06/18：RNA报告 工程优化 流水式并发 并行编程 leetCode5

06/26：RNA报告 leetcode5 并发第五章 mall+项目完善 编译码

L1: 如果后一个状态与前一个状态相加>0，则tmp延续

如果<0，状态无效，tmp置为0；

最后Math.max()一波

L392: 建表+跳跃 双指针 二维dp

7/14：leetcode3 改BPRNN的bug

「 二叉树常规操作、 」

7/24：\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**科研：**ComNet输出的相位和幅度误差大，调研基于MIMO的天线通道幅度相位误差矫正，再优化。现在对单通道测试效果良好，对于多通道有没有联合学习的前人工作？

**极客SB：**

**单数据源自动配置和手动配置所需的Bean：**

**数据源相关 【DataSource 根据选择的连接池决定】**

**事务相关 【PlatformTransactionMannger TransactionTemplate】**

**操作相关 【JdbcTemplate】**

**多数据源自动配置和手动配置所需的Bean：**

**自动——与SpringBoot协同工作（二选一）：**

**1 配置@Primary类型的Bean 2 排除SpringBoot自动配置**

**连接池Druid：**

**SQL防注入，定制数据库操作filter=conn, config, state, slf4j，监控（慢sql...）**

**内部**

**连接池选择的考量点**

**Sping JDBC操作类：**

**【1】常用的Bean注解 @Componet 通用 @Repository 数据库操作 @Service 业务服务**

**@Controller SpringMVC @RestControlle restfulWebService**

**【2】 Bean\_1 JdbcTemplate —— RowMapper 对象查询**

**【3】 Bean\_2 namedParameterJdbcTemplate**

**【4】 单sql处理 批量sql处理**

**【5】 JPA是规范 Hibernate(ORM) Mybatis(ORM) spring-data-jdbc（简化JDBC）**

**底层都是由JDBC实现**

**事务抽象**

**PlatformTransactionMannger 不同框架有不同的Mannger**

**TransactionDefinition 事务传播特性7个 隔离特性 超时设置 是否只读**

**事务的传播特性：规定事务方法和事务方法发生嵌套调用时事务如何进行传播。**

**service1.method1 -> service2.method2**

**【REQUIRED】  
service1：REQUIRED, service2:REQUIRED  
REQUIRED:支持事务，没有事务时创建新的事务，如果存在事务则直接加入该事务执行  
method1执行时没有事务，则会新建事务Tx1,在method1中调用method2时，发现已经存在事务，则会加入事务执行，最后执行commit。**

**【NOT\_SUPPORTED】  
service1：NOT\_SUPPORTED, service2:REQUIRED  
NOT\_SUPPORTED：以非事务方式执行，如果当前存在事务则将当前事务挂起  
method1执行不会创建事务，执行method2时会创建新的事务，执行sql之后会进行commit**

**service1: NOT\_SUPPORTED,service2:NOT\_SUPPORTED  
两个方法都不会创建事务执行  
NOT\_SUPPORTED相当于没有Spring事务，每条执行语句单独执行，单独提交**

**【REQUIRES\_NEW】  
REQUIRES\_NEW：不管是否存在事务，都创建一个新的事务，原来的方法挂起，新的方法执行完毕后，继续执行老的事务  
service1:REQUIRED,service2:REQUIRES\_NEW  
执行method1时会创建新的事务Tx1，执行method2时会先将Tx1挂起，然后创建新的事务Tx2,执行完后Tx2进行commit,然后将Tx1解除挂起，Tx1进行commit  
REQUIRES\_NEW为当前方法创建一个新的事务，并且当前事务先提交，然后再提交老的事务。  
【新的事务提交成功之后，老的事务提交失败时新的事务的sql不会回滚】  
【新的事务提交失败回滚，老的事务是否会进行回滚？】应该会的**

**【MANDATORY】  
必须在一个已有的事务中执行，否则报错  
service1: NOT\_SUPPORTED, service2:MANDATORY  
method1不会创建事务执行，method2执行时检查当前没有事务，则会抛出异常不会执行，method1已经在数据库层面执行成功并完成commit,所以抛出的异常不影响method1的执行结果  
MANDATORY必须在已有事务下被调用，否则报错**

**【NEVER】  
须在一个没有事务中执行，否则报错  
service1:REQUIRED,service2:NEVER  
method1执行时没有事务，则会新建事务Tx1,method2执行时检查到存在事务，则会进行报错抛出异常，method1由于还没有提交事务，则会进行回滚。**

