# Redis集群知识(二)

## 专业名称介绍

### 缓存穿透

条件:访问一个不存在的数据

说明:当访问一个不存在的数据时,因为缓存中没有这个key,导致缓存形同虚设.最终访问后台数据库.但是数据库中没有该数据所以返回null.

隐患:如果有人恶意频繁查询一个不存在的数据,可能会导致数据库负载高导致宕机.

总结:业务系统访问一个不存在的数据,称之为缓存穿透.

### 缓存击穿

条件:当缓存key失效/过期/未命中时,高并发访问该key

说明:如果给一个key设定了失效时间,当key失效时有一万的并发请求访问这个key,这时缓存失效,所有的请求都会访问后台数据库.称之为缓存击穿.

场景:微博热点消息访问量很大,如果该缓存失效则会直接访问后台数据库,导致数据库负载过高.

### 缓存雪崩

前提:高并发访问,缓存命中较低或者失效时

说明:假设缓存都设定了失效时间,在同一时间内缓存大量失效.如果这时用户高并发访问.缓存命中率过低.导致全部的用户访问都会访问后台真实的数据库.

场景:在高并发条件下.缓存动态更新时

### 脑裂

说明:由于推选机制同时选举出多台主机,导致程序不能正常的执行.该问题称之为脑裂.

解决办法:

1. 机器的数量是奇数台
2. 配置的服务器的量一定要多.

## 关于redis集群说明

### 什么时候集群崩溃

说明:如果主节点在宕机时没有从节点则集群崩溃.

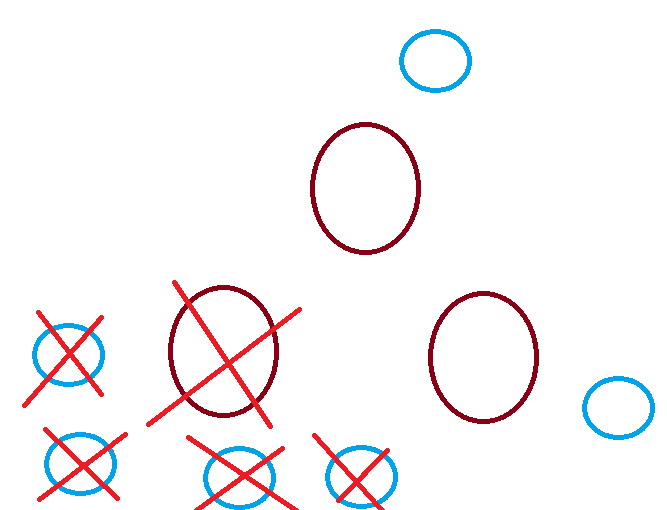
问题:3主6从 宕机几次集群崩溃?????

特点:集群中如果主机宕机,那么从机可以继续提供服务,

当主机中没有从机时,则向其他主机借用多余的从机.继续提供服务.

如果主机宕机时没有从机可用,则集群崩溃.

答案:9台机器,宕机5-7次集群崩溃了.



### Java程序连接异常

问题描述:

如果java程序链接redis时,报错没有**可用的集群节点**时.检测虚拟机的IP地址和配置文件的地址是否匹配.

为什么:因为虚拟机的IP地址发生了变化.

## Spring整合Redis集群

### 入门案例

@Test

**public** **void** test01(){

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("1805", "学习集群使用");

System.***out***.println("获取数据:"+jedisCluster.get("1805"));

}

## Spring中工厂模式

### 工作模式的作用

作用:通过工厂模式实例化对象

原因:

1. 有些对象需要经过业务处理,添加参数才可以使用.
2. 有些类可能是接口或者是抽象类.不能直接实例化.需要工厂模式帮助创建对象.

### 静态工厂

特点:静态工厂中必须有静态方法.

调用逻辑:类名.静态方法

调用过程:

1. 编辑spring配置文件

<!--配置静态工厂 -->

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

1. 编辑工厂模式代码

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

**public** **static** Calendar getInstance(){

//JDK原生提供了实例化方法.

