

学习Google的guava--让代码飞翔起来

笔记本： java

创建时间： 2018\11\15 星期四 15:42

更新时间： 2018\11\15 星期四 21:09

作者： 804790605@qq.com

一、引言

Guava是一种基于开源的Java库，其中包含谷歌正在由他们很多项目使用的很多核心库。这个库是为了方便编码，并减少编码错误。这个库提供用于集合，缓存，支持原语，并发性，常见注解，字符串处理，I/O和验证的实用方法。

Guava工程包含了若干被Google的Java项目广泛依赖的核心库，例如：集合 [collections]、缓存 [caching]、原生类型支持 [primitives support]、并发库 [concurrency libraries]、通用注解 [common annotations]、字符串处理 [string processing]、I/O 等等。所有这些工具每天都在被Google的工程师应用在产品服务中。

依据guava的git上的说明，主要解决以上那些问题。网址是：<https://github.com/google/guava/wiki> 由于guava本身内容比较多，可以见下图，我这边只简单介绍一部分。

] | <https://github.com/google/guava/wiki>

淘宝 京东 天猫 系统之家 电影大全 IntelliJ git ancur 前端 springboot project tars JPA my 从E中导入 vue JWT jenkins bigdata

用户指南

第66页

Guava项目包含我们在基于Java的项目中依赖的几个Google核心库：集合，缓存，基元支持，并发库，通用注解，字符串处理，I/O等。Google员工每天都在生产服务中使用这些工具。

但是，通过Javadoc拖网并不总是学习如何充分利用图书馆的最有效方式。在这里，我们尝试为番石榴的一些最流行和最强大的功能提供可读和愉快的解释。

这个维基是一项正在进行的工作，其中一部分可能仍在建设中。

- 基本实用程序：使用Java语言更愉快。
 - [使用和避免null](#)：null可能不明确，可能导致混淆错误，有时只是简单的不愉快。许多Guava实用程序拒绝并在null上快速失败，而不是盲目地接受它们。
 - [前提条件](#)：更轻松地测试方法的前提条件。
 - [常用对象方法](#)：简化实现Object方法，如hashCode()和toString()。
 - [订购](#)：Guava强大的“流利Comparator”课程。
 - [Throwables](#)：简化传播和检查异常和错误。
- 集合：Guava对JDK集合生态系统的扩展。这些是番石榴最成熟和最受欢迎的部分。
 - [不可变的集合](#)，用于防御性编程，不断收集和提高效率。
 - [新的集合类型](#)，用于JDK集合无法解决的用例：多集，多图，表，双向映射等。
 - [强大的收集实用程序](#)，用于未提供的常见操作 java.util.Collections。
 - [扩展实用程序](#)：编写Collection装饰器？实施Iterator？我们可以让这更容易。
- [图](#)：用于建模图形结构化数据的库，即实体及它们之间的关系。主要功能包括：
 - [图表](#)：一个图表，其边缘是匿名实体，没有自己的身份或信息。
 - [ValueGraph](#)：一个图形，其边缘具有关联的非唯一值。
 - [网络](#)：边缘为唯一对象的图形。

- [介绍](#)
- 基本公用事业
 - [使用/避免null](#)
 - [可选的](#)
 - [前提条件](#)
 - [订购](#)
 - [创建](#)
 - [链接](#)
 - [应用](#)
 - [对象方法](#)
 - [等于](#)
 - [的hashCode](#)
 - [比较/的compareTo](#)
 - [将Throwable](#)
- 集合
 - [不可变的收藏品](#)
 - [新的集合类型](#)
 - [多集](#)
 - [Multimap之](#)
 - [BIMAP](#)
 - [表](#)
 - [ClassToInstanceMap](#)
 - [RangeSet](#)
 - [RangeMap](#)
 - [实用程序类](#)
 - [Iterables](#)
 - [清单](#)

- **网络**：边缘为唯一对象的图形。
- 支持可变和不可变，有向和无向的图形以及其他几个属性。
- **缓存**：本地缓存，完成正确，并支持各种过期行为。
- **功能习语**：谨慎使用，Guava的功能习语可以显着简化代码。
- **并发**：功能强大，简单的抽象，使编写正确的并发代码变得更容易。
 - **ListenableFuture**：期货，完成后回调。
 - **服务**：启动和关闭的事情，为您处理困难的状态逻辑。
- **字符串**：一些非常有用的字符串实用程序：拆分，连接，填充等。
- **原语**：在原始类型，如操作 `int` 和 `char`，而不是由JDK提供的，包括某些类型的无符号的变种。
- **范围**：Guava强大的API，用于处理 `Comparable` 类型的范围，包括连续和离散。
- **I / O**：简化的I / O操作，尤其适用于Java 5和6的整个I / O流和文件。
- **哈希**：用于比 `Object.hashCode()` 包括Bloom过滤器提供的更复杂哈希的工具。
- **EventBus**：组件之间的发布 - 订阅式通信，无需组件明确地相互注册。
- **数学**：JDK未提供的经过优化，经过全面测试的数学实用程序。
- **反思**：Java的反射功能的番石榴实用程序。
- **提示**：使用Guava以您希望的方式运行应用程序。
 - **哲学**：番石榴是什么和不是什么，以及我们的目标。
 - **在构建中使用Guava**，使用包括Maven，Gradle等构建系统。
 - **使用ProGuard**避免捆绑Guava的部分，而不是与JAR一起使用。
 - **Apache Commons等价物**，帮助您使用Apache Commons Collections翻译代码。
 - **兼容性**，Guava版本之间的细节。
 - **Idea Graveyard**，功能请求已被最终拒绝。
 - **朋友**，我们喜欢和欣赏的开源项目。
 - **HowToContribute**，如何为Guava做贡献。

注意：要讨论此Wiki的内容，请使用guava-discuss邮件列表。

- Iterables
- 清单
- 集
- 地图
- 多集
- 屈德宁
- 表
- 扩展实用程序
 - 转发装饰器
 - PeekingIterator
 - AbstractIterator
- 图表
 - 定义
 - 功能
 - 图表类型
 - 图形
 - ValueGraph
 - 网络
 - 构建图形实例
 - 构建器约束与优化提示
 - 可变和不可变的图
 - 可变*类型
 - 不可变的*实现
 - 图形元素（节点和边）
 - 图书馆合同和行为
 - 突变
 - equals（），hashCode（）和图等价
 - 访问方法
 - 同步
 - 元素对象

二、快速使用

a、环境设置

1、**jdk环境设置**，只要熟悉java开发的，基本都会，不会的也可以百度一下jdk的path、classpath、javahome的配置，应该很快就能百度出来，并且能够有很多。

2、Guava设置，

下载guava.jar，并且放入项目，或者配置到系统环境中，当然我是直接使用maven管理的

```
<!-- https://mvnrepository.com/artifact/com.google.guava/guava -->
<dependency>
  <groupId>com.google.guava</groupId>
  <artifactId>guava</artifactId>
  <version>27.0-jre</version>
</dependency>
```

