Redis 快速提高系统性能的银弹

说明：阅读该文章需要一定 Web 开发经验，最好对 Redis 有一个基本的认知，文章最后的附录也会为大家提供一些相关的文章，本文章只是为了让那些对 Redis 的应用仅仅局限于缓存 的开发人员了解到 Redis 更多可能的应用场景，由于篇幅限制，文中很多场景只是阐述了实现的思想及部分原理，仅仅提供了部分功能的具体实现。

现代高并发复杂系统面临的挑战

现代系统随着功能的复杂化，各种各样需求层出不穷，面对愈加复杂话的业务系统、越来越庞大的用户群体，以及用户对体验的要求越来越高，性能就变得更加重要。

抛开代码逻辑、服务器性能的相关问题外，提高性能的方式有以下几种：

1. 动静分离
2. 复杂均衡
3. 分布式
4. 集群化
5. 缓存
6. 限流处理
7. 数据压缩
8. 其他

我们来分析一下负载均衡、分布式、集群化涉及的问题：

1. 配置管理变得复杂，因此需要设置配置中心来解决该问题。
2. 同一个用户的请求会转发至不同的 Web 服务器，从而导致 Session 丢失等问题。
3. 同一个请求在分布式环境中需要不同服务来提供不同处理，从而需要分布式事务来确保数据的一致性。
4. 分布式唯一 ID 问题。

另外针对不同部分系统中的一些特定问题又有其他的一些特殊业务需求：

1. IP统计

2. 用户登录记录统计
3. 实时的排行榜
4. 原子计数
5. 最新评论

诚然，以上各种问题都有花样繁多的解决方法，例如：

配置中心可以使用 Zookpeer、Redis 等实现。

Session 丢失可以使用 Session 同步、客户端 token、Session 共享等解决，其中 Session 共享又可以细分不同实现方式。

面对层出不穷的概念，以及各种新兴的技术，我们往往会显得力不从心，那么有没有一个银弹可以解决这些问题呢？

Redis 非银弹却无比接近

我这里为大家推荐的就是 Redis ，虽然它离真正意义的银弹还是有些距离，但是他是为数不多的接近银弹的解决方案：

1. Redis 使用 C 开发，是一款内存 K/V 数据库，架构设计极简，性能卓著。
2. Redis 采用 单线程 多路复用的设计，可以从而避免了并发带来的锁性能损耗等问题。
3. Redis 安装、测试、配置、运维较其他产品更为容易。
4. Redis 是目前为止最受欢迎的 K/V 数据库，支持持久化，value 支持多种数据结构。
5. Redis 命令语法简单，极易掌握。
6. Redis 提供了一种通用的协议，使得各种编程语言都能很方便的开发出与其交互的客户端。
7. Redis 开放源码，我们可以对其进行二次开发来定制优化。
8. Redis 目前有较好的社区维护，版本迭代有所保障，新的功能也在有条不紊的添加完善。
9. Redis 有较好的主从复制、集群相关支持。
10. 最新版本提供模块化功能，可以方便的扩展功能。

接下来我们就来说说怎么使用 Redis 解决之前提到的问题：

1. 配置中心

   Redis 本身就是内存 K/V 数据库，支持 哈希、集合、列表等五种数据结构，从而配置信息的存储、读取速度都能够得到满足，Redis 还提供订阅/发布功能从而可以在配置发生改变时通知不同服务器来进行更新相关配置。

2. 分布式锁

   使用 Redis 的 SETNX 命令或者 SET 命令配合 NX 选项的方式以及过期时间等功能可以很方便的实现一个性能有约定分布式锁。

3. 缓存

   Redis 支持多种过期淘汰机制，本身性能的优势也使 Redis 在缓存方面得到广泛使用。

4. Lua 脚本

   Lua 是一种轻量小巧的脚本语言，用标准C语言编写并开放源代码。Redis 支持 Lua 脚本的运行，从而可以扩展 Redis 中的命令实现很多复杂功能。

   Redis 支持使用 Lua 脚本来实现一些组合命令逻辑处理，从而可以使用 Redis 做为限流、分布式唯一 ID 相关技术的实现。

5. Redis 支持 BitMaps

   位图（ bitmap ）是一种非常常用的结构，在索引，数据压缩等方面有广泛应用,能同时保证存储空间和速度最优化（ 而不必空间换时间 ）。

   使用 Redis 的 BitMaps 做为用户登录记录统计，不仅统计速度极快，而且内存占用极低。

6. Redis 支持 HyperLogLog 算法

   Redis HyperLogLog是一种使用随机化的算法，以少量内存提供集合中唯一元素数量的近似值。

   HyperLogLog 可以接受多个元素作为输入，并给出输入元素的基数估算值：

   - **基数**：集合中不同元素的数量。比如 {‘apple’, ‘banana’, ‘cherry’, ‘banana’, ‘apple’} 的基数就是3。
   - **估算值**：算法给出的基数并不是精确的，可能会比实际稍微多一些或者稍微少一些，但会控制在合理的范围之内。

