# Redis介绍

Redis是一个key-value存储系统。和Memcached类似，它支持存储的value类型相对更多，包括string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和hash（哈希类型）。这些数据类型都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，redis支持各种不同方式的排序。与memcached一样，为了保证效率，数据都是缓存在内存中。区别的是redis会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了master-slave(主从)同步。

Redis 是一个高性能的key-value数据库。 redis的出现，很大程度补偿了memcached这类key/value存储的不足，在部 分场合可以对关系数据库起到很好的补充作用。它提供了Java，C/C++，C#，PHP，JavaScript，Perl，Object-C，Python，Ruby，Erlang等客户端，使用很方便。[1]

Redis支持主从同步。数据可以从主服务器向任意数量的从服务器上同步，从服务器可以是关联其他从服务器的主服务器。这使得Redis可执行单层树复制。从盘可以有意无意的对数据进行写操作。由于完全实现了发布/订阅机制，使得从数据库在任何地方同步树时，可订阅一个频道并接收主服务器完整的消息发布记录。同步对读取操作的可扩展性和数据冗余很有帮助。

redis目前还只能作为小数据量存储（全部数据能够加载在内存中） ，海量数据存储方面并不是redis所擅长的领域。

与同类产品memcached比较:

1.网络IO模型方面：Memcached是多线程，分为监听线程、worker线程，引入锁，带来了性能损耗。Redis使用单线程的IO复用模型，将速度优势发挥到最大，也提供了较简单的计算功能

2.内存管理方面：Memcached使用预分配的内存池的方式，带来一定程度的空间浪费 并且在内存仍然有很大空间时，新的数据也可能会被剔除，而Redis使用现场申请内存的方式来存储数据，不会剔除任何非临时数据 Redis更适合作为存储而不是cache

3.数据的一致性方面：Memcached提供了cas命令来保证.而Redis提供了事务的功能，可以保证一串 命令的原子性，中间不会被任何操作打断

4.存储方式方面：Memcached只支持简单的key-value存储，不支持枚举，不支持持久化和复制等功能

总结: Redis是一个高性能的key-value数据库。 redis的出现，很大程度补偿了memcached这类key/value存储的不足，在部分场合可以对关系数据库起到很好的补充作用。

# 集群条件下用户会话数据共享

在介绍用户会话数据共享前，先讲个生活场景，以此来说明用户会话数据共享。

一家咖啡店有喝5杯咖啡免费赠一杯咖啡的优惠，然而一次性消费5杯咖啡的机会微乎其微，这时就需要某种方式来纪录某位顾客的消费数量。

想象一下其实也无外乎下面的几种方案：

1、该店的店员很厉害，能记住每位顾客的消费数量，只要顾客一走进咖啡店，店员就知道该怎么对待了。这种做法就是协议本身支持状态。

2、发给顾客一张卡片，上面记录着消费的数量，一般还有个有效期限。每次消费时，如果顾客出示这张卡片，则此次消费就会与以前或以后的消费相联系起来。这种做法就是在客户端保持状态，就是下面要介绍的“以cookie的方式共享用户会话数据”。

3、发给顾客一张会员卡，除了卡号之外什么信息也不纪录，每次消费时，如果顾客出示该卡片， 则店员在店里的纪录本上找到这个卡号对应的纪录添加一些消费信息。这种做法就是在web应用服务端保持状态，就是下面要介绍的“以cookie+session的方式共享用户会话数据”，

4、发给顾客一张会员卡，除了卡号之外什么信息也不纪录，每次消费时，如果顾客出示该卡片， 店员不在将消费信息记录在记录本里，因为会员增多，记录本不够用，所以将消费信息输入到“电脑里”。这种做法就是最后要重点介绍的“以cookie+第三方存储的方式共享用户会话数据”，此处的用户数据是记录在“第三方电脑里”

重点说明:无论采用哪种方式，可以看到cookie在其中都起着重大的作用，用来作为客户端和服务器端沟通的桥梁。当初Netscape发明cookie，正式为了解决http这种无状态的协议的，所以cookie的使用一般是有服务器端生成的（chrome不支持用js生成cookie）。对cookie在各浏览器，各域下的大小、个数均有限制、以及cookie跨域的问题要注意。

## 以cookie的方式共享用户会话数据

在一些简单的应用场景中，用户会话数据的安全性不高、存储容量不大的情况下，可以采取此方案，即将用户的会话数据加密放到cookie里，浏览器下次访问时，会将这个cookie发送到服务器，服务器解密。

