答应我， 不要再用 if (obj != null) 判空了

Java专栏 2022-10-09 12:20 发表于甘肃

Java专栏
一个Java、数据库、中间件、业内资讯、面试、学习资源等干货的知识分享社区。每...
94篇原创内容

公众号

相信不少小伙伴已经被java的NPE(Null Pointer Exception)所谓的空指针异常搞的头昏脑涨,有大佬说过"防止 NPE，是程序员的基本修养。"但是修养归修养，也是我们程序员最头疼的问题之一，那么我们今天就要尽可能的利用Java8的新特性 Optional来尽量简化代码同时高效处理NPE（Null Pointer Exception 空指针异常）

认识Optional并使用

简单来说，Opitonal类就是Java提供的为了解决大家平时判断对象是否为空用 会用 null!=obj 这样的方式存在的判断，从而令人头疼导致NPE（Null Pointer Exception 空指针异常），同时Optional的存在可以让代码更加简单，可读性跟高，代码写起来更高效.

常规判断：

```
//对象 人
//属性有 name，age
Person person=new Person();
if (null==person){
   return "person为null";
}
return person;
```

使用Optional：

```
//对象 人
//属性有 name，age
Person person=new Person();
```

```
return Optional.ofNullable(person).orElse("person为null");
```

测试展示类Person代码(如果有朋友不明白可以看一下这个)：

```
public class Person {
    private String name;
    private Integer age;

    public Person(String name, Integer age) {
        this.name = name;
        this.age = age;
    }

    public Person() {
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }
}
```

下面，我们就高效的学习一下神奇的Optional类！

2.1 Optional对象创建

首先我们先打开Optional的内部,去一探究竟 先把几个创建Optional对象的方法提取出来

```
public final class Optional<T> {
```

```
    private static final Optional<?> EMPTY = new Optional<>();
    private final T value;
    //我们可以看到两个构造方格都是private 私有的
    //说明 我们没办法在外面去new出来Optional对象
    private Optional() {
        this.value = null;
     }
    private Optional(T value) {
        this.value = Objects.requireNonNull(value);
     }
     //这个静态方法大致 是创建出一个包装值为空的一个对象因为没有任何参数赋值
    public static<T> Optional<T> empty() {
        @SuppressWarnings("unchecked")
        Optional<T> t = (Optional<T>) EMPTY;
        return t;
     }
     //这个静态方法大致 是创建出一个包装值非空的一个对象 因为做了赋值
    public static <T> Optional<T> of(T value) {
        return new Optional<>(value);
     }
     //这个静态方法大致是 如果参数value为空，则创建空对象，如果不为空，则创建有参对象
    public static <T> Optional<T> ofNullable(T value) {
        return value == null ? empty() : of(value);
     }
}
```

再做一个简单的实例展示 与上面对应

```
// 1、创建一个包装对象值为空的Optional对象
Optional<String> optEmpty = Optional.empty();
// 2、创建包装对象值非空的Optional对象
Optional<String> optOf = Optional.of("optional");
// 3、创建包装对象值允许为空也可以不为空的Optional对象
Optional<String> optOfNullable1 = Optional.ofNullable(null);
Optional<String> optOfNullable2 = Optional.ofNullable("optional");
```

我们关于创建Optional对象的内部方法大致分析完毕 接下来也正式的进入Optional的学习与使用中。

2.2 Optional.get()方法(返回对象的值)

get()方法是返回一个option的实例值 源码：

```
public T get() {
    if (value == null) {
        throw new NoSuchElementException("No value present");
    }
    return value;
}
```

也就是如果value不为空则做返回，如果为空则抛出异常 "No value present" 简单实例展示

```
Person person=new Person();
person.setAge(2);
Optional.ofNullable(person).get();
```

2.3 Optional.isPresent()方法(判读是否为空)

isPresent()方法就是会返回一个boolean类型值，如果对象不为空则为真，如果为空则false 源码:

```
public Boolean isPresent() {
    return value != null;
}
```

简单的实例展示:

```
Person person=new Person();
person.setAge(2);
if (Optional.ofNullable(person).isPresent()){
    //写不为空的逻辑
    System.out.println("不为空");
} else{
    //写为空的逻辑
    System.out.println("为空");
}
```

2.4 Optional.ifPresent()方法(判读是否为空并返回函数)

这个意思是如果对象非空，则运行函数体 源码:

```
    public void ifPresent(Consumer<? super T> consumer) {
          //如果value不为空，则运行accept方法体
          if (value != null)
              consumer.accept(value);
      }
```