   HyperLogLog 的优点是，即使输入元素的数量或者体积非常非常大，计算基数所需的空间总是固定的、并且是很小的。

   在 Redis 里面，每个 HyperLogLog 键只需要花费 12 KB 内存，就可以计算接近 2^64 个不同元素的基数。这和计算基数时，元素越多耗费内存就越多的集合形成鲜明对

比。使用 HyperLogLog 算法，我们可以轻而易举的实现 IP 统计等对数据容许些许误差的统计功能。

7. Redis 支持 Geo 功能

   我们可以使用基于 Redis 来实现地理位置相关管理，附近的人、两地理位置间距离计算等功能变得极为容易实现。

8. 简单消息队列

   Redis 列表 + 发布/订阅功能可以很方便的实现一个简单的消息队列，将消息存入 Redis 列表中，通过 发布/订阅功能通知指定成员，成员获取到通知后可以根据通知内容进行对应处理。

9. 全文检索

   Redis 官方团队开发了 RediSearch 模块，可以实现使用 Redis 来做全文检索的功能。

10. 分布式唯一ID

    Redis 的设计使其可以避免并发的多种问题，使其命令都是原子执行，这些特性都天生匹配分布式唯一ID生成器的要求。

    而且通过与 Lua 脚本的结合使用更是能生成复杂的有某些规律的唯一ID。

部分代码实现

下面我们以 Java代码作为演示（编程语言实现方式原理类似只是具体实现方式有些许差别而已）讲解几个功能的实现：

Session 共享

原理：将不同 Web 服务器的 Session 信息统一存储在 Redis 中，并且获取 Session 也是从 Redis 中获取

实现方法：

方法一：基于 Tomcat 实现 Sessioin 共享：

Tomcat 配置步骤(相关代码资源可以从 https://gitee.com/coderknock/Tomcat-Redis-Session-Manager-Demo 获取)：

1. 将 commons-pool2-2.4.2.jar、jedis-2.9.0.jar、commons-pool2-2.4.2.jar 三个 jar 包放到 Tomcat 下的 lib 目录下（注意：不是项目的 lib 目录）。

2. 修改 Tomcat conf 下 context.xml：

```
XML
<Context>
    ......
    <Valve
className="com.orangefunction.tomcat.redissessions.RedisSessionHan
dlerValve" />
    <Manager
className="com.orangefunction.tomcat.redissessions.RedisSessionMan
ager"
            host="127.0.0.1"
            port="6379"
            database="0"
            maxInactiveInterval="60"
            password="admin123" />
        ......
</Context>
```

方法二：基于 Fileter 、 自行实现 HttpServletRequestWrapper 、 HttpSession ：

关键代码：

HttpSessionWrapper.java


```
java
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONException;
import com.coderknock.jedis.executor.JedisExecutor;
import com.coderknock.pojo.User;
import org.apache.commons.lang3.StringUtils;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpSessionContext;
import java.util.Enumeration;

/**
 * <p></p>
 *
 * @author 三产
 * @version 1.0
 * @date 2017-08-26
 * @QQGroup 213732117
 * @website http://www.coderknock.com
 * @copyright Copyright 2017 拿客 coderknock.com  All rights
reserved.
 * @since JDK 1.8
```

```
    */
   public class HttpSessionWrapper implements HttpSession {
       protected final Logger logger =
LogManager.getLogger(HttpSessionWrapper.class);
       private String sid = "";
       private HttpServletRequest request;
       private HttpServletResponse response;
       private final long creationTime =
System.currentTimeMillis();
       private final long lastAccessedTime =
System.currentTimeMillis();
       //过期时间单位秒
       private int expire_time = 60;

       public HttpSessionWrapper() {
       }

       public HttpSessionWrapper(String sid, HttpServletRequest
request,
                                 HttpServletResponse response) {
           this.sid = sid;
           this.request = request;
           this.response = response;
       }

       public Object getAttribute(String name) {
           logger.info(getClass() + "getAttribute(),name:" +
name);
           try {
               Object obj = JedisExecutor.execute(jedis -> {
                   String jsonStr = jedis.get(sid + ":" + name);
                   if (jsonStr != null ||
StringUtils.isNotEmpty(jsonStr)) {
                       jedis.expire(sid + ":" + name,
expire_time);// 重置过期时间
                   }
                   return jsonStr;
               });
               return obj;
           } catch (JSONException je) {
               logger.error(je);
           } catch (Exception e) {
               logger.error(e.getMessage());
           }
           return null;
       }

       public void setAttribute(String name, Object value) {
           logger.info(getClass() + "setAttribute(),name:" +
name);
           try {
               JedisExecutor.executeNR(jedis -> {
                   if (value instanceof String) {
```

```
                        String value_ = (String) value;
                        jedis.set(sid + ":" + name, value_);//普通字
符串对象
                    } else {
                        jedis.set(sid + ":" + name,
JSON.toJSONString(value));//序列化对象
                    }

                    jedis.expire(sid + ":" + name, expire_time);//
重置过期时间
                });
            } catch (Exception e) {
                logger.error(e);
            }

        }


        public void removeAttribute(String name) {
            logger.info(getClass() + "removeAttribute(),name:" +
name);
            if (StringUtils.isNotEmpty(name)) {
                try {
                    JedisExecutor.executeNR(jedis -> {
                        jedis.del(sid + ":" + name);
                    });
                } catch (Exception e) {
                    logger.error(e);
                }
            }

        }
        //...... 省略部分代码
    }
```

