# Bigdecimal

**构造函数尽量用字符串形式 或者采用valueOf()初始化, 但是一定不要使用数字作为构造函数参数,**

**正确使用方式如下:**

BigDecimal correctWay1 = new BigDecimal("0.1");

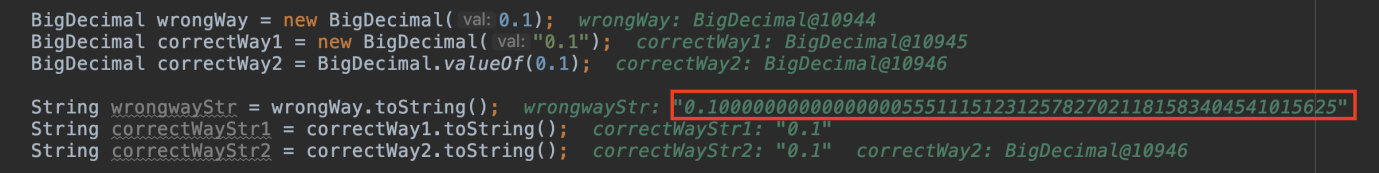
BigDecimal correctWay2 = BigDecimal.valueOf(0.1);

错误使用方式如下:

BigDecimal wrongWay = new BigDecimal(0.1);

原因: 采用上述构造方式，0.1是当成double传入的，double做科学计算的时候精度是有问题的

两种方式使用对比图



# 单例模式与多例模式

单例模式

类似静态类, 全局只有一个, 所以在 web应用中, spring 容器托管的组件内不能有私有属性(原因: 多个请求进来会修改当前的私有属性值) , 如果要有属性,可以不要用spring容器托管,作为一个纯 java 类使用 或者 使用多例模式

多例模式对象注入方式:

1. 对象工厂模式注入:

组件开启多例模式 Scope(value=”prototype”) , 调用方使用对象工厂获取spring容器托管的组件 ,此时拿到的组件都非同一个对象

@Service  
@Scope(value = "prototype")  
public class CategoryService {

}

@RestController  
@RequestMapping("/category")  
@Validated  
public class CategoryController {  
  
 @Autowired  
 private ObjectFactory<CategoryService> objectFactory;  
  
 @GetMapping()  
 public void test(){  
 CategoryService categoryService = objectFactory.getObject();  
 }

2. 动态代理模式注入:

类使用proxyMode = ScopedProxyMode.TARGET\_CLASS

接口使用 proxyMode = ScopedProxyMode.INTERFACES)

组件使用动态代理方式开启多例模式 Scope(value = "prototype", proxyMode = ScopedProxyMode.*TARGET\_CLASS*), 调用方获取spring容器托管的组件 ,此时拿到的组件都非同一个对象 (使用动态代理模式可以很好解决spring 中 ocp 问题)

@Service  
@Scope(value = "prototype", proxyMode = ScopedProxyMode.*TARGET\_CLASS*)  
public class CategoryService {

}

@RestController  
@RequestMapping("/category")  
@Validated  
public class CategoryController {  
  
 @Autowired  
 private CategoryService categoryService;  
  
 @GetMapping()  
 public void test(){  
 System.*out*.println(categoryService);  
 }

# 系统中浮点数价格处理几种方式

1. 向上取整, 如 1.8944 🡪 1.90

2. 直接舍去, 如 1.8944 🡪 1.89

3. 四舍五入,

4. 银行家模式: 奇进偶不进, 逢六进一

# ThreadLocal

ThreadLocal 可以很好的解决多线程安全问题，同一个ThreadLocal所包含的对象，在不同的线程中有不同的副本

1. 因为每个线程中有自己的ThreadLocal实例副本，且该副本只能由当前线程访问，所以就不存在线程间数据共享的问题
2. ThreadLocal 变量通常被private static修饰。当一个线程结束时，它所使用的所有 ThreadLocal 相对的实例副本都可被回收

ThreadLocal 为每一个线程都提供了一个变量副本,使得每一个线程在同一时间访问到的都不是同一个对象,这样在多线程环境中就隔离了数据的共享,

将一些非线程安全的变量以ThreadLocal存放，在同一次请求响应的调用线程中，所有对象所访问的同一ThreadLocal变量都是当前线程所绑定的。

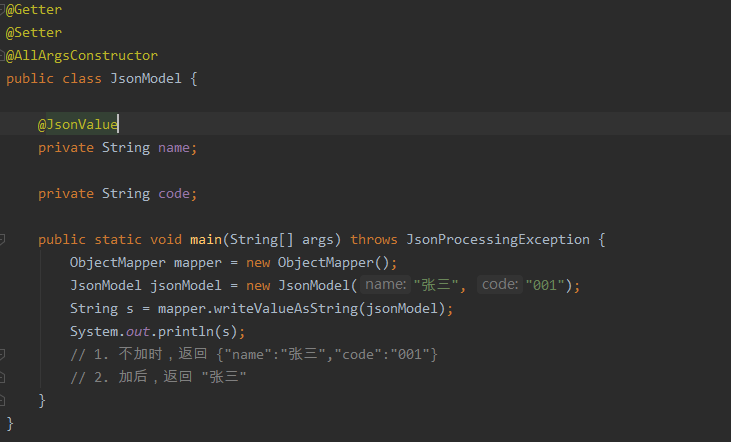
因为ThreadLocal与线程的生命周期共存, 通常线程池对于线程的管理都是采用线程复用的方法, 在线程池中的线程很难甚至永远都不会结束,这将意味着线程的持续时间不可预测,甚至于JVM的生命周期一致,如果每次都往同一个线程中存取数据,可能导致对象占用空间持续膨胀,导致内存泄漏, 所以需及时清除当前线程资源

参考文章: <https://www.jianshu.com/p/f956857a8304>

<https://www.cnblogs.com/luxiaoxun/p/8744826.html>

# FastJson

@JsonValue

1. 用在get或属性字段上，一个类中能用一个
2. 当加上@JsonValue, 序列化时，只返回一个字段的值

@JsonCreate

1. Json 反序列化时，是用的无参构造函数反序列化，如果没有无参构造函数，则会报错
2. @JsonCreate 指定一个方法或者有参构造函数反序列化

@JsonProperty

注解用于属性上，作用是把该属性的名称序列化为另外一个名称

@JsonFormat(shape = JsonFormat.Shape.OBJECT)

# 枚举的序列化和反序列化