3、测试

话不多说，先写个例子用一用再说。

（很多人写正确的代码很厉害，但是错误的代码不会写，个人觉得牛逼的人应该是不仅正确的代码顺手拈来，错误的代码例子也能够轻而易举的写出来）

下面写一个我们经常写代码觉得很烦的基础封装类null值得问题代码。

```

public static void test1(){
    GuavaTest1 guavaTester = new GuavaTest1();
    Integer a = null;
    Integer b = new Integer(10);
    System.out.println(guavaTester.sum(a,b));
}

public Integer sum(Integer a, Integer b){
    return a + b;
}

```

得到的结果是：

```

D:\ProgramFiles\Java\jdk1.8.0_181\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:49294', transport: 'socket'
Exception in thread "main" java.lang.NullPointerException
    at main.zf.guavalearn.GuavaTest1.sum(GuavaTest1.java:22)
    at main.zf.guavalearn.GuavaTest1.test1(GuavaTest1.java:18)
    at main.zf.guavalearn.GuavaTest1.main(GuavaTest1.java:11)
Disconnected from the target VM, address: '127.0.0.1:49294', transport: 'socket'

```

很难受是不是，大部分情况下，我们都是先判断这个值是否是null，自己写代码太难受了，用一下guava的optional，再写个代码看看。

```

public static void test1Guava(){
    GuavaTest1 guavaTester = new GuavaTest1();
    Integer invalidInput = null;
    Optional<Integer> a = Optional.of(invalidInput);
    Optional<Integer> b = Optional.of(new Integer(10));
    System.out.println(guavaTester.sum(a,b));
}

public Integer sum(Optional<Integer> a, Optional<Integer> b){
    return a.get() + b.get();
}

```

得到的结果是：

```

D:\ProgramFiles\Java\jdk1.8.0_181\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:50122', transport: 'socket'
Exception in thread "main" java.lang.NullPointerException
    at com.google.common.base.Preconditions.checkNotNull(Preconditions.java:877)
    at com.google.common.base.Optional.of(Optional.java:103)
    at main.zf.guavalearn.GuavaTest1.test1Guava(GuavaTest1.java:30)
    at main.zf.guavalearn.GuavaTest1.main(GuavaTest1.java:13)
Disconnected from the target VM, address: '127.0.0.1:50122', transport: 'socket'

```

可以看到，optional会先帮我们处理null的情况，以免我们自己忘记处理。

三、详细了解

guava就是一个耐看的美女，发现越深入了解，越觉得guava美丽。

1、String

为什么要先研究String，因为String是一个好东西，个人觉得java要是没有String，那么java不可能应用这么广泛，String在java里面占了半壁江山。

比如针对String的对象判断null和空的问题，代码如下：

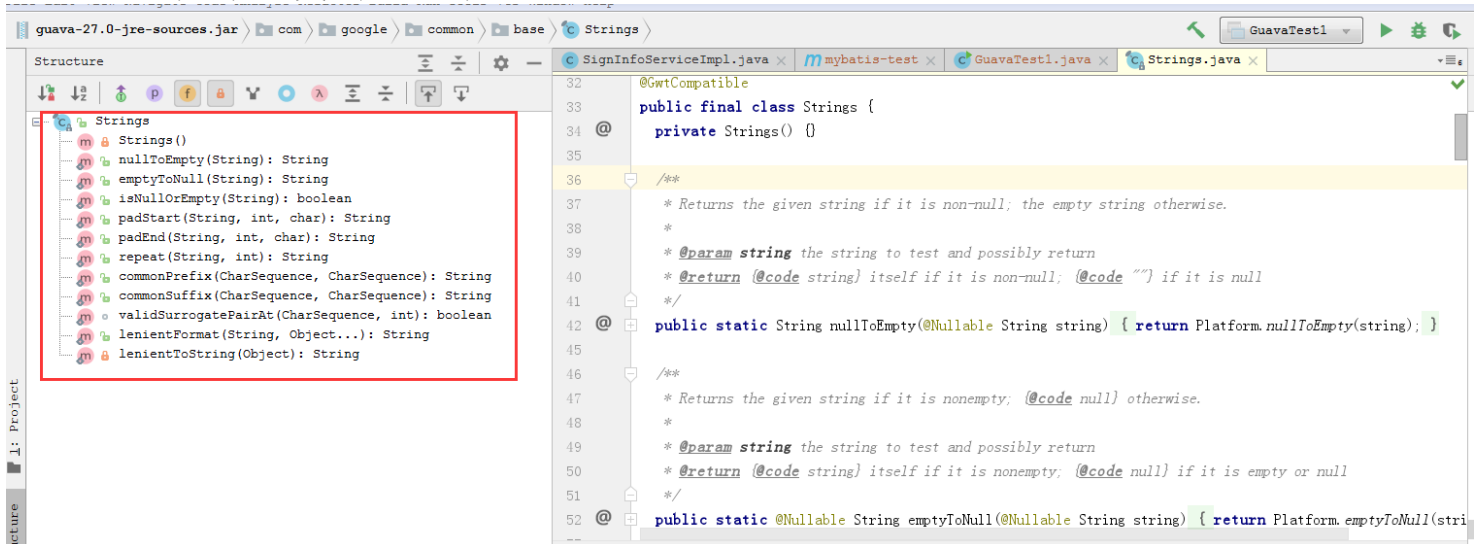
```
public void javaString(String input){
    if( input==null || input.equals("") ){
        System.out.println("输入字符串为空");
    }
}

public void guavaString(String input){
    if(Strings.isNullOrEmpty(input)){
        System.out.println("输入字符串为空");
    }
}
```

当然Strings这个工具类非常好用，个人觉得里面还有一个非常实用的方法是：

`Strings.commonSuffix("aac", "aac");` // 该方法返回两个字符串的最长公共后缀。

查看源码可以看见该类的方法有以下这些：



具体的使用代码，我就不多展示，因为看方法命名基本就知道是干嘛用的，当然也可以进入方法，看源码是怎么实现这些功能的。

2、基础工具包

a、Optional

guava的Optional类似于Java 8新增的Optional类，都是用来处理null的，不过guava的是抽象类，其实实现类为Absent和Present，而java.util的是final类。其中一部分方法名是相同的。

Guava用Optional表示可能为null的T类型引用。一个Optional实例可能包含非null的引用（我们称之为引用存在），也可能什么也不包括（称之为引用缺失）。它从不说包含的是null值，而是用存在或缺失来

