("hello activemq1111");

//8.发送消息

producer.send(textMessage);

//9.关闭资源

producer.close();

session.close();

connection.close();

接收消息:

//创建一个连接工厂对象

ConnectionFactory connectionFactory = **new** ActiveMQConnectionFactory("tcp://192.168.25.168:61616");

//使用连接工厂对象创建一个连接

Connection connection = connectionFactory.createConnection();

//开启连接

connection.start();

//使用连接对象创建一个Session对象

Session session = connection.createSession(**false**, Session.***AUTO\_ACKNOWLEDGE***);

//使用Session创建一个Destination，Destination应该和消息的发送端一致。

Queue queue = session.createQueue("test-queue");

//使用Session创建一个Consumer对象

MessageConsumer consumer = session.createConsumer(queue);

//向Consumer对象中设置一个MessageListener对象，用来接收消息

consumer.setMessageListener(**new** MessageListener() {

@Override

**public** **void** onMessage(Message message) {

//取消息的内容

**if** (message **instanceof** TextMessage) {

TextMessage textMessage = (TextMessage) message;

**try** {

String text = textMessage.getText();

//打印消息内容

System.***out***.println(text);

} **catch** (JMSException e) {

e.printStackTrace();

}

}

}

});

//系统等待接收消息

/\*while(true) {

Thread.sleep(100);

}\*/

System.***in***.read();

//关闭资源

consumer.close();

session.close();

connection.close();

### 单点登录

Sso登录执行流程:



登录的处理流程：

1. 登录页面提交用户名密码。
2. 登录成功后生成token。Token相当于原来的jsessionid，字符串，可以使用uuid。
3. 把用户信息保存到redis。Key就是token，value就是TbUser对象转换成json。
4. 使用String类型保存Session信息。可以使用“前缀:token”为key
5. 设置key的过期时间。模拟Session的过期时间。一般半个小时。
6. 把token写入cookie中。
7. Cookie需要跨域。例如www.taotao.com\sso.taotao.com\order.taotao.com，可以使用工具类。
8. Cookie的有效期。关闭浏览器失效。
9. 登录成功。

总结:将登录成功后的用户信息存储到redis中(使用string类型存储,key用UUID生成,并设置存活时间)

将key存储到cookie;跨工程时去cookie中查找key,通过key查询用户信息