# Redis高级(二)

## Spring整合哨兵

### 编辑配置文件

<!--哨兵配置项 -->

<bean id=*"sentinelPool"* class=*"redis.clients.jedis.JedisSentinelPool"*>

<constructor-arg name=*"masterName"* value=*"mymaster"*/>

<constructor-arg name=*"sentinels"*>

<set>

<value>192.168.126.166:26379</value>

</set>

</constructor-arg>

</bean>

### 编辑工具类

//通过哨兵链接Redis

@Autowired(required=**false**)

**private** JedisSentinelPool sentinelPool;

**public** String get(String key){

Jedis jedis = sentinelPool.getResource();

String result = jedis.get(key);

sentinelPool.returnResource(jedis);

**return** result;

}

//为数据设定超时时间

**public** **void** set(String key,String value,**int** seconds){

Jedis jedis = sentinelPool.getResource();

jedis.setex(key, seconds, value);

}

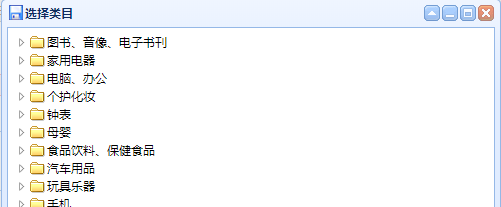
**public** **void** set(String key,String value){

Jedis jedis = sentinelPool.getResource();

jedis.set(key, value);

}

### 测试结果



## Redis集群搭建

### Redis分片和哨兵存在的问题

优点:

1. 分片实现内存数据的动态扩容
2. 分片可以分散的保存数据
3. 哨兵可以实现redis高可用

缺点:

1. 分片中如果有一个节点宕机,则整个分片将不能正常使用
2. 哨兵本身没有实现高可用,如果哨兵宕机,则可能导致整合redis服务宕机.

### Redis集群介绍

说明:redis集群包含了分片的特点和哨兵的高可用的机制.同时内部不需要单独的配置哨兵,而所有的redis集群中的节点,都有选举权.得票数高的,当选主机.

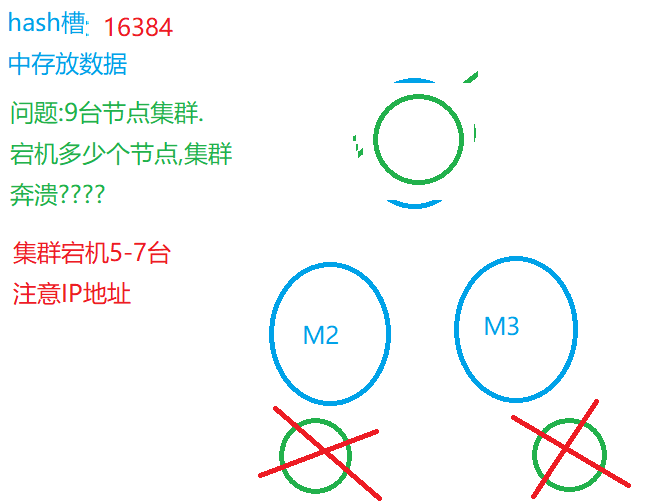
### Redis集群内部通讯原理

搭建原则:一般主节点一般都大于等于3,同时主节点的个数是奇数个.

搭建:

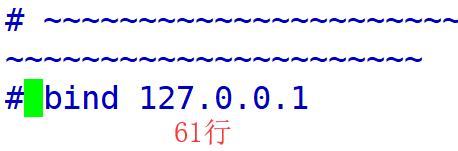
3主6从共9台redis.

