# 资料、文章备留：

## PM

《如果你是新晋的leader， 你可能需要了解这些》

https://mp.weixin.qq.com/s/xdpmB8ne8NKQluJBNMJVRg

《初次走上技术管理岗位的思考总结》

https://mp.weixin.qq.com/s/WFiYcc0tlMzAdKTWn7f60A

《一个小公司老板的日常管理总结》

https://mp.weixin.qq.com/s/NaoVRgRzOwB0XTuLE4DX4Q

## Redis

《硬核！15张图解Redis为什么这么快 》

https://mp.weixin.qq.com/s/abjEHiMjPC8dnXweN1da-A

源码C语言、纯内存访问、单线程、IO多路复用

《四个大点，搞懂 Redis 到底快在哪里？》

https://mp.weixin.qq.com/s/4kPlBE3C6lTuSvt5mY5hUQ

（I/O复用模型，epoll）：将用户socket对应的fd注册进epoll（实际上服务器和操作系统之间传递的不是socket的fd而是fd\_set的数据结构），然后epoll只告诉哪些需要读/写的socket，只需要处理那些活跃的、有变化的socket fd的就好了。这样，整个过程只在调用epoll的时候才会阻塞，收发客户消息是不会阻塞的。

《为什么 Redis 单线程能支撑高并发？》

https://mp.weixin.qq.com/s/wINh2\_FA6B-B9jrG5V7WnA

《Redis 性能问题分析》

https://www.jianshu.com/p/67093716547b

减少网络延时，减少耗时操作，优化数据结构，OS，硬件，持久化方面，分布式，读写分离

《性能测试 —— Redis 基准测试 》

https://mp.weixin.qq.com/s/wt-qSslcka7KauiweXWwDQ

基于redis-benchmark的性能压力测试。

《Java 中 Redis 五大基本类型的用法》Java 基基

https://mp.weixin.qq.com/s/ZfE5yI7uOlquFzHqyU6Q-w

《Redis 如何保持和 MySQL 数据一致 》

https://mp.weixin.qq.com/s/CCbS61JRPbYaIXXIAamCvg

缓存先删，后查DB库，成功后再写回redis。

高并发情况下，可以将请求放在队列里，已有相同操作的情况下，轮询200ms等待读缓存。

《80% 人不知道的 Redis 分布式锁的正确实现方式（Java 版）》

https://mp.weixin.qq.com/s/7ze2v9HQH07rvYoNpUTmzw

保障锁的设置和过期具有原子性。

操作完需要手动解锁，保证自己的锁自己解，可以将锁的值设置为请求的唯一索引。

可以使用lua脚本，jedis.eval()。

if redis.call('get', KEYS[1]) == ARGV[1] then

return redis.call('del', KEYS[1])

else

return 0

end

《业务性能优化之redis批处理利器 》

https://mp.weixin.qq.com/s/BJ\_TfFTabxKztGfT69mCXA

远程调用redis的get/set会影响性能：

1，改用 pipeline，使用messagepack序列化/反序列化业务实体，以增加网络传输效率。

2，使用mq解耦

《Redis浅入浅出 -- 敖丙在蘑菇街的技术分享 》

https://mp.weixin.qq.com/s/xS2dqi5IG10AF7cHbzguAA

偏实操层面，有具体使用场景。可以参考书籍《Redis深度历险》

《面试被问 Redis 锁，我哭了 qaq 。。。》

https://mp.weixin.qq.com/s/EDR3ooUMQNF-g716Qzl8aA

讲解 Redisson的用法。更擅长分布式、集群场景下的加锁和解锁。

## Zookeeper

《架构师必备：Zookeeper集群“脑裂”问题处理大全》

https://mp.weixin.qq.com/s/rtHUZyKCzvJhIEEb1X1LLA

脑裂的情况，多个分布式节点之间协调选取出Leader节点，正常就1个，但是异常情况下，由于各种原因，可能出现多个leader。

要解决Split-Brain脑裂的问题，一般有下面几种种方法：

1，Quorums (法定人数) 方式: 比如3个节点的集群，Quorums = 2, 也就是说集群可以容忍1个节点失效，这时候还能选举出1个lead，集群还可用。比如4个节点的集群，它的Quorums = 3，Quorums要超过3，相当于集群的容忍度还是1，如果2个节点失效，那么整个集群还是无效的。这是Zookeeper防止"脑裂"默认采用的方法；

2， Redundant communications (冗余通信)方式：集群中采用多种通信方式，防止一种通信方式失效导致集群中的节点无法通信。

3，Fencing (共享资源) 方式：比如能看到共享资源就表示在集群中，能够获得共享资源的锁的就是Leader，看不到共享资源的，就不在集群中。

4，仲裁机制方式；

5，启动磁盘锁定方式。

## Resource-master

01JavaStep01

第一阶段资料

第二阶段资料

02JavaStep02

01java进阶13天资料

02资料-爪哇JDK新特性详解jdk8-12，14

03Ja-va快速入门教程资料

04Ja-va基础教程-小白的福音资料

03JavaWeb

01linux+redis实战教程资料

02玩转MySQL-资料

03深入Sharding-JDBC分库分表

04Web前端入门教程资料

05Linux入门到精通

06Maven精品教程资料

07超系统的Ja-vaWeb快速入门

04StructrueStep01

01Ja-vaWeb网上商城资料

02资料-2018spring框架

03黑马程序员\_Springmvc视频教程-资料

04Mybatis资料

05hibernate2017版资料

06struts2资料

07资料-zookeeper dubbo

08资料-3天全面深入学习zookeeper

05ProjectStep01

01ssh框架综合项目开发资料

02MUI快速实现自己的“微信App”资料

03资料-MySQL高级教程

04管家婆项目资料

05使用redis消息队列完成秒杀过期订单处理资料

06资料-Spring高级之注解驱动开发详解

07基于struts2拦截器实现数据和视图的分离资料

06StructrueStep02

01SpringBoot

02资料-SpringCloud 微服务架构

03资料-springcloud alibaba

04资料-SpringSecurity

05资料-深入解析docker容器化技术

06资料-RabbitMQ深入浅出-分布式系统开发必会的技术

07资料-Spring Security Oauth2.0认证授权专题-热门技术框架

08资料-Elastic Stack 从入门到实践

09资料-mongodb基础到进阶

10黑马商城资料（ssm）

07ProjectStep02

01资料-数据层全栈方案 SpringData 高级应用

02高可用的并发解决方案nginx+keepalived资料

03Spring Boot+Vue.js+FastDFS实现分布式图片服务器资料

04资料-微服务架构的分布式事务控制解决方案

05分布式架构单点登录+授权认证实战 CAS+SpringSecurity资料

06资料-全面解剖RocketMQ和项目实战

07SSO单点登录技术-CAS统一身份认证服务资料

08基于oracle数据库的PLSQL编程以及存储过程的创建和使用资料

09资料-深入解读大厂ja-va面试必考基本功-HashMap集合

10资料-Kafka深入探秘者来了

08Advanced

01资料-Redis入门到精通

02资料-解密JVM

03资料-2020版数据结构与算法

04资料-Tomcat核心原理解析

05资料-数据结构与算法

06资料-全面学习Mybatis插件之Mybatis-Plus

07Mybatis注解开发及注解配置复杂关系映射资料

08基于MongoDB实现商品管理系统资料

09ELasticJob分布式任务调度资料

10资料-Jenkins教程

11利用jQuery+Ajax+HighCharts打造项目图表资料

12资料-并发编程

13资料-Spring Cloud Alibaba Nacos

14oracle数据库单行和多行函数的使用资料

15资料-分布式事务

16资料-分布式链路追踪skywalking

17资料-Java日志

18资料-Apollo分布式配置中心-服务中间件

19资料-Elastic-Job分布式任务调度-服务中间件

20资料-oracle轻松入门

21资料-synchronized

22精品详解Quartz资料

23资料-视频-入门到精通ActiveMQ+企业面试题讲解

24资料-并发编程volatile精讲

26资料-ELK高级搜索

27资料-由浅入深，全面解析ThreadLocal

28资料-jvm垃圾回收机制

29资料-全面深入解析ArrayList原理

30资料-虚拟机性能分析和故障解决工具

31资料-百度地图

32资料-详细分析LinkedList数据链表的实现原理

33资料-jvm内存结构

35资料-深入解读大厂ja-va面试必考基本功-HashMap集合

# Redis

## 【安装】

apt install redis-server

【gcc源码编译安装】

wget http://download.redis.io/releases/redis-5.0.8.tar.gz

curl --output redis.tar.gz http://download.redis.io/releases/redis-5.0.8.tar.gz

mv redis-5.0.8 /usr/redis

make

make install

https://www.jb51.net/article/136019.htm

## 【配置】

/etc/redis/redis.conf

更改

bind 127.0.0.1

protected-mode yes

更改为

# bind 127.0.0.1

protected-mode no

设置密码

requirepass dddd123 指定密码dddd123

设置防火墙的6379端口可以进行访问

### 【Redis 主从模式】

从属于某个 master，实时数据同步

1，从属命令

redis> SLAVEOF 127.0.0.1 6379

2，断开从属命令，保存同步的数据，可以作为 master 的备份体

redis> SLAVEOF NO ONE

类似功能的命令

从 5.0.0. 以后的版本

REPLICAOF hostname port

REPLICAOF NO ONE

配置文件里，可以改：

replicaof 192.168.118.131 6379

### 【Redis 哨兵模式】

安装哨兵

apt install redis-sentinel

修改sentinel配置文件

vim /usr/local/redis/6379/26379.conf

# 主要修改这一项：sentinel monitor；最后的数字表示：主掉线后，从变主的最小投票数

sentinel monitor macrog-master 192.168.24.131 6379 2

# 添加守护进程模式

daemonize yes

# 添加指明日志文件名

logfile "/usr/local/redis/6379/sentinel26379.log"