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

}

}

1. 编辑测试类

@Test

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

//1.实例化spring容器

ApplicationContext app =

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

//2.获取对象

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

System.***out***.println("获取当前时间:"+calendar.getTime());

}

### 实例化工厂

特点:首先实例化工厂,之后通过工厂对象调用工厂方法创建对象

调用特点:对象.方法

调用方法:

1. 编辑配置文件

<!--实例化工厂 -->

<bean id=*"instanceFactory"* class=*"com.jt.manage.factory.InstanceFactory"*/>

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

factory-method=*"getInstance"*/>

1. 编辑工厂模式

**public** **class** InstanceFactory {

**public** Calendar getInstance(){

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

}

}

1. 测试类

@Test

**public** **void** testInstanceFacotory(){

//1.实例化spring容器

ApplicationContext app =

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

//2.获取对象

Calendar calendar = (Calendar) app.getBean("calendar2");

System.***out***.println("获取当前时间:"+calendar.getTime());

}

### Spring工厂

特点:实现Spring中特定接口FactoryBean,容器内部自动调用

调用方法:

1. 编辑配置文件

<!--spring工厂 -->

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

1. 编辑工厂模式

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

@Override

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

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

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

}

@Override

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

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

}

//是否单例

@Override

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

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

**return** **false**;

}

}

1. 编辑测试类

@Test

**public** **void** testSpringFacotory(){

//1.实例化spring容器

ApplicationContext app =

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

//2.获取对象

Calendar calendar = (Calendar) app.getBean("calendar3");

System.***out***.println("获取当前时间:"+calendar.getTime());

}

## Spring接口扩展

### 接口扩展

1. BeanNameAware

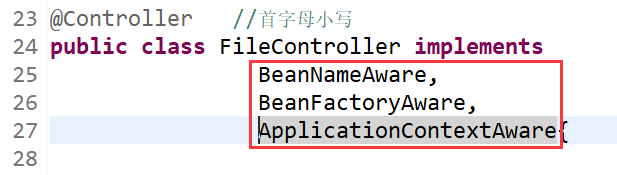
获取beanId名称

1. BeanFactoryAware

获取Spring中bean工厂

1. ApplicationContextAware

获取Spring容器对象



实现类方法:

@Override

**public** **void** setBeanName(String name) {

System.***out***.println("获取当前类在Spring中的ID值:"+name);

}

@Override

**public** **void** setBeanFactory(BeanFactory beanFactory) **throws** BeansException {

System.***out***.println("获取spring工厂对象:"+beanFactory);

}

@Override

**public** **void** setApplicationContext(ApplicationContext applicationContext) **throws** BeansException {

System.***out***.println("获取整个spring容器:"+applicationContext);

}

## Spring整合Redis集群

### 修改配置文件

#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

### 编辑工厂模式

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

**private** JedisPoolConfig poolConfig; //注入配置文件

**private** String redisNodePrefix; //注入配置前缀

**private** Resource propertySource; //注入redis配置文件

@Override

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

Set<HostAndPort> nodes = getNodes();

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

**return** jedisCluster;

}

/\*\*

\* 1.定义Set集合

\* 2.获取配置文件中的IP和端口

\* 3.封装Set集合

\* **@return**

\*/

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

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

**try** {

//获取Property数据

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

properties.load(propertySource.getInputStream());

//获取Redis节点数据

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

String strKey = (String) key;

//判断当前遍历的key是否为redis节点的key

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

//IP:端口号

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;

}

@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** String getRedisNodePrefix() {

**return** redisNodePrefix;

}

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

**this**.redisNodePrefix = redisNodePrefix;

}

**public** Resource getPropertySource() {

**return** propertySource;

}

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

**this**.propertySource = propertySource;

}

}

项目编辑完成后,将项目打包

### 编辑Spring配置文件

<!--配置redis集群 -->

<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.facotry.JedisClusterFactory"*>