**【NESTED】  
如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行与REQUIRED类似的操作。  
service1:REQUIRED,service2:NESTED  
method1执行时没有事务，则会新建事务Tx1,执行sql后不会提交，method2创建一个NESTED嵌套事务执行sql，然后一起提交  
method1方法创建一个事务，则再调用method2方法时，直接在该事务的基础上创建一个嵌套事务，本质上还是同一个事务，做一次提交；  
service1:NOT\_SUPPORTED,service2:NESTED  
method1执行不会创建事务，sql执行在数据库commit,method2会创建一个事务执行sql后commit。  
method1方法不创建事务，则调用method2方法时，直接创建一个新的事务，单独提交。**

**事务、sql语句和连接池连接数的关系：**

**事务是在连接上的，一个事务开启后的多条SQL都是执行在这一个连接上的，不会更换连接。 没加事务，大部分情况下就是各条SQL单独的事务，分别对应一个连接。**

**事务隔离特性：**



**事务特性设置：**

**编程式事务 TransactionTemplate 【TransactionCallback】**

**PlatformTransactionManager 【TransactionDefinition】**

**声明式事务 AOP拦截（动态代理和静态代理）用代理对象调用本类增强后的方法**

**XML / 基于注解的配置方式**

**Spring AOP用一个方法调用被事务声明的方法，被调用方法的事务会失效，原因是同级无法创建代理对象，从而无法调用增强方法。**

**声明式事务的本质是通过AOP来增强类的功能，为类做了一个代理，看似在调用自己写的类，实际是用的增强后的代理类。**

**解决方案: 注入实例，内部方法调用改为直接调用实例的方法。**

**获取代理对象（AopContext.currentProxy()）.方法()**

**异常抽象：**

**Spring将数据操作的异常全部转换为DataAccessException**

**多数据源、分库分表、读写分离的关系 数据库中间件**

**7/26\_\_\_晴\_\_：**

**极客SB：**

**O/R Mapping(对象关系映射)框架**

**Hibernate成为JPA（提供了一种基于POJO的持久化模型）标准的实现**

**LazyInitializationException：懒加载异常**

**(加事务) 调用Service方法操作数据库时，如果在方法前不加事务就会报异常。Hibernate给我们做了LazyLoading优化，在使用时才会去加载。没有事务时，数据库操作后连接后还回去了，Session关闭，那么之后访问该操作拿到的items自然就会因为没有session而报错。加了事务，保证Session连接在提交或者回滚前都会持有，Session也在，自然就不报错。**

(加多对对或多对一的映射关系)我们在开发的时候经常会遇到延迟加载问题，在实体映射时，多对一和多对多中，多的一样的属性默认是lazy="true"(即，默认是延迟加载)**也可以设置@ManyToMany的fetch，默认是LAZY。ID是直接加载，不缓存，所有取ID的数据操作不会受懒加载影响。**

**其他解决：**<https://www.cnblogs.com/loaderman/p/10037878.html>

**Mybatis：**

**insert、update、delete都是返回影响结果的条数，Options(useGeneratedKeys=True)可以将ID填入到mapper入参的实体类实例。type-handler拦截并实现字段和对象域的类型转换（Money类型转换）。**

**辅助工具：Mybatis Generator、Mybatis pageHelper**

**NoSQL实战**

**Docker辅助开发：**

**容器比虚拟机轻量，简化了重复搭建开发环境的工作。**

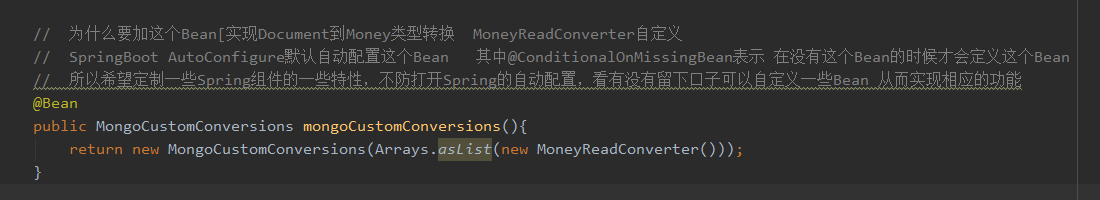
**Spring中访问MongoDB：Spring Data MongoDB**