# 修改工作目录

dir "/usr/local/redis/6379"

# 修改启动端口

port 26379

# 添加关闭保护模式

protected-mode no

启动哨兵

redis-sentinel [配置文件]

哨兵工作原理，哨兵的工作流程主要分为以下几个阶段：

状态感知

心跳检测

选举哨兵领导者

选择新的master

故障恢复

客户端感知新master

## 【操作命令】

Redis 命令参考网站 http://doc.redisfans.com/

--启动 Redis

./redis-server [redis.conf]

--启动 Windows客户端连接工具

redis-cli.exe -h 127.0.0.1 -p 6379

-- Linux 下连接

redis-cli

redis-cli -h 127.0.0.1

【重启】

systemctl restart redis-server

--密码认证登录

auth "admin123456"

auth "ffpredis123" //在线版

--远程连接

telnet 172.21.126.164 6379

telnet 172.21.138.36 6379

$redis-cli -h 172.21.126.164 -p 6379 -a "ffpredis123"

【get属性】

config get \* // 获取全部属性

config get paramName // 获取单个属性

config get dir //输出 redis 安装目录

【set属性】

config set protected-mode no --设置不需要安全模式

config set requirepass admin123456 --设置属性"认证密码"

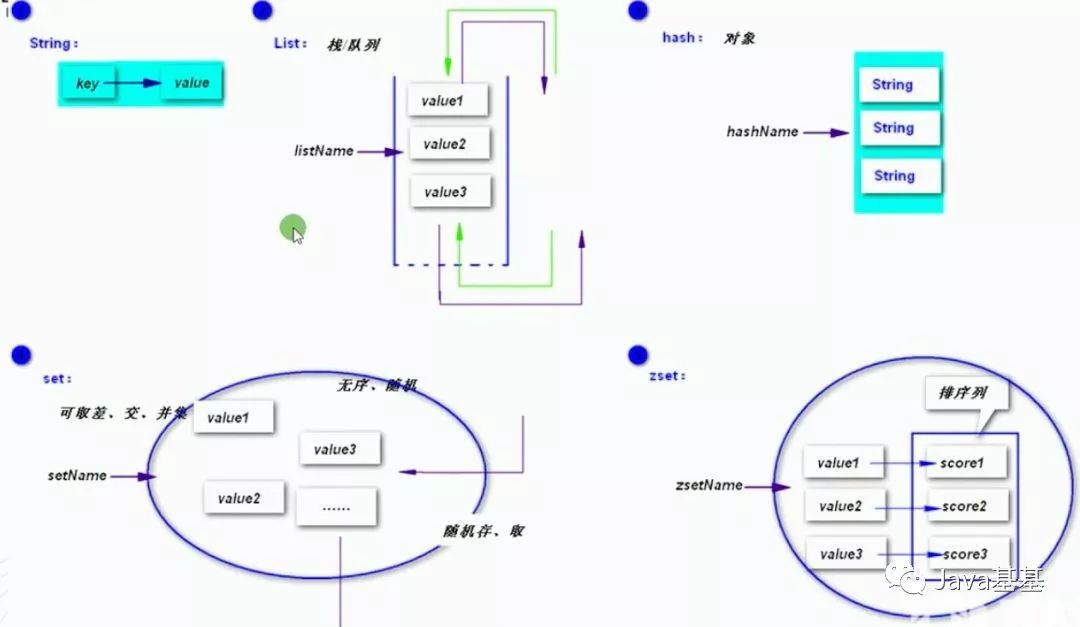
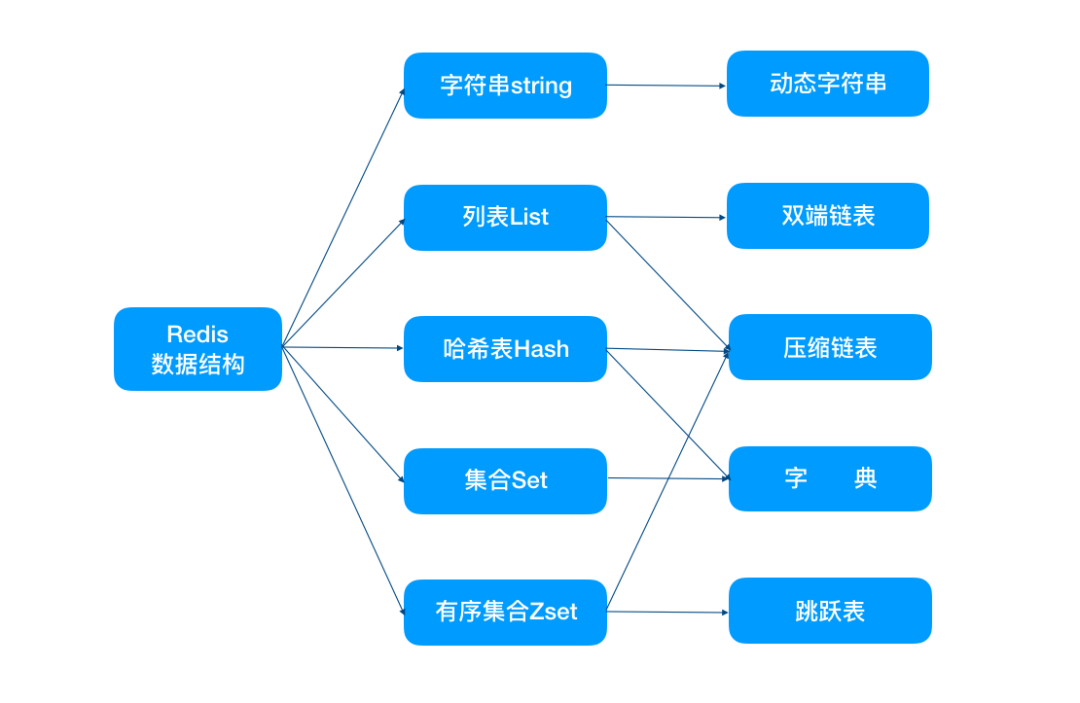
--测试是否正常

ping

-- 监控redis处理的每一条命令

monitor

## 【数据结构及操作】



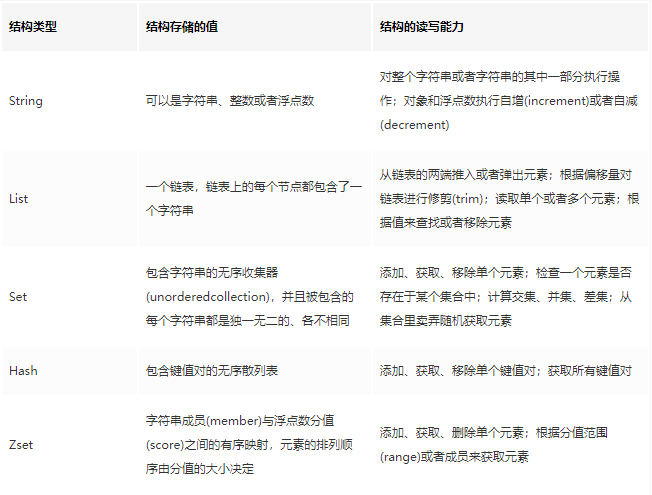
备记：hash类型，操作的是map对象，

//存值 参数第一个变量的名称， map键名(key)， map键值(value)

Long num = jedis.hset("hash1", "username", "caopengfei");

String res = jedis.hmset("hash2", map); // 存取多个key-value

jedis.hgetAll("hash2");



-- 获取全部key

keys \*

-- 设置值

set key value

setNx key value -- set if not exist

-- 不存在值即设置，返回 1；存在不做操作，返回 0

-- 获取值

get key

-- 设置过期时间

expires key 秒数

-- 获取过期剩余时间

TTL key

-- 查看redis服务

ps -ef |grep redis

【查看当前 redis 角色等信息】

info replication

【sortedset】有序集合

127.0.0.1:6379> ZADD b 1 zookeeper

127.0.0.1:6379> zadd b 1 redis

127.0.0.1:6379> zincrby b 2 "zookeeper"

127.0.0.1:6379> add b 2 mongodb

127.0.0.1:6379> zrevrange b 0 1 withscores // 降序前2

1) "zookeeper"

