# ssm框架

### SpringMVC的运行原理



（1）发起请求到前端控制器DispatcherServlet

（2）前端控制器请求handlerMapping查找handler（xml、注解）

（3）处理器映射器向前端控制器返回handlerExecutionChain(处理器执行链，包含了拦截器和handler)

（4）前端控制器调用处理器适配器去执行handler

（5）处理器适配器去执行handler

（6）handler执行完成之后，给适配器返回ModelAndView

（7）处理器适配器向前端控制器返回ModelAndView

（8）前端控制器请求视图解析器去进行视图解析（生成jsp）

（9）视图解析器向前端返回view

（10）前端控制器进行视图渲染

（11）返回相应结果

### Springmvc的注解

@restControl-----------类上，responseBody+control

@control-----------扫描

@requestMapping----------------加在类和方法上,类的路径+方法路径

@requestParamete---------------形参上,请求参数可省略（名称一致）

@pathvariable-----------------形参上，必须写，springmvc的restful风格,路径参数

@requestBody----------------形参上，封装前端的参数为java对象

@responseBody---------------方法上，把java对象解析为json传递到前端，异步请求一定要加！！！！

SpringMVC三大组件:处理器映射器(handlerMapping)，处理器适配器（handlerAdapter）,视图解析器（viewResolver）

需要用户开发的组件为handler和view

00

### 全局异常处理如何实现

为了区别不同的异常通常根据异常类型自定义异常类，这里我们创建一个自定义系统异常，如果controller、service、dao抛出此类异常说明是系统预期处理的异常信息。

自定义异常处理器实现HandlerExceptionResolver接口,在实现方法内做判断是否为自定义异常

系统的dao、service、controller出现都通过throws Exception向上抛出，最后由springmvc前端控制器交由异常处理器进行异常处理

自定义一个异常类,实现handlerExceptionResolve并实现resolvelException方法

在springmvc.xml中配置这个异常bean对象

### 拦截器如何实现的

自定义一个拦截器类,实现handlerInterceptor并实现preHandler(执行前),postHandler(执行handler之后,返回modlerAndView之前),afterComplication(modlerAndView之后)这三个方法

在springmvc.xml中配置拦截器

应用场景:前台用户登陆验证

当在web.xml文件拦截路径为”/”,表示拦截所以资源，此时需要在spring.mvx中配置静态资源映射

### Spring事务管理

1. 事务的4个特性：原子性，一致性，隔离性，持久性
2. 两种事务管理方式：声明式和编程式。我们一般采用声明式事务管理。
3. 声名式事务：基于AOP技术实现的声明式事务管理，实质就是：在方法执行前后进行拦截，然后在目标方法开始之前创建并加入事务，执行完目标方法后根据执行情况提交或回滚事务。

声明式事务管理又有两种方式：基于XML配置文件的方式；另一个是在业务方法上进行@Transactional注解，将事务规则应用到业务逻辑中

### Spring的事务传播行为

PROPAGATION\_REQUIRED--支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。

PROPAGATION\_SUPPORTS--支持当前事务，如果当前没有事务，就以非事务方式执行。

PROPAGATION\_MANDATORY--支持当前事务，如果当前没有事务，就抛出异常。

PROPAGATION\_REQUIRES\_NEW--新建事务，如果当前存在事务，把当前事务挂起。

PROPAGATION\_NOT\_SUPPORTED--以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

PROPAGATION\_NEVER--以非事务方式执行，如果当前存在事务，则抛出异常。

Hibernate架构，对象有三种状态

瞬时态(Transient)、持久态(Persistent)、脱管态(Detached)。

瞬时态：没有oid，不被session关联

持久态：有oid，被session关联

脱管态：有oid，不被session关联

Hibernate中什么是延迟加载，延迟的作用是什么？

延迟加载机制是为了避免一些无谓的性能开销而提出来的，所谓延迟加载就是当在真正需要数据的时候，才真正执行数据加载操作。在Hibernate中提供了对实体对象的延迟加载以及对集合的延迟加载，另外在Hibernate3中还提供了对属性的延迟加载。

Hibernate的缓存

Hibernate的Session就是一种缓存，我们通常将之称为Hibernate的一级缓存，当想使用session从数据库中查询出一个对象时，Session也是先从自己内部查看是否存在这个对象，存在则直接返回，不存在才去访问数据库，并将查询的结果保存在自己内部。Session关闭后一级缓存会消失

Hibernate的二级缓存就是要为Hibernate配置一种全局缓存，让多个线程和多个事务都可以共享这个缓存。二级缓存是独立于Hibernate的软件部件，属于第三方的产品，多个厂商和组织都提供有缓存产品，例如，EHCach等等

Hibernate 和 JDBC 的优缺点

Hibernate就是封装了JDBC，他可以写一条hql语句，可以再不同数据库中使用，不用修改hql语句，但是关联查询效率低。

JDBC是基础的链接数据库的框架，效率高，但是mysql、oracle、sql service等不同的数据库要写不同的sql语句。

SSH三者作用

Struts在项目中的作用

**Struts 在项目主要起控制作用**，只要用于web层（即视图层和控制层）

Struts本身是使用典型的MVC结构实现的，项目中使用了struts之后就等于项目也是一个MVC结构，使项目结构更清晰，分工更明细。

**Struts在项目中主要负责视图层、控制层**，在视图层提供了大量的标签库，提高视图层的开发速度。在控制层使用中央控制器（Actionsupport）和配置文件(struts.xml)，**实现拦截用户请求、封装请求参数及页面导航**。

Spring在项目中的作用

**Spring是一个全方位的整合框架，在项目中对hibernate和struts进行整合，解决层与层之间的耦合问题**。

Spring的作用贯穿了整个中间层，将Web层、Service层、DAO层及PO无缝整合。

**Spring的IoC**来实现组件之间的依赖关系注入，上层框架不会渗透到下层组件，提高组件移植性和重用性，使得程序更灵活，上层框架不依赖实现而是依赖于抽象(委托接口)、使得实现类的变化不会影响上层组件，解决了层与层之间的藕合带来的维护或升级困难