**NoSQL一般分为四大块：**

**Key-Value类型、文档型、列存储、图数据库**

**MongoDB是一款开源的文档型数据库。**

**类型转换/自定义组件特性实现：**



**MongoDB Template和Repository对于数据库操作的区别**

**Redis【缓存】：**

**开源的内存KV存储、支持多种数据结构**

**Spring对Redis存储：**

**Spring Data Redis：**

**支持的客户端Jedis/Lettuce RedisTemplate Repository支持**

**Jedis客户端的：**

**Jedis不是线程安全的，即不能在多个线程中共享Jedis实例**

**JedisPool来做一个数据库连接池的效果，通过Jedis获得Jedis实例。**

**Redis哨兵模式和集群模式：**

**注意版本控制 最好都是一致的**

<https://juejin.im/post/6844903661223542792>

**可以在单线程/进程中处理多个事件流的方法呢？IO多路复用。Redis就是单线程模型。**

<https://zhuanlan.zhihu.com/p/115220699>

**读写分离：。。。**

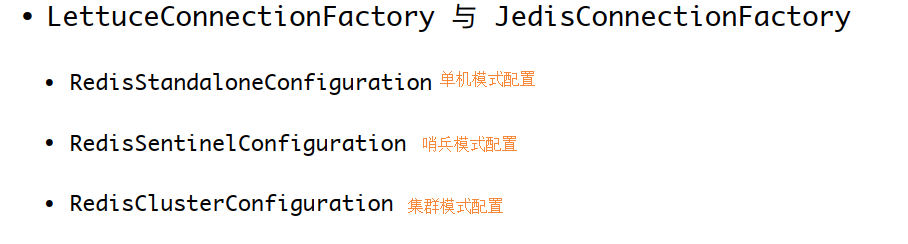
**Spring的缓存抽象【缓存】：**

**读写比比较高，考虑使用缓存。**

**基于注解的缓存@EnableCaching**

**Redis在Spring中的其他应用：**

**新版本Redis已经用Lettuce（支持读写分离。1 只读主节点、只读从 2 优先读主、优先读从）代替Jedis（不支持读写分离）**



**Redis Template、Repository和Cachexxx：**

**Template 万能 需要手工操作**

**Repository 很少用到，不会像操作数据库一样操作Redis，如果针对一些数据有二级索引的需求，可以用。**

**Cachexxx 用在把某个方法返回值缓存的情况，可以不用自己写。**

**处理不同类型数据源的Reposity，如何区分这些Reposity：**

**根据实体的注解【@Entity @Document @RedisHash】**

**根据继承的接口类型**

**扫描不同的包【不同方式放在不同的包下】**

public class Coffee extends BaseEntity implements Serializable {  
 private String name;  
 @Type(type = "org.jadira.usertype.moneyandcurrency.joda.PersistentMoneyMinorAmount",  
 parameters = {@org.hibernate.annotations.Parameter(name = "currencyCode", value = "CNY")})  
 private Money price;  
}

**2020/8/5\_\_晴天暴晒\_\_：**

**数据访问进阶**

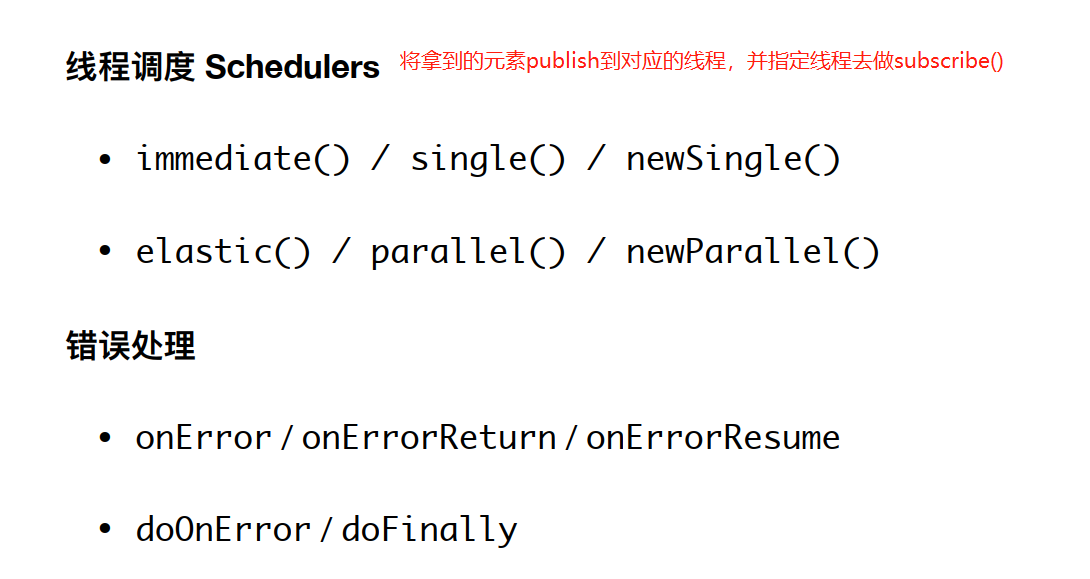
**Reactive Programming（响应式编程）： 面向数据流和变化传播的编程范式。**