2) "3"

3) "mongodb"

4) "2"

127.0.0.1:6379> zrange b 0 1 withscores // 升序前2

1) "redis"

2) "1"

3) "mongodb"

4) "2"

127.0.0.1:6379> ZRANGE b 0 -1 withscores // 全部

1) "redis"

2) "1"

3) "mongodb"

4) "2"

5) "zookeeper"

6) "3"

### 【持久化】

save

该命令将在 redis 安装目录中创建dump.rdb文件，将 cache 数据保存到文件中

【恢复数据】

如果需要恢复数据，只需将备份文件 (dump.rdb) 移动到 redis 安装目录并启动服务即可。

bgsave

在后台新的子进程进行备份，可以使用lastsave查看最新的进度。

Redis有RDB和AOF的持久化机制。

RDB：基于快照，将某一时刻的所有数据保存到一个RDB文件中，默认方式。

AOF(append-only-file)，当Redis服务器执行写命令的时候，将执行的写命令保存到AOF文件中，默认关闭。

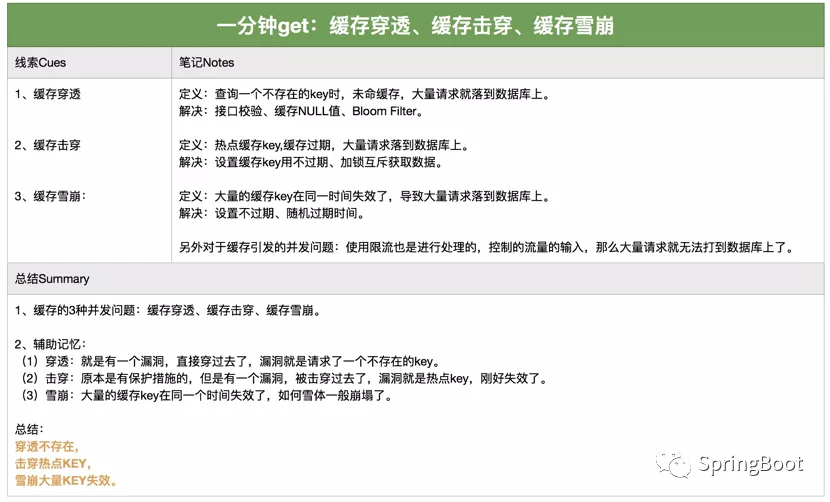
AOF持久化功能的实现可以分为3个步骤：

命令追加：命令写入aof\_buf缓冲区

文件写入：调用flushAppendOnlyFile函数，考虑是否要将aof\_buf缓冲区写入AOF文件中

文件同步：考虑是否将内存缓冲区的数据真正写入到硬盘

## 【实践】



# Zookeeper

ZooKeeper “动物管理员”

ZK是一个树形的目录服务

作用：协调分布式的系统

做 “配置中心”，Eureka 与 ZK 功能类似

## 【安装】

apt install zookeeper

## 【配置】

Connection() 链接

//server：172.21.126.163:2181,172.21.126.164:2181,172.21.126.165:2181

this.zk = new ZooKeeper(server, 6000, this);// 初始化 zk

[SYSTEM]

THREADS=100

COMMANDTIMEOUT=60

[JMS]

#AQ\_URL=tcp://172.21.126.164:61616

AQ\_URL=failover:(tcp://172.21.126.163:61617,tcp://172.21.126.164:61617,tcp://172.21.126.165:61617)?Randomize=false

[REDIS]

#IP=172.21.126.164

IP=172.21.126.165:26380,172.21.126.165:26381

PASSWORD=ffpredis123

【ZK节点类型】

1.持久节点(persistent)

持久节点，创建后一直存在，直到主动删除此节点。

2.持久顺序节点(persistent\_sequential)

持久顺序节点，创建后一直存在，直到主动删除此节点。

在ZK中，每个父节点会为它的第一级子节点维护一份时序，记录每个子节点创建的先后顺序。

3.临时节点(ephemeral)

临时节点在客户端会话失效后节点自动清除。临时节点下面不能创建子节点。

4.顺序临时节点(ephemeral\_sequential)

临时节点在客户端会话失效后节点自动清除。临时节点下面不能创建子节点。

父节点 getChildren 会获得顺序的节点列表。

## 【操作命令】

Create() 创建节点

String zkPath = "/ffp/commands/" + SystemConfig.getAppId();// /ffp/commands/FFPSERVER\_ZHOUMIN\_1

this.zk.create(zkPath, json.getBytes(), Ids.OPEN\_ACL\_UNSAFE, CreateMode.EPHEMERAL);

GetChildren()获取子节点

this.zk.getChildren(name, watch);// name:/ffp/commands，获取子节点

[FFPSERVER\_SYSTEM\_1, FFPSMSEMAIL\_SYSTEM\_1, FFPSERVER\_XUPING\_1]

删除节点

//if (zk.exists("/mcp/commands/"+SystemConfig.getAppId(), true)==) {

zk.delete("/mcp/commands/" + SystemConfig.getAppId(), 0);// 创建节点前不能存在同名节点？

//}

GetData()获取数据

this.zk.getData(name, false, null);// name:/ffp/config/FFPWEB，获取节点（的配置信息？）

监控数据变化

@Override

public void process(WatchedEvent event)

根据 WatchedEvent 的不同枚举类型，进行相应的操作

*// ZkClient client = new ZkClient("192.xxx:2181");*

client.subscribeChildChanges(path, new IZkChildListener() {  
 @Override  
 public void handleChildChange(String s, List<String> list) throws Exception {  
 System.*out*.println("service change:" + s);  
 list.forEach(s1 -> System.*out*.println(s1));  
  
 serviceList = list;  
 }  
});  
client.subscribeDataChanges(path, new IZkDataListener() {  
 @Override  
 public void handleDataChange(String s, Object o) throws Exception {  
 System.*out*.println("data change:" + s + " → " + o);  
 }  
  
 @Override  
 public void handleDataDeleted(String s) throws Exception {  
 System.*out*.println("node deleted:" + s);  
 }  
});

## 【zkcli脚本命令介绍】

zkcli 连接默认zookeeper服务器

zkcli -server ip:port 连接指定的zookeeper服务器

create -s -e path data [acl] 创建节点，-s表示顺序，-e表示临时，默认是持久节点，acl缺省表示不做任何权限限制

ls path [watch] 显示path下的节点，不递归显示，watch注册监听，命令行可忽视

ls2 path 显示当前节点下的节点和当前节点的属性信息

get path [watch] 获取path的属性信息和数据内容

set path data [version] 更新path的数据内容，version是做类似cas的功能的对应dataversion，命令行可忽略

delete path [version] 删除节点，不能递归删除，只能删除叶子节点

setacl path acl 设置节点acl，例子(scheme:id:password=:perm)-(digest:example:sha-1(base64(pwd))=:cdrwa) create delete read write admin

getacl path 获取path节点的acl

stat path 查看path的属性信息

quit 退出zkcli

# Oshi

oshi

https://github.com/oshi/oshi

JNA - java native access

https://github.com/java-native-access/jna

https://www.cnblogs.com/songxingzhu/p/9107878.html

测试案例：

public class test {  
 public static void main(String[] args) {  
 SystemInfo systemInfo = new SystemInfo();  
 HardwareAbstractionLayer hardware = systemInfo.getHardware();  
 GlobalMemory memory = hardware.getMemory();  
 for (PhysicalMemory m : memory.getPhysicalMemory()) {  
 System.*out*.println(m.getManufacturer() + " " + m.getCapacity());  
 }  
 }  
}

## SystemInfo 总类（门面？）

包含：

PlatformEnum

1，OperatingSystem

不同的OS

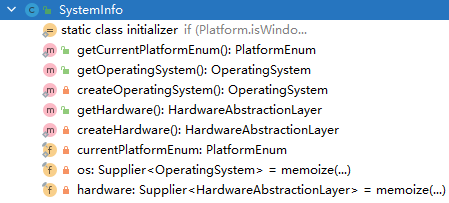
2，HardwareAbstractionLayar

基于不同OS的硬件抽象

Eg. WindowsHardwareAbstractionLayer

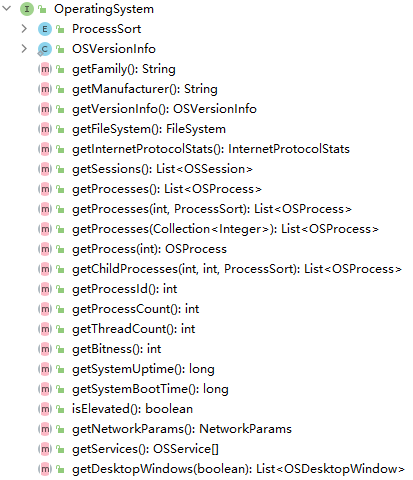
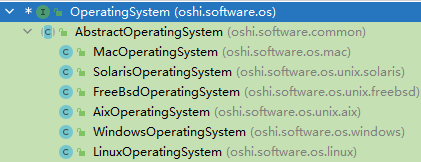
具体不同的硬件

