## 1.java集合

### List

List为接口，其实现类为ArrayList和LinkedList。

ArrayList底层结构为数组。LinkedList底层结构为链表。

#### 为什么不用数组

**java本身有数组，为什么还要用ArrayList呢？**

因为java数组在定义时必须要指定长度，相当于数组是定长的，如果初始化数组大小太大的话会浪费内存，太小了不够用。但ArrayList不同，它可以动态扩容。

#### 如何扩容

**ArrayList是如何实现动态扩容的？**

首先在new ArrayList()时，会默认生成一个长度为0的Object数组，都会去计算这个数组空间是否够用，第一次调用如果够用直接追加元素，如果不够就要扩容，扩容依靠的是ArrayList中的grow()方法，每次扩容为原来的1.5倍。第一次调用add()方法时，由于数组长度为0，会调用grow()将数组长度增加到10，之后每次扩容都为原来的1.5倍。

#### 为什么用的最多

**为什么ArrayList在日常开发用的最多？**

由于底层是数组，日常开发中，遍历比增删要多，即便是增删也是往往在尾部，时间复杂度也就O(1),另外ArrayList增删调用的copyOf()被优化过

#### Vector

**Vector也是List的实现类，理解吗？**

vector底层也是数组，现在用的比较少，相对于ArrayList,它是线程安全的，扩容的时候扩容2倍

#### 线程安全CopyOnWrite

**如果不用Vector，还有哪些线程安全的List？**

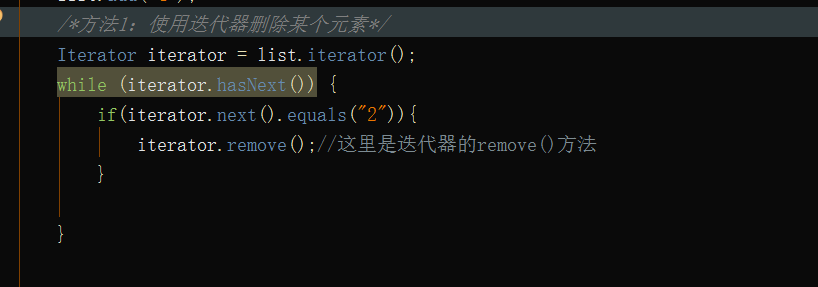
CopyOnWriteArrayList是线程安全的，CopyOnWrite容器很常见（通俗地讲，当我们往容器中添加一个元素的时候，不是直接添加，而是对当前容器copy，复制一个容器，在这个复制的容器中添加元素，添加完之后，再将引用指向这个新容器。）

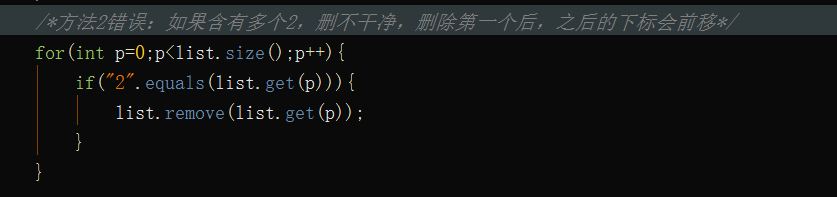
缺点：

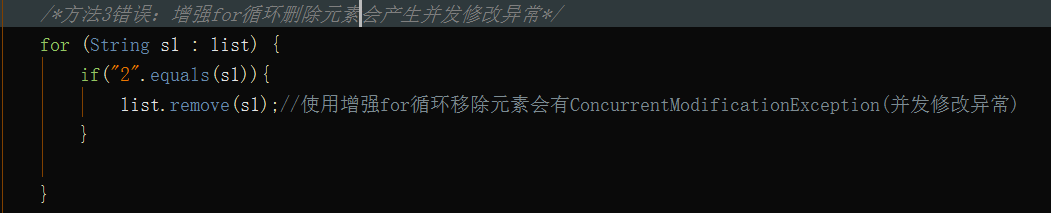
1.内存占用问题，产生了两个容器

2.只能保持数据的最终一致性，无法保持实时性

#### 移除元素Iterator

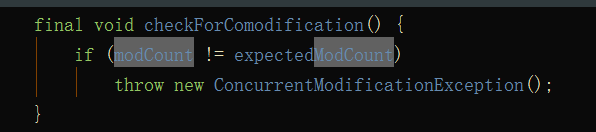






增强for修改元素报异常的原因：

在遍历时都会调用下面的checkForComodificaiton()方法，如果有元素被移除或者增加了就回导致modCount！=expectedMoCount，从而报异常。



### Map

Map是java中util包下的一个接口。常见的实现类有HashMap、LinkedHashMap、TreeMap、ConcurrentHashMap

#### 底层结构

哈希表是由数组+链表组成

HashMap、ConcurrentHashMap底层为数组+链表/红黑树

LinkedHashMap底层为数组+链表+双向链表

TreeMap底层结构为红黑树

#### HashMap初始值、负载因子

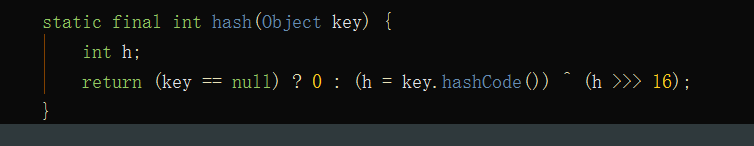
HashMap有几个构造方法，可以指定它的初始大小和负载因子。如果给参数，默认HashMap的大小为16，负载因子的大小为0.75；

public HashMap(int initialcapacity,float loadFactor){}为HashMap的有参构造器，参数分别为初始值大小和负载因子大小；如果初始值传的7，我们会将HashMap的初始大小定义为8（2次幂大小），是因为HashMap里采用的是**位运算**来代替取模，当我们想要把元素放进HashMap的时候，需要取出这个元素位置，这一步利用位运算效率更高。