SessionFilter.java

```
java
    import com.coderknock.wrapper.DefinedHttpServletRequestWrapper;
    import org.apache.commons.lang3.StringUtils;
    import org.apache.logging.log4j.LogManager;
    import org.apache.logging.log4j.Logger;

    import javax.servlet.*;
    import javax.servlet.http.Cookie;
    import javax.servlet.http.HttpServletRequest;
    import javax.servlet.http.HttpServletResponse;
    import java.io.IOException;
    import java.util.UUID;

    /**
```

```
    * <p></p>

    *

    * @author 三产

    * @version 1.0

    * @date 2017-08-26

    * @QQGroup 213732117

    * @website http://www.coderknock.com

    * @copyright Copyright 2017 拿客 coderknock.com  All rights
reserved.

    * @since JDK 1.8

    */
   public class SessionFilter implements Filter {
       protected final Logger logger =
LogManager.getLogger(getClass());
       private static final String host = "host";
       private static final String port = "port";
       private static final String seconds = "seconds";

       public void init(FilterConfig filterConfig) throws
ServletException {
           logger.debug("init filterConfig info");
       }

       public void doFilter(ServletRequest request,
ServletResponse response,
                            FilterChain chain) throws IOException,
ServletException {
           //从cookie中获取sessionId，如果此次请求没有sessionId，重写为
这次请求设置一个sessionId

           HttpServletRequest httpRequest = (HttpServletRequest)
request;
           HttpServletResponse httpResponse =
(HttpServletResponse) response;
           String sid = null;
           if (httpRequest.getCookies() != null) {
               for (Cookie cookie : httpRequest.getCookies()) {
                   if (cookie.getName().equals("JSESSIONID")) {
                       sid = cookie.getValue();
                       break;
                   }
               }
           }
```

```
            if (StringUtils.isEmpty(sid)) {
                try {
                    Cookie cookie = new Cookie("JSESSIONID",
httpRequest.getLocalAddr() + ":" + request.getLocalPort() + ":" +
UUID.randomUUID().toString().replaceAll("-", ""));
                    httpResponse.addCookie(cookie);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }

            logger.info("JSESSIONID:" + sid);
            chain.doFilter(new
DefinedHttpServletRequestWrapper(sid, httpRequest, httpResponse),
response);
        }

        public void destroy() {
        }

    }
```

排行榜

原理：通过 Redis 有序集合可以很便捷的实现该功能

关键命令：

`ZADD key [NX|XX][CH][INCR] score member [score member ...]`：初始化排行榜中成员及其分数。

`ZINCRBY key increment member` ：为某个成员增加分数，如果该成员不存在则会添加该成员并设定分数为 `increment` 。

`ZUNIONSTORE destination numkeys key [key ...][WEIGHTS weight [weight ...]][AGGREGATE SUM|MIN|MAX]`：可以合并多个排行榜，该操作会将几个集合的并集存储到 `destination` 中，其中各个集合相同成员分数会叠加或者取最大、最小、平均值等(根据 [AGGREGATE SUM|MIN|MAX] 参数决定，默认是叠加)，从而可以实现根据多个分排行榜来计算总榜排行的功能。