Eg.Processor(CPU), Memory



## 重要类的层级关系

### OS 操作系统



★下面以 WindowsOperatingSystem为例，逐项进行分析

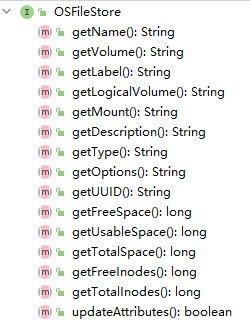
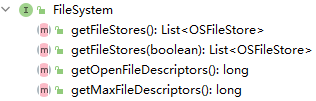
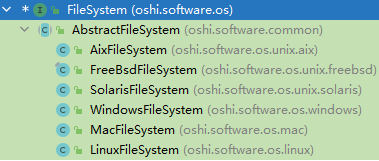
Arrays.*asList*(OperatingSystem.class.getDeclaredMethods()).forEach((method -> {  
 System.*out*.println(method.getName().replaceFirst("get", ""));  
}));

#### Family

#### Manufacturer

#### VersionInfo

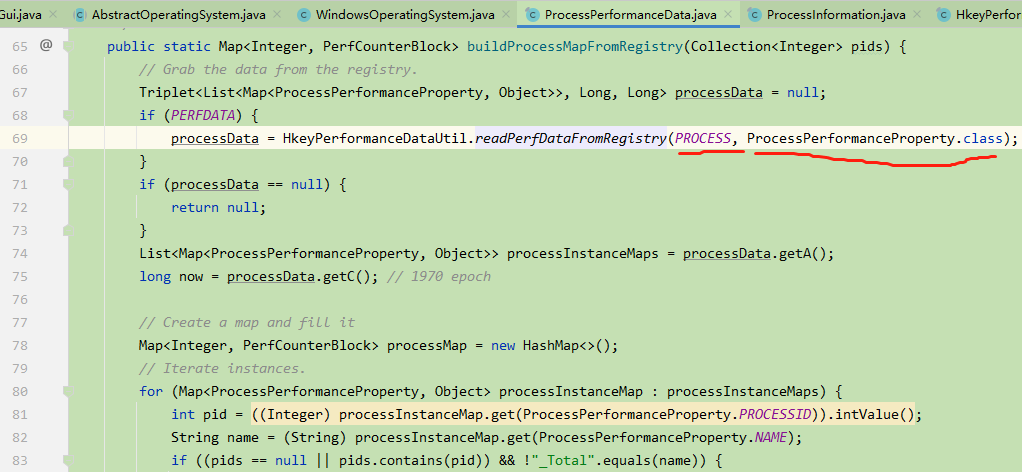
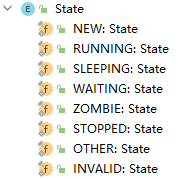
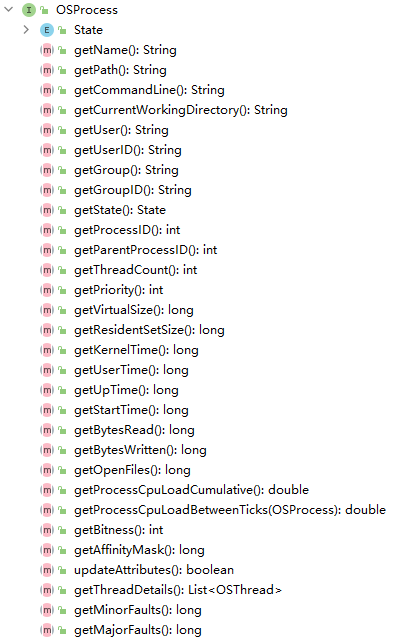
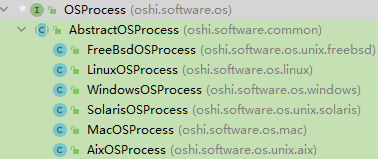
#### FileSystem



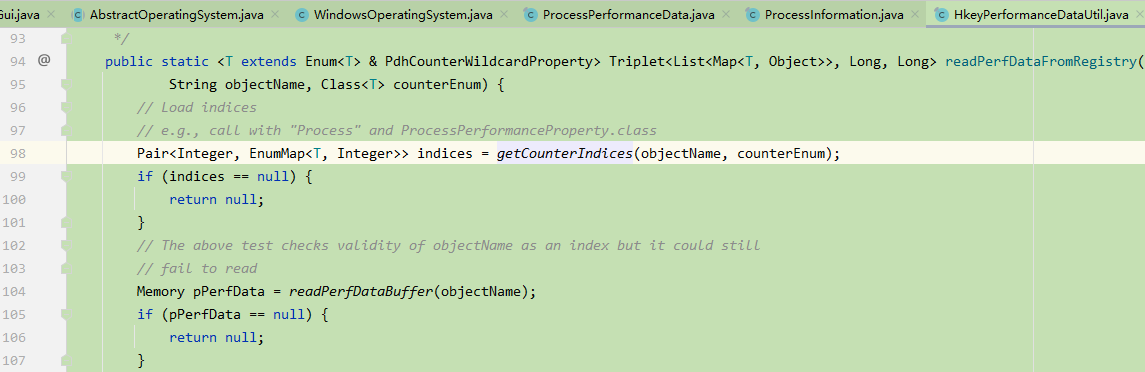
#### InternetProtocolStats

#### Sessions

#### ★ Processes

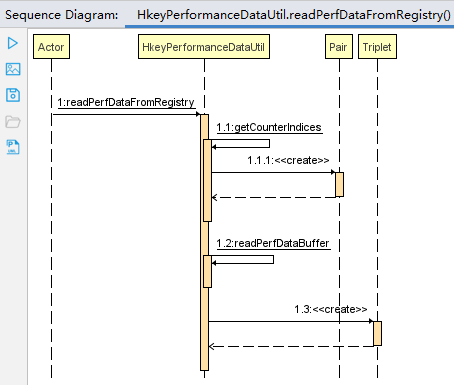


*// call with "Process" and ProcessPerformanceProperty.class*



*// Reads and parses a block of performance data from the registry.*

oshi.driver.windows.registry.HkeyPerformanceDataUtil#readPerfDataFromRegistry



#### ProcessCount

#### ChildProcesses

#### ThreadCount

#### Bitness

#### SystemUptime

#### SystemBootTime

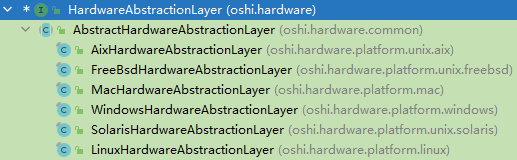
#### isElevated

#### NetworkParams

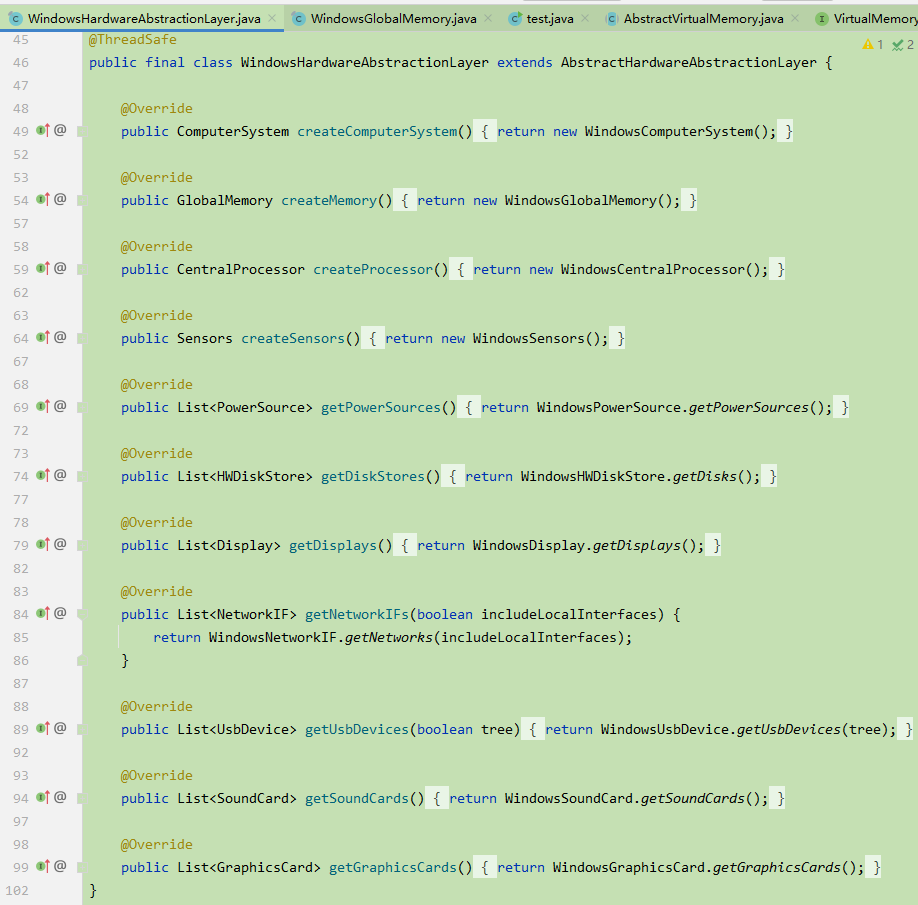
#### Services

#### DesktopWindows

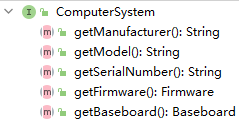
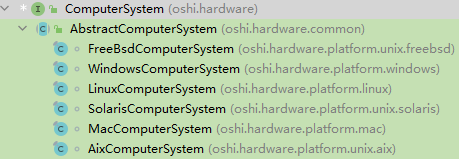
### Hardware硬件