内部通信原理:

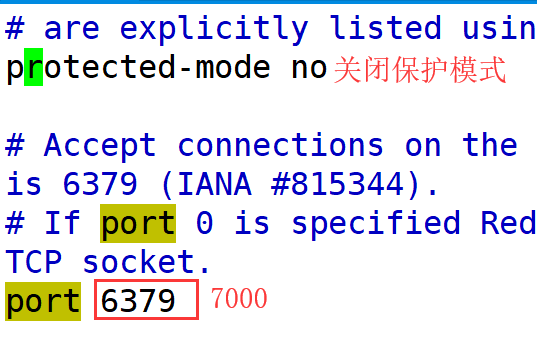


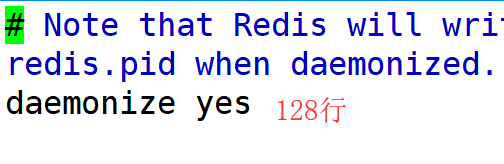
### 编辑redis.conf配置文件

1. 注释IP绑定

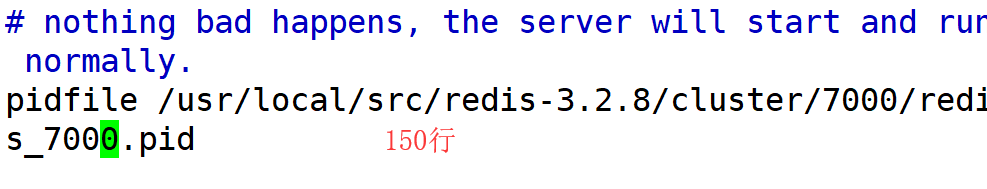


1. 修改保护模式和端口

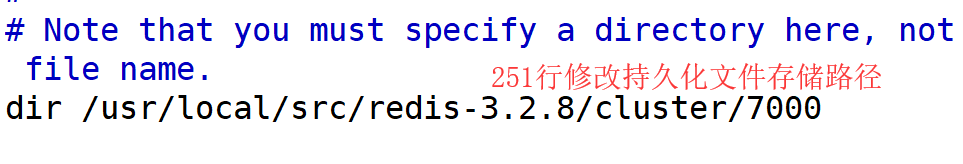
 3.开启后台启动



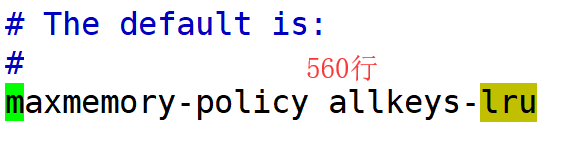
4.修改PID路径



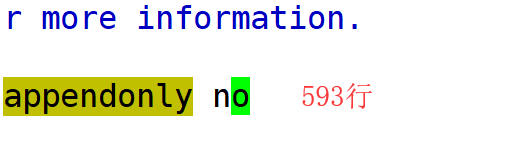
5.修改存储路径



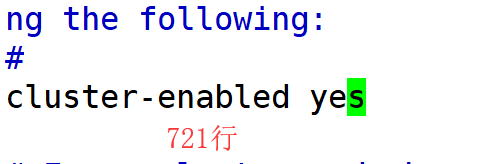
6.修改内存策略



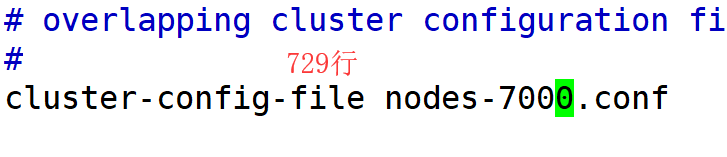
7.关闭AOF模式



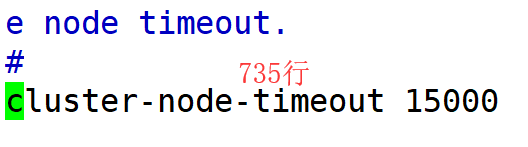
8.开启集群配置



9.开启集群配置



10.设定超时时间



### 复制配置文件

[root@localhost cluster]# cp 7000/redis.conf 7001/

[root@localhost cluster]# cp 7000/redis.conf 7002/

[root@localhost cluster]# cp 7000/redis.conf 7003/

[root@localhost cluster]# cp 7000/redis.conf 7004/

[root@localhost cluster]# cp 7000/redis.conf 7004/

[root@localhost cluster]# cp 7000/redis.conf 7005/

[root@localhost cluster]# cp 7000/redis.conf 7006/

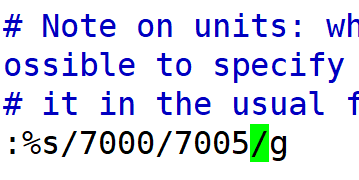
[root@localhost cluster]# cp 7000/redis.conf 7007/

