Atititi 过滤对象list列表集合的实现 方便实现cache缓存cache ，where功能 filt功能

Function接口的情况是，我们不会去详细的讲解equals方法。apply方法会返回对输入断定后的结果。在Fucntion接口使用的地方来转换对象，Predicate接口则用于过滤对象。Pr

Predicate接口则用于过滤对象

使用Predicates.and方法

Predicates.and方法接受多个Predicate对象并且返回一个Predicate对象，因此调用返回的Predicate对象的apply方法当所有Predicate对象的apply方法都返回true的时候会返回true。如果其中一个Predicate对象返回false，其他的Predicate对象的执行就会停止。例如，假如我们只允许城市人口小于500,000并且年降雨量小于45.7英寸的。

Predicate smallAndDry = Predicates.**and**(smallPopulationPredicate, lowRainFallPredicate);

下面是Predicates.and方法的签名：

Predicates.and(Iterable<Predicate<T>> predicates);

Predicates.and(Predicate<T> ...predicates);

使用Predicates.or方法

Predicates.or方法接受多个Predicate对象并且返回一个Predicate对象,如果当中有一个Predicate对象的apply方法返回true则总方法就返回true。

如果有一个Predicate实例返回true，就

Function接口提高给我们转换对象和Predicate接口可以给我们一个强大的过滤机。Functions和Predicates类也帮助我们写代码更加简单。Suppliers通过提供必要的协作对象，而完全隐藏了这些对象创建的细节。 使用依赖注入框架spring或者guice，这些接口将允许我们无缝地通过简单地提供不同的实现改变我们的程序的行为。下一章我们讲Guava的重点，Collections。

## Functions[函数]和Predicates[断言]

[Google Guava] 4-函数式编程 \_ 并发编程网 – ifeve.com.html

**public** **class** ListCacheTest {

**public** **static** **void** main(String[] args) {

List<Map> **li**=Lists.*newArrayList*();

li.add( **new** Function<Object, Map>() {

@Override

**public** Map apply(Object t) {

Map **m**= Maps.*newConcurrentMap*();

m.put("c1", "c1v");

m.put("rowid", "r1");

**return** m;

}

}.apply(**null**) );

li.add( **new** Function<Object, Map>() {

@Override

**public** Map apply(Object t) {

Map **m**= Maps.*newConcurrentMap*();

m.put("c1", "c1v2");

m.put("rowid", "r2");

**return** m;

}

}.apply(**null**) );

// select \* from list where rowid=r2

//Function<String,Map> lookup = Functions.

Collection **rzt**= Collections2.*filter*(li, **new** Predicate<Map>(){

@Override

**public** **boolean** apply(Map rowMap) {

**if**(rowMap.get("rowid").equals("r2"))

**return** **true**;

**else**

**return** **false**;

}});

System.***out***.println( JSON.*toJSONString*(rzt));

}

}