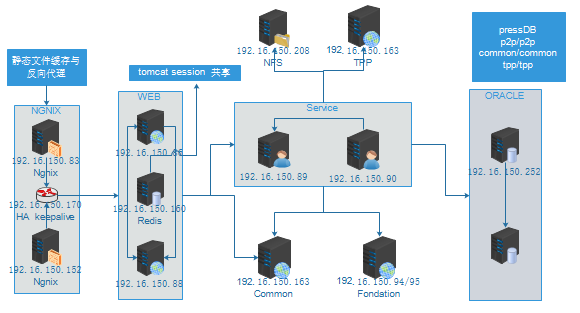
# 善林宝缓存系统

## 善林宝网络拓扑图



## nginx缓存

### 缓存配置

proxy\_cache\_path /var/cache/nginx/proxy\_cache levels=1:2 keys\_zone=slb:500m inactive=1d max\_size=20G;

location = / {

proxy\_cache slb;

proxy\_cache\_valid 200 304 3m;

proxy\_cache\_valid any 5h;

proxy\_cache\_key $host$uri$is\_args$args;

proxy\_set\_header Host $host;

proxy\_set\_header X-Forwarded-For $remote\_addr;

add\_header Nginx-Cache "$upstream\_cache\_status";

proxy\_pass http://http\_web;

}

location ~\* \.(gif|jpg|png|bmp|js|html|htm|css|swf|jpeg)$ {

proxy\_next\_upstream http\_502 http\_504 error timeout invalid\_header;

proxy\_cache slb;

proxy\_cache\_valid 200 304 301 302 12h;

proxy\_cache\_valid any 1d;

proxy\_cache\_key $host$uri$is\_args$args;

proxy\_set\_header Host $host;

proxy\_set\_header X-Forwarded-For $remote\_addr;

expires 1d;

proxy\_pass http://http\_static;

}

location / {

proxy\_set\_header Host $host;

proxy\_set\_header X-Forwarded-For $remote\_addr;

proxy\_pass http://http\_web;

}

### 缓存目录细节

缓存图片：test.shanlinbao.com/images/index/weixin.jpg

缓存路径：/var/cache/nginx/proxy\_cache/f/22/8a1fa2f8517a6624cbbe73ec2caec22f

算法：

1. /var/cache/nginx/proxy\_cache/ 是配置proxy\_cache\_path确定。
2. 8a1fa2f8517a6624cbbe73ec2caec22f

MD5(test.shanlinbao.com/images/index/weixin.jpg)的hash值。

1. /f/22路径 是hash值得后3位，其根据 levels=1:2确定。

### 缓存清除

Purge清除缓存配置

server {

listen 10000;

server\_name test.shanlinbao.com shanlinbao.com;

location ~ /purge(/.\*) {

allow 172.16.0.0/16;

allow 192.16.0.0/16;

deny all;

proxy\_cache\_purge slb $host$1$is\_args$args;

}

}

Test.shanlinbao.com:10000/purge/index.htm

## 前端Web缓存

### Session共享缓存

Web的session共享是基于spring-session项目。通过redis来提供session的共享。

@EnableRedisHttpSession

public class HttpSessionConfig { }

#### Session共享实现细节

通过springSessionRepositoryFilter 来实现http session的共享，实现的细节参考类org.springframework.session.web.http.SessionRepositoryFilter

主要实现细节：

public class SessionRepositoryFilter<S extends ExpiringSession> extends OncePerRequestFilter {

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {

request.setAttribute(SESSION\_REPOSITORY\_ATTR, sessionRepository);

SessionRepositoryRequestWrapper wrappedRequest

= new SessionRepositoryRequestWrapper(request, response, servletContext);

SessionRepositoryResponseWrapper wrappedResponse

= new SessionRepositoryResponseWrapper(wrappedRequest,response);

HttpServletRequest strategyRequest =

httpSessionStrategy.wrapRequest(wrappedRequest, wrappedResponse);

HttpServletResponse strategyResponse =

httpSessionStrategy.wrapResponse(wrappedRequest, wrappedResponse);

try {

filterChain.doFilter(strategyRequest, strategyResponse);

} finally {

wrappedRequest.commitSession();

