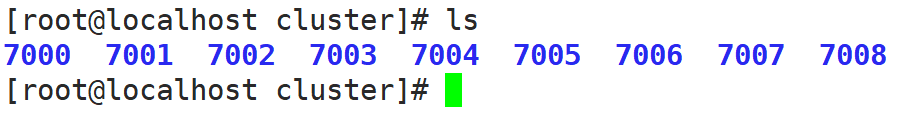
# 搭建redis集群

## 集群搭建步骤

### 创建文件夹

说明:采用1主2从.3个主节点的形式.共9台

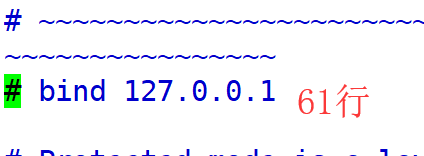


### 编辑配置文件

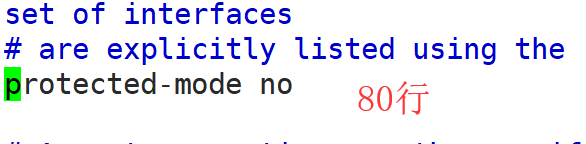
1. 将redis根目录中的redis.conf文件导入7000/下

cp redis.conf cluster/7000/

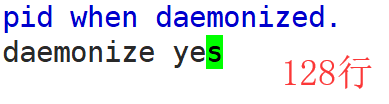
1. 去除IP绑定



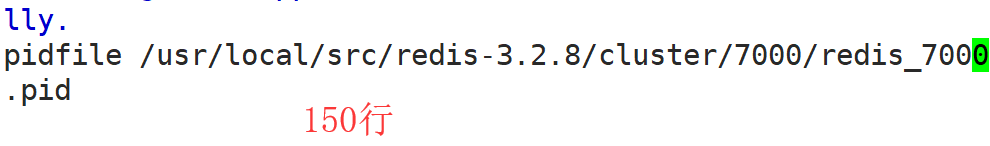
1. 关闭保护模式



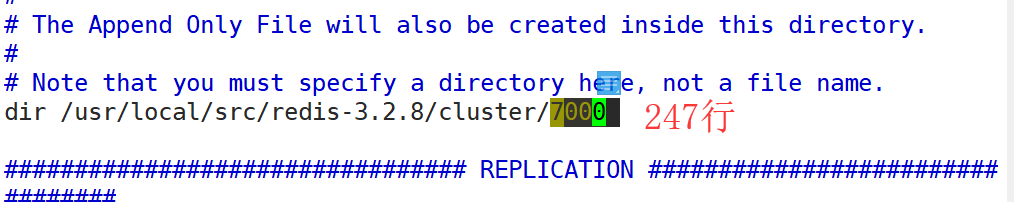
1. 开启后台启动



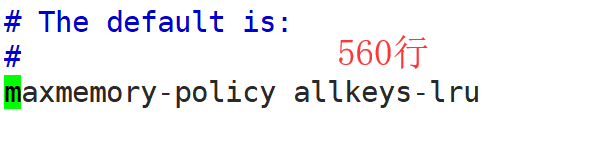
1. 修改PID的保存路径



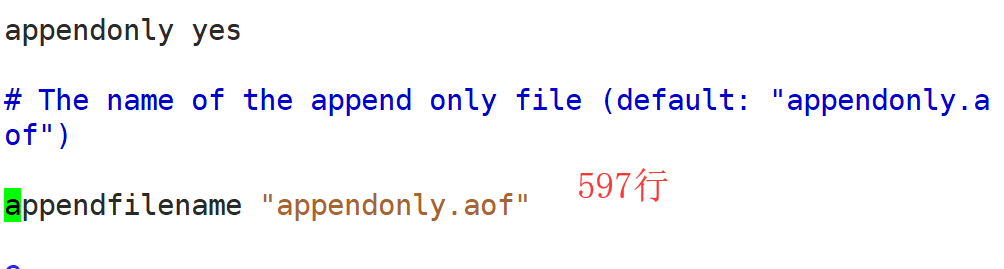
1. 修改文件夹路径



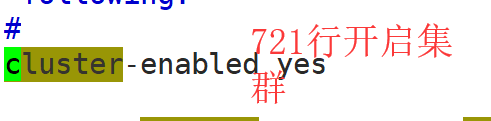
1. 修改内存策略



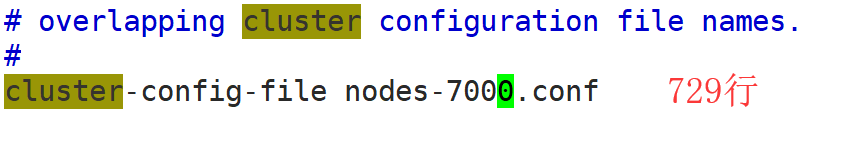
1. 修改持久化策略