**SpringAOP** 是面向方面编程，实现事务处理、日志服务等与业务无关的系统服务，实现插件式编程。

Hibernate在项目中的作用

**Hibernate应用于数据持久化层，是对JDBC的轻量级封装**。是一种对象、关系的映射工具， 提供了从 Java类到数据表的映射，也提供了数据查询和恢复等机制， 大大减少数据访问的复杂度。把对数据库的直接操作 , 转换为对持久对象的操作。

在项目中的主要作用就是1、**解决程序与数据库的依赖**，即使用了Hibernate之后，以后更改数据库不需要更改代码，因为Hibernate会根据数据库方言来生成对应的SQL语句；2、是**对JDBC的轻量级封装，简化持久层的代码，提高开发速度**。

### 简单介绍下springMVC和struts2的区别有哪些

1. springmvc的入口是一个servlet即前端控制器，而struts2入口是一个filter过虑器。

2. springmvc是基于方法开发(一个url对应一个方法)，请求参数传递到方法的形参，可以设计为单例或多例(建议单例)，struts2是基于类开发，传递参数是通过类的属性，只能设计为多例。

3. Struts采用值栈存储请求和响应的数据，通过OGNL存取数据， springmvc通过参数解析器是将request请求内容解析，并给方法形参赋值，将数据和视图封装成ModelAndView对象，最后又将ModelAndView中的模型数据通过reques域传输到页面。Jsp视图解析器默认使用jstl。

分布式：一个业务分拆多个子业务，部署在不同的服务器上

集群：同一个业务，部署在多个服务器上

举例：就比如新浪网，访问的人多了，他可以做一个群集，前面放一个响应服务器，后面几台服务器完成同一业务，如果有业务访问的时候，响应服务器看哪台服务器的负载不是很重，就将给哪一台去完成。

而分布式，从窄意上理解，也跟集群差不多， 但是它的组织比较松散，不像集群，有一个组织性，一台服务器垮了，其它的服务器可以顶上来。

分布式的每一个节点，都完成不同的业务，一个节点垮了，哪这个业务就不可访问了。

Sql优化

1. 可以对主键和外键建立索引
2. 避免对列进行操作。把数据库函数，计算表达式等这些操作放到等式的右边
3. 避免一些潜在的数据类型转化。比如字符型数据和数值型数据比较，会自动调用to number()函数，从而导致全局扫描
4. 避免使用IN OR NOT IN <> 。在where子句中使用这些关键字会使索引失效
5. 去掉where子句中的is null和is not null，。不然会让索引失效
6. 使用存储过程，实现读写分离

# Struts2

Struts2的执行流程

1：从客户端发出请求（HTTPServletRequest）。

2：请求经过各种过滤器（filter）

3：经过各种过滤器后，到达核心控制器FilterDispatcher，被调用后，通过struts.xml配置文件中Action标签中的映射来确定调用的Action方法，

4：当确定了调用的Action之后会找到反射生成的指定Action的代理对象（ActionProxy），

5：ActionProxy会通过配置文件信息找到指定的Action，

6：然后会经过系统和自定义拦截器栈，拦截器中会通过判断有执行invocation.invoke()返回一个字符串，来连接到下一个拦截器或者到达执行Action。

7：action执行完毕后，会返回一个字符串，该字符串与配置文件Action中result的name属性相对应，将处理得到的数据封装到此视图来，

8：在依次出了拦截器栈之后，将数据响应（HttpServletResponse）到客户端。

Struts2和servlet 的优缺点

Servlet属于tomcat集成的，加载速率快。获取参数只能通过getparemeter,需要封装。可以通过jsp域对象。单例的，值不会清除，不安全。

Struts2属于外部集成，并且加载后需要执行18个拦截器效率低。优点是以一种面向切面思想处理除业务逻辑外的一些操作，如国际化，字符编码集的操作，事务的操作。参数的获取方式比servlet多，可以通过属性驱动，模型驱动。Ognl表达式必须压入到值栈中才能返回到前端。多例的用完直接销毁，安全的。重定向，调转的方式，有4种

拦截器与过滤器的区别 ：

过滤器，是在java web中，你传入的request,response提前过滤掉一些信息，或者提前设置一些参数，然后再传入servlet或者struts的 action进行业务逻辑，比如过滤掉非法url（不是login.do的地址请求，如果用户没有登陆都过滤掉）,或者在传入servlet或者 struts的action前统一设置字符集，或者去除掉一些非法字符

拦截器，是在面向切面编程的就是在你的service或者一个方法，前调用一个方法，或者在方法后调用一个方法比如动态代理就是拦截器的简单实现，在你调用方法前打印出字符串（或者做其它业务逻辑的操作），也可以在你调用方法后打印出字符串，甚至在你抛出异常的时候做业务逻辑的操作。