[root@localhost cluster]# cp 7000/redis.conf 7008/

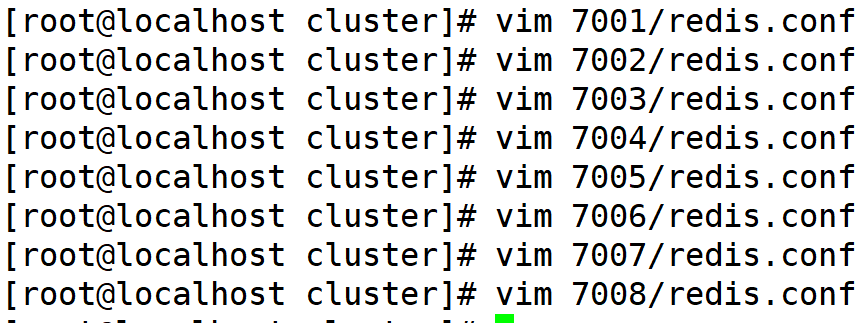
### 批量修改redis配置文件

说明:将redis.conf中7000改为对应的端口号.

修改命令:



分别修改7001-7008端口号



### 批量启动redis

#!/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 &

批量启动:

[root@localhost cluster]# **sh start.sh**

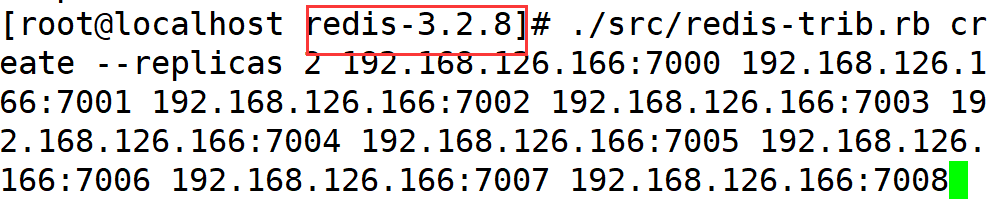
### 创建集群

命令:

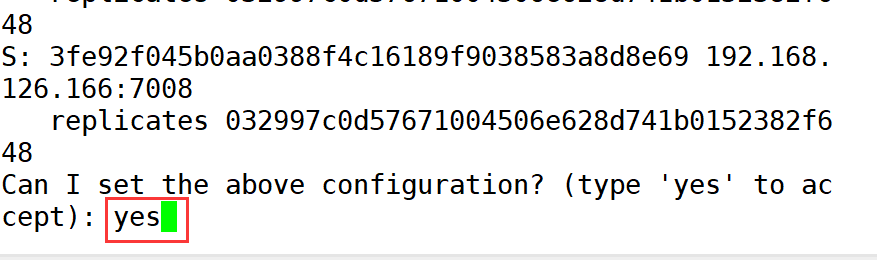
./src/redis-trib.rb create --replicas 2 192.168.126.166:7000 192.168.126.166:7001 192.168.126.166:7002 192.168.126.166:7003 192.168.126.166:7004 192.168.126.166:7005 192.168.126.166:7006 192.168.126.166:7007 192.168.126.166:7008

2:代表一台机器几台从机

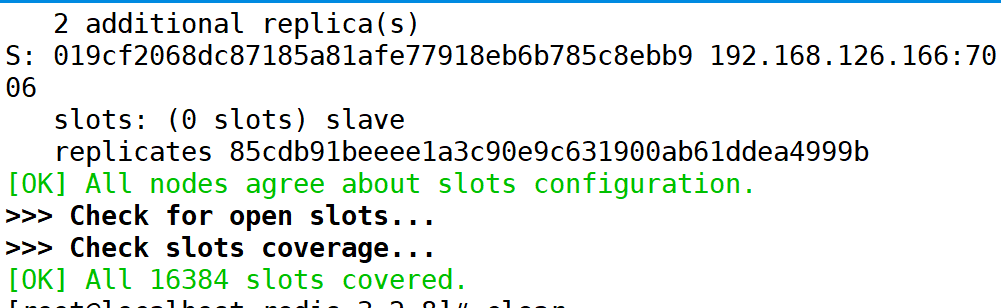
执行要求:



同意策略:输入yes



集群搭建成功!!!