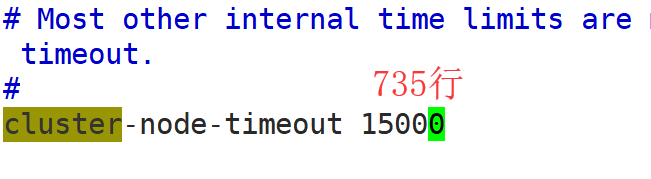
1. 开启集群



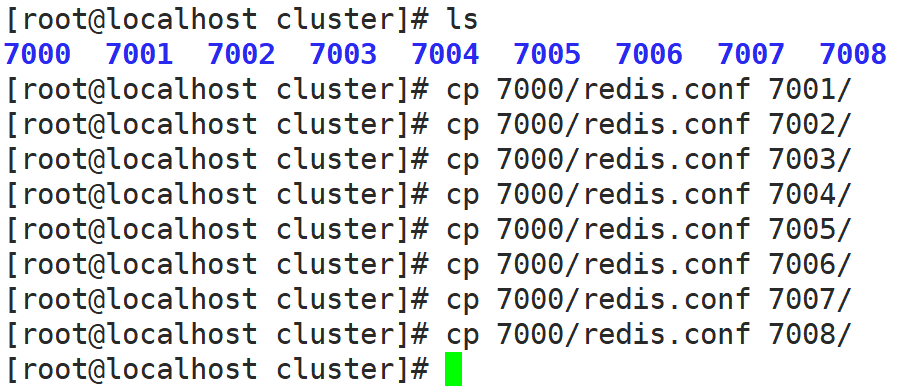
1. 开启集群配置文件



10.开启超时时间



11.将配置文件复制



12.批量修改端口号

分别修改7000-7008中的全部端口号

:%s/7000/7001/g

### 编辑脚本

启动脚本编辑:

#!/bin/sh

redis-server 7000/redis.conf &

redis-server 7001/redis.conf &

redis-server 7002/redis.conf &

redis-server 7003/redis.conf &

redis-server 7004/redis.conf &

redis-server 7005/redis.conf &

redis-server 7006/redis.conf &

redis-server 7007/redis.conf &

redis-server 7008/redis.conf &

~

~

~

:wq

执行启动脚本

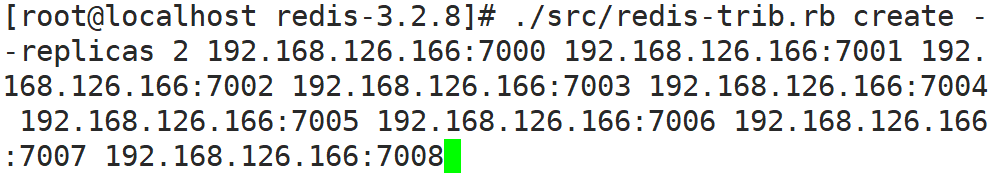
sh start.sh

### 搭建redis集群

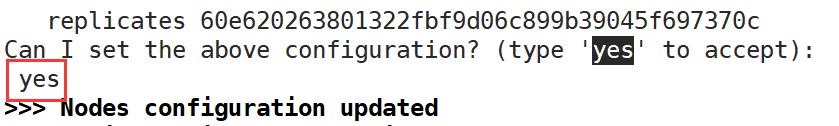
命令:

./src/redis-trib.rb create --replicas 2 192.168.126.166:7000 192.168.126.166:7001 192.168.126.166:7002 192.168.126.166:7003 192.168.126.166:7004 192.168.126.166:7005 192.168.126.166:7006 192.168.126.166:7007 192.168.126.166:7008

replicas 2 一台主机有2台从机

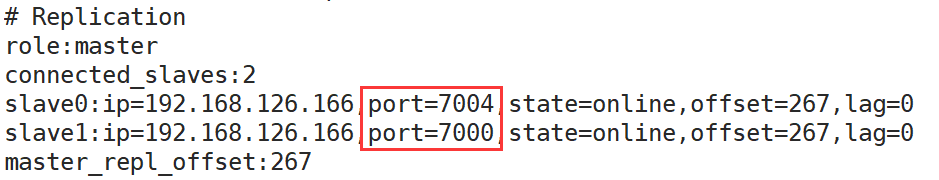


输入yes确定消息



### 集群检测

当关闭redis主节点时,检测redis能否实现高可用.



## Spring整合集群

### 入门案例

@Test

**public** **void** test04(){

JedisPoolConfig poolConfig = **new** JedisPoolConfig();

poolConfig.setMaxTotal(1000);

//添加集群的节点信息

Set<HostAndPort> nodes = **new** HashSet<HostAndPort>();

nodes.add(**new** HostAndPort("192.168.126.166",7000));

nodes.add(**new** HostAndPort("192.168.126.166",7001));

nodes.add(**new** HostAndPort("192.168.126.166",7002));

nodes.add(**new** HostAndPort("192.168.126.166",7003));

nodes.add(**new** HostAndPort("192.168.126.166",7004));

nodes.add(**new** HostAndPort("192.168.126.166",7005));

nodes.add(**new** HostAndPort("192.168.126.166",7006));

nodes.add(**new** HostAndPort("192.168.126.166",7007));

nodes.add(**new** HostAndPort("192.168.126.166",7008));

JedisCluster cluster =

**new** JedisCluster(nodes, poolConfig);

cluster.set("1804", "集群搭建成功!!!!!");

System.***out***.println(cluster.get("1804"));

}

### 作业(选作)

自己根据redis集群入门案例.实现spring整合.通过配置文件实例化jedisCluster对象.

## Spring中工厂模式

### 工厂模式的作用

主旨:实例化对象

为什么使用工厂模式:

1. 简化代码的编辑 减低耦合性
2. 有些类可能是接口或者抽象类不能直接实例化.

### 静态工厂

核心:通过静态方法实例化对象

调用方式:Class.静态方法()

例子:获取当前时间 Calendar对象

1.配置文件

<!--Spring中静态工厂

spring解析方式:

1.当spring容器加载bean标签时,这时实例化对象

2.spring容器加载工厂类.通过factory-method中标识的方法

实例化对象.

3.spring将id和对象进行保存.方便对象获取.

-->

<bean id=*"calendar1"*

class=*"com.jt.manage.factory.StaticFactory"*

factory-method=*"getInstance"*/>

2.测试代码

@Test

**public** **void** testStaticFactory(){

ApplicationContext context =

**new** ClassPathXmlApplicationContext("/spring/factory.xml");

Calendar calendar = (Calendar) context.getBean("calendar1");

System.***out***.println("获取时间:"+calendar.getTime());

}

### 实例化工厂

核心:通过对象.方法创建对象

配置文件

<!--spring中实例工厂 -->

<bean id=*"instanceFacotry"*

class=*"com.jt.manage.factory.InstanceFacotry"*/>

<bean id=*"calendar2"* factory-bean=*"instanceFacotry"*

factory-method=*"getInstance"*/>

### Spring工厂模式

说明:通过实现特定的接口,之后回调实例化对象

1. 配置文件

<!--Spring工厂模式 -->

<bean id=*"calendar3"* class=*"com.jt.manage.factory.SpringFactory"*/>

1. 工厂模式编辑

**public** **class** SpringFactory **implements** FactoryBean<Calendar>{

@Override

**public** Calendar getObject() **throws** Exception {

System.***out***.println("我是spring工厂模式");