1. 拦截器是基于java的反射机制的，而过滤器是基于函数回调。
2. 拦截器不依赖与servlet容器，过滤器依赖与servlet容器。
3. 拦截器只能对action请求起作用，而过滤器则可以对几乎所有的请求起作用。
4. 拦截器可以访问action上下文、值栈里的对象，而过滤器不能访问。
5. 在action的生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化时被调用一次

**执行顺序** ：过滤前 - 拦截前 - Action处理 - 拦截后 - 过滤后。个人认为过滤是一个横向的过程，首先把客户端提交的内容进行过滤(例如未登录用户不能访问内部页面的处理)；过滤通过后，拦截器将检查用户提交数据的验证，做一些前期的数据处理，接着把处理后的数据发给对应的Action；Action处理完成返回后，拦截器还可以做其他过程(还没想到要做啥)，再向上返回到过滤器的后续操作。

线程中的锁

悲观锁(Pessimistic Lock), 顾名思义，就是很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会block直到它拿到锁。传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁。

乐观锁(Optimistic Lock), 顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制。乐观锁适用于多读的应用类型，这样可以提高吞吐量，像数据库如果提供类似于write\_condition机制的其实都是提供的乐观锁。

死锁：是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。

# Mybatis

### Mybatis的工作原理

sqlsessionfactoryBuilder类加载核心配置文件（sqlmapConfig.xml）parse方法会产生会产生一个configration对象, configration对象通过构造方法的方式交给sqlsessionfactory对象。sqlsessionfactory产生sqlsession对象，并且将configuration对象通过sqlsession 构造方法的方式传给sqlsession对象。Sqlsession通过configration对象拿到excutor执行器，调用select方法时，底层该方法会从configuration对象中去找sql语句（mappedstatement）,执行并返回结果集.

### mybatis与spring整合

导入jar

ApplicationContext中配置：sqlsessionFactoryBean的配置，实体类别名包扫描，接口和mapper配置文件的包扫描

### Hibernate与mybatis区别

1、Hibernate全自动orm框架，mybatis是半自动化的orm框架

2、**mybatis动态sql**

Dao代理模式

5个一致:

namespace,

接口方法名和statmentid一致

ParmeType

ResultType

包扫描接口文件

代理对象=sqlsession.GetMapper.(类.calss)

### #{},${}区别

#可以作为占位符

$为字符串的拼接

为了防止sql注入,我们使用#.

### Mapper.xml配置文件

1、 Mapper.xml文件中的namespace与mapper接口的全限定名相同。

2、 Mapper接口方法名和Mapper.xml中定义的每个statement的id相同

3、 Mapper接口方法的输入参数类型和mapper.xml中定义的每个sql 的parameterType的类型相同

4、 Mapper接口方法的返回参数类型和mapper.xml中定义的每个sql的resultType的类型相同

5、 Mapper接口和mapper配置文件必须在同一个包下面而且名称要一致生成代理对象:

<mapper namespace=*"接口的全限名 "*>

<select id=*"方法名 "* parameterType=*"接口传入参数类型"*

resultType=*"接口返回值类型 "*>

select \* from user where id=#{yonghudeid}

</select>

</mapper>

标签

Select:查询

Insert:插入

Delete删除

Update更新

Foreach循环遍历

If判断

Where判断

### MapperSqlConfig核心配置文件

加载数据库属性配置文件

配置environment中的事务管理和数据库连接池

加载mapper.xml配置文件

标签

**properties**（属性）

settings（全局配置参数）

**typeAliases**（类型别名）

typeHandlers（类型处理器）

objectFactory（对象工厂）

plugins（插件）

environments（环境集合属性对象）

environment（环境子属性对象）

transactionManager（事务管理）

dataSource（数据源）

**mappers**（映射器）

### 代理模式开发步骤

1：一个接口对应一个mapper.xml

2:遵循5个一致方针

a、namespace必须时候接口的全限定名

b、Statementid必须和接口的方法名称一致

c、接口方法的参数类型要和parameterType要一致

d、接口方法的返回值类型要和resultType一致

e、mapper接口和mapper配置文件必须在同一个包下面而且名称要一致

生成代理对象:

### Mybatis分页插件

使用PageHelper分页插件

1. 导入pom插件PageIntercepto
2. 在sqlMapConfig.xml配置plugin插件PageHelper，其实质就是配置一个拦截器并配置方言指定具体的数据库。
3. 设置分页信息：PageHelper.starPage(起始页，每页数量)------设置在当前线程中
4. 获取分页信息：创建pageInfo对象，把list结果集传入构造方法。PageInfo中提供了丰富的api

# Jvm虚拟机

* 启动类加载器：Bootstrap ClassLoader，跟上面相同。它负责加载存放在JDK\jre\lib(JDK代表JDK的安装目录，下同)下，或被-Xbootclasspath参数指定的路径中的，并且能被虚拟机识别的类库（如rt.jar，所有的java.\*开头的类均被Bootstrap ClassLoader加载）。启动类加载器是无法被Java程序直接引用的。
* 扩展类加载器：Extension ClassLoader，该加载器由sun.misc.Launcher$ExtClassLoader实现，它负责加载JDK\jre\lib\ext目录中，或者由java.ext.dirs系统变量指定的路径中的所有类库（如javax.\*开头的类），开发者可以直接使用扩展类加载器。
* 应用程序类加载器：Application ClassLoader，该类加载器由sun.misc.Launcher$AppClassLoader来实现，它负责加载用户类路径（ClassPath）所指定的类，开发者可以直接使用该类加载器，如果应用程序中没有自定义过自己的类加载器，一般情况下这个就是程序中默认的类加载器。