}

}

}

SessionRepositoryRequestWrapper 这个类包装了Redis Session的操作。

### 本地缓存配置

本地缓存主要是通过ehcache缓存不变的数据或者长期不变的数据。例如数据字典。

#### ehcache配置

**Java Config**

@Configuration

@EnableCaching

public class CachingConfig implements CachingConfigurer{

@Bean

@Override

public CacheManager cacheManager() {

EhCacheCacheManager cacheManager = new EhCacheCacheManager();

cacheManager.setCacheManager(ehCacheManagerFactoryBean().getObject());

return cacheManager;

}

@Bean

public EhCacheManagerFactoryBean ehCacheManagerFactoryBean() {

EhCacheManagerFactoryBean ehCacheManagerFactoryBean =

new EhCacheManagerFactoryBean();

ehCacheManagerFactoryBean.setConfigLocation(new ClassPathResource("ehcache.xml"));

ehCacheManagerFactoryBean.setCacheManagerName("webCache");

ehCacheManagerFactoryBean.setShared(true);

return ehCacheManagerFactoryBean;

}

}

**XML config**

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

xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd"

updateCheck="false" name="webCache">

<diskStore path="java.io.tmpdir/ehcache/web" />

<!-- DefaultCache setting. -->

<defaultCache maxElementsInMemory="1000" eternal="false"

timeToIdleSeconds="3000" timeToLiveSeconds="6000" overflowToDisk="false"

maxElementsOnDisk="100000" />

<!-- Special objects setting. -->

<cache name="param" maxElementsInMemory="1000" eternal="false"

overflowToDisk="false"

maxElementsOnDisk="1500" timeToIdleSeconds="43200" timeToLiveSeconds="43200" />

</ehcache>

#### 缓存使用

@Cacheable(value="param", key="'getInvestStartTime'")

public Map<String,String> getInvestStartTime(){...}

#### 清除缓存

@Controller

@RequestMapping("/localCache")

public class LocalCacheController extends BaseController {

/\*\*

\* 获取所有key

\*

\* @return

\*/

@RequestMapping(value="{cache}/keys",method = RequestMethod.GET)

@ResponseBody

public List<String> getKeys(@PathVariable String cache) {

return CacheUtils.getKes(cache);

}

/\*\*

\* 得到相应的缓存值

\*

\* @return

\*/

@RequestMapping(value = "{cache}/{key}", method = RequestMethod.GET)

@ResponseBody

public Object get(@PathVariable String cache,@PathVariable String key) {

return CacheUtils.get(cache, key);

}

/\*\*

\* 删除相应的缓存值

\*

\* @return

\*/

@RequestMapping(value = "{cache}/{key}", method = RequestMethod.DELETE)

public void delete(@PathVariable String cache,@PathVariable String key) {

CacheUtils.remove(cache, key);;

}

/\*\*

\* 删除某缓存中的的所有缓存值

\*

\* @return

\*/

@RequestMapping(value = "{cache}", method = RequestMethod.DELETE)

public void deleteAll(@PathVariable String cache) {

CacheUtils.removeAll(cache);

}

}

## 后台服务缓存

### 共享缓存配置

共享缓存主要通过Redis缓存一下统计数据，变化不频繁，实时性要求不高的数据，如首页的统计数据。

#### 配置Redis

**Build.gradle**

compile("org.springframework.boot:spring-boot-starter-redis")

**Application.properties**

spring.redis.host=192.16.150.103

spring.redis.password=

spring.redis.port=6379

spring.redis.pool.max-idle=8

spring.redis.pool.min-idle=0

spring.redis.pool.max-active=50

spring.redis.pool.max-wait=-1

**Java Configuration**

**public** **class** CachingConfig **implements** CachingConfigurer{

@Autowired

RedisOperations<Object, Object> redisTemplate;

@Bean(name="ehcache")

@Override

**public** CacheManager cacheManager() {

EhCacheCacheManager cacheManager = **new** EhCacheCacheManager();

cacheManager.setCacheManager(ehCacheManagerFactoryBean().getObject());

**return** cacheManager;

}

@Bean(name="redis")

**public** CacheManager redisCacheManger() {

**return** **new** RedisCacheManager((RedisTemplate<Object, Object>)redisTemplate);

}

}

#### Banner

Banner主要缓存单个banner的信息及不同source的banner列表。

**查询缓存**

@Cacheable(value="appbanner",key="#params['appSource']", cacheManager="redis")