表示。但Optional从不会包含null值引用。

例子如下：

```
public static void test3() {
    Integer value1 = null;
    Integer value2 = 10;
    /*创建指定引用的Optional实例，若引用为null则快速失败返回absent()
    absent()创建引用缺失的Optional实例*/
    Optional<Integer> a = Optional.fromNullable(value1);
    Optional<Integer> b = Optional.of(value2); //返回包含给定的非空引用Optional实例
    System.out.println(sumInteger(a,b));
}

private static Integer sumInteger(Optional<Integer> a,Optional<Integer> b){
    //isPresent():如果Optional包含非null的引用（引用存在），返回true
    System.out.println("First param is present: " + a.isPresent());
    System.out.println("Second param is present: " + b.isPresent());
    Integer value1 = a.or(0); //返回Optional所包含的引用,若引用缺失,返回指定的值
    Integer value2 = b.get(); //返回所包含的实例,它必须存在,通常在调用该方法时会调用isPresent()判断是否为null
    return value1 + value2;
}
```

结果是：

```
D:\ProgramFiles\Java\jdk1.8.0_181\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:53906', transport: 'socket'
Disconnected from the target VM, address: '127.0.0.1:53906', transport: 'socket'
ERROR: JDWP Unable to get JNI 1.2 environment, jvm->GetEnv() return code = -2
JDWP exit error AGENT_ERROR_NO_JNI_ENV(183): [util.c:840]
First param is present: false
Second param is present: true
10
```

b、Preconditions

前置条件Preconditions提供静态方法来检查方法或构造函数，被调用是否给定适当的参数。它检查的先决条件。其方法失败抛出IllegalArgumentException。

例子如下：

```
package main.zf.guavalearn;

import com.google.common.base.Preconditions;

/**
 * @Author: Administrator
 * @Date: 2018\11\15 0015 17:09
 * @Description:
```

```

*/
public class PreconditionsDemo {
    public static void main(String[] args) {
        try {
            getValue(5);
        } catch (IndexOutOfBoundsException e){
            System.out.println(e.getMessage());
        }

        try {
            sum(4,null);
        } catch (NullPointerException e){
            System.out.println(e.getMessage());
        }

        try {
            sqrt(-1);
        } catch (IllegalArgumentException e){
            System.out.println(e.getMessage());
        }

    }

    private static double sqrt(double input){
        Preconditions.checkNotNull(input>0.0,
            "Illegal Argument passed: Negative value %s.",input);
        return Math.sqrt(input);
    }

    private static int sum(Integer a,Integer b){
        a= Preconditions.checkNotNull(a,
            "Illegal Argument passed: First parameter is Null.");
        b=Preconditions.checkNotNull(b,
            "Illegal Argument passed: First parameter is Null.");
        return a+b;
    }

    private static int getValue(int input){
        int[] data={1,2,3,4,5};
        int index=Preconditions.checkNotNull(input,data.length,
            "Illegal Argument passed: Invalid index.");
        return data[index];
    }
}

```

得到的结果是：


```
D:\ProgramFiles\Java\jdk1.8.0_181\bin\java.exe ...
```

```
Connected to the target VM, address: '127.0.0.1:54294', transport: 'socket'  
Illegal Argument passed: Invalid index. (5) must be less than size (5)  
Illegal Argument passed: First parameter is Null.  
Illegal Argument passed: Negative value -1.0.  
Disconnected from the target VM, address: '127.0.0.1:54294', transport: 'socket'
```

c、Joiner

Joiner 提供了各种方法来处理字符串加入操作，对象等。

Joiner的实例不可变的，因此是线程安全的。

```
public static void testJoiner(){  
    /*  
    on:制定拼接符号，如：test1-test2-test3 中的 "- " 符号  
    skipNulls()：忽略NULL,返回一个新的Joiner实例  
    useForNull( "Hello" )：NULL的地方都用字符串" Hello" 来代替  
    */  
    StringBuilder sb=new StringBuilder();  
    //skipNulls() does nothing!分开写跳过null就不起作用了，因为实例不可改变  
    Joiner.on(",").skipNulls().appendTo(sb,"Hello","guava");  
    System.out.println(sb);  
    System.out.println(Joiner.on(",").useForNull("none").join(1,null,3));  
    System.out.println(Joiner.on(",").skipNulls().join(Arrays.asList(1,2,3,4,null,6)));  
    Map<String,String> map=new HashMap<>();  
    map.put("key1","value1");  
    map.put("key2","value2");  
    map.put("key3","value3");  
    System.out.println(Joiner.on(",").withKeyValueSeparator("=").join(map));  
}
```

结果是：

```
D:\ProgramFiles\Java\jdk1.8.0_181\bin\java.exe ...
```

```
Connected to the target VM, address: '127.0.0.1:54812', transport: 'socket'  
Disconnected from the target VM, address: '127.0.0.1:54812', transport: 'socket'  
Hello, guava  
1, none, 3  
1, 2, 3, 4, 6  
key1=value1, key2=value2, key3=value3
```

d、Splitter

Splitter 能够将一个字符串按照指定的分隔符拆分成可迭代遍历的字符串集合，Iterable

```
public static void testSplitter(){  
    /*
```

on():指定分隔符来分割字符串

limit():当分割的子字符串达到了*limit*个时则停止分割

fixedLength():根据长度来拆分字符串

trimResults():去掉子串中的空格

omitEmptyStrings():去掉空的子串

withKeyValueSeparator():要分割的字符串中*key*和*value*间的分隔符,分割后的子串中*key*和*value*间的分隔符默认是=

*/

```
System.out.println(Splitter.on(",").limit(3).trimResults().split(" a, b, c, d"));//[ a, b, c,d]
```

```
System.out.println(Splitter.fixedLength(3).split("1 2 3"));//[1 2, 3]
```

```
System.out.println(Splitter.on(" ").omitEmptyStrings().splitToList("1 2 3"));
```

```
System.out.println(Splitter.on(",").omitEmptyStrings().split("1,,,2,,,3"));//[1, 2, 3]
```

```
System.out.println(Splitter.on(" ").trimResults().split("1 2 3")); //[1, 2, 3],默认的连接符是,
```

```
System.out.println(Splitter.on(";").withKeyValueSeparator(":").split("a:1;b:2;c:3"));://{a=1,  
b=2, c=3}
```

```
}
```

结果是：

```
D:\ProgramFiles\Java\jdk1.8.0_181\bin\java.exe ...
```

```
Connected to the target VM, address: '127.0.0.1:57389', transport: 'socket'
```

```
[a, b, c, d]
```

```
[1 2, 3]
```

```
[1, 2, 3]
```

```
[1, 2, 3]
```