类加载器的工作原理

双亲委派模型的工作流程是：如果一个类加载器收到了类加载的请求，它首先不会自己去尝试加载这个类，而是把请求委托给父加载器去完成，依次向上，因此，所有的类加载请求最终都应该被传递到顶层的启动类加载器中，只有当父加载器在它的搜索范围中没有找到所需的类时，即无法完成该加载，子加载器才会尝试自己去加载该类。

### 委托机制

当一个类加载和初始化的时候，类仅在有需要加载的时候被加载。假设你有一个应用需要的类叫作Abc.class，首先加载这个类的请求由Application类加载器委托给它的父类加载器Extension类加载器，然后再委托给Bootstrap类加载器。Bootstrap类加载器会先看看rt.jar中有没有这个类，因为并没有这个类，所以这个请求由回到Extension类加载器，它会查看jre/lib/ext目录下有没有这个类，如果这个类被Extension类加载器找到了，那么它将被加载，而Application类加载器不会加载这个类；而如果这个类没有被Extension类加载器找到，那么再由Application类加载器从classpath中寻找。记住classpath定义的是类文件的加载目录，而PATH是定义的是可执行程序如javac，java等的执行路径。

### 可见性机制

根据可见性机制，子类加载器可以看到父类加载器加载的类，而反之则不行。所以下面的例子中，当Abc.class已经被Application类加载器加载过了，然后如果想要使用Extension类加载器加载这个类，将会抛出java.lang.ClassNotFoundException异常。

### 单一性机制

根据这个机制，父加载器加载过的类不能被子加载器加载第二次。虽然重写违反委托和单一性机制的类加载器是可能的，但这样做并不可取。你写自己的类加载器的时候应该严格遵守这三条机制。

# Redis（6379）

### Redis的优缺点

优点:

1. 性能高效。读写性能可以达到10w/s的频率。
2. 支持数据类型丰富。除了String类型的value，还支持hash，set，sortedset，list等数据类型
3. 支持主从复制。主机可以将数据同步到从机，可以经行读写分离。
4. 支持数据持久化。采用AOF和RDB两种方式将数据存储在本地。

缺点:

1. 数据库容量受到了物理内存的限制，不能用作海量数据的高性能读写，
2. 内存消耗较大
3. Redis不支持回滚(Roll Back)

### 集群的实现方式

创建 Redis 节点

 redis-trib.rb 这个工具

redis cluster在设计的时候，就考虑到了去中心化，去中间件，也就是说，集群中的每个节点都是平等的关系，都是对等的，每个节点都保存各自的数据和整个集群的状态。每个节点都和其他所有节点连接，而且这些连接保持活跃，这样就保证了我们只需要连接集群中的任意一个节点，就可以获取到其他节点的数据。

Redis 集群会把数据存在一个 master 节点，然后在这个 master 和其对应的salve 之间进行数据同步。当读取数据时，也根据一致性哈希算法到对应的 master 节点获取数据。只有当一个master 挂掉之后，才会启动一个对应的 salve 节点，充当 master

### 哨兵模式

如果我们在使用主从复制的情况下，Master服务器进行了down机的情况，我们的系统就不能再进行写的操作，所以此时redis在2.6版本引入了哨兵模式，但是并不稳定，2.8版本之后哨兵模式才稳定了起来。

顾名思义Redis的哨兵模式就是对redis系统进行实时的监控，其主要功能有下面两点

1.监测主数据库和从数据库是否正常运行。

2.当我们的主数据库出现故障的时候，可以自动将从数据库转换为主数据库，实现自动的切换。

修改配置

说明当master主服务器down机之后，剩下的服务器要进行一个投票选举出一个主服务器

默认配置哨兵0.5ms进行一次检查这个集群。

可以配置几个从节点将支持转换为主服务器的这个事情。

### 主从复制

1、slave与Master建立连接，发送sync同步命令。

也就是说当用户在Master写入一条命令后，他们之间会通过一些算法把数据同步到每一个slave上。

2、Msater会开启一个后台进程，将数据库快照保存到文件中。同时Master主进程会开始收集新的写命令并缓存。

3、后台完成保存后，就将文件发送给slave

4、slave将此文件保存在硬盘上。

我们将一台Redis服务器作主库(Matser)，其他三台作为从库(Slave)，主库只负责写数据，每次有数据更新都将更新的数据同步到它所有的从库，而从库只负责读数据。这样一来，就有了两个好处：

1、读写分离，不仅可以提高服务器的负载能力，并且可以根据读请求的规模自由增加或者减少从库的数量

2、数据被复制成了了好几份，就算有一台机器出现故障，也可以使用其他机器的数据快速恢复。

### 事务

1、MULTI

该命令用来开启事务，它总是返回ok结果，当其执行之后，客户端可以继续发送任意条数量的指令，这些指令不会立即被执行，而是被放到了队列中，直到EXEC被调用之后，所有命令才会被序列化执行。

2、EXEC

该命令负责触发并执行队列中所有的命令。

NOTE:

如果MULTI开启之后，因为某些原因没有成功执行EXEC，那么事务中所有的命令都不会被执行的。

3、DISCARD

该命令用来刷新事务中所有排队等待执行的指令，它总是返回ok结果，并且将服务连接状态恢复到正常。如果已经使用WATCH，那么其会将释放所有被WATCH的key。

4、WATCH

标记所有指定的key被监控起来，使其在事务中有条件的执行（乐观锁）。

### 雪崩和缓存击穿

**雪崩**：设置缓存时采用了相同的过期时间，导致缓存在某一时刻同时失效，请求全部转发到DB，DB瞬时压力过重雪崩。

解决：用加锁或者队列的方式保证缓存的单线程（进程）写，从而避免失效时大量的并发请求落到底层存储系统上。比如缓存失效时间分散开，比如我们可以在原有的失效时间基础上增加一个随机值，比如1-5分钟随机，这样每一个缓存的过期时间的重复率就会降低，就很难引发集体失效的事件。

**缓存击穿**：缓存穿透是指查询一个一定不存在的数据，而因为缓存中没有该数据，所以请求落到了DB中查询，如果流量大时，DB就挂掉了。

解决：最常见的则是采用布隆过滤器将所有可能存在的数据哈希到一个足够大的bitmap中，一个一定不存在的数据会被 这个bitmap拦截掉，从而避免了对底层存储系统的查询压力。

如果一个查询返回的数据为空（不管是数 据不存在，还是系统故障），我们仍然把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。

### 管道技术

在C/S交互中，客户端可发送任意数请求命令，不需要等服务端响应后再行发送请求；而服务端可以一次性完成并返回所有客户端请求结果，也可以分批次返回请求结果，这主要看服务端处理结果的速度而定。

### 发布订阅

Redis 发布订阅(pub/sub)是一种消息通信模式：发送者(pub)发送消息，订阅者(sub)接收消息。

Redis 客户端可以订阅任意数量的频道。

客户端向频道中发布消息，多个订阅者可以同时收到。

### Redis的存储方式和存储结构

**存储结构**：Hash，Set，Sorted set，List，String

**RDB（性能高）**：将Reids在内存中数据快照定时dump到磁盘上，在 Redis 重启动时，RDB程序可以通过载入 RDB文件来还原数据库的状态

例如：save 300 10 #在300秒(5分钟)之后，如果至少有10个key发生变化，则dump内存快照。

**AOF(数据完整)**：将Reids的写入命令以协议文本的方式追加到磁盘文件。

1. AOF\_FSYNC\_NO：不保存。

2. AOF\_FSYNC\_EVERYSEC：每一秒钟保存一次。

3. AOF\_FSYNC\_ALWAYS：每执行一个命令保存一次

RDB与AOF的策略选择：

主要根据自己的应用场景来分析，在效率和可用性之间做选择，愿意牺牲一些性能换取数据的高可用（AOF），还是愿意丢失一些数据换取性能的提高(RDB)。

倒排索引

倒排索引是实现“单词-文档矩阵”的一种具体存储形式，通过倒排索引，可以根据单词快速获取包含这个单词的文档列表。倒排索引主要由两个部分组成：“单词词典”和“倒排文件”。

全局异常处理

自定义一个异常类,实现handlerExceptionResolve并实现resolvelException方法

在springmvc.xml中配置这个异常bean对象

拦截器

自定义一个拦截器类,实现handlerInterceptor并实现preHandler(执行前),postHandler(执行handler之后,返回modlerAndView之前),afterComplication(modlerAndView之后)这三个方法

在springmvc.xml中配置拦截器

应用场景:前台用户登陆验证

Lucene全文检索

1. 创建索引库
2. 获取源文件
3. 将源文件转化为document对象，一条数据就是一个document对象

Document对象通过field存数据，一个document可以包含多个field

1. 每个field需要指定域属性（是否分词，是否建立索引，是否存储）
2. 将document放入索引库
3. 查询索引库
4. 建立连接索引库（indexReader）
5. IndexSearch
6. Query=new TermQuery(new Term(“域”，“关键字”))
7. indexSearh执行TermQuery得到TopDocs结果

# ActiveMQ

### ActiveMQ的作用与原理

ActiveMQ的作用就是系统之间进行通信。可以让系统直接的调用进行解耦，实现系统间的异步通信。原理就是生产者生产消息，把消息发送给MQ。MQ接收到消息后，查询有多少个消费者，然后把消息转发消费者，此过程中，生产者无需参与。消费者接收到消息后做出的相应处理与生产者没有任何关系。

MQ有两种通信方式，点对点和订阅模式。如果是点对点，消费没有发送成功此消息会保存到MQ服务端知道有消费者来消费，消息不会丢失。发布订阅的通信方式，默认情况下只通知一次，如果接收不到此消息就没有。

# Webservice

#### Webservice介绍

Webservice是一个跨平台跨系统的远程调用技术。解决了跨域问题而且也方便数据做二次处理。目前用的较多的是webservice的CXF框架，而CXF框架分为jax-ws和jax-rs两种服务方式。ws采用的是soap协议只支持xml格式，rs采用的是http协议，支持json和xml。考虑到数据的轻量级，所以在项目中一般会使用jax-rs的restful风格来编程。

#### 说说restful风格

Restful是一种编程风格，并不是规定。其特点是服务器的服务以资源的形式对外提供访问，而且每一个资源对应一个特定的URI。对应资源的操作我们使用http的四种请求来完成（GET,PUT,POST,DELETE）

#### Spring整合jax-rs

1. 导入cxf依赖jar包

2. 提供Service接口和实现，Service接口上面用注解配置服务路径

3. 需要传输的domain对象，添加@XmlRootElement注解

(修改后执行 maven install)

4. applicationContext里面配置接口地址

5. web.xml里面配置CXFServlet处理web service请求

# Quartz

Quartz是一个任务调度框架,在项目中主要是用来做为定时处理,如商品限时抢购等.