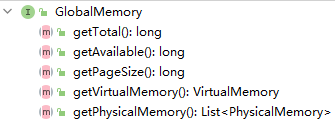
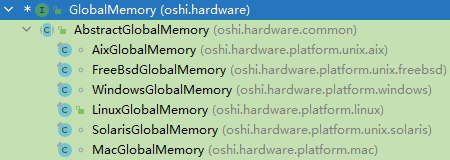
★下面以 WindowsHardwareAbstractionLayer为例进行分析：



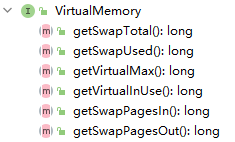
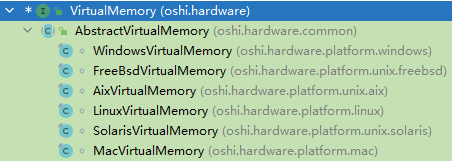
#### ComputerSystem



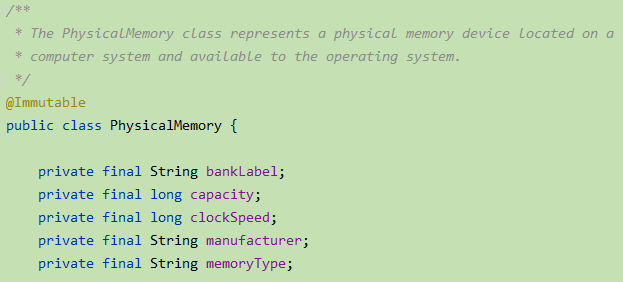
#### Memory



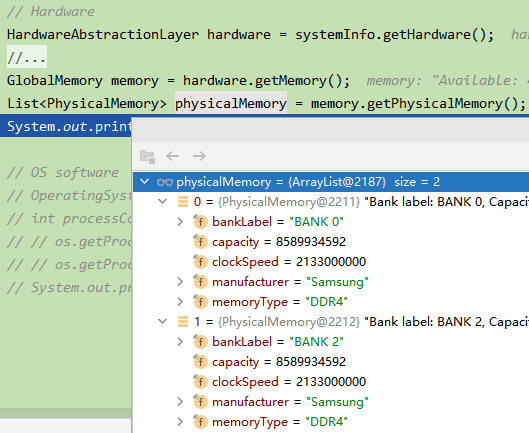
1）Virtual Memory



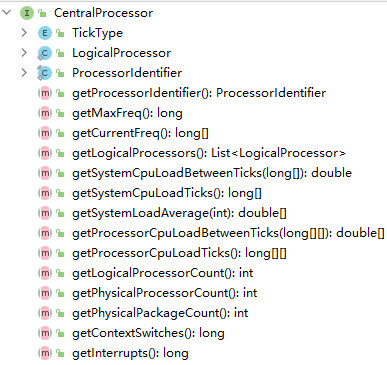
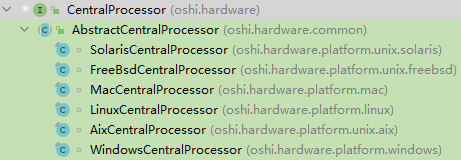
2）Physical Memory



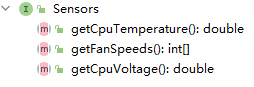
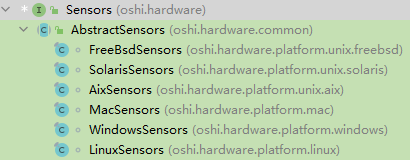
Eg.



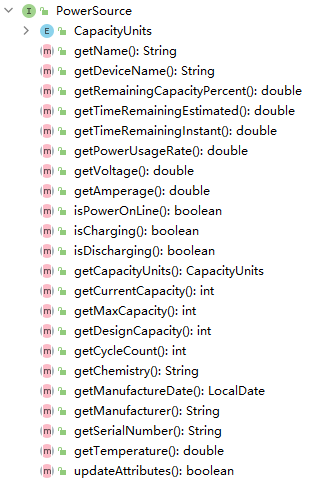
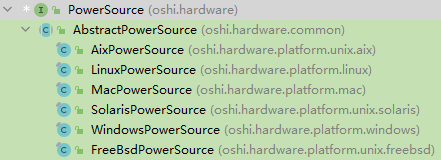
#### Processor



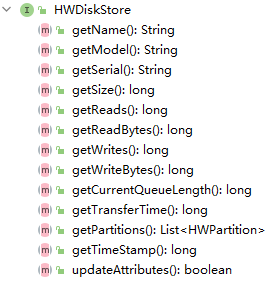
#### Sensors



#### PowerSource



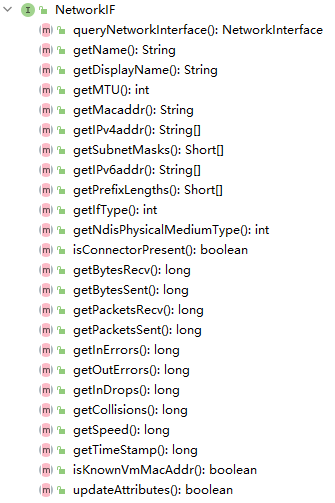
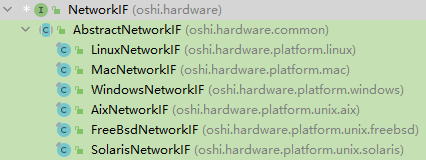
#### DiskStores



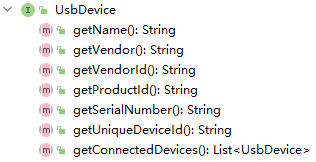
#### Displays



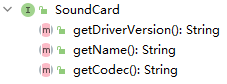
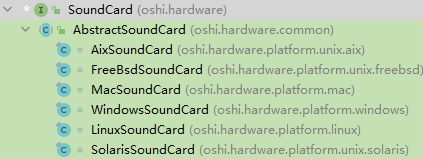
#### NetworkIFs



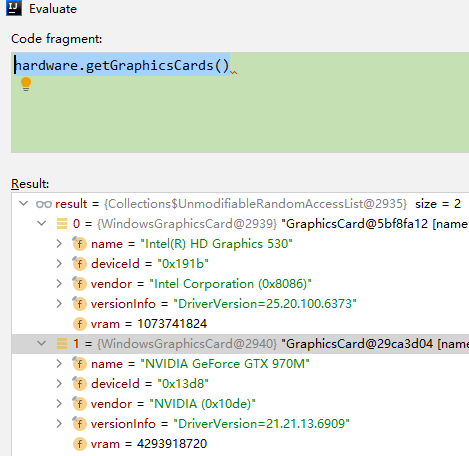
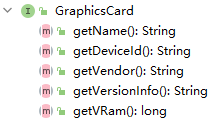
#### UsbDevices



#### SoundCards



#### GraphicsCards



# EhCache

介绍教程 https://www.jianshu.com/p/154c82073b07

整合 SpringBoot https://www.cnblogs.com/xzmiyx/p/9897623.html

## 配置

### 配置文件 ehcache-shiro.xml

com.ruoyi.framework.config.ShiroConfig#getCacheManagerConfigFileInputStream 这个里面绑定配置

*<!-- 磁盘缓存位置 -->*<diskStore path="java.io.tmpdir"/>  
  
*<!-- maxEntriesLocalHeap:堆内存中最大缓存对象数，0没有限制 -->  
<!-- maxElementsInMemory： 在内存中缓存的element的最大数目。-->  
<!-- eternal:elements是否永久有效，如果为true，timeouts将被忽略，element将永不过期 -->  
<!-- timeToIdleSeconds:失效前的空闲秒数，当eternal为false时，这个属性才有效，0为不限制 -->  
<!-- timeToLiveSeconds:失效前的存活秒数，创建时间到失效时间的间隔为存活时间，当eternal为false时，这个属性才有效，0为不限制 -->  
<!-- overflowToDisk： 如果内存中数据超过内存限制，是否要缓存到磁盘上 -->  
<!-- statistics：是否收集统计信息。如果需要监控缓存使用情况，应该打开这个选项。默认为关闭（统计会影响性能）。设置statistics="true"开启统计 -->  
  