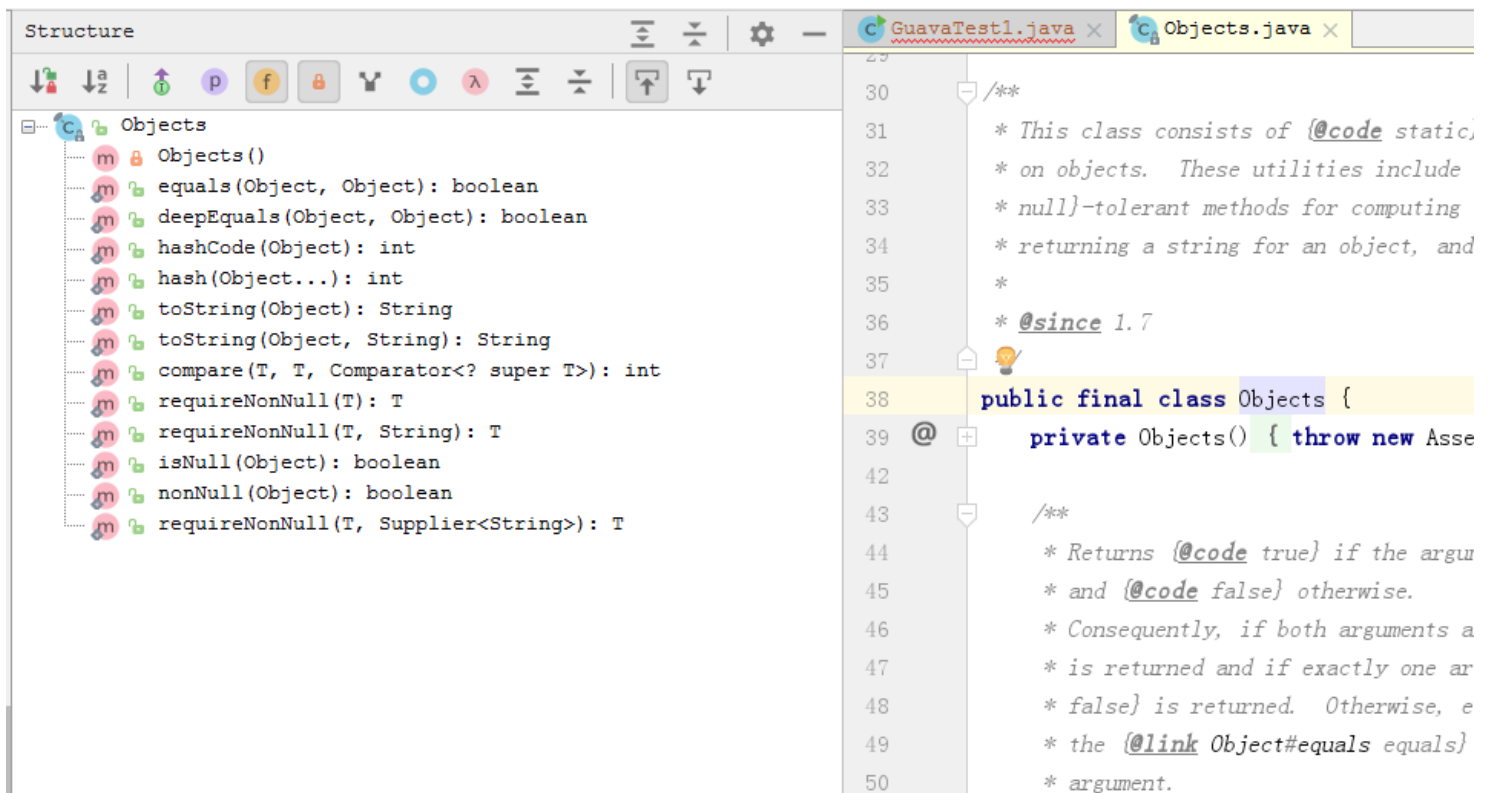
```
[1, 2, 3]
```

```
{a=1, b=2, c=3}
```

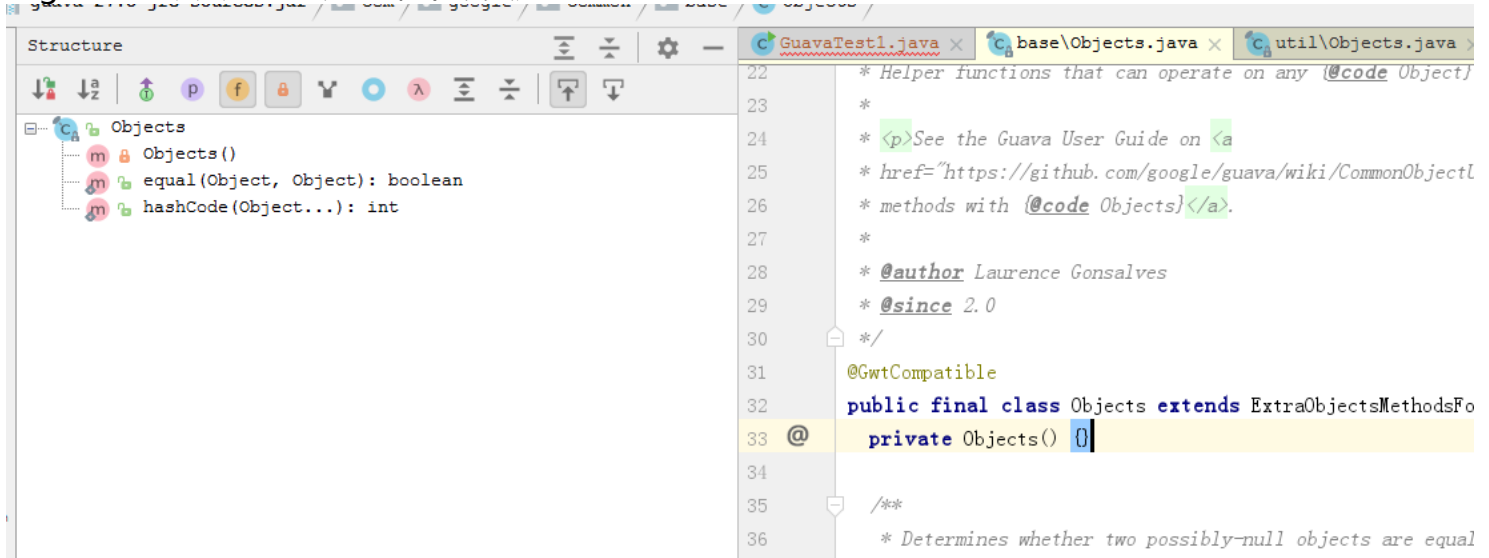
```
Disconnected from the target VM, address: '127.0.0.1:57389', transport: 'socket'
```

e、Objects

java7及以后的版本建议使用jdk中的Objects类，已经很好用了。



而guava的就只有两个方法，不够实用。



f、 EventBus

Guava为我们提供了事件总线EventBus库，它是事件发布-订阅模式的实现，让我们能在领域驱动设计(DDD)中以事件的弱引用本质对我们的模块和领域边界很好的解耦设计。

Guava为我们提供了同步事件EventBus和异步实现AsyncEventBus两个事件总线，他们都不是单例的。Guava发布的事件默认不会处理线程安全的，但我们可以标注@AllowConcurrentEvents来保证其线程安全

如果Listener A监听Event A, 而Event A有一个子类Event B, 此时Listener A将同时接收Event A和B消息话不多说，直接上代码：