**return** Calendar.*getInstance*();

}

@Override

**public** Class<?> getObjectType() {

**return** Calendar.**class**;

}

@Override

**public** **boolean** isSingleton() {

**return** **true**;

}

}

### 测试方法

@Test

**public** **void** testStaticFactory(){

ApplicationContext context =

**new** ClassPathXmlApplicationContext("/spring/factory.xml");

Calendar calendar = (Calendar) context.getBean("calendar1");

Calendar calendar2 = (Calendar) context.getBean("calendar2");

Calendar calendar3 = (Calendar) context.getBean("calendar3");

System.***out***.println("获取时间:"+calendar.getTime());

System.***out***.println("获取时间:"+calendar2.getTime());

System.***out***.println("获取时间:"+calendar3.getTime());

}

## Spring整合redis集群

### 编辑pro配置文件

#最小空闲数

redis.minIdle=100

#最大空闲数

redis.maxIdle=300

#最大连接数

redis.maxTotal=1000

#客户端超时时间单位是毫秒

redis.timeout=5000

#最大建立连接等待时间

redis.maxWait=1000

#是否在从池中取出连接前进行检验,如果检验失败,则从池中去除连接并尝试取出另一个

redis.testOnBorrow=true

#redis cluster

redis.cluster0.host.port=192.168.126.166:7000

redis.cluster1.host.port=192.168.126.166:7001

redis.cluster2.host.port=192.168.126.166:7002

redis.cluster3.host.port=192.168.126.166:7003

redis.cluster4.host.port=192.168.126.166:7004

redis.cluster5.host.port=192.168.126.166:7005

redis.cluster6.host.port=192.168.126.166:7006

redis.cluster7.host.port=192.168.126.166:7007

redis.cluster8.host.port=192.168.126.166:7008

### 编辑redis配置文件

<!-- jedis 配置-->

<bean id=*"poolConfig"* class=*"redis.clients.jedis.JedisPoolConfig"* >

<!--最大空闲数-->

<property name=*"maxIdle"* value=*"${redis.maxIdle}"* />

<!--最大建立连接等待时间-->

<property name=*"maxWaitMillis"* value=*"${redis.maxWait}"* />

<!--是否在从池中取出连接前进行检验,如果检验失败,则从池中去除连接并尝试取出另一个-->

<property name=*"testOnBorrow"* value=*"${redis.testOnBorrow}"* />

<property name=*"maxTotal"* value=*"${redis.maxTotal}"* />

<property name=*"minIdle"* value=*"${redis.minIdle}"* />

</bean>

<!--利用spring工厂实例化对象 -->

<bean id=*"jedisCluster"* class=*"com.jt.common.fatory.JedisClusterFactory"* >

<property name=*"redisSource"*>

<value>classpath:/properties/redis.properties</value>

</property>

<property name=*"perKey"* value=*"redis.cluster"* />

<property name=*"poolConfig"* ref=*"poolConfig"* />

</bean>

### 编辑工具类

**public** **class** JedisClusterFactory **implements** FactoryBean<JedisCluster> {

**private** JedisPoolConfig poolConfig;

**private** Resource redisSource;

**private** String perKey; //表示前缀

@Override

**public** JedisCluster getObject() **throws** Exception {

Set<HostAndPort> nodes = getNodes();

JedisCluster jedisCluster =

**new** JedisCluster(nodes, poolConfig);

**return** jedisCluster;

}

**public** Set<HostAndPort> getNodes(){

Set<HostAndPort> nodes =

**new** HashSet<HostAndPort>();

Properties properties = **new** Properties();

**try** {

properties.load(redisSource.getInputStream());

//遍历Pro中的数据获取redis节点数据信息

**for** (Object node : properties.keySet()) {

String strNode = (String) node;