**Project Reactor：基础使用 --支持并发操作数据库**

**Flux[0..N] — N个元素的序列 Mono[0..1] — 0或1个元素**

**直到出现subscribe()时，之前的代码才开始执行。**

**onRequest()、onCancel()、onDispose()控制订阅者一次性订阅的元素，不需要全部请求到。**



**通过Reactive的方式访问Redis：**

**Lettuce客户端支持Reactive方式**



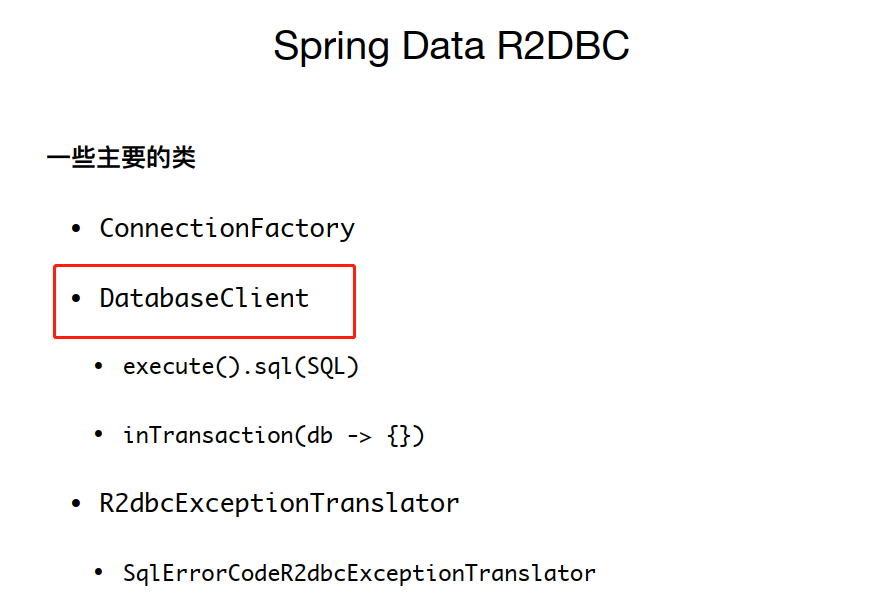
**通过Reactive的方式访问MongoDB：**



**CountDownLatch控制主线程等待子线程执行完后结束。**

**通过Reactive的方式访问RDBMS(传统型关系数据库)：**

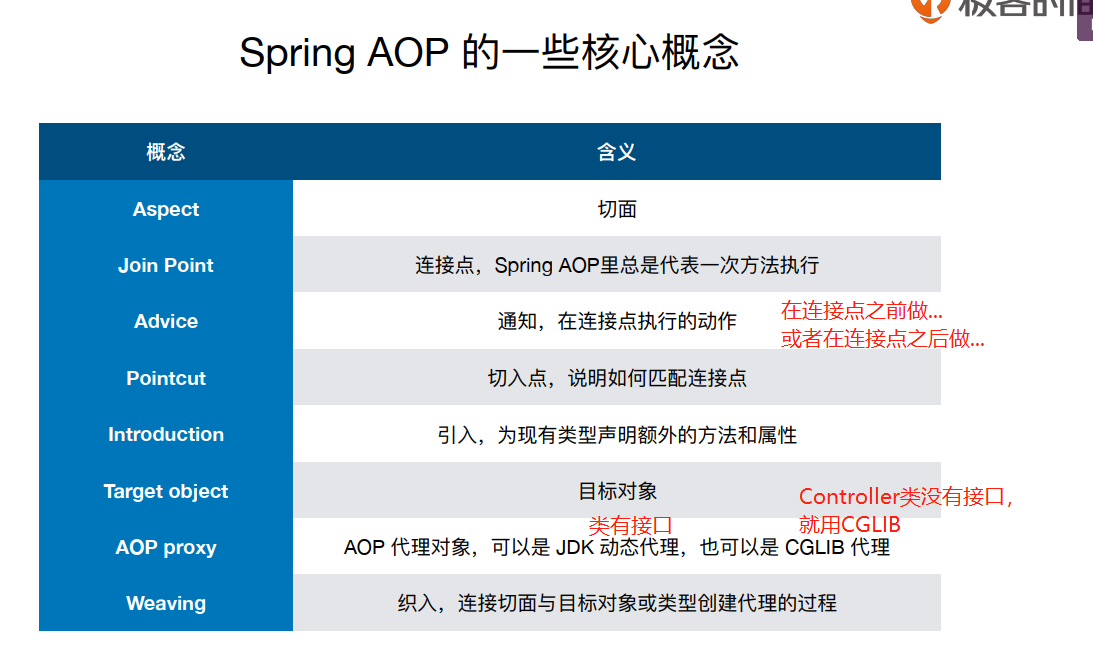
**Spring Data R2DBC 或者 Repositery**





**AOP打印数据访问层的摘要**

**Spring AOP：**



**声明式事务@EnableTransactionManagement由AOP实现**

**如何打印SQL：监控你的Spring boot数据库操作**

**HikariCP：使用P6Spy外部库 配置data-P6Spy**

**Alibaba Druid：内置SQL输出 或者 log4j2进行日志输出**

**@EqualsAndHashCode()**

**2020/8/13：\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Keras使用中的花式技巧：**

**自定义loss:**

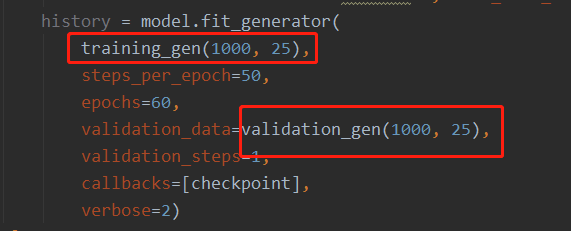