1、 事件

```
package main.zf.guavalearn;
```

```
/**
 * @Author: Administrator
```

```

* @Date: 2018\11\15 0015 18:38
* @Description:
*///Guava 发布-订阅模式中传递的事件,是一个普通的POJO类
public class OrderEvent { //事件
private String message;

public OrderEvent(String message) {
this.message = message;
}

public String getMessage() {
return message;
}
}

```

2、订阅

```

package main.zf.guavalearn;

import com.google.common.eventbus.Subscribe;

/**
 * 订阅者
 * @Author: Administrator
 * @Date: 2018\11\15 0015 18:39
 * @Description:
 */
public class EventListener {

    //@Subscribe保证有且只有一个输入参数,如果你需要订阅某种类型的消息,只需要在指定的方法上加上@Subscribe注解即可
    @Subscribe
    public void listen(OrderEvent event){
        System.out.println("EventListener receive message: "+event.getMessage());
    }

    /**
     * 一个subscriber也可以同时订阅多个事件
     * Guava会通过事件类型来和订阅方法的形参来决定到底调用subscriber的哪个订阅方法
     */
    @Subscribe
    public void listen(String message){
        System.out.println("EventListener receive message: "+message);
    }
}

```

3、多个订阅者

```

package main.zf.guavalearn;

```

```

import com.google.common.eventbus.Subscribe;

/**
 * @Author: Administrator
 * @Date: 2018\11\15 0015 18:41
 * @Description: 多个订阅者
 */
public class MultiEventListener {
    @Subscribe
    public void listen(OrderEvent event){
        System.out.println("MultiEventListener receive msg: "+event.getMessage());
    }

    @Subscribe
    public void listen(String message){
        System.out.println("MultiEventListener receive msg: "+message);
    }
}

```

4、测试类

```

package main.zf.guavalearn;

import com.google.common.eventbus.EventBus;

/**
 * @Author: Administrator
 * @Date: 2018\11\15 0015 18:43
 * @Description:
 */
public class EventBusDemo {
    public static void main(String[] args) {
        EventBus eventBus=new EventBus("jack");
        /*
        如果多个subscriber订阅了同一个事件,那么每个subscriber都将收到事件通知
        并且收到事件通知的顺序跟注册的顺序保持一致
        */
        eventBus.register(new EventListener()); //注册订阅者
        eventBus.register(new MultiEventListener());
        eventBus.post(new OrderEvent("hello")); //发布事件
        eventBus.post(new OrderEvent("world"));
        eventBus.post("!");
    }
}

```

5、得到的结果是：

```
D:\ProgramFiles\Java\jdk1.8.0_181\bin\java.exe ...
```

```
Connected to the target VM, address: '127.0.0.1:50258', transport: 'socket'
EventListener receive message: hello
MultiEventListener receive msg: hello
EventListener receive message: world
MultiEventListener receive msg: world
EventListener receive message: !
MultiEventListener receive msg: !
Disconnected from the target VM, address: '127.0.0.1:50258', transport: 'socket'
```

6、Dead Event

```
package main.zf.guavalearn;

import com.google.common.eventbus.DeathEvent;
import com.google.common.eventbus.Subscribe;

/**
 * @Author: Administrator
 * @Date: 2018\11\15 00:15 18:47
 * @Description: 如果EventBus发送的消息都不是订阅者关心的称之为Dead Event。
 */
public class DeathEventListener {
    boolean isDelivered=true;

    @Subscribe
    public void listen(DeathEvent event){
        isDelivered=false;
        System.out.println(event.getSource().getClass()+" "+event.getEvent()); //source通常是
        EventBus
    }

    public boolean isDelivered() {
        return isDelivered;
    }
}
```

3、Collection

1、不可变集合

1)、不可变对象有很多优点，包括：

- 当对象被不可信的库调用时，不可变形式是安全的；
- 不可变对象被多个线程调用时，不存在竞态条件问题
- 不可变集合不需要考虑变化，因此可以节省时间和空间。所有不可变的集合都比它们的可变形式有更好的内存利用率（分析和测试细节）；
- 不可变对象因为有固定不变，可以作为常量来安全使用。

2)、JDK也提供了Collections.unmodifiableXXX方法把集合包装为不可变形式，但：

- 笨重而且累赘：不能舒适地用在所有想做防御性拷贝的场景；
- 不安全：要保证没人通过原集合的引用进行修改，返回的集合才是事实上不可变的；
- 低效：包装过的集合仍然保有可变集合的开销，比如并发修改的检查、散列表的额外空间，等等。

3)、创建不可变集合方法：

- copyOf方法，如ImmutableSet.copyOf(set);
- of方法，如ImmutableSet.of("a" , "b" , "c")或 ImmutableMap.of("a" , 1, "b" , 2);
- Builder工具

代码例子如下：

```
public static void testImmutable(){
    ImmutableSet<String> set=ImmutableSet.of("a","b","c","d");
    System.out.println(set);
    ImmutableSet<String> set1=ImmutableSet.copyOf(set);
    System.out.println(set);
    ImmutableSet<String> set2=ImmutableSet.<String> builder().addAll(set).add("e").build();
    System.out.println(set);
    ImmutableList<String> list=set.asList();
    System.out.println(list);
}
```

结果如下：

```
D:\ProgramFiles\Java\jdk1.8.0_181\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:51587', transport: 'socket'
[a, b, c, d]
[a, b, c, d]
[a, b, c, d]
[a, b, c, d]
Disconnected from the target VM, address: '127.0.0.1:51587', transport: 'socket'
```

2、新型集合类

S.N.	集合名称和说明
1	<u>Multiset</u> 一个扩展来设置界面，允许重复的元素。
2	<u>Multimap</u> 一个扩展来映射接口，以便其键可一次被映射到多个值
3	<u>BiMap</u> 一个扩展来映射接口，支持反向操作。
4	<u>Table</u> 表代表一个特殊的图，其中两个键可以在组合的方式被指定为单个值。

1) 、 Multiset

Multiset可统计一个词在文档中出现了多少次

例子代码如下：

```
public static void testMultiset(){
    Multiset<String> set= LinkedHashMultiset.create();
    set.add("a");
    set.add("a");
    set.add("a");
    set.add("a");
    set.add("b");
    set.add("b");
    set.add("c");
    set.add("c");
    set.add("c");
    set.add("d");
    System.out.println(set);
    set.setCount("a",5); //添加或删除指定元素使其在集合中的数量是count
    System.out.println(set);
    System.out.println(set.count("a")); //给定元素在Multiset中的计数
    System.out.println(set.count("b")); //给定元素在Multiset中的计数
    System.out.println(set);
    System.out.println(set.size()); //所有元素计数的总和,包括重复元素
    System.out.println(set.elementSet().size()); //所有元素计数的总和,不包括重复元素
    set.clear(); //清空集合
    System.out.println(set);
}
```