<!--引入配置文件源文件 -->

<property name=*"propertySource"*>

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

</property>

<!--引入池配置文件 -->

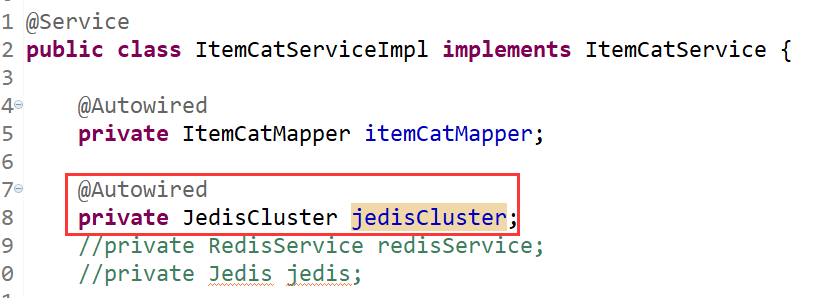
<property name=*"poolConfig"* ref=*"poolConfig"*/>

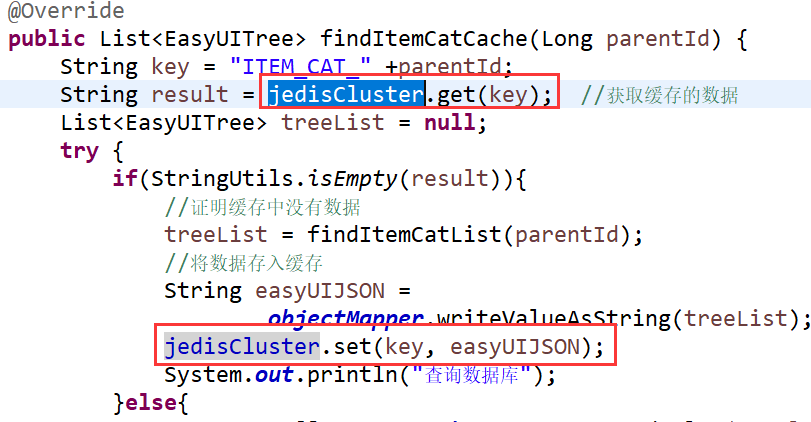
<!--添加配置前缀-->

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

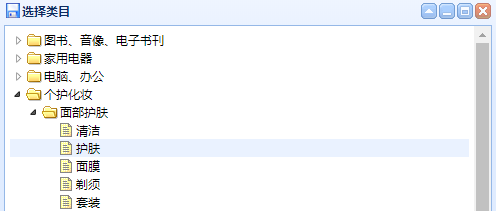
</bean>

### 实现集群操作





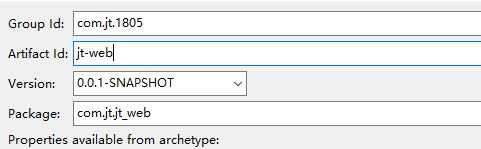
### 页面效果



# 京淘项目前台实现

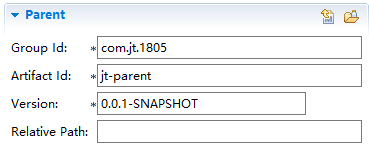
## 构建项目

### 选择骨架

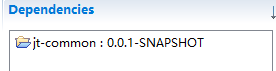


### 添加继承和依赖

1. 添加继承



1. 添加依赖



### 添加tomcat插件

<build>

<plugins>

<plugin>

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

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

<version>2.2</version>

<configuration>

<port>8092</port>

<!--项目的发布路径 -->

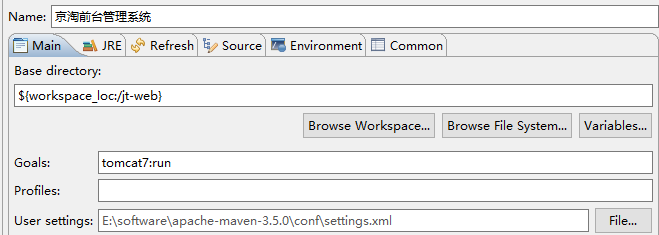
<path>/</path>

</configuration>

</plugin>

</plugins>

</build>



### 实现Nginx反向代理

要求:用户输入[www.jt.com时跳转到服务器8092](http://www.jt.com时跳转到服务器8092)中

修改Nginx配置文件

#前台管理系统

server {

listen 80;

server\_name www.jt.com;

location / {

#实现服务器代理

proxy\_pass http://localhost:8092;

proxy\_connect\_timeout 3;

proxy\_read\_timeout 3;

proxy\_send\_timeout 3;