类实现job接口,重写ecxcute方法

在spring容器汇总定义实现JOB接口的类对象,Trigger对象和Schedule对象.

Trigger为定时器,定义执行的时间

Job为目标类

通过Schedule为启动定时框架

# Spring

##### AOP

面向切面编程.在oop中,我们将事务纵向抽象为一个个的对象.在aop编程中,我们是将对象的某些类似的方法横向抽象为一个切面.对这个切面进行一些如权限校验,事务管理,记录日志等公用操作处理的过程就是面向切面编程的思想.

# MySql

##### 事务的隔离级别和特性

事务具有四个特征：原子性（ Atomicity ）、一致性（ Consistency ）、隔离性（ Isolation ）和持久性（ Durability ）。

四种隔离级别

**Read Uncommitted（读取未提交内容）**

**Read Committed（读取提交内容）**

**Repeatable Read（可重读）**

**Serializable（可串行化）**

##### MySQL两种引擎的区别

Innodb引擎: Innodb引擎提供了对数据库ACID事务的支持，并且实现了SQL标准的四种隔离级别。(事务)

MyIASM引擎: MyIASM是MySQL默认的引擎，但是它没有提供对数据库事务的支持，也不支持行级锁和外键，((非事务)

1、MyIASM是非事务安全的，而InnoDB是事务安全的

2、MyIASM锁的粒度是表级的，而InnoDB支持行级锁

3、MyIASM支持全文类型索引，而InnoDB不支持全文索引

4、MyIASM相对简单，效率上要优于InnoDB，小型应用可以考虑使用MyIASM

5、MyIASM表保存成文件形式，跨平台使用更加方便

##### Mysql数据库优化

1. 如果已知查询的结果只有一条，可以使用limit 1,这样数据库引擎就会在找到一条数据后停止搜索，增加性能。
2. 选择合适的存储引擎。MyISAM适合大量查询的应用，它的select count（\*）速度非常快。但是对于写入操作不是太好，因为它属于表级锁，操作时会把整个表锁起来。

InnoDB支持行级锁，在写操作较多时候会更加优秀。同时它还支持事务。

1. 对字段建立索引，增加查询效率。
2. 使用Not Existsts 替代Not In。
3. 避免使用in , not in , is null , is not null , < >等，防止索引失效
4. 进行分表分库
5. 垂直分库：按照表功能进行分类。如商品DB,用户DB等
6. 水平分库：当一个表中的数据过大时，把表分多多个表。逻辑上是一张表，物理上是两张表。
7. 优化sql语句
8. 尽量用类名替代\*
9. Where语句的解析顺序。
10. 使用where替代having
11. 使用多表查询替代子查询

##### 数据库分页

Mysql 分页 limit start,size; start从0开始,seize每页条数

Orcal 分页 三层嵌套

##### 索引的原理

索引的原理大致概括为以空间换时间，数据库在未添加索引的时候进行查询默认的是进行全量搜索，也就是进行全局扫描，有多少条数据就要进行多少次查询，然后找到相匹配的数据就把他放到结果集中，直到全表扫描完。而建立索引之后，会将建立索引的KEY值放在一个n叉树上（BTree）。因为B树的特点就是适合在磁盘等直接存储设备上组织动态查找表，每次以索引进行条件查询时，会去树上根据key值直接进行搜索，次数约为log总条数，底数为页面存储数，例如一个100万数据的表，页面存储数为100，那么有索引的查询次数为3次log1000000100，但是全量搜索为100万次搜索，这种方式类似于二分法，但是这个是n分法。

# Dubbo

##### 为什么要用dubbo

ws采用的是soap协议,效率低

采用restful:http+json效率不高,服务需要自己治理

采用dubbo:使用rpc协议,直接用socket通信.效率高,服务自动治理.没有api入侵

##### Dubbo的工作原理

Dubbo就是资源调度和治理中心的管理工具

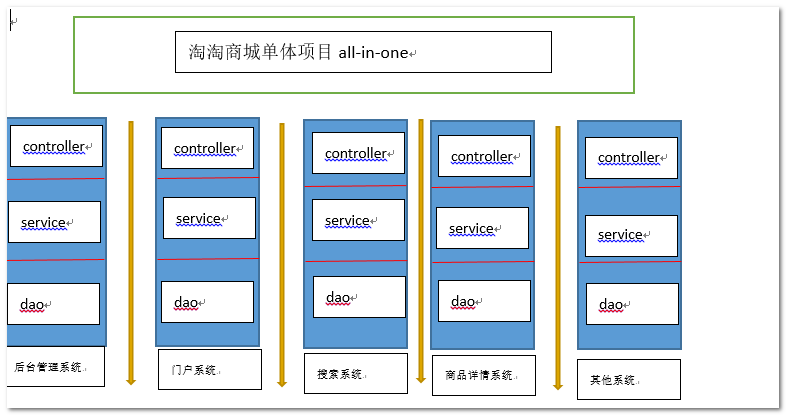
1. 服务端与zookeeper（注册中心）建立长连接，将**ip，端口号**和**接口信息**及**应用名称**交给zookeeper。
2. 消费者在zookeeper中查询相关的接口信息
3. Zookeeper返回ip和端口给消费者
4. 消费者通过ip端口利用**RPC协议**直接调用服务端接口
5. 通过monitor获取到zookeeper中的数据,方便监控.