结果是：

```
D:\ProgramFiles\Java\jdk1.8.0_181\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:52209', transport: 'socket'
[a x 4, b x 2, c x 3, d]
[a x 5, b x 2, c x 3, d]
5
2
[a x 5, b x 2, c x 3, d]
11
4
[]
```

2) 、 Multimap

Multimap可以很容易地把一个键映射到多个值。换句话说，Multimap是把键映射到任意多个值的一般方式。

示例代码如下：

```
public static void testMutimap(){
```



```

Multimap<String,Integer> map= HashMultimap.create(); //Multimap是把键映射到任意多个值
的一般方式
map.put("a",1); //key相同时不会覆盖原value
map.put("a",2);
map.put("a",3);
System.out.println(map); //{a=[1, 2, 3]}
System.out.println(map.get("a")); //返回的是集合
System.out.println(map.size()); //返回所有"键-单个值映射"的个数,而非不同键的个数
System.out.println(map.keySet().size()); //返回不同key的个数
//可以解决如下代码
Map<String, Collection<Integer>> mapView=map.asMap();
}

```

运行结果如下：

```

D:\ProgramFiles\Java\jdk1.8.0_181\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:52563', transport: 'socket'
{a=[1, 2, 3]}
[1, 2, 3]
3
1
Disconnected from the target VM, address: '127.0.0.1:52563', transport: 'socket'

```

3)、BiMap

BiMap是一种特殊的映射其保持映射，同时确保没有重复的值是存在于该映射和一个值可以安全地用于获取键背面的倒数映射。

示例代码如下：

```

public static void testBimap(){
BiMap<String,String> biMap= HashBiMap.create();
biMap.put("sina","sina.com");
biMap.put("qq","qq.com");
biMap.put("sina","sina.cn"); //会覆盖原来的value
/*
在BiMap中,如果你想把键映射到已经存在的值, 会抛出IllegalArgumentException异常
如果对特定值,你想要强制替换它的键, 请使用 BiMap.forcePut(key, value)
*/
biMap.put("tencent","qq.com"); //抛出异常
biMap.forcePut("tencent","qq.com"); //强制替换key
System.out.println(biMap);
System.out.println(biMap.inverse().get("sina.com")); //通过value找key
System.out.println(biMap.inverse().inverse()==biMap); //true
}

```

运行结果如下：

```
D:\ProgramFiles\Java\jdk1.8.0_181\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:52825', transport: 'socket'
Exception in thread "main" java.lang.IllegalArgumentException: value already present: qq.com
    at com.google.common.collect.HashBiMap.put(HashBiMap.java:279)
    at com.google.common.collect.HashBiMap.put(HashBiMap.java:260)
    at main.zf.guavalearn.GuavaTest1.testBimap(GuavaTest1.java:140)
    at main.zf.guavalearn.GuavaTest1.main(GuavaTest1.java:148)
Disconnected from the target VM, address: '127.0.0.1:52825', transport: 'socket'
```

这个类里的方法有：

S.N.	方法及说明
1	V forcePut(K key, V value) 另一种put的形式是默默删除，在put(K, V)运行前的任何现有条目值。
2	BiMap<V,K> inverse() 返回此bimap，每一个bimap的值映射到其相关联的键的逆视图。
3	V put(K key, V value) 关联指定值与此映射中(可选操作)指定的键。
4	void putAll(Map<? extends K,? extends V> map) 将所有从指定映射此映射(可选操作)的映射。
5	Set<V> values() 返回此映射中包含Collection的值视图。

4)、Table

Table它有两个支持所有类型的键：“行”和“列”。Table代表一个特殊的映射，其中两个键可以在组合的方式被指定为单个值。它类似于创建映射的映射。

示例代码如下：

```
public static void testTable(){
    //记录学生在某门课上的成绩
    Table<String,String,Integer> table= HashBasedTable.create();
    table.put("jack","java",100);
    table.put("jack","c",90);
    table.put("mike","java",93);
    table.put("mike","c",100);
    System.out.println(table);
    Set<Table.Cell<String,String,Integer>> cells=table.cellSet();
    for (Table.Cell<String,String,Integer> cell : cells) {
        System.out.println(cell.getRowKey()+" "+cell.getColumnKey()+" "+cell.getValue());
    }
    System.out.println(table.row("jack"));
    System.out.println(table);
    System.out.println(table.rowKeySet());
}
```

```

System.out.println(table.columnKeySet());
System.out.println(table.values());
}

```

运行结果如下：

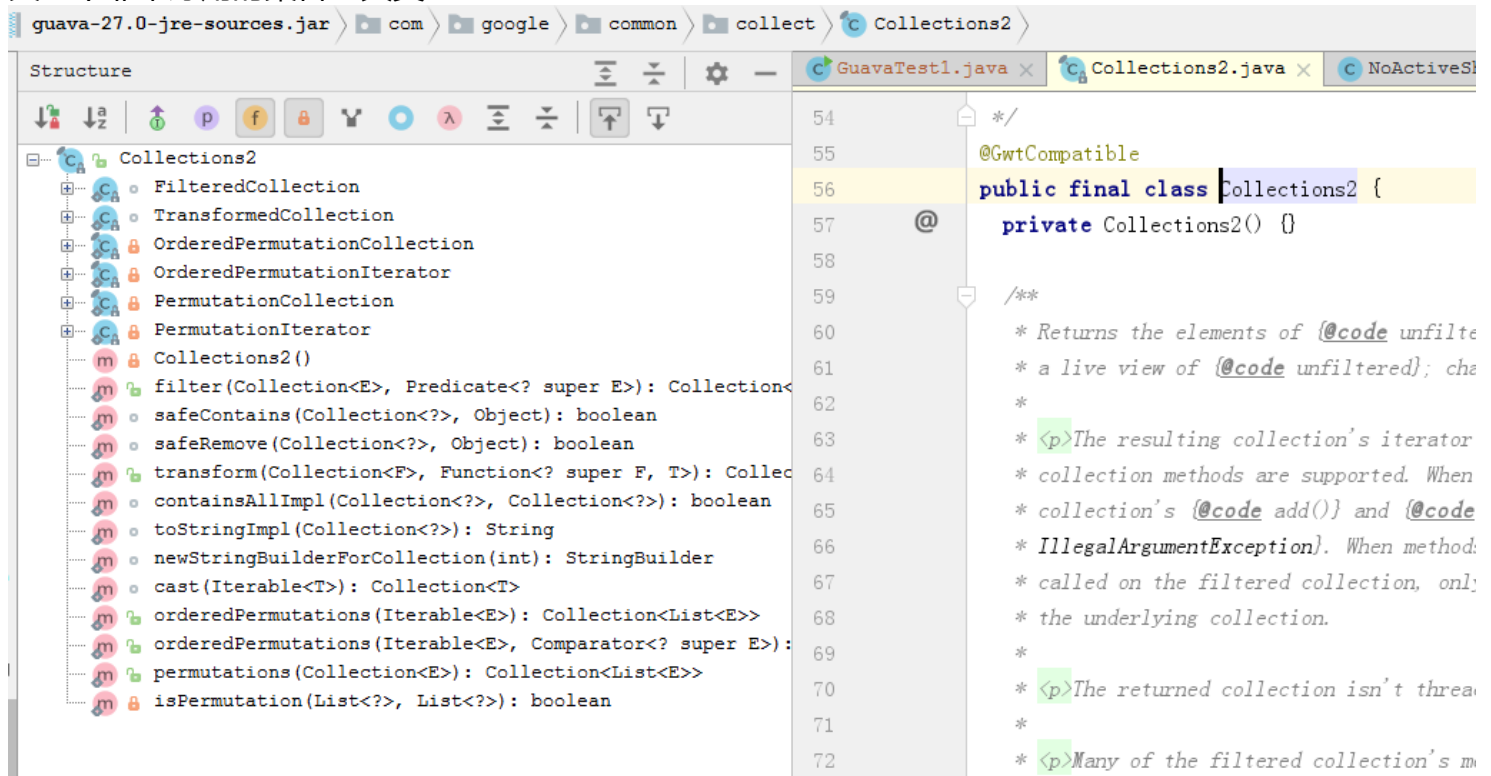
```

D:\ProgramFiles\Java\jdk1.8.0_181\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:53728', transport: 'socket'
{jack={java=100, c=90}, mike={java=93, c=100}}
jack java 100
jack c 90
mike java 93
mike c 100
{java=100, c=90}
{jack={java=100, c=90}, mike={java=93, c=100}}
[jack, mike]
[java, c]
[100, 90, 93, 100]

