总结

笔记本:	Spring问题		
创建时间:	2019/10/26 9:21	**更新时间:**	2019/10/31 18:38
作者:	1455352355@qq.com		
URL:	about:blank		

1：spring复习 注入方式
（1）：注解
Bean加返回值
（2）：Xml方式
2:
IOC容器的生命周期 初始化容器（init）->创建，使用容器，销毁容器
bean的生命周期 :Ioc对象初始化时候，会自动创建对象->init->....->当容器关闭时 调用destry
3：@Autowire在属性前标注 不调用setXXX,如果加载set前则调用set；还可以加在方法参数前（Bean加返回值形式不可 ）
自动注入 一：@Autiwried （spring提供）默认根据类型
自动注入二：@Resource （JSR250提供）默认根据名称 找不到在根据类型
4：注意 log文件是log4j 还是log4j2 这个很重要

```
<!-- LOG4J2 日志实现配置接口 -->
<setting name="logImpl" value="LOG4J" />
```

4：xml映射文件中#{} 应该填 不然会报错**There is no getter for property named 'USER_NAME' in 'class com.zz.chatroom.bean.UserInfoBean'**

```
后面的#{USER_NAME} 这里错了，应该
写成#{userName} 应该是属性值
```

5: @Sharable 注解用来说明ChannelHandler是否可以在多个channel直接共享使用。

6:枚举基本用法

在JDK1.5 之前，我们定义常量都是： public static fianl.... 。现在好了，有了枚举，可以把相关的常量分组到一个枚举类型里，而且枚举提供了比常量更多的方法。

```
public enum Color {
RED, GREEN, BLANK, YELLOW
}
```

用法二：switch

JDK1.6之前的switch语句只支持int,char,enum类型，使用枚举，能让我们的代码可读性更强。

```
enum Signal {
        GREEN, YELLOW, RED
    }
    public class TrafficLight {
        Signal color = Signal.RED;
        public void change() {
            switch (color) {
            case RED:
                color = Signal.GREEN;
                break;
            case YELLOW:
                color = Signal.RED;
                break;
            case GREEN:
                color = Signal.YELLOW;
                break;
```

```
            }
        }
    }
```

7：Js中Date 转换

```
newDate.format("yyyy-MM-dd hh:mm:ss")
```

8：lambda表达式

9：netty问题：https://www.jianshu.com/p/96a50869b527 || https://www.w3cschool.cn/netty_4_user_guide/6worbozt.html

10：数字类型的toString()方法可以接收表示转换基数(radix)的可选参数，如果不指定此参数，转换规则将是基于十进制。同样，也可以将数字转换为其他进制数(范围在2-36)

```
var n = 17;
n.toString();//'17'
n.toString(2);//'10001'
n.toString(8);//'21'
n.toString(10);//'17'
n.toString(12);//'15'
n.toString(16);//'11'
```

11：Java 8 Stream 概念 https://www.runoob.com/java/java8-streams.html

12：for 和foreach 和迭代器： for循环便于访问顺序存储的记录，而foreach和迭代器便于访问链接存储。

13：MessageFormat.format()用法

```
java.text.MessageFormat类
MessageFormat提供一种语言无关的方式来组装消息，它允许你在运行时刻用指定的参数来替换掉消息字符串中的一部分。你可以为MessageFormat定义一个模式，在其中你可以用占位符来表示变化的部分：
```

14：HashMap和ConcurrentHashMap的区别 https://www.cnblogs.com/heyonggang/p/9112731.html

```
锁分段技术：首先将数据分成一段一段的存储，然后给每一段数据配一把锁，当一个线程占用锁访问其中一个段数据的时候，其他段的数据也能被其他线程访问。
ConcurrentHashMap提供了与Hashtable和SynchronizedMap不同的锁机制。Hashtable中采用的锁机制是一次锁住整个hash表，从而在同一时刻只能由一个线程对其进行操作；而ConcurrentHashMap中则是一次锁住一个桶。
ConcurrentHashMap默认将hash表分为16个桶，诸如get、put、remove等常用操作只锁住当前需要用到的桶。这样，原来只能一个线程进入，现在却能同时有16个写线程执行，并发性能的提升是显而易见的
```

15：链式编程

```
链式编程可以使得代码可读性高，链式编程的原理就是返回一个this对象，就是返回本身，达到链式效果
entity.setName("fdf").setAge(21).setSex('m').setEmailAddress("123453612@qq.com").setPhoneNum("113-1213-4478");
```

16：mybatis-plus 主键自增问题
https://blog.csdn.net/qq_37186247/article/details/85238506

```
org.apache.ibatis.reflection.ReflectionException: Could not set property 'id' of 'class com.pojo.sallerPojo.TbBrand' with value '1077177904745537538' Cause: java.lang.IllegalArgumentException: argument type mismatch
```

17： **instanceof** ： instanceof 运算符只能用作对象的判断。
严格来说是Java中的一个双目运算符（*运算*所需变量为两个的*运算符*叫做*双目运算符*），用来测试一个对象是否为一个类的实例

```
boolean result = obj instanceof Class
```

