在我们的开发中，NullPointerException可谓是随时随处可见，为了避免空指针异常，我们常常需要进行一些防御式的检查，所以在代码中常常可见if(obj != null) 这样的判断。幸好在JDK1.8中，java为我们提供了一个Optional类，Optional类能让我们省掉繁琐的非空的判断，下面是该类提供的方法

方法	描述
of	把指定的值封装为Optional对象，如果指定的值为null，则抛出NullPointerException
empty	创建一个空的Optional对象
ofNullable	把指定的值封装为Optional对象，如果指定的值为null，则创建一个空的Optional对象
get	如果创建的Optional中有值存在，则返回此值，否则抛出NoSuchElementException
orElse	如果创建的Optional中有值存在，则返回此值，否则返回一个默认值
orElseGet	如果创建的Optional中有值存在，则返回此值，否则返回一个由Supplier接口生成的值
orElseThrow	如果创建的Optional中有值存在，则返回此值，否则抛出一个由指定的Supplier接口生成的异常
filter	如果创建的Optional中的值满足filter中的条件，则返回包含该值的Optional对象，否则返回一个空的Optional对象
map	如果创建的Optional中的值存在，对该值执行提供的Function函数调用
flagMap	如果创建的Optional中的值存在，就对该值执行提供的Function函数调用，返回一个Optional类型的值，否则就返回一个空的Optional对象
isPresent	如果创建的Optional中的值存在，返回true，否则返回false
ifPresent	如果创建的Optional中的值存在，则执行该方法的调用，否则什么也不做

of

```
//创建一个值为张三的String类型的Optional
Optional<String> ofOptional = Optional.of("张三");
//如果我们用of方法创建Optional对象时，所传入的值为null，则抛出NullPointerException如下图所示
Optional<String> nullOptional = Optional.of(null);
```

empty

```
//创建一个空的String类型的Optional对象
Optional<String> emptyOptional = Optional.empty();
```

ofNullable

```
//为指定的值创建Optional对象，不管所传入的值为null不为null，创建的时候都不会报错
Optional<String> nullOptional = Optional.ofNullable(null);
Optional<String> nullOptional = Optional.ofNullable("lisi");
```

如果传递的是Null，会自动调用Empty方法，创建一个空对象

get

如果我们创建的Optional对象中有值存在则返回此值，如果没有值存在，则会抛出

NoSuchElementException异常。小demo如下：

```
Optional<String> stringOptional = Optional.of("张三");
System.out.println(stringOptional.get());
```

orElse

如果创建的Optional中有值存在，则返回此值，否则返回一个默认值

```
Optional<String> stringOptional = Optional.of("张三");
System.out.println(stringOptional.orElse("zhangsan"));

Optional<String> emptyOptional = Optional.empty();
System.out.println(emptyOptional.orElse("李四"));
```

orElseGet

如果创建的Optional中有值存在，则返回此值，否则返回一个由Supplier接口生成的值

```
Optional<String> stringOptional = Optional.of("张三");
System.out.println(stringOptional.orElseGet(() -> "zhangsan"));

Optional<String> emptyOptional = Optional.empty();
System.out.println(emptyOptional.orElseGet(() -> "orElseGet"));
```

orElseThrow

如果创建的Optional中有值存在，则返回此值，否则抛出一个由指定的Supplier接口生成的异常

```
Optional<String> stringOptional = Optional.of("张三");
System.out.println(stringOptional.orElseThrow(CustomException::new));

Optional<String> emptyOptional = Optional.empty();
System.out.println(emptyOptional.orElseThrow(CustomException::new));

private static class CustomException extends RuntimeException {
  private static final long serialVersionUID = -4399699891687593264L;

  public CustomException() {
     super("自定义异常");
  }

  public CustomException(String message) {
     super(message);
  }
 }
```

filter

如果创建的Optional中的值满足filter中的条件，则返回包含该值的Optional对象，否则返回一个空的Optional对象

```
Optional<String> stringOptional = Optional.of("zhangsan");
System.out.println(stringOptional.filter(e -> e.length() > 5).orElse("王五"));
stringOptional = Optional.empty();
System.out.println(stringOptional.filter(e -> e.length() > 5).orElse("lisi"));
```

注意：Optional中的filter方法和Stream中的filter方法是有点不一样的，Stream中的filter方法是对一堆元素进行过滤，而Optional中的filter方法只是对一个元素进行过滤，可以把Optional看成是最多只包含一个元素

的Stream

map

如果创建的Optional中的值存在，对该值执行提供的Function函数调用

```
return user.map(u -> u.getOrders()).orElse(Collections.emptyList())

//上面避免了我们类似 Java 8 之前的做法
if(user.isPresent()) {
  return user.get().getOrders();
} else {
  return Collections.emptyList();
}
```

map 是可能无限级联的, 比如再深一层, 获得用户名的大写形式

```
return user.map(u -> u.getUsername())
           .map(name -> name.toUpperCase())
           .orElse(null);
```

这要搁在以前, 每一级调用的展开都需要放一个 null 值的判断

```
User user = .....
if(user != null) {
  String name = user.getUsername();
  if(name != null) {
    return name.toUpperCase();
  } else {
    return null;
  }
} else {
  return null;
}
```

flagMap

flatMap与map（Funtion）方法类似，区别在于flatMap中的mapper返回值必须是Optional，map方法的mapping函数返回值可以是任何类型T。调用结束时，flatMap不会对结果用Optional封装。

ifPresent

存在才做点什么

```
user.ifPresent(System.out::println);

//而不要下边那样
if (user.isPresent()) {
  System.out.println(user.get());
}
```