```

5)、Collections2

又一个非常好用的集合工具类



The screenshot shows an IDE with the 'Collections2' class selected in the 'Structure' view on the left. The main editor displays the source code of 'Collections2.java'. The code includes a package declaration, a GWT-compatible annotation, and a public final class 'Collections2' with a private constructor. It also contains several Javadoc comments describing the class's purpose and usage.

```

guava-27.0-jre-sources.jar > com > google > common > collect > Collections2
Structure
Collections2
  FilteredCollection
  TransformedCollection
  OrderedPermutationCollection
  OrderedPermutationIterator
  PermutationCollection
  PermutationIterator
  Collections2()
  filter(Collection<E>, Predicate<? super E>): Collection<E>
  safeContains(Collection<?>, Object): boolean
  safeRemove(Collection<?>, Object): boolean
  transform(Collection<F>, Function<? super F, T>): Collection<T>
  containsAllImpl(Collection<?>, Collection<?>): boolean
  toStringImpl(Collection<?>): String
  newStringBuilderForCollection(int): StringBuilder
  cast(Iterable<T>): Collection<T>
  orderedPermutations(Iterable<E>): Collection<List<E>>
  orderedPermutations(Iterable<E>, Comparator<? super E>): Collection<List<E>>
  permutations(Collection<E>): Collection<List<E>>
  isPermutation(List<?>, List<?>): boolean

54  */
55  @GwtCompatible
56  public final class Collections2 {
57      @
58      private Collections2() {}
59
60      /**
61       * Returns the elements of {@code unfiltered} as a live view of
62       * the elements of {@code unfiltered}; changes to the underlying
63       * collection are reflected in the view. The resulting collection's
64       * iterator supports the {@code Collection<E>#iterator()} method.
65       * When the {@code add()} method is called on the filtered collection,
66       * an {@code IllegalArgumentException} is thrown. When the
67       * {@code remove()} method is called on the filtered collection, only
68       * the underlying collection is modified.
69       *
70       * The returned collection isn't thread-safe.
71       *
72       * Many of the filtered collection's methods are supported.

```

示例代码如下：

```

public static void testCollections2(){
    List<String> list= Lists.newArrayList("moon","dad","refer","son");
    // filter ( ) : 只保留集合中满足特定要求的元素
    Collection<String> palindromeList= Collections2.filter(list, input -> {

```

```

return new StringBuilder(input).reverse().toString().equals(input); //找回文串
});
System.out.println(palindromeList);

Set<Long> times= Sets.newHashSet();
times.add(91299990701L);
times.add(9320001010L);
times.add(9920170621L);
// transform ( ) : 类型转换
Collection<String> timeStrCol= Collections2.transform(times, new Function<Long, String>()
{
    @Nullable
    @Override
    public String apply(@Nullable Long input) {
        return new SimpleDateFormat("yyyy-MM-dd").format(input);
    }
});
System.out.println(timeStrCol);
// 多个Function组合
List<String> list3= Lists.newArrayList("abcde","good","happiness");
//确保容器中的字符串长度不超过5
Function<String,String> f1=new Function<String, String>() {
    @Nullable
    @Override
    public String apply(@Nullable String input) {
        return input.length()>5?input.substring(0,5):input;
    }
};
//转成大写
Function<String,String> f2=new Function<String, String>() {
    @Nullable
    @Override
    public String apply(@Nullable String input) {
        return input.toUpperCase();
    }
};
Function<String,String> function= Functions.compose(f1,f2);
Collection<String> results=Collections2.transform(list3,function);
System.out.println(results);
}

```

运行结果是：

```
↑ D:\ProgramFiles\Java\jdk1.8.0_181\bin\java.exe ...
↓ Connected to the target VM, address: '127.0.0.1:56980', transport: 'socket'
↵ [dad, refer]
↵ [1970-04-19, 1970-04-26, 1972-11-23]
↵ [ABCDE, GOOD, HAPPY]
☐ Disconnected from the target VM, address: '127.0.0.1:56980', transport: 'socket'
```

6)、集合操作：交集、差集、并集

示例代码如下：

```
public static void testCollectionOperator(){
    Set<Integer> set1= Sets.newHashSet(1,2,3,4,5);
    Set<Integer> set2=Sets.newHashSet(3,4,5,6);
    Sets.SetView<Integer> inter=Sets.intersection(set1,set2); //交集
    System.out.println(inter);
    Sets.SetView<Integer> diff=Sets.difference(set1,set2); //差集,在A中不在B中
    System.out.println(diff);
    Sets.SetView<Integer> union=Sets.union(set1,set2); //并集
    System.out.println(union);
}
```

运行结果如下：

```
D:\ProgramFiles\Java\jdk1.8.0_181\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:58111', transport: 'socket'
[3, 4, 5]
[1, 2]
[1, 2, 3, 4, 5, 6]
Disconnected from the target VM, address: '127.0.0.1:58111', transport: 'socket'
```

4、原语类型

作为Java的原语类型，不能用来传递在泛型或于类别作为输入。Guava提供大量包装工具类来处理原始类型的对象。以下是有用的原始处理工具的列表：

S.N.	类名	程序名称和说明
1	<u>Bytes</u>	实用程序的原始字节。
2	<u>Shorts</u>	实用的原始short。
3	<u>Ints</u>	实用为基本整型。
4	<u>Longs</u>	实用的原始长整型。
5	<u>Floats</u>	实用为基本float。