1. 两个入参(y\_true, y\_pred)的简单损失函数
2. 多个入参的复杂损失函数

def my\_loss(y\_true, y\_pred, another\_input\_01, another\_input\_02, ...)

model.addloss()

**注意代码缺失：**



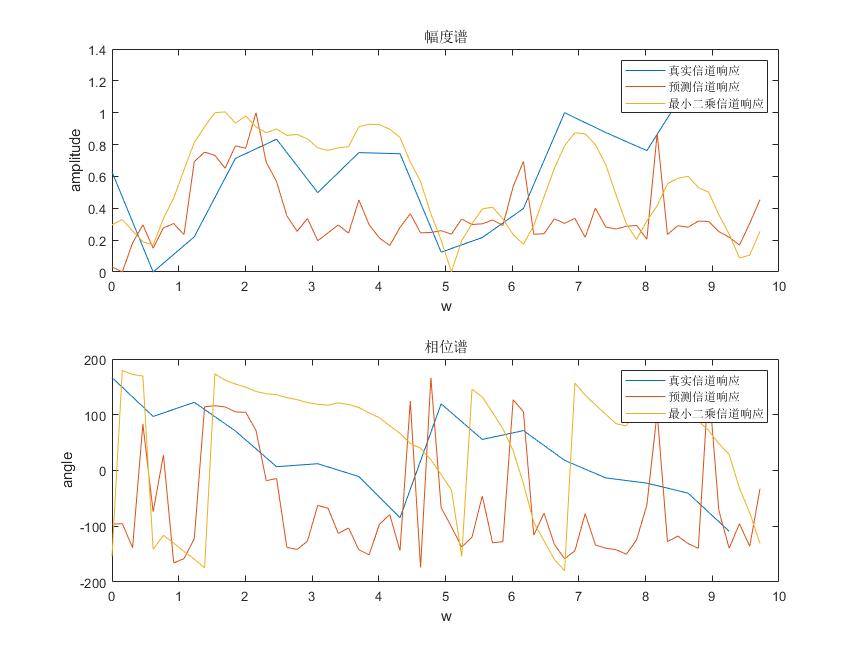


**更随意的metric等等...详情见链接:**

<https://www.cnblogs.com/gczr/p/12421412.html>

<https://www.jianshu.com/p/4283c25f2a8c>

**科研：加入正则化后效果并不理想，BER也比原来高不到0.04。**

****

**算法：复杂度分析**

**2020/8/14：**

**极客SB:**

**[1]Hibernate的常用注解**：

@Entity @Table @Id @GeneratedValue @Column

@OneToMany @ManytoOne @ManytoMany

**[2] @EqualsAndHashCode: 配合继承+@Data**

<https://blog.csdn.net/qq_27093465/article/details/90056695?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase>

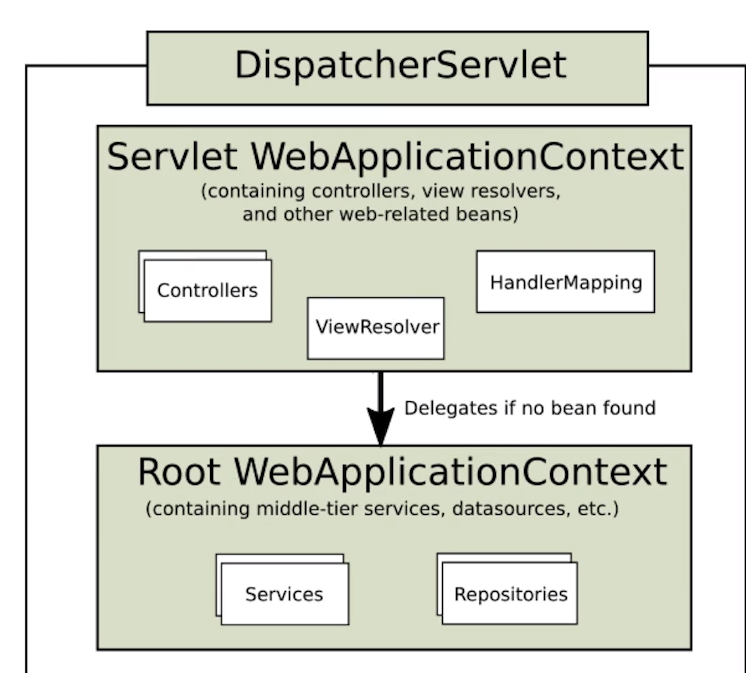
**[3] 出现No identifier specified for entity 错误：**