<!-- 默认缓存 -->*<defaultCache  
 maxEntriesLocalHeap="1000"  
 eternal="false"  
 timeToIdleSeconds="3600"  
 timeToLiveSeconds="3600"  
 overflowToDisk="false">  
</defaultCache>

*<!-- 登录记录缓存，10分钟过期 -->*<cache name="loginRecordCache"  
 maxEntriesLocalHeap="2000"  
 eternal="false"  
 timeToIdleSeconds="600"  
 timeToLiveSeconds="0"  
 overflowToDisk="false"  
 statistics="false">  
</cache>

### 定义bean

*/\*\*  
 \* 缓存管理器 使用Ehcache实现*

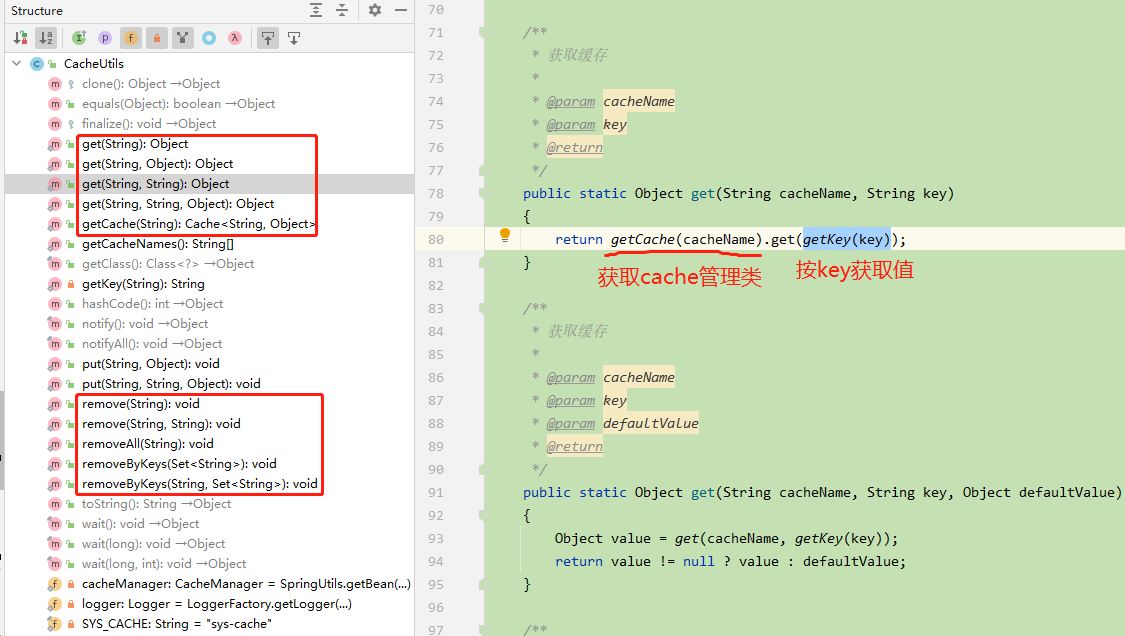
*\* org.apache.shiro.cache.ehcache.EhCacheManager*

*\* 这个类本质上就是包装了 protected net.sf.ehcache.CacheManager manager;  
 \*/*@Bean  
public EhCacheManager getEhCacheManager() {  
 net.sf.ehcache.CacheManager cacheManager = net.sf.ehcache.CacheManager.*getCacheManager*("ruoyi");  
 EhCacheManager em = new EhCacheManager();  
 if (StringUtils.*isNull*(cacheManager)) {  
 em.setCacheManager(new net.sf.ehcache.CacheManager(getCacheManagerConfigFileInputStream()));  
 return em;  
 }  
 else {  
 em.setCacheManager(cacheManager);  
 return em;  
 }  
}

## 核心类

### 若依CacheUtils操作方法：

com.ruoyi.common.utils.CacheUtils



private static CacheManager *cacheManager* = SpringUtils.*getBean*(CacheManager.class);

*/\*\*  
 \* 获得一个Cache，没有则显示日志  
 \*/*public static Cache<String, Object> getCache(String cacheName)  
{  
 Cache<String, Object> cache = *cacheManager*.getCache(cacheName);  
 if (cache == null)  
 {  
 throw new RuntimeException("当前系统中没有定义“" + cacheName + "”这个缓存。");  
 }  
 return cache;  
}

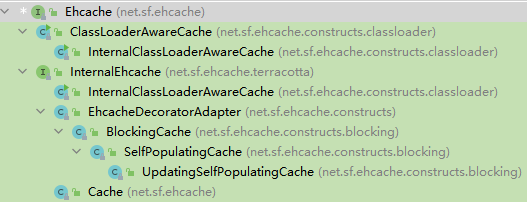
### 若依CacheManager

public final <K, V> Cache<K, V> getCache(String name) throws CacheException {  
 try {

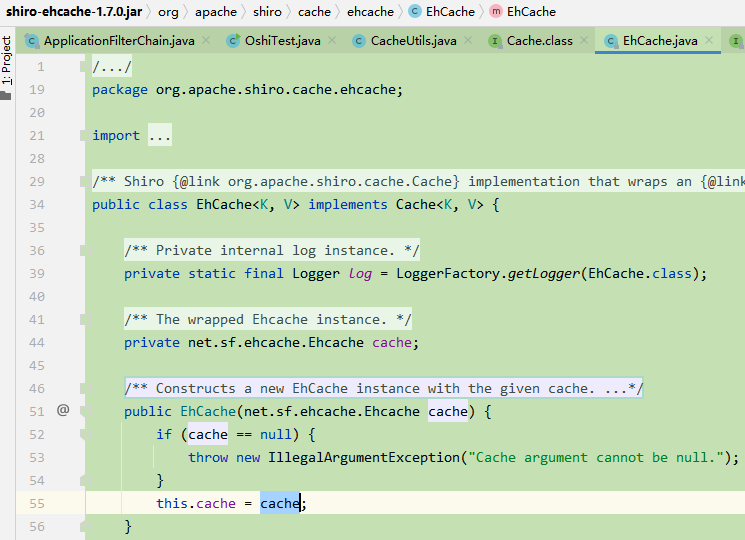
*// 这里就是shiro框架调用EcCache的地方了*  
 net.sf.ehcache.Ehcache cache = ensureCacheManager().getEhcache(name);  
 if (cache == null) {  
 this.manager.addCache(name);  
 cache = manager.getCache(name);  
 } else {  
 if (*log*.isInfoEnabled()) {  
 *log*.info("Using existing EHCache named [" + cache.getName() + "]");  
 }  
 }  
 return new EhCache<K, V>(cache);  
 } catch (net.sf.ehcache.CacheException e) {  
 throw new CacheException(e);  
 }  
}

