equals

当一个对象中的字段可以为null时，实现Object.equals方法会很痛苦，因为不得不分别对它们进行null检查。使用Objects.equal帮助你执行null敏感的equals判断，从而避免抛出NullPointerException。例如:

```
1. Objects.equal("a", "a"); // returns true
2. Objects.equal(null, "a"); // returns false
3. Objects.equal("a", null); // returns false
4. Objects.equal(null, null); // returns true
```

hashCode

用对象的所有字段作散列[hash]运算应当更简单。Guava的Objects.hashCode(Object...)会对传入的字段序列计算出合理的、顺序敏感的散列值。你可以使用Objects.hashCode(field1, field2, …, fieldn)来代替手动计算散列值。

toString

好的toString方法在调试时是无价之宝，但是编写toString方法有时候却很痛苦。使用 Objects.toStringHelper可以轻松编写有用的toString方法。例如：

```
1. // Returns "ClassName{x=1}"
2. Objects.toStringHelper(this).add("x", 1).toString();
3. // Returns "MyObject{x=1}"
4. Objects.toStringHelper("MyObject").add("x", 1).toString();
```

compare/compareTo

实现一个比较器[Comparator]，或者直接实现Comparable接口有时也伤不起。考虑一下这种情况：

```
1. class Person implements Comparable<Person> {
2.   private String lastName;
3.   private String firstName;
4.   private int zipCode;
5.
6.   public int compareTo(Person other) {
7.     int cmp = lastName.compareTo(other.lastName);
8.     if (cmp != 0) {
9.       return cmp;
10.     }
11.     cmp = firstName.compareTo(other.firstName);
12.     if (cmp != 0) {
13.       return cmp;
14.     }
15.     return Integer.compare(zipCode, other.zipCode);
16.   }
17. }
```

这部分代码太琐碎了，因此很容易搞乱，也很难调试。我们应该能把这种代码变得更优雅，为此，Guava提供了ComparisonChain。

```
1. public int compareTo(Foo that) {
2.     return ComparisonChain.start()
3.             .compare(this.aString, that.aString)
4.             .compare(this.anInt, that.anInt)
5.             .compare(this.anEnum, that.anEnum,
Ordering.natural().nullsLast())
6.             .result();
7. }
```

ComparisonChain执行一种懒比较：它执行比较操作直至发现非零的结果，在那之后的比较输入将被忽略。

在新版本的JDK中等于零的结果会抛异常，必须返回非零的结果