**导包错误**：org.springframework.data.annotation.Id & import javax.persistence.Id

**Spring MVC**

**[4] SpringBoot上下文：用于管理所包含组件的生命周期**

**（Servlet【子上下文】+Root【父上下文】）**



**各个context相互独立，每个context的aop增强只对本context的bean生效。如果想将切面配置成通用的，对父和子上下文的bean均支持增强，则：**

1. **切面fooAspect定义在父上下文。**
2. **父上下文和子上下文，均要开启aop的增强，即 @EnableAspectJAutoProxy 或<aop: aspectj-autoproxy /> 的支持。**

**因为存在委托机制，在自己的 context 中找不到 bean，会委托父 context 查找该 bean。**

**那么在子上下文中getBean()查找在父上下文【增强】定义的bean，就会产生增强效果。**

**上下文配置：编程式、xml定义上下文 、 定义父子关系。**

**一个spring应用可以存在多个web上下文，但是只有一个root上下文。**

**例子：geektime context-hierarchy**

1. **@Configurable & @Configuration**

一般情况下，我们使用Spring Ioc容器为我们管理的对象，并注入依赖。但是，如果不是Spring Ioc容器管理的类，我们可以注入吗？非Spring容器管理的对象是指的由我们自己new出来的对象。想让Spring Ioc容器注入依赖的话，可以使用\*@Configurable\*注解

**@Configurable: 解决new 对象()的依赖注入问题**

<http://shzhangji.com/cnblogs/2015/09/12/model-dependency-injection-with-spring-aop/>‘