`ZREVRANGE key start stop [WITHSCORES]` ：该命令就是最关键的获取排行信息的命令，可以获取从高到低的成员。

Redis 命令演示(“#”之后为说明)：

```
    # 1、存储几个排行榜成员数据（这里可以理解为把自己系统已有数据加载到 Redis
中）
    ZADD testTop 23 member1 25 member2
    # 2、增加某个人的分数（这里的分数就是排行的依据可以是浮点类型）
```

```
   ZINCRBY  testTop 20 member1   # 此时 testTop 中 member1 的分数就编程了 43
   ZINCRBY  testTop -10 member2  # 此时 testTop 中 member2 的分数就编程了 15
   ZINCRBY  testTop 20 member3   # 此时向 testTop 中添加了 member3 成员，分数为 20
   # 3、查询排行榜前两名，并且查询出其分数【WITHSCORES 选项用于显示分数，不带该参数则只会查出成员名称】
   ZREVRANGE testTop 0 1 WITHSCORES
   #结果：
   # 1) "member1"
   # 2) "43"
   # 3) "member3"
   # 4) "20"
   # 假设此时还有一个 排行榜
   ZADD testTop2  100 member2 200 member3 123 member4
   # 将 testTop testTop2 合成一个总榜 top
   ZUNIONSTORE  top 2 testTop testTop2
   # 查询总榜所有成员排行情况
   ZREVRANGE top 0 -1 WITHSCORES
   1) "member3"
   2) "220"
   3) "member4"
   4) "123"
   5) "member2"
   6) "115"
   7) "member1"
   8) "43"
```


Java 相关实现代码（模拟了 sf.gg 的名望榜）可以查看。

https://gitee.com/coderknock/Redis-Top-And-Around /src/test/java/TopDemo.java 有具体测试用例

Geo 相关功能

Redis 的 Geo 功能提供了查询两个成员距离、某个成员附近范围成员等功能可以用其实现一个简单的附近的人

Java 相关实现代码可以查看：

https://gitee.com/coderknock/Redis-Top-And-Around /src/test/java/GeoDemo.java 有具体测试用例。

缓存

原理：将经常会访问的数据根据一定规则设置一个 Key 后存入 Redis，每次查询时先查询 Redis 中是否包含匹配数据，如果缓存不存在再查询数据库。

注意点：对于不存在的数据应该存入一个自己设定的空值并设置过期时间，这样可以避免缓存击穿（由于数据不存在，所以设置 Key 对应的值为 null（Java中的表示形式），因为 Redis 会移除值为 null 的 key 这样会导致，每次查询还是会访问数据库）。

Java 相关实现代码可以查看：

https://gitee.com/coderknock/Redis-Cache

结束语

本文只是问了发散大家的思维，如对具体功能实现由兴趣可以在之后的交流中共同探讨。

由于个人的局限性，文中可能存在错误表述，大家可以在评论区中提出共同探讨。

附录

Redis环境搭建

在线体验： http://try.redis.io/

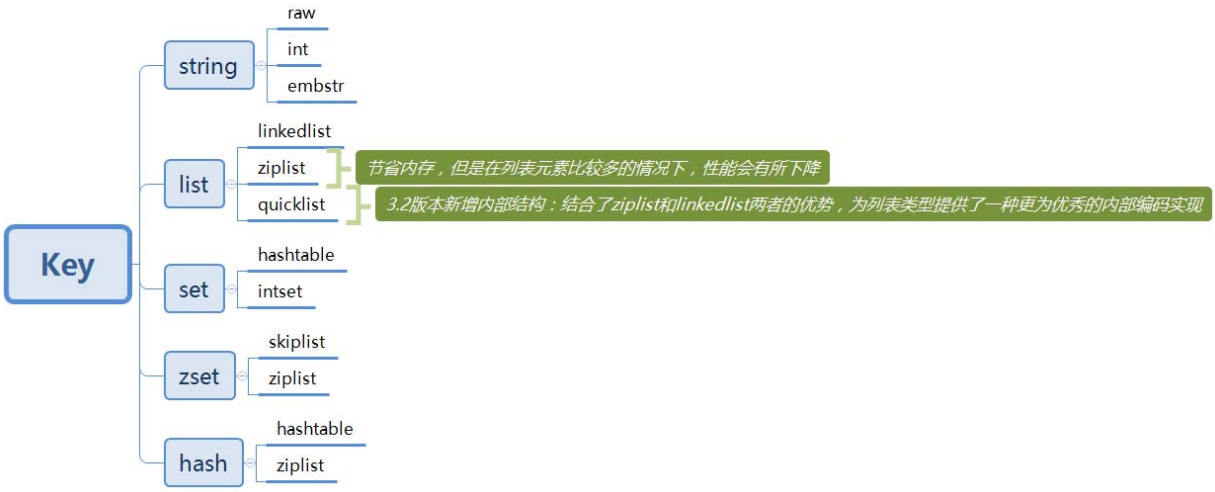
Windows版本： https://github.com/MSOpenTech/redis

Linux安装： https://www.coderknock.com/blog/2016/05/28/LinuxRedis.html

Redis 配置

https://www.coderknock.com/blog/2017/06/14/Redis%20%E9%85%8D%E7%BD%AE.html

Redis 支持的五大数据结构



Redis 基础知识扩展阅读

Redis 发布订阅图解