# Guava学习之Preconditions

在编写程序的时候，很多时候都需要检查输入的参数是否符合我们的需要，比如人的年龄需要大于0，名字不能为空；如果不符合这两个要求，我们将认为这个对象是不合法的，这时候我们需要编写判断这些参数是否合法的函数，我们可能这样写：

```
package com.wyp;

import java.util.ArrayList;
import java.util.List;

/**
 * Created with IntelliJ IDEA.
 * User: 过往记忆
 * Blog:
 * Date: 13-7-24
 * Time: 下午9:02
 */
public class TestObject {
    List personList = new ArrayList();

    public void addPerson(Person person) {
        if (check(person)) {
            personList.add(person);

        }
    }

    private boolean check(Person person) {
        if (person.getAge() > 0 && person.getName() != null) {
            return true;
        }
        return false;
    }
}
```

这样看起来很不错，但是如果还有很多对象需要进行验证呢？这样你可能会想，我再去写几个函数去验证。但是你可能发现，这样的函数很难维护，而且可用性不高，扩展性也比较差，复用性就更谈不上了。

这就是今天要谈的Guava中提供的Preconditions类，Preconditions提供了大量的用于参数的

检测函数，在项目中使用了这个类，不仅可以简化我们的代码，而且可读性大大提高，何乐而不为呢？强烈建议在您的代码中使用这个类，而且在Preconditions类中的所有函数都是static修饰的，这意味着你不需要声明（你不可以申请一个Preconditions对象，因为Preconditions的构造函数用private修饰的）一个Preconditions类对象，而可以直接使用这个类提供的方法。在Preconditions中提供的校验类主要有以下几个：

1、static void checkArgument(boolean expression)：这个函数在expression为true的时候将不会发生任何事情，如果expression为false将会抛出IllegalArgumentException异常，说明输入的参数非法；Guava中的实现如下：

```
public static void checkArgument(boolean expression) {
   if (!expression) {
     throw new IllegalArgumentException();
   }
}
```

2、static void checkArgument(boolean expression, Object errorMessage)：这个函数和上面的类似，只不过在抛出异常的同时多数出了errorMessage提示信息；Guava中的实现如下：

```
public static void checkArgument(
     boolean expression, @Nullable Object errorMessage) {
   if (!expression) {
     throw new IllegalArgumentException(String.valueOf(errorMessage));
   }
}
```

3、static void checkArgument(boolean expression, String errorMessageTemplate, Object... errorMessageArgs)：和上面的类似，只不过输出的提示信息可以被格式化成一定的格式；Guava中的实现如下：

```
 public static void checkArgument(boolean expression,
     @Nullable String errorMessageTemplate,
     @Nullable Object... errorMessageArgs) {
   if (!expression) {
     throw new IllegalArgumentException(
        format(errorMessageTemplate, errorMessageArgs));
   }
}
```

4、static int checkElementIndex(int index, int size)检查index是否为在一个长度为size的list、string或array合法的范围。 index的范围区间是[0, size](包含0和size)。无需直接传入list、string或array， 只需传入大小。返回index。失败时抛出的异常类型：IndexOutOfBoundsException；Guava中的实现如下：

```
public static int checkElementIndex(int index, int size) {
  return checkElementIndex(index, size, "index");
}
```

5、static int checkElementIndex(int index, int size, String desc)和上面的类似，只不过在出现异常的时候多数出一些提示信息(desc)；Guava中的实现如下：

```
public static int checkElementIndex(
    int index, int size, @Nullable String desc) {
  // Carefully optimized for execution by hotspot (explanatory comment above)
  if (index < 0 || index >= size) {
    throw new IndexOutOfBoundsException(badElementIndex(index, size, desc));
  }
  return index;
}
```

6、static T checkNotNull(T reference)检查reference不为null时，则直接返回value；否则抛出的异常类NullPointerException；Guava中的实现如下：

```
public static  T checkNotNull(T reference) {
  if (reference == null) {
    throw new NullPointerException();
  }
  return reference;
}
```

7、static T checkNotNull(T reference, Object errorMessage)和上面的功能一样，只不过在抛出异常的时候输出了一些提示信息；Guava中的实现如下：

```
public static  T checkNotNull(T reference, @Nullable Object errorMessage) {
  if (reference == null) {
    throw new NullPointerException(String.valueOf(errorMessage));
```

```
    }
    return reference;
}
```