## 以cookie+session的方式共享用户会话数据

在一些并发量不高的后台管理项目中，用户的会话数据，可以存储在服务器的内存中，但是这里需要用到session复制。如在tomcat中就实现了session复制功能。但是当并发量上去，后台的tomcat服务器增多，session复制会造成很大的性能开销。

## [重点]以cookie+第三方存储的方式共享用户会话数据

在高并发的情况下，随着web应用服务器的增多，session复制已不能在满足性能的需要，有必要将用户的共享数据存储到第三方的存储服务上，如Memcached、redis、甚至数据库。

考虑到redis的势头很强劲，这里用redis的实现方案。

参考文章:

<http://www.tuicool.com/articles/EFnaUz>

<http://blog.fens.me/linux-redis-install/>

### 逻辑架构



### Haproxy、keepalived搭建

请参考：《HAProxy and Keepalived安装指导》



实施的时候，修改你环境的IP即可！

### JDK1.7、Tomcat7安装

* 到oracle官网，下载与操作系统相匹配的JDK1.7安装包，下载tomcat7.搭建好Tomcat7环境。
* 将“tomcat7从redis中访问用户会话数据”所需要的相关jar放到%TOMCAT\_HOME%/lib下。

tomcat-redis-session-manager-1.2-tomcat-7-java-7.jar

jedis-2.0.0.jar

commons-pool-1.3.jar

commons-pool2-2.2.jar

* 将%TOMCAT\_HOME%/conf/context.xml中加入redis配置，这样在应用程序中不用修改任何操作session的代码，tomcat会自动调用RedisSessionMananger将用户会话数据保存到redis

<Valve className="com.radiadesign.catalina.session.RedisSessionHandlerValve" />

<Manager className="com.radiadesign.catalina.session.RedisSessionManager"

host="192.168.202.149"

port="6379"

database="0"

maxInactiveInterval="60" />

### Redis安装、简单命令

#apt-get install redis-server //在线安装

#ps -ef|grep redis //查看redis进程

#netstat -an|grep :6379 //查看是否开启监听端口

#/etc/init.d/redis-server status //查看服务状态，若正常则为：redis-server is running

#/etc/init.d/redis-server start //启动

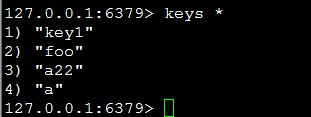
#/etc/init.d/redis-server stop //停止

#/etc/init.d/redis-server stop //重启

#redis-cli //进入cli命令界面



#keys \* //查看存储的键



//set “yourkey” "yourvalue" //设置key对应的value

//get “yourkey” //获取key对应的value

//del “yourkey” //删除key以及对应的value

其他命令请网上查询

### 程序代码测试

新建1个web工程。

在controller层，设置session的参考代码如下：

/\*\*

\* 通过POST请求,ajax增加session

\* **@return**

\* **@throws** Exception

\*/

@ResponseBody

@RequestMapping(value="/addSession",method=RequestMethod.*POST*)

**public** String addSession(HttpServletRequest request,HttpServletResponse response) **throws** Exception{

*logger*.debug("==============addSession(POST)==================");

System.*out*.println("addSession....");

HttpSession session = request.getSession(**true**);

Demo demo = **new** Demo(); //Demo必须要实现序列化接口

demo.setId(1l);

demo.setUsername("xiaogang");

demo.setPassword("password");

session.setAttribute("userinfo", demo);

**return** "success add session";

}

获取session的参考代码如下

/\*\*

\* 通过GET请求,获取session

\* **@return**

\* **@throws** Exception

\*/

@ResponseBody

@RequestMapping(value="/getSession",method=RequestMethod.*GET*)

**public** String getSession(HttpServletRequest request,HttpServletResponse response) **throws** Exception{

*logger*.debug("==============getSession(GET)==================");

Demo demo = (Demo)request.getSession().getAttribute("userinfo");

String result = demo.getId()+"," + demo.getUsername() + ","+demo.getPassword();

System.*out*.println("getSession:"+result);

**return** result;

}

测试步骤如下：

* 当用户发起新建session请求时，通过虚拟IP访问haproxy，haproxy将请求落到Tomcat7-1，新建的session会通过tomcat7-1保存到redis中。当用户再次发起访问session请求时，由于算法为简单的轮询，haproxy将请求落到另外一台Tomcat7-2上，该Tomcat7-2会从redis中获取用户session。
* 在上面的步骤基础上，如果手工清空redis中的用户session数据，再次发起访问session请求时，后台报找不到用户session异常。

### 架构演进(待补充)

有时间可以将架构复杂化，增加redis集群，当某个主redis服务器挂掉，从服务器可以马上接管工作。