其中 obj 为一个对象，Class 表示一个类或者一个接口，
当 obj 为 Class 的对象，或者是其直接或间接子类，或者是其接口的实现类，结果 result 都返回 true，否则返回false。

```
1:obj 必须为引用类型，不能是基本类型//编译不通过
2:obj 为 null //返回false
3:obj 为 class 类的实例对象
4:obj 为 class 接口的实现类
5:obj 为 class 类的直接或间接子类
```

18：HTTP**协议的**Keep-Alive **模式** **https://www.jianshu.com/p/49551bda6619**

19：Json.parse()

```
JSON 通常用于与服务端交换数据。
在接收服务器数据时一般是字符串。
我们可以使用 JSON.parse() 方法将数据转换为 JavaScript 对象。
```

20:Js中：setTimeout() 方法用于在指定的毫秒数后调用函数或计算表达式。

21： **Iterator 迭代器越界** java.util.NoSuchElementException问题定位

```
Iterator i = set.iterator();
  while (i.hasNext()) {
   System.out.println(i.next());
   pw.println(i.next());
  }
改为：
while (i.hasNext()) {
   String ss = (String) i.next();
   System.out.println(ss);
   pw.println(ss);
  }
```

22：是否可以将一些客户端所需的基本信息且在后续经常用到 在服务器加载时候就注入常量中 省去后续数据库查询

23：http和websocket的长连接区别

https://www.cnblogs.com/Catherine001/p/8359153.html

24：WebSocket和HTTP的关系

https://blog.csdn.net/osle123/article/details/52755575

18:netty问题：

{Netty 提供了几种拥有相同编程接口的基本传输实现：

- 基于 NIO 的 TCP/IP 传输 (见 io.netty.channel.nio),
- 基于 OIO 的 TCP/IP 传输 (见 io.netty.channel.oio),
- 基于 OIO 的 UDP/IP 传输, 和
- 本地传输 (见 io.netty.channel.local).

}

1：

ChannelInitializer简介

https://www.cnblogs.com/stevenczp/p/7597903.html

```
1.channelActive() 方法将会在连接被建立并且准备进行通信时被调用。


2.ByteBuf 之所以没有这个flip()方法;因为有两个指针，一个对应读操作一个对应写操作。
当你向 ByteBuf 里写入数据的时候写指针的索引就会增加，同时读指针的索引没有变化。读指针索引和写指针索引分别代表了消息的开始和结束。

3.当一个写请求已经完成是如何通知到我们？这个只需要简单地在返回的 ChannelFuture 上增加一个ChannelFutureListener。
这里我们构建了一个匿名的 ChannelFutureListener 类用来在操作完成时关闭 Channel。
或者，你可以使用简单的预定义监听器代码：
```

```
    f.addListener(ChannelFutureListener.CLOSE);
```

4.ChannelHandler 有2个生命周期的监听方法：handlerAdded()和 handlerRemoved()。
你可以完成任意初始化任务只要他不会被阻塞很长的时间。

5：**@Sharable** 注解用来说明ChannelHandler是否可以在多个channel直接共享使用。

```
注意：ChannelHandlerContext.write() (和 writeAndFlush() )方法会返回一个
ChannelFuture 对象，
一个 ChannelFuture 代表了一个还没有发生的 I/O 操作。
这意味着任何一个请求操作都不会马上被执行，因为在 Netty 里所有的操作都是异步的。
举个例子下面的代码中在消息被发送之前可能会先关闭连接。
Channel ch = ...;
ch.writeAndFlush(message);
ch.close();
close() 方法也可能不会立马关闭，他也会返回一个ChannelFuture。
```

Handler执行顺序

1. 对于channelInboundHandler,总是会从传递事件的开始，向链表末尾方向遍历执行可用的inboundHandler。

2. 对于channelOutboundHandler，总是会从write事件执行的开始，向链表头部方向遍历执行可用的outboundHandler。

```
ch.pipeline().addLast(new OutboundHandler1());
ch.pipeline().addLast(new OutboundHandler2());
ch.pipeline().addLast(new InboundHandler1());
ch.pipeline().addLast(new InboundHandler2());
```

链表中的顺序为head->out1->out2->in1->in2->tail
那么Inbound的执行顺序为read->in1->in2

在Inbound执行write后，outbound执行顺序为out1<-out2<-write

1.所以实际使用中，如果添加的顺序不好，很可能会意外跳过某些inbount或者outbound。

所以建议实际使用上，先通过addFirst插入所有outBound

再通过addLast插入所有inBound

这样inBound与outBound的插入顺序与执行顺序完全一致，且不会出现跳过的情况

2.所以一些统一编码解码的handler，例如ssl，httpcodec，最好是按照顺序放在链表头！这样才会保证进出都会执行到并且业务逻辑可以正常插入

retain/release

每一个新分配的ByteBuf的引用计数值为1，
每对这个ByteBuf对象增加一个引用，需要调用ByteBuf.retain()方法，
而每减少一个引用，需要调用ByteBuf.release()方法。当这个ByteBuf对象的引用计数值为0时，表示此对象可回收。 这种机制可以很简单的进行对象的生命周期的管理