http://outofmemory.cn/java/guava/cache/how-to-use-guava-cache

google guava中有cache包，此包提供内存缓存功能。内存缓存需要考虑很多问题，包括并发问题，缓存失效机制，内存不够用时缓存释放，缓存的命中率，缓存的移除等等。 当然这些东西guava都考虑到了。
guava中使用缓存需要先声明一个CacheBuilder对象，并设置缓存的相关参数，然后调用其build方法获得一个Cache接口的实例。请看下面的代码和注释，注意在注释中指定了Cache的各个参数。

```
public static void main(String[] args) throws ExecutionException,
InterruptedException{
            //缓存接口这里是LoadingCache，LoadingCache在缓存项不存在时可以自动加
载缓存
            LoadingCache<Integer,Student> studentCache
                //CacheBuilder的构造函数是私有的，只能通过其静态方法newBuilder()
来获得CacheBuilder的实例
                = CacheBuilder.newBuilder()
                //设置并发级别为8，并发级别是指可以同时写缓存的线程数
                .concurrencyLevel(8)
                //设置写缓存后8秒钟过期
                .expireAfterWrite(8, TimeUnit.SECONDS)
                //设置缓存容器的初始容量为10
                .initialCapacity(10)
                //设置缓存最大容量为100，超过100之后就会按照LRU最近虽少使用算法来
移除缓存项
                .maximumSize(100)
                //设置要统计缓存的命中率
                .recordStats()
                //设置缓存的移除通知
                .removalListener(
                      new RemovalListener<Object, Object>() {
                        @Override
                        public void
onRemoval(RemovalNotification<Object, Object> notification) {

System.out.println(notification.getKey()
                                        + " was removed, cause is " +
notification.getCause());
```

```
                    }
                })
                //build方法中可以指定CacheLoader，在缓存不存在时通过CacheLoader
的实现自动加载缓存
                .build(
                        new CacheLoader<Integer, Student>() {
                            @Override
                            public Student load(Integer key) throws
Exception {
                                System.out.println("load student "
+ key);
                                Student student = new Student();
                                student.setId(key);
                                student.setName("name " + key);
                                return student;
                            }
                        }
                );

        for (int i=0;i<20;i++) {
            //从缓存中得到数据，由于我们没有设置过缓存，所以需要通过CacheLoader加
载缓存数据
            Student student = studentCache.get(1);
            System.out.println(student);
            //休眠1秒
            TimeUnit.SECONDS.sleep(1);
        }

        System.out.println("cache stats:");
        //最后打印缓存的命中率等 情况
        System.out.println(studentCache.stats().toString());
    }
```

以上程序的输出如下：

```
load student 1
Student{id=1, name=name 1}
Student{id=1, name=name 1}
Student{id=1, name=name 1}
Student{id=1, name=name 1}
Student{id=1, name=name 1}
```

```
Student{id=1, name=name 1}
Student{id=1, name=name 1}
Student{id=1, name=name 1}
1 was removed, cause is EXPIRED
load student 1

......

Student{id=1, name=name 1}
Student{id=1, name=name 1}
Student{id=1, name=name 1}
Student{id=1, name=name 1}
cache stats:
CacheStats{hitCount=17, missCount=3, loadSuccessCount=3,
loadExceptionCount=0, totalLoadTime=1348802, evictionCount=2}
```

看看到在20此循环中命中次数是17次，未命中3次，这是因为我们设定缓存的过期时间是写入后的8秒，所以20秒内会失效两次，另外第一次获取时缓存中也是没有值的，所以才会未命中3次，其他则命中。

guava缓存过期时间分为两种，一种是从写入时开始计时，一种是从最后访问时间开始计时，而且guava缓存的过期时间是设置到整个一组缓存上的；这和EHCache，redis，memcached等不同，这些缓存系统设置都将缓存时间设置到了单个缓存上。guava缓存设计成了一组对象一个缓存实例，这样做的好处是一组对象设置一组缓存策略，你可以根据不同的业务来设置不同的缓存策略，包括弱引用，软引用，过期时间，最大项数等。另外一点好处是你可以根据不同的组来统计缓存的命中率，这样更有意义一些。
这样做也是有缺点的，缺点是首先是每个缓存组都需要声明不同的缓存实例，具体到业务程序中可能就是每个业务对象一个缓存了。这样就把不同的业务缓存分散到不同的业务系统中了，不太好管理。