负载因子：

它的大小决定的是哈希表的扩容和哈希冲突，比如我们默认的HaspMap的大小为16，这样最多能put12个元素，当put第13个元素时，就需要扩容，12/16=0.75；鉴于上面的位运算，HashMap的大小只能是2次幂，所以HashMap每次扩容倍数为原来的2倍

#### put元素时key如何计算哈希值



首先计算正常的hash值，然后与高16位做异或运算，产生最终的hash值，这样做可以减少hash冲突的概率。

#### HashMap什么时候用红黑树

当数组的大小大于64且链表大小大于8时，才会将链表改为红黑树；当红黑树大小为6时，会退化为链表。他们之间的转换主要是出于查询和插入的性能问题。

#### 线程安全的Map

juc包下的ConcurrentHashMap,它通过部分加锁和CAS算法来实现同步

### Set

## 2.java中抽象类和接口的区别

相同点：

1.都不能进行实例化

2.继承抽象类和实现接口都要对其中的抽象方法全部实现

区别：

接口都是公开抽象的方法，没有构造器；抽象类可以有普通方法，并且会有默认无参的构造器

## 3.redis

### linux配置

redis安装：

解压：tar –zxvf redis-5.0.7.tar.gz

将解压后的文件移动到/usr/local ： mv redis-5.0.7 /usr/local/

安装gcc环境：yum install gcc c++

安装redis：进入redis-5.0.7后执行make命令，之后执行make install

移动并修改redis配置文件：在/usr/local/ redis-5.0.7创建bin、etc：mkdir bin/mkdir etc

mv redis.conf /usr/local/ redis-5.0.7/etc

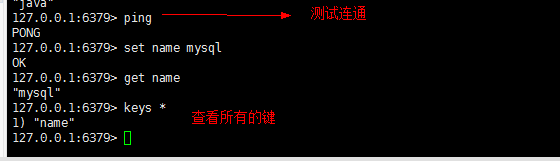
vim redis.conf后将daemonize调整为yes，代表后台运行

redis启动：

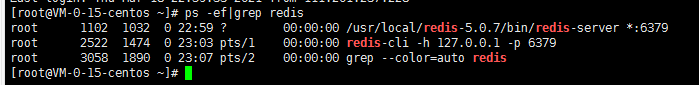
redis-server /usr/local/ edis-5.0.7/etc/redis.conf 通过指定的配置文件启动redis

redis-cli –h xxx -p 6379 启动客户端，并连接机器ip为xxx，端口号为6379的redis

连接测试：



查看redis进程：



注销redis服务：



### 分布式锁

## 4.spring

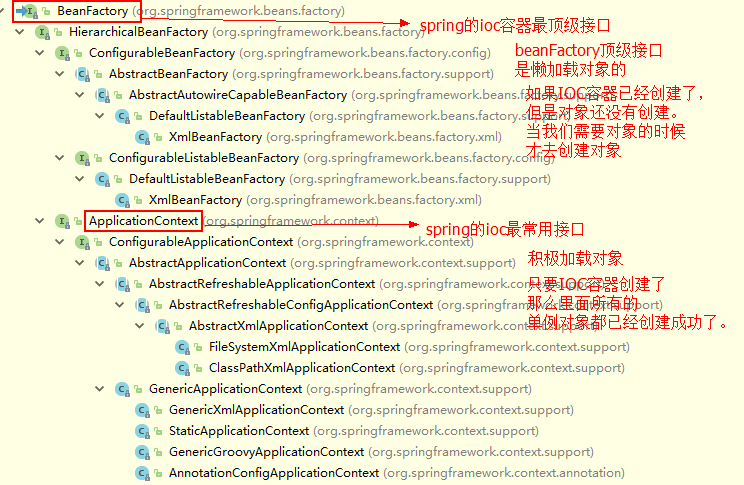
### IOC概念：

Inversion of Control控制反转,创建和管理对象的权利交给了IOC，当我们想要使用对象的时候就去IOC容器中取。

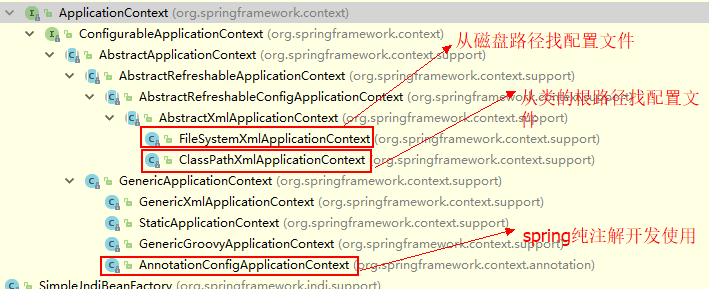
### Spring的四大核心包

beans、context、core、expression

### IOC常见接口



以及常用实现类



### 循环依赖

### bean的常用作用域

singleton:单例，当用scope设置为singleton时，加载spring配置时就会创建单例对象

prototype:多例，当scope设置为prototype时，在调用getBean()时才创建多例对象



### bean的生命周期

1.调用bean对象所属类的无参构造方法

2.调用类中的set方法进行属性注入

3.调用init方法进行对象初始化（在类中自定义initMethod）

4.得到bean，开始使用

5.销毁bean，在类中定义desMethod,调用ClassPathXmlApplicationContext的close方法