**public** ResultVo queryAppBanner(Map<String, Object> params)

**throws** SLException { ... }

@Cacheable(value="banner",key="#params['bannerId']", cacheManager="redis")

**public** Map<String, Object> queryBannerById(Map<String, Object> params) { ... }

**缓存失效**

@Caching(evict = { @CacheEvict(value = "appguid", allEntries = **true**, cacheManager="redis"),

@CacheEvict(value = "appstart", allEntries = **true**, cacheManager="redis"),

@CacheEvict(value = "appbanner", allEntries = **true**, cacheManager="redis"),

@CacheEvict(value = "banner", key= "#result.result['data']", cacheManager="redis") })

**public** ResultVo saveBanner(Map<String, Object> params) **throws** SLException { ... }

#### 公告

#### 动态

#### 客户信息

* 查找客户信息时缓存，更新时候失效缓存。更新时不要缓存跟数据库事务有关。

@Cacheable(value="customer", key="'mobile\_' + #mobile", unless = "#result == null", cacheManager="redis")

**public** CustInfoEntity findCustByMobile(String mobile) {

CustInfoEntity cie = custInfoRepository.findByMobile(mobile);

**return** cie;

}

@Caching(evict = {

@CacheEvict(value = "customer", key= "#result.id", cacheManager="redis"),

@CacheEvict(value = "customer", key= "'mobile\_' + #result.mobile", cacheManager="redis") })

**public** CustInfoEntity saveCust(**final** CustInfoEntity cie) {

CustInfoEntity cust = custInfoRepository.save(cie);

**return** cust;

}

//@Cacheable(value="customer", key="'custId\_' + #custId", unless = "#result == null", cacheManager="redis")

**public** CustInfoEntity findByCustId(String custId) {

CustInfoEntity cie = custInfoRepository.findOne(custId);

**return** cie;

}

* 缓存在Customer Service里维护，利用BeanSelfAware注入自身的代理类来保证查询结果保存到缓存。

**public** ResultVo loginMobile(Map<String, Object> param) {

String mobile = (String) param.get("mobile");

String password = (String) param.get("loginPassword");

CustInfoEntity cie = self.findCustByMobile(mobile);

**if** (**null** == cie)

**return** **new** ResultVo(**false**, "手机号或密码错误!");

**else** **if** (!password.equals(cie.getLoginPassword()))

**return** **new** ResultVo(**false**, "手机号或密码错误!");

**return** **new** ResultVo(**true**, "登录成功", cie);

}

**public** ResultVo setLoginPassword(Map<String, Object> params) {

String mobile = params.get("mobile") + "";

String passwd = params.get("loginPasswd") + "";

CustInfoEntity cie = self.findCustByMobile(mobile);

cie.setLoginPassword(Hashing.*md5*().hashString(passwd, Charsets.***UTF\_8***)

.toString());

self.saveCust(cie);

。。。。。。

}

* 由于客户信息在系统各处使用，因此对开发人员提出了一个要求，他在使用客户信息时，不能使用Customer repository来直接访问数据，而是要通过Customer Service的接口来访问数据。
* 确保更新客户时调用Customer Service的SaveCust的方法来失效缓存，不要依赖JPA的更新机制。
* 如果你使用JDBCTemplate直接操作客户信息，请谨慎考虑缓存问题，缓存是否需要失效。
* 需要考虑redis的内存问题，如果所有的客户信息保存在redis里。

#### 清除redis缓存

Redis管理工具 redis-desktop-manager-0.8.3.3850

#### Spring cache部分细节

RedisCache

CacheInterceptor

## Redis3

### 安装

下载代码： <http://download.redis.io/releases/redis-3.0.7.tar.gz>

yum install gcc //c语言编译器

make MALLOC=libc

yum install tcl //测试依赖

Make test

yum install ruby

yum install rubygems

Linux注意事项

* Add vm.overcommit\_memory = 1 to /etc/sysctl.conf and then reboot or run the command sysctl vm.overcommit\_memory=1 for this to take effect immediately.
* disable Linux kernel feature transparent huge pages, using the following command: echo never > /sys/kernel/mm/transparent\_hugepage/enabled
* Set up swap memory as much as memory
* Set maxmemory in configuration file
* Redis may use up to 2 times the memory normally used when persistence .
* If you are using replication, make sure that either your master has persistence enabled, or that it does not automatically restarts on crashes

### 高可用性

Sentinel(哨兵)