## 工厂模式

### 什么时候使用工厂模式?

核心:工厂模式的核心思想就是实例化对象.

说明:某个写对象可能需要特定的参数,可能是抽象类,也可能是接口等等.不能直接实例化的,需要采用工厂模式.

### 静态工厂

核心:通过类名.静态方法的方式创建对象

工厂代码:

**public** **class** StaticFactory {

//静态工厂必须有静态方法

**public** **static** Calendar getIn(){

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

}

}

1. 工厂配置文件

<!--配置静态工厂 -->

<bean id=*"calendar1"* class=*"com.jt.manage.factory.StaticFactory"* factory-method=*"getIn"*/>

### 实例化工厂

核心:先实例化工厂对象,之后利用工厂对象.工厂方法实例化对象

代码:

**public** **class** InsFactory {

**public** Calendar getIns(){

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

}

}

配置文件:

<!--实例化工厂 -->

<bean id=*"insFactory"* class=*"com.jt.manage.factory.InsFactory"*/>

<bean id=*"calendar2"* factory-bean=*"insFactory"* factory-method=*"getIns"*/>

### Spring工厂

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

//当程序加载该类时,容器内部会自动的调用getObject方法

@Override

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

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

}

@Override

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

// **TODO** Auto-generated method stub

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

}

@Override

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

// **TODO** Auto-generated method stub

**return** **false**;

}

}

2.编辑配置文件

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

## 集群测试入门案例

### 跳过测试类打包

<!--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>

### 入门案例

/\*\*

\* 1.创建集群操作对象

\* 2.将集群操作的节点存入Set集合中

\*/

@Test

**public** **void** testRedisCluster(){

Set<HostAndPort> nodes =

**new** HashSet<>();

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 jedisCluster =

**new** JedisCluster(nodes);

jedisCluster.set("1807","集群测试成功了!!!");

System.***out***.println(jedisCluster.get("1807"));

}

### 如何优化redisAPI

问题:如果需要操作redis节点,则必须通过new关键字实例化很多的HostAndPort对象.如果将来redis节点数量变化,则必须修改原始代码.

能否实现只改配置文件IP:端口,不改代码.

### 编辑pro文件

1.编辑配置文件

#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=192.168.126.166:7000

redis.cluster1=192.168.126.166:7001

redis.cluster2=192.168.126.166:7002

redis.cluster3=192.168.126.166:7003

redis.cluster4=192.168.126.166:7004

redis.cluster5=192.168.126.166:7005

redis.cluster6=192.168.126.166:7006

redis.cluster7=192.168.126.166:7007

redis.cluster8=192.168.126.166:7008

2.spring容器加载配置文件

<!-- <context:property-placeholder location="classpath:/properties/\*.properties"/> -->

<bean class=*"org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"*>

<property name=*"locations"*>

<list>

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

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

</list>

</property>

</bean>

### 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>

<!--通过工厂模式获取数据 -->

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

<!--引入配置文件源文件 -->

<property name=*"propertySource"*>

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

</property>

<!--引入池配置文件 -->

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

<!--添加配置前缀-->

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

</bean>

### 编辑工具类

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

**private** Resource propertySource; //导入配置文件 加载节点信息

**private** JedisPoolConfig poolConfig; //注入池配置文件

**private** String redisNodePrefix; //定义节点前缀

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

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

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

**try** {

properties.load(propertySource.getInputStream());

//获取redis节点信息

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

//判断哪些是redis节点

String strKey = (String) key;

**if**(strKey.startsWith(redisNodePrefix)){

//192.168.126.166:7000

String value = properties.getProperty(strKey);

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

HostAndPort hostAndPort =

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

nodes.add(hostAndPort);

}

}

} **catch** (Exception e) {

e.printStackTrace();

}

**return** nodes;

}

//通过工厂对象 创建jedisCluster对象

@Override

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

Set<HostAndPort> nodes = getNodes();