### Cache

缓存核心类



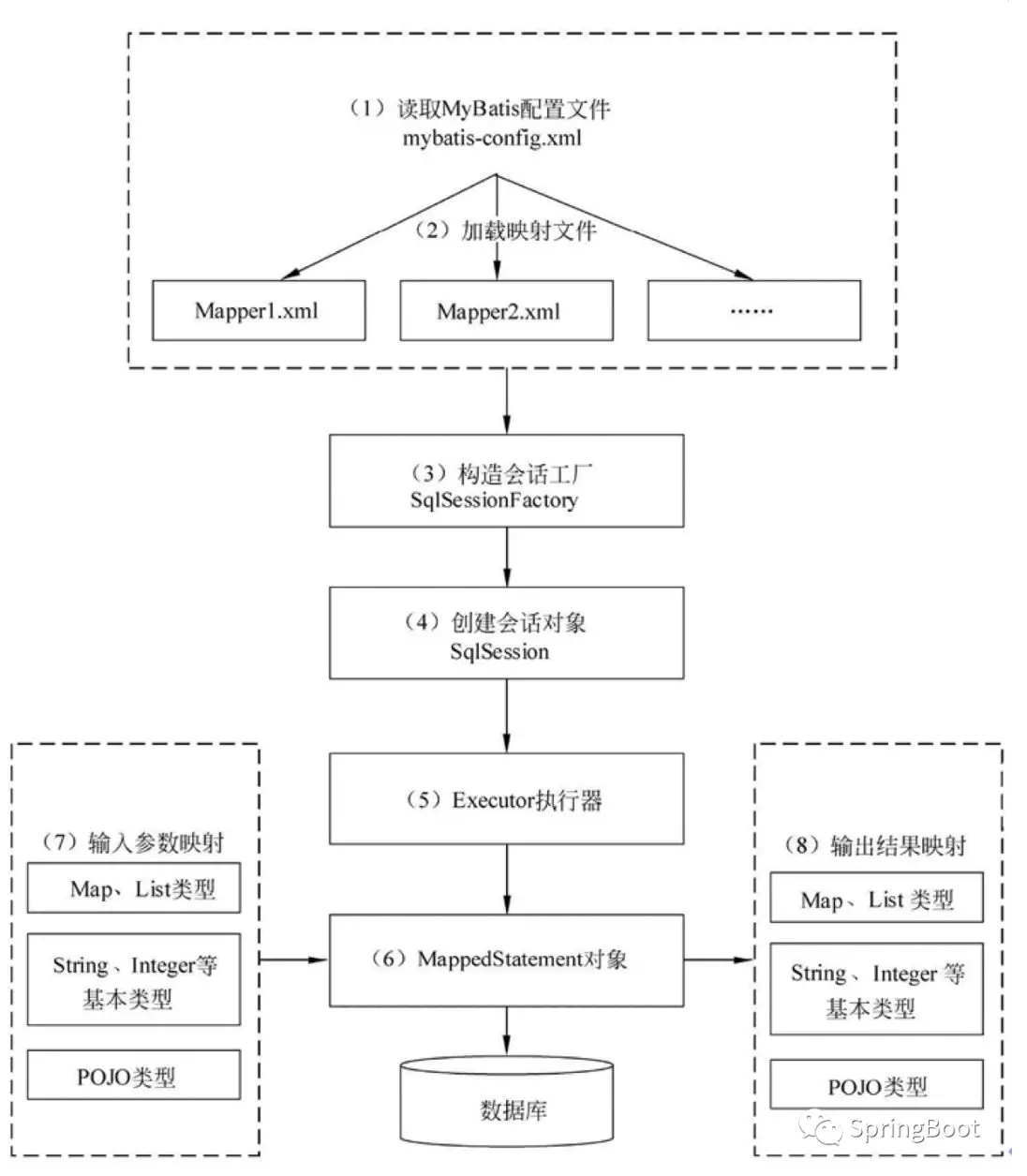
Shiro和ehcache关联的类：



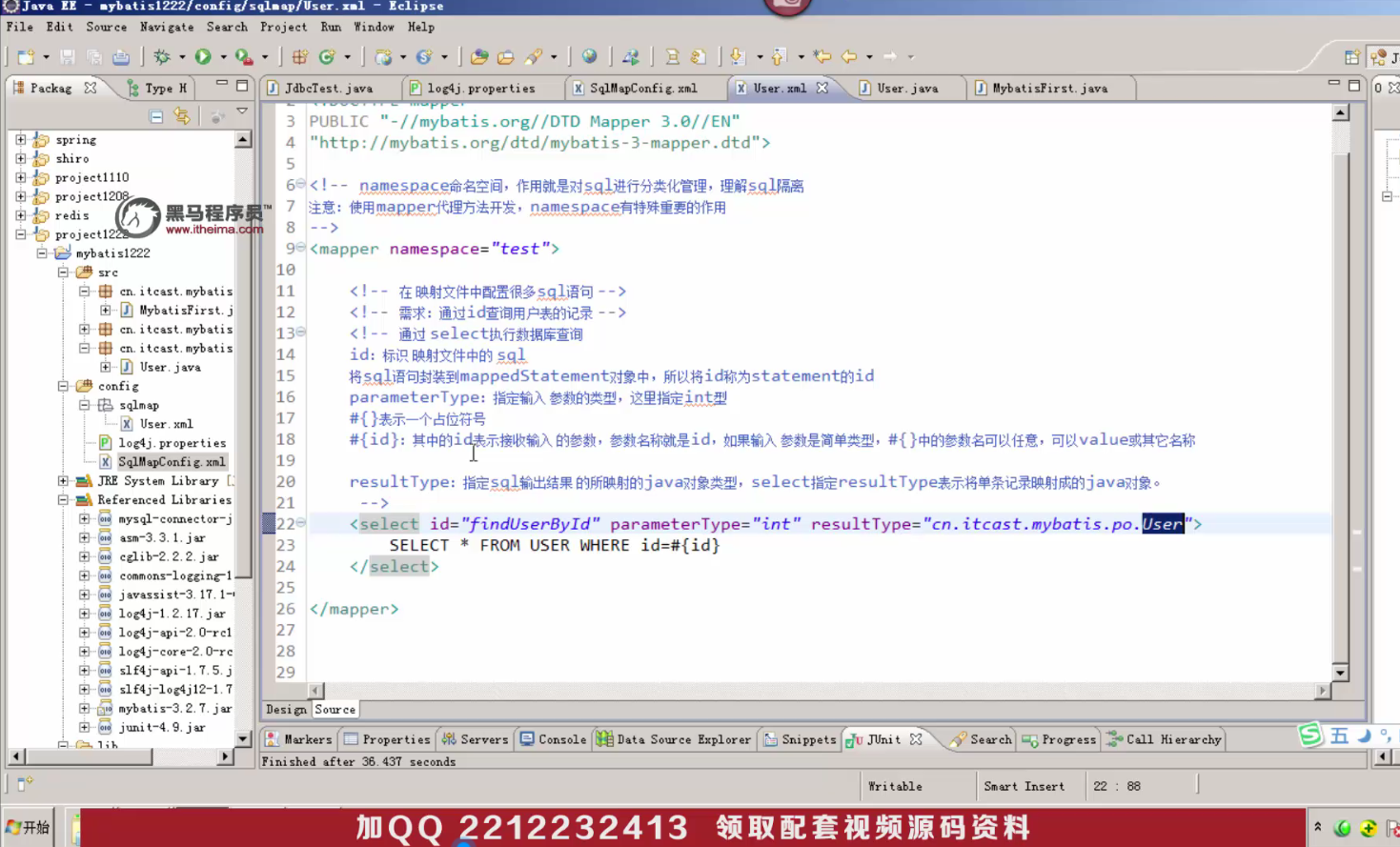
# Mybatis

结合代码和UML图看效果更好，demo/mybatis/readme.md

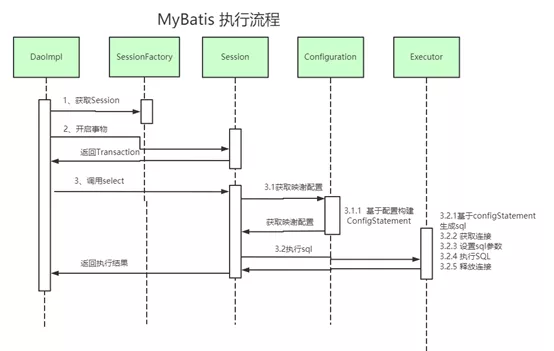
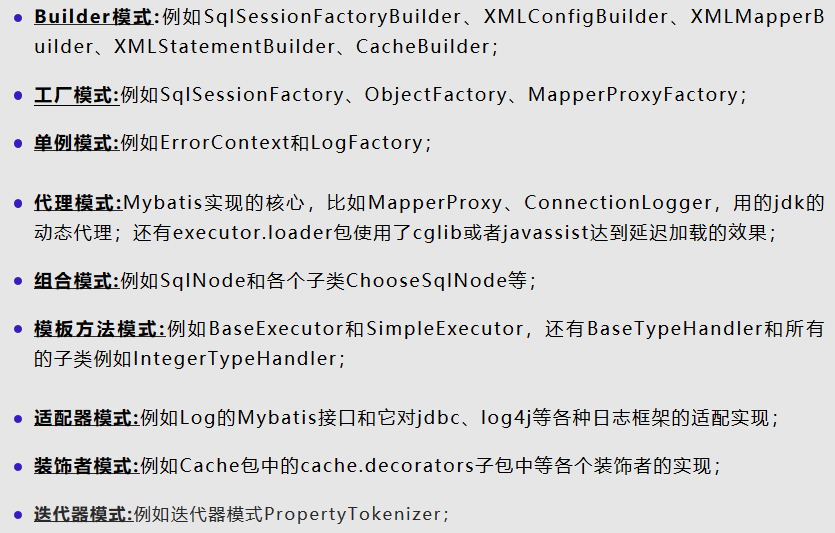
一图纵览 Mybatis 执行流程



定义 xml mapper的样例



设计模式



# Mybatis-plus

结合代码和UML图看效果更好

demo/SpringBoot：com.example.springboot.mybatisplus.MybatisPlusConfig

# RocketMQ里的netty

教程

【Netty】工作原理解析 https://www.cnblogs.com/haif/p/14129494.html

rocketmq之源码分析netty简要分析（五）https://my.oschina.net/wangshuaixin/blog/3055707

## Server 端：

org.apache.rocketmq.remoting.netty.NettyRemotingServer

ServerBootstrap 核心类

private final ServerBootstrap serverBootstrap;

