一句话理解：减少大量判断，简介代码，使用try/catch来处理参数的判断逻辑。

在日常开发中，我们经常会对方法的输入参数做一些数据格式上的验证，以便保证方法能够按照正常流程执行下去。对于可
一定要做事前检测和判断，来避免程序流程出错，而不是完全通过错误处理来保证流程正确执行，毕竟错误处理是比较消耗
对参数的判断都需要自己来逐个写方法判断，代码量不少并且复用性不高，如下所示

```
1. public class PreconditionsTest {
2.
3.     @Test
4.     public void Preconditions() throws Exception {
5.
6.         getPerson(8,"peida");
7.
8.         getPerson(-9,"peida");
9.
10.         getPerson(8,"");
11.
12.         getPerson(8,null);
13.     }
14.     public static void getPerson(int age,String neme)throws Exception{
15.         if(age>0&&neme!=null&&neme.isEmpty()!=true){
16.             System.out.println("a person age:"+age+",neme:"+neme);
17.         }else{
18.             System.out.println("参数输入有误！");
19.         }
20.     }
21. }
```

说明：参数验证，我们每次都要添加if语句来做判断，重复的工作会做好多次。getPerson方法只有2个参数，验证规则也不
则复杂后，上面代码的可读性都会很差的，复用性就更谈不上了。

Guava类库中提供了一个作参数检查的工具类--Preconditions类， 该类可以大大地简化我们代码中对于参数的预判断和
验证实现起来更加简单优雅，下面我们看看Preconditions类的使用实例：

```
1. import com.google.common.base.Preconditions;
2.
3. public class PreconditionsTest {
4.
5.     @Test
6.     public void Preconditions() throws Exception {
7.
8.         getPersonByPrecondition(8,"peida");
9.
10.         try {
11.             getPersonByPrecondition(-9,"peida");
12.         } catch (Exception e) {
13.             System.out.println(e.getMessage());
14.         }
15.
16.         try {
17.             getPersonByPrecondition(8,"");
18.         } catch (Exception e) {
19.             System.out.println(e.getMessage());
20.         }
21.
22.         try {
23.             getPersonByPrecondition(8,null);
24.         } catch (Exception e) {
25.             System.out.println(e.getMessage());
26.         }
27.     }
28.
29.     public static void getPersonByPrecondition(int age,String neme)throws Exception{
30.         Preconditions.checkNotNull(neme, "neme为null");
31.         Preconditions.checkArgument(neme.length()>0, "neme为\'\'");
32.         Preconditions.checkArgument(age>0, "age 必须大于0");
33.         System.out.println("a person age:"+age+",neme:"+neme);
34.     }
35. }
```

方法说明：

方法声明（不包括额外参数）	描述	检查失败时抛出
checkArgument(boolean)	检查boolean是否为true，用来检查传递给方法的参数。	IllegalArgumen
checkNotNull(T)	检查value是否为null，该方法直接返回value，因此可以内嵌使用checkNotNull。	NullPointerExc
checkState(boolean)	用来检查对象的某些状态。	IllegalStateExc
checkElementIndex(int index, int size)	检查index作为索引值对某个列表、字符串或数组是否有效。index>=0 && index<size *	IndexOutOfBo
checkPositionIndex(int index, int size)	检查index作为位置值对某个列表、字符串或数组是否有效。index>=0 && index<=size *	IndexOutOfBo
checkPositionIndexes(int start, int end, int size)	检查[start, end]表示的位置范围对某个列表、字符串或数组是否有效*	IndexOutOfBo