看实例:

```
Person person=new Person();
person.setAge(2);
Optional.ofNullable(person).ifPresent(p -> System.out.println("年龄"+p.getAge()));
```

如果对象不为空，则会打印这个年龄，因为内部已经做了NPE（非空判断），所以就不用担心空指针异常了。

2.5 Optional.filter()方法(过滤对象)

filter()方法大致意思是，接受一个对象，然后对他进行条件过滤，如果条件符合则返回Optional对象本身，如果不符合则返回空Optional

源码:

```
public Optional<T> filter(Predicate<? super T> predicate) {
    Objects.requireNonNull(predicate);
    //如果为空直接返回this
    if (!isPresent())
                return this; else
            //判断返回本身还是空Optional
    return predicate.test(value) ? this : empty();
}
```

简单实例：

```
Person person=new Person();
person.setAge(2);
Optional.ofNullable(person).filter(p -> p.getAge()>50);
```

2.6 Optional.map()方法(对象进行二次包装)

map()方法将对应Funcation函数式接口中的对象，进行二次运算，封装成新的对象然后返回在Optional中 源码：

```
public<U> Optional<U> map(Function<? super T, ? extends U> mapper) {
        Objects.requireNonNull(mapper);
        //如果为空返回自己
        if (!isPresent())
            return empty();
        else {
        //否则返回用方法修饰过的Optional
            return Optional.ofNullable(mapper.apply(value));
        }
    }
```

实例展示：

```
Person person1=new Person();
person.setAge(2);
String optName = Optional.ofNullable(person).map(p -> person.getName()).orElse("name为空");
```

2.7 Optional.flatMap()方法(Optional对象进行二次包装)

map()方法将对应 `Optional< Funcation >` 函数式接口中的对象，进行二次运算，封装成新的对象然后返回在Optional中 源码:

```
public<U> Optional<U> flatMap(Function<? super T, Optional<U>> mapper) {
    Objects.requireNonNull(mapper);
    if (!isPresent())
```

```
                return empty(); else {
            return Objects.requireNonNull(mapper.apply(value));
        }
    }
```

实例：

```
Person person=new Person();
person.setAge(2);
Optional<Object> optName = Optional.ofNullable(person).map(p -> Optional.ofNullable(p.getName
```

2.8 Optional.orElse()方法(为空返回对象)

常用方法之一，这个方法意思是如果包装对象为空的话，就执行orElse方法里的value，如果非空，则返回写入对象 源码:

```
public T orElse(T other) {
    //如果非空，返回value，如果为空，返回other
    return value != null ? value : other;
}
```

2.9 Optional.orElseGet()方法(为空返回Supplier对象)

这个与orElse很相似，入参不一样，入参为Supplier对象，为空返回传入对象的.get()方法，如果非空则返回当前对象 源码:

```
public T orElseGet(Supplier<? extends T> other) {
    return value != null ? value : other.get();
}
```

实例：

```
Optional<Supplier<Person>> sup=Optional.ofNullable(Person::new);
```

```
//调用get()方法，此时才会调用对象的构造方法，即获得到真正对象
Optional.ofNullable(person).orElseGet(sup.get());
```

说真的对于Supplier对象我也懵逼了一下，去网上简单查阅才得知 Supplier也是创建对象的一种方式,简单来说，Suppiler是一个接口，是类似Spring的懒加载，声明之后并不会占用内存，只有执行了get()方法之后，才会调用构造方法创建出对象创建对象的语法的话就是 `Supplier supPerson = Person::new;` 需要使用时 `supPerson.get()` 即可

2.10 Optional.orElseThrow()方法(为空返回异常)

这个我个人在实战中也经常用到这个方法，方法作用的话就是如果为空，就抛出你定义的异常，如果不为空返回当前对象，在实战中所有异常肯定是要处理好的，为了代码的可读性

源码：

```
public <X extends Throwable> T orElseThrow(Supplier<? extends X> exceptionSupplier) throw
    if (value != null) {
        return value;
    } else {
        throw exceptionSupplier.get();
    }
}
```

实例：这个就贴实战源码了

```
//简单的一个查询
Member member = memberService.selectByPhone(request.getPhone());
Optional.ofNullable(member).orElseThrow(() -> new ServiceException("没有查询的相关数据"));
```