Worker Group

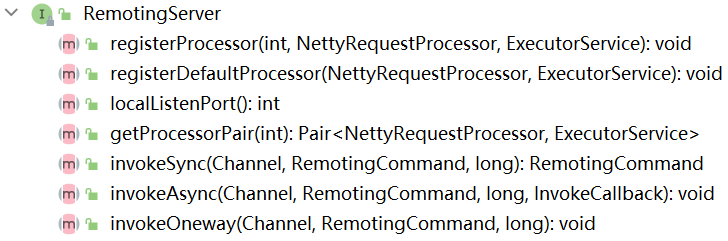
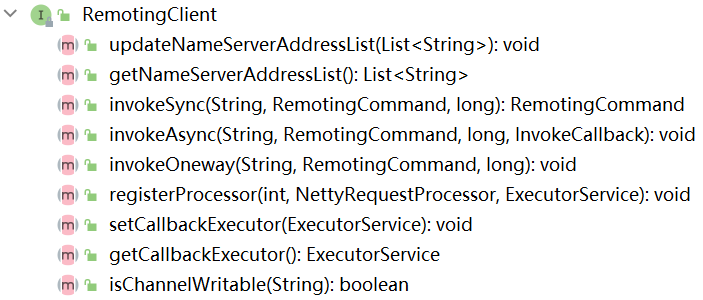
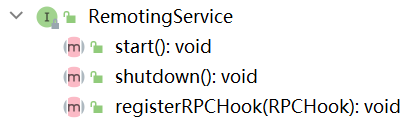
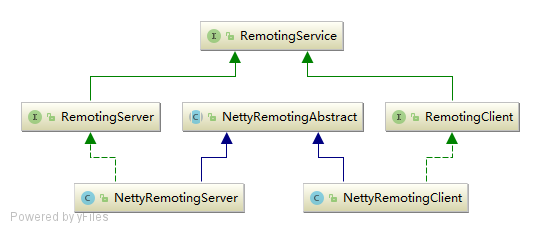
private final EventLoopGroup eventLoopGroupSelector;

Boss Group

private final EventLoopGroup eventLoopGroupBoss;

org.apache.rocketmq.namesrv.processor.DefaultRequestProcessor #processRequest

关键类层级关系：

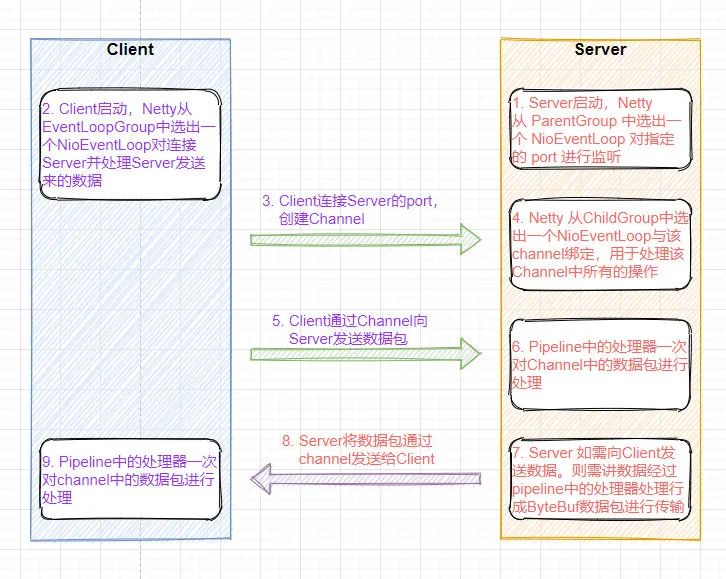
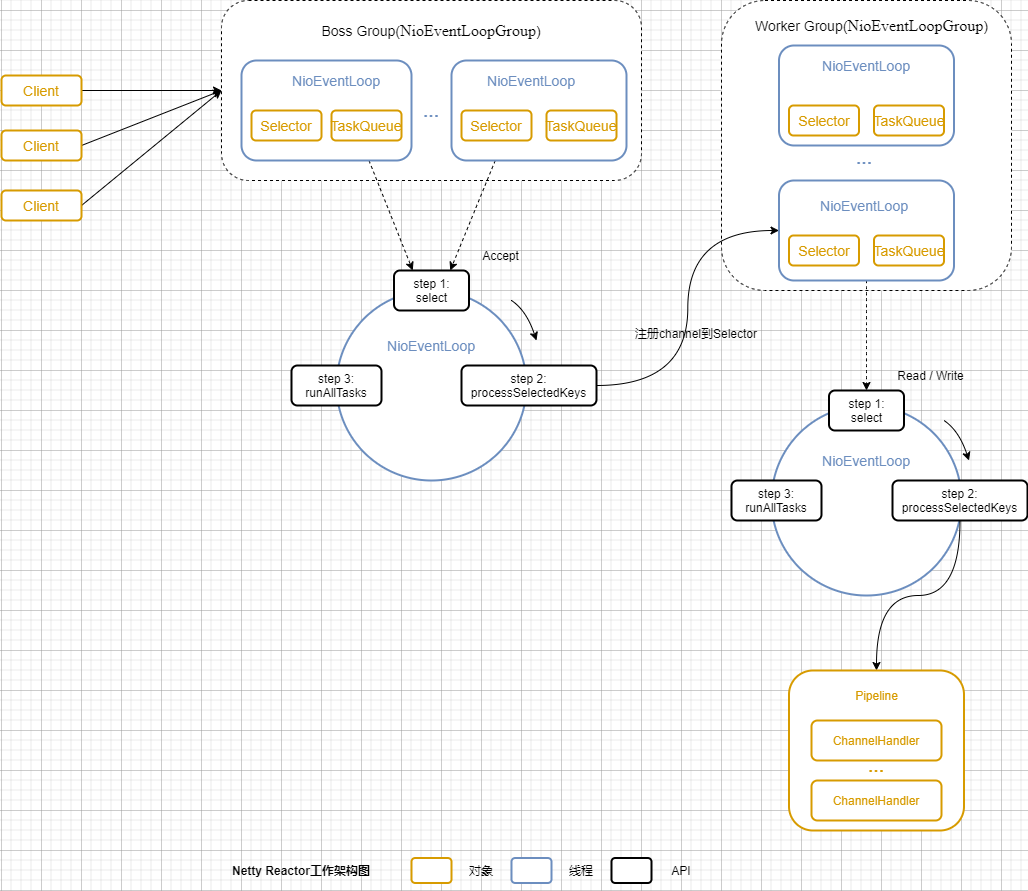


## Client 端：

org.apache.rocketmq.broker.out.BrokerOuterAPI #registerBroker

## Netty原理

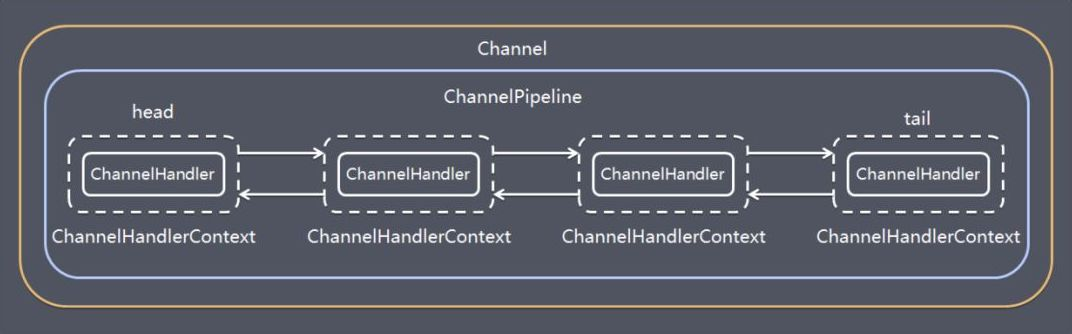
工作原理示意图（Netty Reactor 工作架构图）



工作流程说明：

1. Netty 抽象出两组线程池：BossGroup 专门负责接收客户端的连接，WorkerGroup 专门负责网络的读写。
2. BossGroup 和 WorkerGroup 类型都是 NioEventLoopGroup。
3. NioEventLoopGroup 相当于一个事件循环组，这个组中含有多个事件循环，每一个事件循环是 NioEventLoop。
4. NioEventLoop 表示一个不断循环的执行处理任务的线程，每个 NioEventLoop 都有一个 selector ，用于监听绑定在其上的 socket 的网络通讯。其内部采用串行化设计，从消息的读取 -> 解码 -> 处理 -> 编码 -> 发送，始终由 IO 线程 NioEventLoop 负责。
5. NioEventLoopGroup 可以有多个线程, 即可以含有多个 NioEventLoop。
6. 每个 Boss NioEventLoop 循环执行的步骤：
   * 轮询 accept 事件。
   * 处理 accept 事件，与 client 建立连接，生成 NioScocketChannel ，并将其注册到某个 Worker NIOEventLoop 上 的 selector。
   * 处理任务队列的任务，即 runAllTasks。
7. 每个 Worker NIOEventLoop 循环执行的步骤：
   * 轮询 read，write 事件。
   * 处理 I/O 事件， 即read ，write 事件，在对应 NioScocketChannel 处理业务。
   * 处理任务队列的任务，即 runAllTasks。
8. 每个 Worker NIOEventLoop 处理业务时，会使用pipeline（管道），pipeline 中包含了 channel , 即通过pipeline 可以获取到对应通道，管道中维护了很多的handler 处理器用来处理 channel 中的数据。

在 Netty 中每个 Channel 都有且仅有一个 ChannelPipeline 与之对应，它们的组成关系如下：



异步网络模型 Proactor，事件驱动？