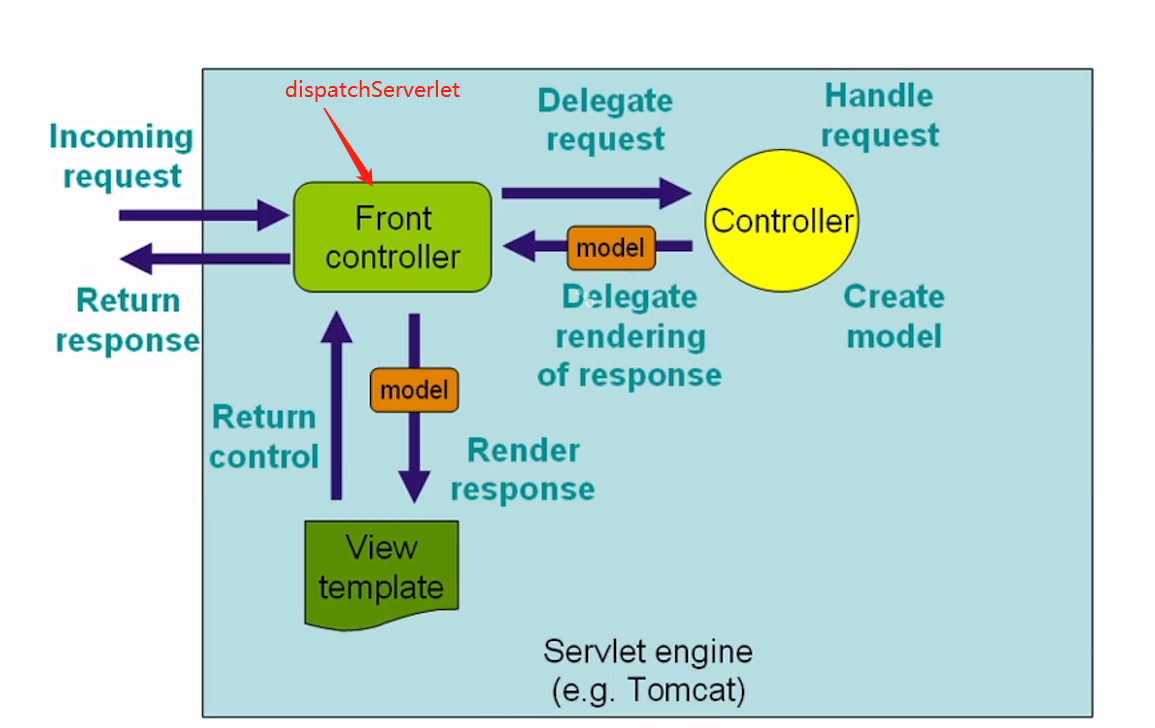
<https://zhuanlan.zhihu.com/p/61587264>

<https://blog.csdn.net/u013066244/article/details/89061676>

**@Configuration: 相当于xml中的< beans>**

和AnnotationConfigApplicationContext功能类似

**【5】Debug dispathchServelt源码 请求处理机制 Render视图呈现**



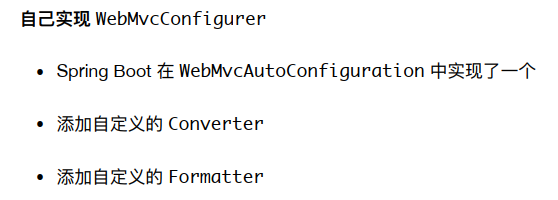
**图x、请求处理流程图**



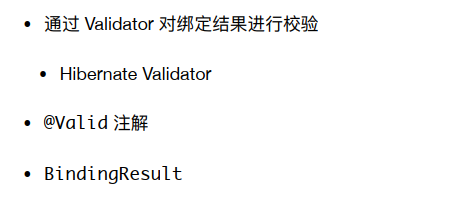
**[6] Debug dispathchServelt源码 定义处理方法**

**【类型转换】**

**自定义的Coverter注册为bean，送给spring IOC容器管理，就会在post请求正文时，自动识别字段类型和转换为需要的类型。**



**【定义校验】**



**【multipart上传】**

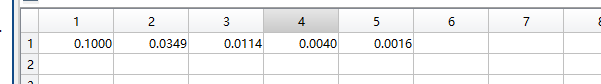


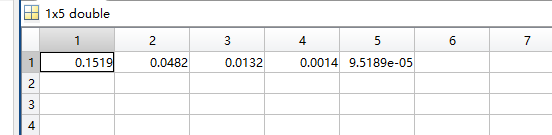
**【7】Debug dispathchServelt源码 视图解析机制 Render**

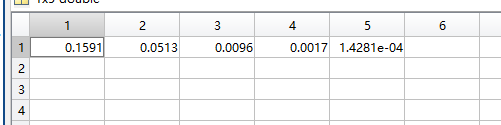
**ModelAndView 返回视图 处理流程 ResponseBody 返回响应正文 处理流程**

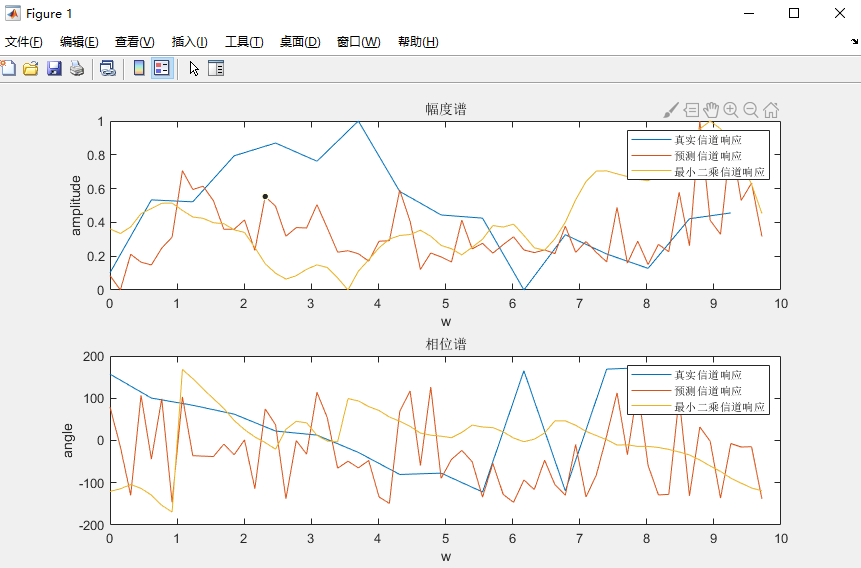
**重定向：redirect:: 客户端发起 uri会改变 forward:: 服务端发起 uri不会改变**

**科研： 实验结果**









**2020/08/21:**

**Spring MVC**

**极客SB:**

【Spring MVC的常用视图】

视图列表 支持的模板引擎(freemark thymeleaf等)

格式化响应正文：

MessageCoverter 自动查找HttpMessageConverters 进行注册

支持Jackson 1、@JsonComponet注册 序列化和反序列器

2、Jackson2ObjectMapperBuilderCustomizer

@RestController = @Controller + @ResponseBody

如果不只有响应正文，还有视图返回，那么加@Controller，更加灵活

【静态资源和缓存】

@JsonIgnoreProperties(value={“hibernateLazyInitializer”})