}

}

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

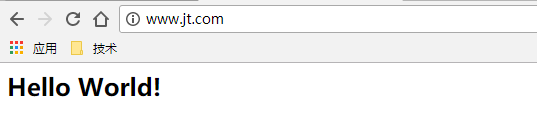
127.0.0.1 order.jt.com

127.0.0.1 solr.jt.com

192.168.126.148 [www.easymall.com](http://www.easymall.com)

说明:修改完成后重启Nginx服务器

效果展现:

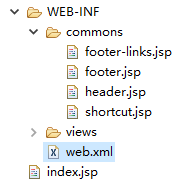


## 添加配置文件

### 配置文件说明

说明:jt-web主要负责与用户进行交互.项目中用到的全部的数据都通过特殊的方式从对应的服务器动态获取.前台只需要配置SpringMVC/Spring/静态资源文件即可.

### 导入静态配置文件



### Web.xml配置文件说明

1. 以监听器的方式启动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>

对于监听器的说明:

有些就的企业现在一直沿用监听器的方式启动spring容器.该方式其实是一种懒加载的方式.当SpringMVC容器需要实例化Controller对象时,并且内部需要注入业务层Service时,该操作会被监听器所拦截.这时启动Spring容器实例化对象.之后将对象注入注入.整个容器启动完成.

1. 前端控制器拦截的策略

<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>

**\*.html 该操作拦截所有以.html结尾的请求**

/service/\* 该配置拦截所有以/Service开头的请求

### 伪静态介绍

静态页面优点:

1. 加载速度快
2. 容易被搜索引擎收录,增加页面的友好性

静态页面缺点:

1. 用户体验感差/交互性差
2. 静态页面数据不安全.

动态页面优点:

1. 用户交互性强
2. 安全性更好AJAX

动态页面缺点:

1. 搜索引擎不会收录动态页面 .jsp .asp

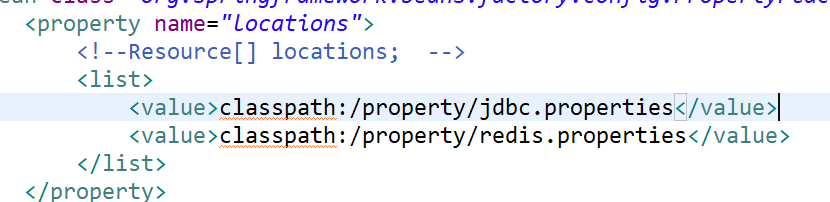
伪静态介绍:

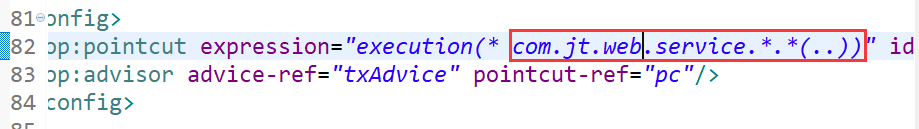
伪静态是相对真实静态来讲的，通常我们为了增强[搜索引擎](https://baike.baidu.com/item/%E6%90%9C%E7%B4%A2%E5%BC%95%E6%93%8E/104812)的友好面，都将文章内容生成静态页面，但是有的朋友为了实时的显示一些信息。或者还想运用动态脚本解决一些问题。不能用静态的方式来展示网站内容。但是这就损失了对搜索引擎的友好面。怎么样在两者之间找个中间方法呢，这就产生了伪静态技术。就是展示出来的是以html一类的静态页面形式，但其实是用ASP一类的动态脚本来处理的。

总结:以html静态页面展现形式的动态页面技术.

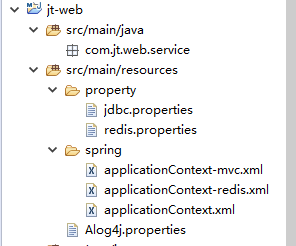
### 修改配置文件

1. 修改Spring配置文件





1. 修改配置文件后效果



### 展现京淘前台首页

1. 编辑Controller

@Controller

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

@RequestMapping("/index")

**public** String index(){

**return** "index";

}

}

1. 页面效果展现