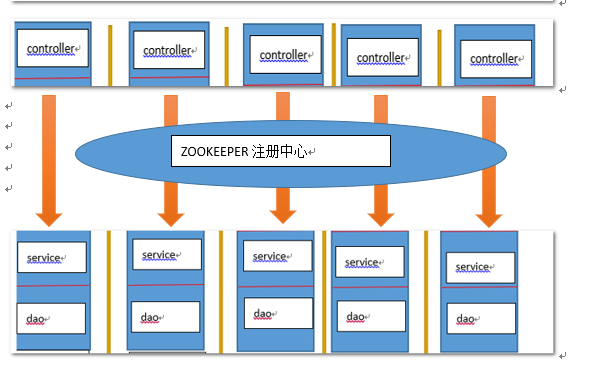
Zookeeper端口号2181

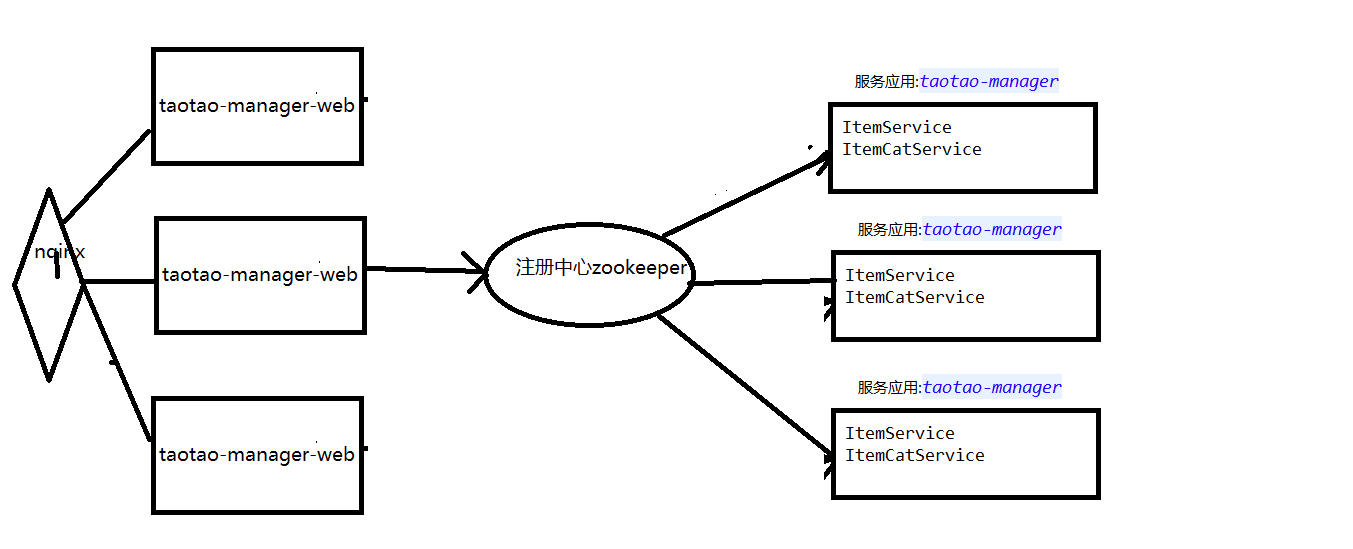
##### Dubbo服务集群,为什么需要集群

同一个服务部署在不同的服务器中,形成服务集群。

1. 使dubbo服务高可用，防止单节点宕机危险
2. 使dubbo服务能承受被高并发调用







##### 负载均衡策略

策略一:采用随机，按权重比设置。

策略二：活跃调用数，慢的提供者收到更少的请求。

设置方式一：在配置文件中配置，可以细粒度到接口或者方法上

设置方式二：在监控中心设置

##### 集群的容错模式

模式一：**failover（缺省）**服务调用失败时自动切换，通过retries=”2”来设置重试次数。一般是读操作

模式二：**Failfast** 只发一次调用，失败立即异常，通常用于非幂等性的写操作例如新增操作

模式三：**failsafe**失败安全，通常用于日志服务

模式四：**Forking**并行调用多个服务，谁最先执行完，则使用谁的结果通常用于实时性较高的读操作，这种模式很浪费服务器资源可以通过forks=”2”来设置最高并发数

总结：

1:在生产环境读操作就使用默认的(failover)模式(retries=“2”)

2:在生产环境中写操作也可以用默认模式,但是记得将retries设置为0。

# FastDFS

##### FastDFS的结构

FastDFS架构包括 Tracker server和Storage server。客户端请求Tracker server进行文件上传、下载，通过Tracker server调度最终由Storage server完成文件上传和下载。

Tracker server作用是负载均衡和调度，通过Tracker server在文件上传时可以根据一些策略找到Storage server提供文件上传服务。可以将tracker称为追踪服务器或调度服务器。

Storage server作用是文件存储，客户端上传的文件最终存储在Storage服务器上，Storage server没有实现自己的文件系统而是利用操作系统 的文件系统来管理文件。可以将storage称为存储服务器。



# JAVAEE

##### 接口和抽象类的区别

1.抽象类和接口都是用来抽象具体对象的. 但是接口的抽象级别最高

2.抽象类可以有具体的方法 和属性, 接口只能有抽象方法和不可变常量

3.抽象类主要用来抽象类别,接口主要用来抽象功能.