或者 Jackson-datatype-hibernate5

作用，不加对hibernate框架产生什么问题？

在hibernate框架中，当我们要访问的数据量过大时，明显用缓存不太合适，因为内存容量有限，为了减少并发量，减少系统资源的消耗，这时Hibernate用懒加载机制来弥补这种缺陷，但是这只是弥补而不是用了懒加载总体性能就提高了。懒加载也被称为延迟加载，它在查询的时候不会立刻访问数据库，而是返回代理对象，当真正去使用对象的时候才会访问数据库。 那么我们在定义完pojo对象后，hibernate会为一对多，，多对多中，多的一样的属性对应的对象生成代理对象。如果我们去查询该pojo并且json序列化返回时，由于无法序列化映射关系中多的一方的属性【此时是代理对象】，所以会报懒加载异常。

1. 维持session常开，可以序列化，不过多的一方的属性是空的。不提取一对多中的数据，不过因为session常开，所以不会报错。
2. 在一对多的关系中，加入@JsonIgnore，这样在转换的时候就会过滤掉这个对象。
3. 配置fetch模式，强制提取一对多关系中的数据，生成很多无用的数据。

2020/09/05:

极客SB:

【**SpringMVC的异常解析】**

**定义方法为异常处理方法：@ExceptionHandler**

**添加位置： @Controller / @RestController**

**【在拦截方法处添加异常】 @ControllerAdvice/@RestControllerAdvice**

**Java8中的重要语言特性： Lambda, Stream**

**针对需要@ResponseBody和ResponseEntity的情况**

**选择ResponseBodyAdvice**

**针对异步请求的接口：AsyncHandlerInterceptor**

**拦截器配置：添加** WebMvcConfigurer.addInterceptors()

拦截器、AOP、过滤器之间的区别和联系。

**访问Web资源**

**SpringBoot框架写客户端请求**

**RestTemplate各种用法 RestTemplate的简单定制**

**WebClient的基本用法**

**一个以Reactive方式处理Http请求的非阻塞的客户端【】**

【常见的会话解决方案】

粘性会话 Sticky Session 使来自一个用户的会话尽可能它的请求落到同一台机器上面，这样就可以把分布式Session变成一个单机的Session.

会话复制 Session Replication 把每台机器上的会话都做复制，这样集群上每台机器都持有相同的一些会话信息。不管请求落在哪一台上面都可以取得Session.

集中会话 Centralized Session 使用JDBC或者Redis集中存储我们的会话信息，只要有相同的JESSIONID，那么我们就可以把这些Session从集中的会话里面取出来。不管在哪台机器上，只要有相同的JESSIONID，那么就能保证它能取到相同的会话。

【SpringSession】

【WebFlux】当传统的SpringMVC需要200个线程做并发时，WebFlux仅仅需要更少的线程和内存实现相同的并发支持。

【重新认识SpringBoot】

1. SpringBoot的组成部分
2. 自动配置 (1) 基于添加的JAR依赖自动对SpringBoot应用添加配置

（2）SpingBoot-autoconfiguration

开启自动配置：@EnableAutoConfiguration (排除自动配置类exclude=Class<?>[])

@SpringBootApplication

定制自动配置： (1) 编写Java Config @Configuration (2) 添加条件判断

@Conditional 只有条件全都有效，才能继续执行下去。

1. 定位自动配置 META-INF/spring.factories

自动配置的执行顺序

@AutoConfigureBefore @AutoConfigureAfter @AutoConfigureOrder

1. 失败分析器 failureAnalyzer
2. 低版本Spring 3.x的Spring没有条件注解（@Conditional），无法自动定位需要加载的自动配置(@EnableAutoConfiguration)如何实现自动配置？

对于SpringBoot而言，它会自动寻找带有@SpringApplication注解这个类的package以下的这些package里面的所有的配置@Configuration。如果package不是上述的关系，那么可以增加@CompontScan。

1. 起步依赖及其实现原理

Maven依赖管理mvn--dependency:tree 解决依赖冲突 mvn的duplicate-finder插件

SpringBoot的起步依赖，Starter Dependencies直接面向功能，一站获得所有相关依赖，不再复制粘贴 spring-boot-starter-\*

【定制Stater Dependencies】

1. 深挖SpringBoot的配置加载机制

PropertySource属性源配置

xml方式<context:property-placeholder>

注解方式@PropertySource

集合@PropertySources

SpringBoot提供的@ConfigurationProperties方式配置

【定制PropertySource】



**运行中的SpringBoot，SpringBoot运维**

**SpringBoot内置容器：【容器替换】**

**spring-boot-starter-tomcat，…-jetty，…-undertow，…-reactor-netty**

**定制Web容器运行参数**

**配置容器支持HTTP/2**

**编写命令行运行的程序：关闭Web容器，设置WebApplicationType为none**

**【微服务和云原生】**

**【服务注册与发现】**

**Eureka来做服务的注册中心**