8、static T checkNotNull(T reference, String errorMessageTemplate, Object... errorMessageArgs)和上面的类是，但是在出现异常的时候输出了格式化的错误信息提示；Guava中的实现如下：

```
public static  T checkNotNull(T reference,
      @Nullable String errorMessageTemplate,
      @Nullable Object... errorMessageArgs) {
    if (reference == null) {
      // If either of these parameters is null, the right thing happens anyway
      throw new NullPointerException(
          format(errorMessageTemplate, errorMessageArgs));
    }
    return reference;
}
```

9、static int checkPositionIndex(int index, int size)检查位置index是否为在一个长度为size的list、string或array合法的范围。index的范围区间是[0， size)(包含0不包含size)。无需直接传入list、string或array， 只需传入大小。成功的时候返回index；失败时抛出的异常类型：IndexOutOfBoundsException；Guava中的实现如下：

```
public static int checkPositionIndex(int index, int size) {
    return checkPositionIndex(index, size, "index");
}
```

10、static int checkPositionIndex(int index, int size, String desc)和上面的功能一样，在失败的时候返回提示信息(desc)；Guava中的实现如下：

```
public static int checkPositionIndex(
      int index, int size, @Nullable String desc) {
    // Carefully optimized for execution by hotspot (explanatory comment above)
    if (index < 0 || index > size) {
      throw new IndexOutOfBoundsException(badPositionIndex(index, size, desc));
```

```
    }
    return index;
}
```

11、static void checkPositionIndexes(int start, int end, int size)检查[start, end)是一个长度为size的list、string或array合法的范围子集。失败时抛出的异常类型：IndexOutOfBoundsException；Guava中的实现如下：

```
public static void checkPositionIndexes(int start, int end, int size) {
    // Carefully optimized for execution by hotspot (explanatory comment above)
    if (start < 0 || end < start || end > size) {
      throw new IndexOutOfBoundsException(badPositionIndexes(start, end, size));
    }
}
```

12、static void checkState(boolean expression)这个判断expression是否为true，如果是则什么都不做，否则抛出IllegalStateException异常；Guava中的实现如下：

```
public static void checkState(boolean expression) {
    if (!expression) {
      throw new IllegalStateException();
    }
}
```

13、static void checkState(boolean expression, Object errorMessage)和上面的功能一样，只不过在抛出异常的时候输出了一些提示信息；Guava中的实现如下：

```
public static void checkState(
      boolean expression, @Nullable Object errorMessage) {
    if (!expression) {
      throw new IllegalStateException(String.valueOf(errorMessage));
    }
}
```

14、static void checkState(boolean expression, String errorMessageTemplate, Object... errorMessageArgs)和上面的类是，但是在出现异常的时候输出了格式化的错误信息提示；Guava中的实现如下：

```
public static void checkState(boolean expression,
    @Nullable String errorMessageTemplate,
    @Nullable Object... errorMessageArgs) {
  if (!expression) {
    throw new IllegalStateException(
       format(errorMessageTemplate, errorMessageArgs));
  }
}
```

说了这么多，如何来用这个类呢？如下所示:

```
import com.google.common.base.Preconditions;

public void TestPre(Person person, List plist, int index) {
      Preconditions.checkArgument(person != null);
      Preconditions.checkNotNull(plist);
      Preconditions.checkState(index > 0);
      Preconditions.checkPositionIndex(index, plist.size());
      Preconditions.checkElementIndex(index, plist.size());
}
```

这样就可以测试一个函数输入的参数是否合法。在Guava中的Strings类中还有Strings.isNullOrEmpty("")和Strings.emptyToNull("");两个用于检测字符串参数是否合法的函数，使用起来和Preconditions差不多，很简单，具体的就不介绍了。

Guava中的Preconditions有以下几个优点：（1）、在静态导入后, 方法很明确无歧义, checkNotNull可以清楚地告诉你它是干什么的, 它会抛出怎样的异常。（2、）简单而又强大的可变参数'printf'风格的自定义错误信息。也去你代码中加入Preconditions类吧！
(完)