2.11 相似方法进行对比分析

可能小伙伴看到这，没用用过的话会觉得orElse()和orElseGet()还有orElseThrow()很相似，map()和flatMap()好相似

哈哈哈不用着急，都是从这一步过来的，我再给大家总结一下不同方法的异同点

orElse()和orElseGet()和orElseThrow()的异同点

方法效果类似，如果对象不为空，则返回对象，如果为空，则返回方法体中的对应参数，所以可以看出这三个方法体中参数是不一样的

- orElse（T 对象）
- orElseGet（Supplier < T > 对象）
- orElseThrow（异常）

map()和orElseGet的异同点

- 方法效果类似，对方法参数进行二次包装，并返回,入参不同
- map（function函数）
- flatmap（Optional < function > 函数）

具体要怎么用，要根据业务场景以及代码规范来定义，下面可以简单看一下我在实战中怎用使用神奇的Optional

3.实战场景再现

场景1：

在service层中查询一个对象，返回之后判断是否为空并做处理

```
//查询一个对象
Member member = memberService.selectByIdNo(request.getCertificateNo());
//使用ofNullable加orElseThrow做判断和操作
Optional.ofNullable(member).orElseThrow(() -> new ServiceException("没有查询的相关数据"));
```

场景2：

我们可以在dao接口层中定义返回值时就加上Optional 例如：我使用的是jpa，其他也同理

```
public interface LocationRepository extends JpaRepository<Location, String> {
Optional<Location> findLocationById(String id);
}
```

然在是Service中

```
public TerminalVO findById(String id) {
//这个方法在dao层也是用了Optional包装了
        Optional<Terminal> terminalOptional = terminalRepository.findById(id);
        //直接使用isPresent()判断是否为空
        if (terminalOptional.isPresent()) {
        //使用get()方法获取对象值
            Terminal terminal = terminalOptional.get();
            //在实战中，我们已经免去了用set去赋值的繁琐，直接用BeanCopy去赋值
            TerminalVO terminalVO = BeanCopyUtils.copyBean(terminal, TerminalVO.class);
            //调用dao层方法返回包装后的对象
            Optional<Location> location = locationRepository.findLocationById(terminal.getLoc
            if (location.isPresent()) {
                terminalVO.setFullName(location.get().getFullName());
            }
            return terminalVO;
        }
        //不要忘记抛出异常
        throw new ServiceException("该终端不存在");
    }
```

4.Optional使用注意事项

Optional真么好用，真的可以完全替代if判断吗?

我想这肯定是大家使用完之后Optional之后可能会产生的想法，答案是否定的

举一个最简单的栗子：

例子1：

如果我只想判断对象的某一个变量是否为空并且做出判断呢?

```
Person person=new Person();
person.setName("");
persion.setAge(2);
//普通判断
```

```
if(StringUtils.isNotBlank(person.getName())){
  //名称不为空执行代码块
}
//使用Optional做判断
Optional.ofNullable(person).map(p -> p.getName()).orElse("name为空");
```

我觉得这个例子就能很好的说明这个问题，只是一个很简单判断，如果用了Optional我们还需要考虑包装值，考虑代码书写，考虑方法调用，虽然只有一行，但是可读性并不好，如果别的程序员去读，我觉得肯定没有if看的明显

5.jdk1.9对Optional优化

首先增加了三个方法:

```
or()、ifPresentOrElse() 和 stream()
```

or() 与orElse等方法相似，如果对象不为空返回对象，如果为空则返回or()方法中预设的值。

ifPresentOrElse() 方法有两个参数：一个 Consumer 和一个 Runnable。如果对象不为空，会执行 Consumer 的动作，否则运行 Runnable。相比ifPresent（）多了OrElse判断。