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

}

@Override

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

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

}

@Override

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

**return** **false**;

}

**public** Resource getPropertySource() {

**return** propertySource;

}

**public** **void** setPropertySource(Resource propertySource) {

**this**.propertySource = propertySource;

}

**public** JedisPoolConfig getPoolConfig() {

**return** poolConfig;

}

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

**this**.poolConfig = poolConfig;

}

**public** String getRedisNodePrefix() {

**return** redisNodePrefix;

}

**public** **void** setRedisNodePrefix(String redisNodePrefix) {

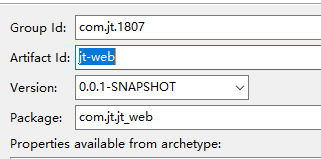
**this**.redisNodePrefix = redisNodePrefix;

}

}

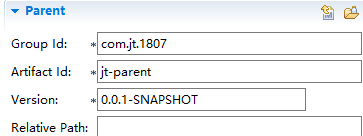
## 京淘前台系统

### 新建项目

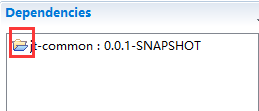


### 添加继承和依赖

1. 添加继承



1. 添加依赖



### 配置tomcat插件

<build>

<plugins>

<!--通过maven启动tomcat插件 -->

<plugin>

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

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

<version>2.2</version>

<configuration>

<port>8092</port>

<path>/</path>

</configuration>

</plugin>

<!--跳过测试类打包 -->

<plugin>

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

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

<configuration>

<skip>true</skip>

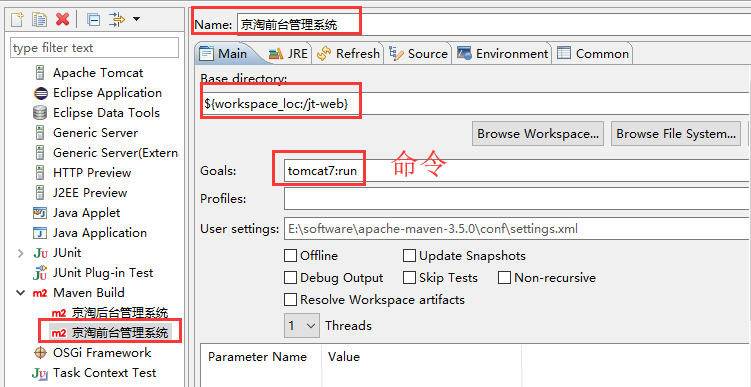
</configuration>

</plugin>

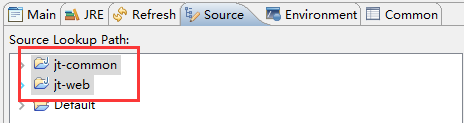
</plugins>

</build>

2.配置tomcat插件



3.导入源码



### 修改nginx配置文件

#配置前台管理服务器

server {

listen 80;

server\_name www.jt.com;

location / {

#发起请求

proxy\_pass http://localhost:8092;

proxy\_connect\_timeout 3;

proxy\_read\_timeout 3;

proxy\_send\_timeout 3;

}

}

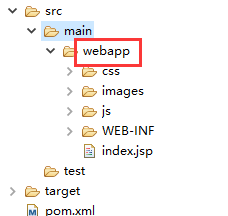
修改完成后,重启nginx

注意配置HOST文件.

## 导入spring配置文件

### 导入静态资源文件

从课前资料中获取静态资源文件



### 编辑web.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

id=*"jt-manage"* version=*"2.5"*>

<display-name>jt-web</display-name>

<!-- <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>

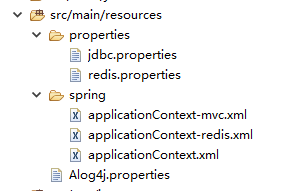
<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

</web-app>

### 导入框架配置文件

因为web不需要直接访问数据库,所以将mybatis操作全部删除.



## 实现京淘前台首页跳转

### 伪静态

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。**

总结:以.html的格式展现动态页面数据

### 实现步骤

访问:http://www.jt.com/index.html时跳转首页

实现:

1. 编辑web.xml添加映射路径

<servlet-mapping>

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

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

</servlet-mapping>

2.编辑前台Controller

@Controller

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

@RequestMapping("/index")

**private** String index(){

**return** "index";

}

}

3.跳转页面