6	<u>Doubles</u>	实用为基本的double。
7	<u>Chars</u>	实用的原始字符。
8	<u>Booleans</u>	实用为基本布尔。

5、数学工具

提供Guava数学相关的实用工具类来处理int，long及BigInteger。以下是有用的工具列表：

S.N.	类名	程序名称和说明
1	IntMath	数学工具为int类型。
2	LongMath	数学工具为long类型。
3	BigIntegerMath	数学实用程序处理BigInteger。

6、缓存工具

Guava通过接口LoadingCache提供了一个非常强大的基于内存的LoadingCache<K，V>。在缓存中自动加载值，它提供了许多实用的方法，在有缓存需求时非常有用。

S.N.	方法及说明
1	V apply(K key) 不推荐使用。提供满足功能接口;使用get(K)或getUnchecked(K)代替。
2	ConcurrentMap<K,V> asMap() 返回存储在该缓存作为一个线程安全的映射条目的视图。
3	V get(K key) 返回一个键在这个高速缓存中，首先装载如果需要该值相关联的值。
4	ImmutableMap<K,V> getAll(Iterable<? extends K> keys) 返回一个键相关联的值的映射，创建或必要时检索这些值。
5	V getUnchecked(K key) 返回一个键在这个高速缓存中，首先装载如果需要该值相关联的值。
6	void refresh(K key) 加载键key，可能是异步的一个新值。

Cache

缓存在很多场景下都是相当有用的。例如，计算或检索一个值的代价很高，并且对同样的输入需要不止一次获取值的时候，就应当考虑使用缓存。

Guava Cache与ConcurrentMap很相似，但也不完全一样。最基本的区别是ConcurrentMap会一直保存所有添加的元素，直到显式地移除。相对地，Guava Cache为了限制内存占用，通常都设定为自动回收元素。在某些场景下，尽管LoadingCache 不回收元素，它也是很有用的，因为它会自动加载缓存。Guava Cache是一个全内存的本地缓存实现，它提供了线程安全的实现机制。

通常来说，Guava Cache适用于：

- 你愿意消耗一些内存空间来提升速度。
- 你预料到某些键会被查询一次以上。
- 缓存中存放的数据总量不会超出内存容量。（Guava Cache是单个应用运行时的本地缓存。它不把数据存放到文件或外部服务器。
- 如果这不符合你的需求，请尝试Memcached这类工具）

Guava Cache有两种创建方式：

- cacheLoader
- callable callback

LoadingCache是附带CacheLoader构建而成的缓存实现refresh机制：

- - LoadingCache.refresh(K) 在生成新的value的时候，旧的value依然会被使用。
- - CacheLoader.reload(K, V) 生成新的value过程中允许使用旧的value
- - CacheBuilder.refreshAfterWrite(long, TimeUnit) 自动刷新cache

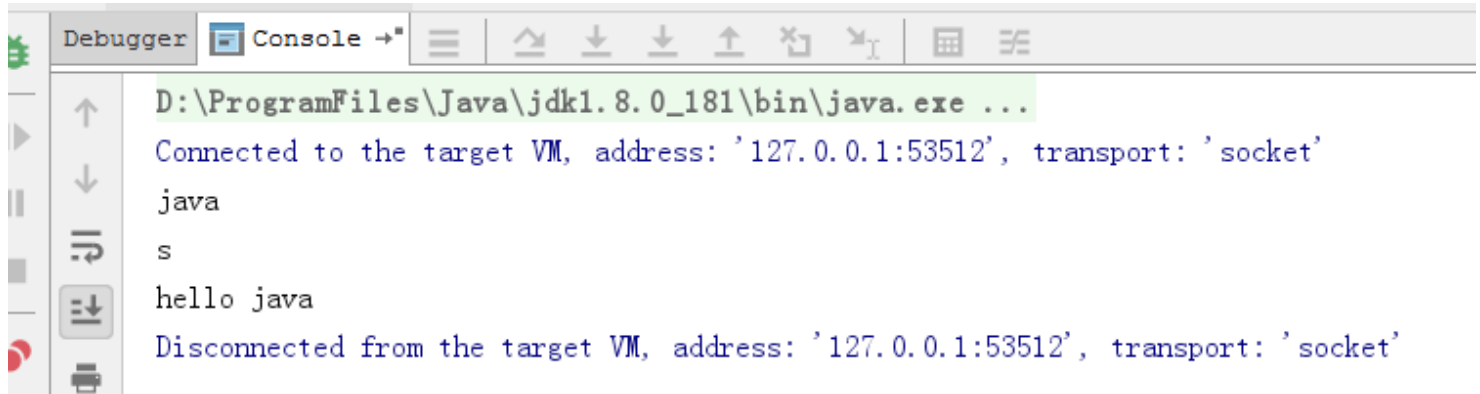
示例代码如下

```
public static void testCache(){
    LoadingCache<String,String> cache= CacheBuilder.newBuilder()
        .maximumSize(100) //最大缓存数目
        .expireAfterAccess(1, TimeUnit.SECONDS) //缓存1秒后过期
        .build(new CacheLoader<String, String>() {
            @Override
            public String load(String key) throws Exception {
                return key;
            }
        });
    cache.put("j","java");
    cache.put("c","cpp");
    cache.put("s","scala");
    cache.put("g","go");
    try {
        System.out.println(cache.get("j"));
        TimeUnit.SECONDS.sleep(2);
        System.out.println(cache.get("s")); //输出s
    } catch (ExecutionException | InterruptedException e) {
        e.printStackTrace();
    }

    Cache<String,String> cache2= CacheBuilder.newBuilder()
        .maximumSize(100)
        .expireAfterAccess(1, TimeUnit.SECONDS)
        .build();
    try {
```

```
String result=cache2.get("java", () -> "hello java");  
System.out.println(result);  
} catch (ExecutionException e) {  
e.printStackTrace();  
}  
}
```

运行结果如下：



当然还有一些其他的，等待慢慢去研究。

参考资料：

guava的git管理地址 <https://github.com/google/guava/wiki>

guava快速入门 <https://blog.csdn.net/dgeek/article/details/76221746>

使用 Google Guava 美化你的 Java 代

码 <https://www.cnblogs.com/SummerinShire/p/6054983.html>

guava第三方教程 <https://www.yiibai.com/guava/>

阿里云guava教程列表 https://www.aliyun.com/jiaocheng/topic_13449.html

guava api <http://tool.oschina.net/apidocs/apidoc?api=guava>

guava 下载 <http://guava.droppages.com/> ,

维基百科 https://en.wikipedia.org/wiki/Google_Guava

guava学习 <https://www.cnblogs.com/haolnu/p/7747717.html>

Google Guava官方教程 <https://blog.csdn.net/axi295309066/article/details/70856889>

<http://code.google.com/p/guava-libraries/wiki/GuavaExplained>

Google Guava官方教程（中文版） <http://ifeve.com/google-guava/>