//判断是否为redis节点数据信息

**if**(strNode.startsWith(perKey)){

//表示是redis节点信息的key 192.168.126.166:7000

String value =

properties.getProperty(strNode);

String[] args = value.split(":");

HostAndPort hostAndPort =

**new** HostAndPort(args[0],Integer.*parseInt*(args[1]));

nodes.add(hostAndPort);

}

}

} **catch** (Exception e) {

e.printStackTrace();

}

**return** nodes;

}

@Override

**public** Class<?> getObjectType() {

**return** JedisCluster.**class**;

}

@Override

**public** **boolean** isSingleton() {

**return** **false**;

}

**public** JedisPoolConfig getPoolConfig() {

**return** poolConfig;

}

**public** **void** setPoolConfig(JedisPoolConfig poolConfig) {

**this**.poolConfig = poolConfig;

}

**public** Resource getRedisSource() {

**return** redisSource;

}

**public** **void** setRedisSource(Resource redisSource) {

**this**.redisSource = redisSource;

}

**public** String getPerKey() {

**return** perKey;

}

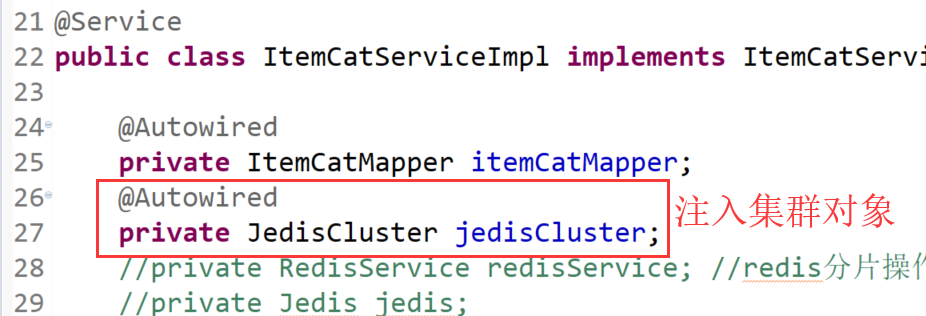
**public** **void** setPerKey(String perKey) {

**this**.perKey = perKey;

}

}

### 集群使用



# 京淘项目前台

## 京淘项目前台搭建

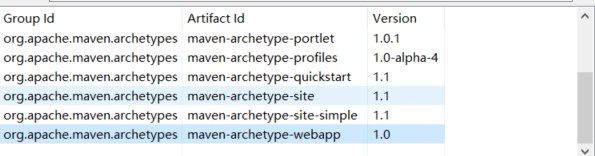
### 搭建说明

京淘前台项目只负责数据的页面展现.不需要连接数据库.所有的业务操作通过HTTP协议/RPC协议.实现参数的远程传输.

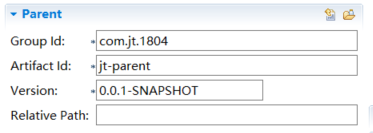
分布式的设计方式为了架构层级的解耦.

### 创建jt-web项目

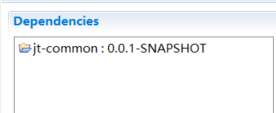
1. 选择骨架



1. 添加继承和依赖
2. 添加继承



1. 添加依赖



3.配置tomcat插件

<build>

<plugins>

<!--跳过测试类打包 -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<configuration>

<skip>true</skip>

</configuration>

</plugin>

<plugin>

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

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

<version>2.2</version>

<configuration>

<!--端口号 -->

<port>8092</port>

<!--项目发布路径 -->

<path>/</path>

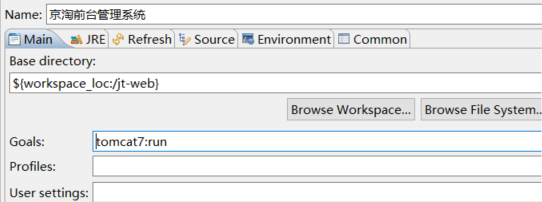
</configuration>

</plugin>

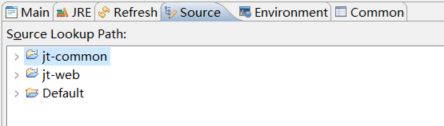
</plugins>

</build>

4.配置tomcat插件



2.添加源码



### 实现反向代理

1. 编辑nginx配置文件,编辑完成后重启Nginx

#前台管理系统

server {

listen 80;