stream() 将Optional转换成stream，如果有值就返回包含值的stream，如果没值，就返回空的stream。

因为这个jdk1.9的Optional具体我没有测试，同时也发现有蛮好的文章已经也能让大家明白jdk1.9的option的优化,我就不深入去说了。

来源:juejin.im/post/5eb9faa26fb9a0437e0e9899

1. 总共88G，时常高达365小时，覆盖所有主流技术栈

2. 均为同一人录制，不是东拼西凑的

3. 对标线下T0级别的培训课，讲师大厂架构师，多年授课经验，通俗易懂

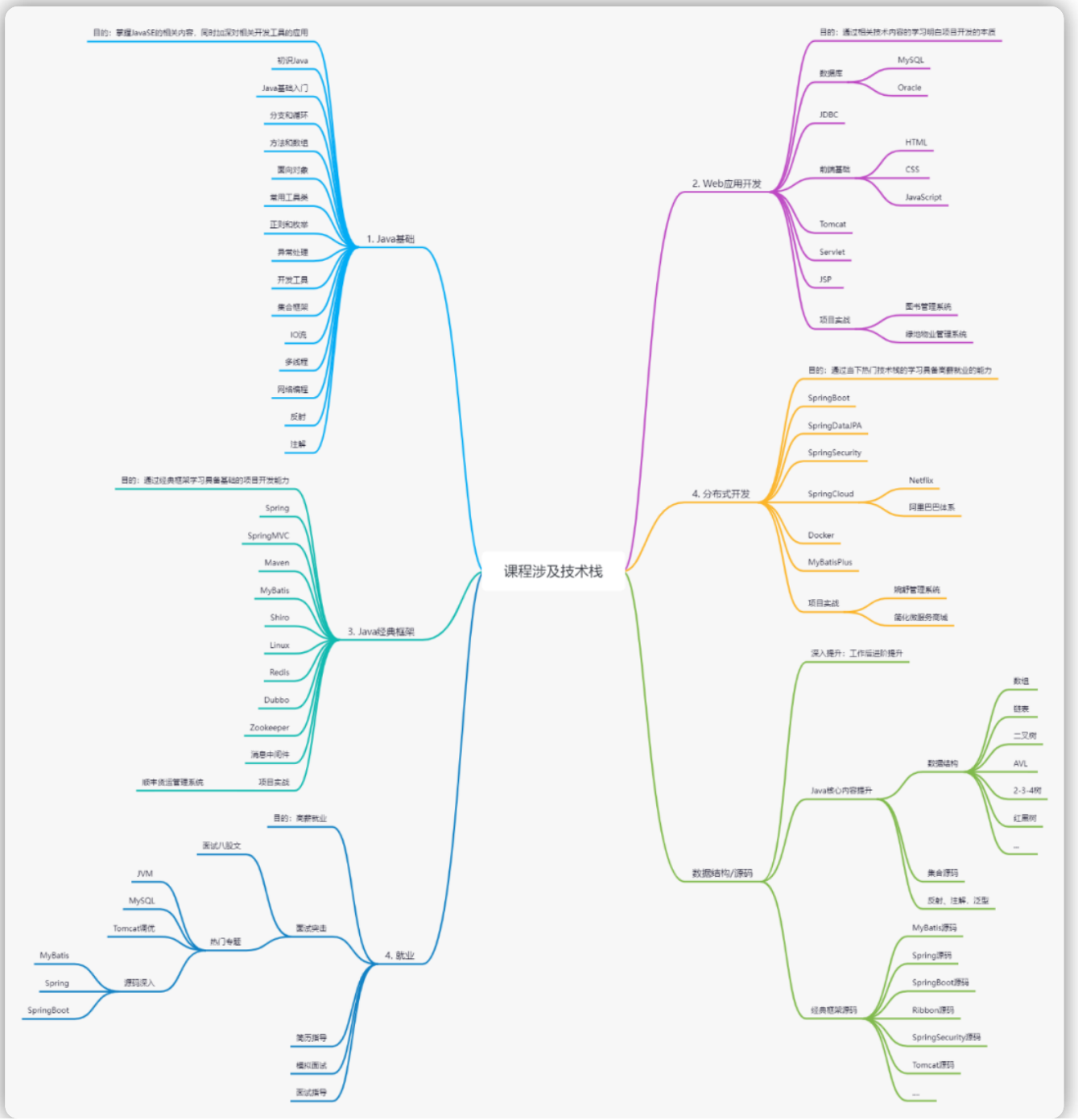
4. 内容丰富，每一个技术点除了视频，还有课堂源码、笔记、PPT、图解

5. 五大实战项目（视频+源码+笔记+SQL+软件）

6. 一次付费，持续更新，永无二次费用

点击下方超链接查看详情〇〇〇（或者点击文末阅读原文）：

(点击查看）88G，超全技术栈的Java入门+进阶+实战！



阅读原文

喜欢此内容的人还喜欢

.Net之接口文档精度丢失处理

鹏祥

给我实现一个前端的 Excel 导入和导出功能

前端瓶子君

C# 11 中的 file local type

amazingdotnet