### 集群

特性Feature

节点自动发现

原理

HASH\_SLOT = CRC16(key) mod 16384

全网状拓扑结构，所有Redis节点之间直接TCP长连接，彼此通过PING-PONG机制互联，通讯协议是Cluster bus protocol

集群总线端口与客户端口(6379, 16379)

重定向方式(Redirection)，每个SLOT绑定一个节点，当客户查询的key的SLOT不属于自己就重定向到SLOT属于的节点。

GET x //查询key x

-MOVED 3999 127.0.0.1:6381 //不属于自己，定向到其他节点 3999是SLOT的值

集群的拓扑结构改变（增加/删除节点）的导致SLOT的再分配(resharding )。

配置

简单配置例子， 集群至少需要3个主节点， 测试最好3主3备

port 7000

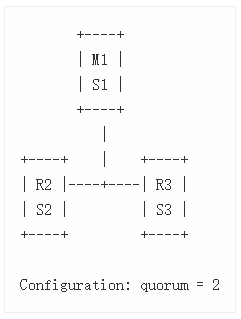
cluster-enabled yes //enable集群

cluster-config-file nodes.conf //节点的配置文件，不能手工配置，集群自动更新

cluster-node-timeout 5000

appendonly yes

### 测试高可用性



mkdir sentinel-test

cd sentinel-test

mkdir 6379 6380 6381 5000 5001 5002

cp /path/to/redis/src/redis-server .

Master 配置：redis.conf

Slave 配置： redis.conf

port 6380

slaveof 127.0.0.1 6379

Sentinel 配置：

port 5000

sentinel monitor mymaster 127.0.0.1 6379 2

sentinel down-after-milliseconds mymaster 5000

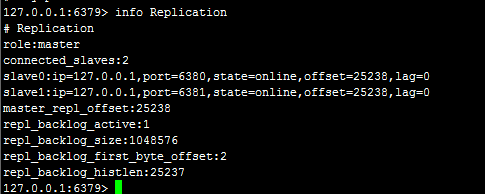
sentinel failover-timeout mymaster 60000

sentinel parallel-syncs mymaster 1

查看master/slave状况

Redis-cli -p 6379

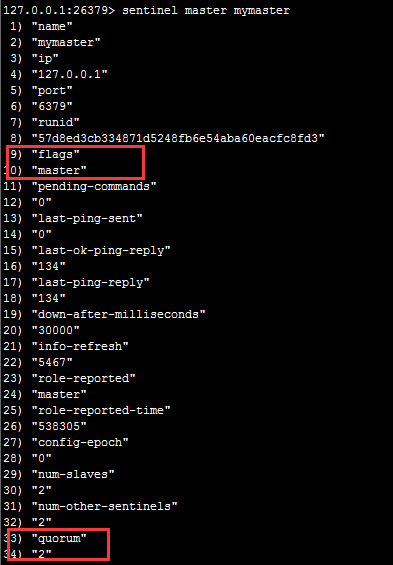
Info Replication



查看sentienal状况

Redis-cli -p 26379

sentinel master mymaster



SNTINEL slaves mymaster

SENTINEL sentinels mymaster

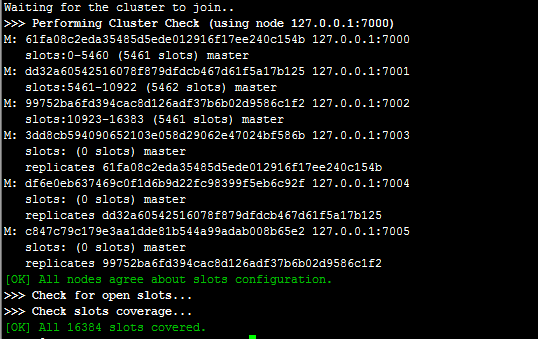
redis-cli -p 6379 DEBUG sleep 30

### 测试集群

1. mkdir cluster-test
2. cd cluster-test
3. mkdir 7000 7001 7002 7003 7004 7005
4. cp /path/to/redis/src/redis-server .
5. cd 7000
6. touch redis.config //内容如上
7. ../redis-server ./redis.config
8. 重复 5-7， 配置的端口改成目录对应的端口
9. gem install redis

./redis-trib.rb create --replicas 1 127.0.0.1:7000 127.0.0.1:7001 \

127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005



### 分布式锁（Distributed Lock Manager）