server\_name www.jt.com;

location / {

proxy\_pass http://localhost:8092;

proxy\_connect\_timeout 1;

proxy\_read\_timeout 1;

proxy\_send\_timeout 1;

}

}

1. 修改hosts文件

# 京淘电商环境

127.0.0.1 image.jt.com

127.0.0.1 manage.jt.com

127.0.0.1 www.jt.com

127.0.0.1 sso.jt.com

127.0.0.1 cart.jt.com

127.0.0.1 order.jt.com

127.0.0.1 solr.jt.com

## 京淘首页展现

### 修改web.xml

<!--

监听器方式启动Spring容器

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/applicationContext\*.xml</param-value>

</context-param>

Spring的ApplicationContext 载入

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener> -->

<!-- 编码过滤器，以UTF8编码 -->

<filter>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 配置SpringMVC -->

<servlet>

<servlet-name>springmvc-web</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- 指定SpringMVC配置文件 -->

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:/spring/applicationContext\*.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>springmvc-web</servlet-name>

<url-pattern>\*.html</url-pattern>

</servlet-mapping>

<!-- 防止springMVC框架返回json时和html冲突报 406 错误 -->

<servlet-mapping>

<servlet-name>springmvc-web</servlet-name>

<url-pattern>/service/\*</url-pattern>

</servlet-mapping>

<welcome-file-list>

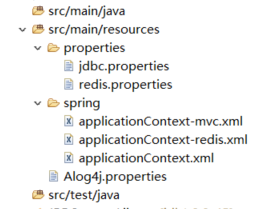
<welcome-file>index.html</welcome-file>

<welcome-file>index.jsp</welcome-file>

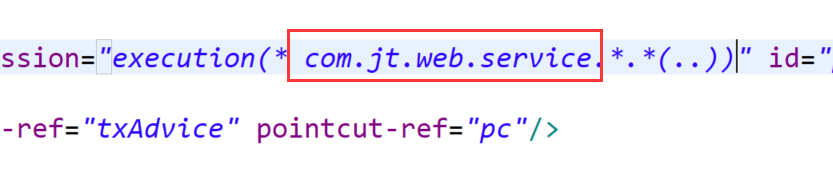
</welcome-file-list>

### 编辑spring配置文件

1. 删除mybatis配置文件后保留图



1. 修改切入点表达式路径



### 实现页面跳转

@Controller

**public** **class** IndexController {

//跳转index.jsp页面

@RequestMapping("/index")

**public** String index(){

**return** "index";

}

}

### 伪静态

1. 静态页面

优点:

1. 浏览器加载速度很快
2. **容易被搜索引擎收录**

缺点:用户体验弱.交互差

2.动态页面

优点:

良好的交互

缺点:

1. 加载速度慢
2. 搜索引擎不会收录动态页面

**伪静态:以.html展现的动态页面**

伪静态是相对真实静态来讲的，通常我们为了增强[搜索引擎](https://baike.baidu.com/item/%E6%90%9C%E7%B4%A2%E5%BC%95%E6%93%8E/104812)的友好面，都将文章内容生成静态页面，但是有的朋友为了实时的显示一些信息。或者还想运用动态脚本解决一些问题。不能用静态的方式来展示网站内容。但是这就损失了对搜索引擎的友好面。怎么样在两者之间找个中间方法呢，这就产生了伪静态技术。就是展示出来的是以html一类的静态页面形式，但其实是用ASP一类的动态脚本来处理的。

### 关于域问题

同域:请求协议://请求域名:请求的端口

如果三者都相同则表示同域.

除此之外都是跨域请求.

\

脑裂