4、抽象类中，且不包含任何实现，派生类必须覆盖它们。接口中所有方法都必须是未实现的。

##### String，StringBuilder，StringBuffer（常用api）

String字符串长度固定，另外两者可以改变。

StringBuilder是线程不安全的，效率高。

StringBuffer是线程安全的，效率低。

String indexOf、substring、trim、equals、split、length()。

##### 接口的默认修饰符

方法是public abstract

属性的默认是public static final

##### Java中创建（实例化）对象的方式

1、用new语句创建对象。

2、通过静态方法返回对象。

3、运用反射手段,调用java.lang.Class或者java.lang.reflect.Constructor类的newInstance()实例方法。

4、调用对象的clone()方法。

5、通过I/O流（包括反序列化），如运用反序列化手段，调用java.io.ObjectInputStream对象的 readObject()方法。

##### 设计模式

1、工厂方法模式：建立一个工厂类，对实现了同一接口的一些类进行实例的创建

2、单例模式：单例对象能保证在一个JVM中，该对象只有一个实例存在。

3、设配器模式：将一个接口转化为另一个接口。

4、装饰者模式：保持原有的接口，通过扩展的方式提高性能。

5、观察者模式：当一个对象变化时，其它依赖该对象的对象都会收到通知，并且随着变化。

6、代理模式：代理模式就是多一个代理类出来，替原对象进行一些操作

7、模板方法模式：解决代码重复的问题。

# 集合

##### 集合的安全性

Vector hashtable concurrenthashmap安全

ArrayList，LinkedList，HashMap，TreeMap，HashSet，TreeSet都不是线程安全的

##### Hashmap底层原理

HashMap()：构建一个初始容量为 16，负载因子为 0.75 的 HashMap, HashMap有两个参数影响其性能：初始容量和加载因子。默认初始容量是16，加载因子是0.75。容量是哈希表中桶(Entry数组)的数量，初始容量只是哈希表在创建时的容量。加载因子是哈希表在其容量自动增加之前可以达到多满的一种尺度。当哈希表中的条目数超出了加载因子与当前容量的乘积时，通过调用 rehash 方法将容量翻倍。

默认情况下，数组大小为16，那么当HashMap中元素个数超过16\*0.75=12的时候，就把数组的大小扩展为 2\*16=32，即扩大一倍，然后重新计算每个元素在数组中的位置，而这是一个非常消耗性能的操作，所以如果我们已经预知HashMap中元素的个数，那么预设元素的个数能够有效的提高HashMap的性能。

HashMap(int initialCapacity, float loadFactor)：以指定初始容量、指定的负载因子创建一个 HashMap。

**简单地说，HashMap 在底层将 key-value 当成一个整体进行处理，这个整体就是一个 Entry 对象。HashMap 底层采用一个 Entry[] 数组来保存所有的 key-value 对，当需要存储一个 Entry 对象时，会根据hash算法来决定其在数组中的存储位置，在根据equals方法决定其在该数组位置上的链表中的存储位置；当需要取出一个Entry时，也会根据hash算法找到其在数组中的存储位置，再根据equals方法从该位置上的链表中取出该Entry。**

##### hashmap和hashtable的区别

1、HashMap可以接受为null的键值(key)和值(value)，而Hashtable则不行。

2、HashMap是非synchronized，线程不安全，但是效率高。而Hashtable是synchronized，这意味着Hashtable是线程安全的，效率低。所以一般我们会选择ConcurrentHashMap，它是HashTable的替代，比HashTable的扩展性更好。

3、HashMap的迭代器(Iterator)是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。所以当有其它线程改变了HashMap的结构（增加或者移除元素），将会抛出ConcurrentModificationException，但迭代器本身的remove()方法移除元素则不会抛出异常。

4、HashMap在不指定容量的情况下的默认容量为16，而HashTable为11。Hashtable扩容时，将容量变为原来的2倍加1，而HashMap扩容时，将容量变为原来的2倍

##### Arraylist为什么有索引就查询快

数组名这个变量中存放的是你这个数组中第一个元素的地址

数组下标的实际意义是：相对于数组起始地址的地址偏移量。

有如下的等价关系：a[i] = \*(a + i)

##### ConcurrentHashMap和Hashtable的区别

它们都可以用于多线程的环境，但是当Hashtable的大小增加到一定的时候，性能会急剧下降，因为迭代时需要被锁定很长的时间。因为ConcurrentHashMap引入了分割(segmentation)，不论它变得多么大，仅仅需要锁定map的某个部分，而其它的线程不需要等到迭代完成才能访问map。简而言之，在迭代的过程中，ConcurrentHashMap仅仅锁定map的某个部分，而Hashtable则会锁定整个map。ConcurrentHashMap所使用的锁分段技术，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。

##### 如何让hashmap同步

HashMap可以通过Collections类的Collections.synchronizeMap(hashMap);

##### HashMap和HashSet的区别

1. HashMap实现了map接口；HashSet实现了set接口。
2. HashMap储存键值对；HashSet仅仅存储对象。
3. HashMap使用put()方法将元素放入map中；HashSet使用add()方法将元素放入set中。
4. HashMap中使用键对象来计算hashcode值；HashSet使用成员对象来计算hashcode值，对于两个对象来说hashcode可能相同，所以equals()方法用来判断对象的相等性，如果两个对象不同的话，那么返回false。

##### TreeMap

自定义排序规则：要实现Comparator接口并实现compare()方法

自然排序： compareTo 实现机制

# 线程IO

##### 同步

当一个同步调用发出后，调用者要一直等待返回消息（结果）通知后，才能进行后续的执行；

（选择排队等候）

##### 异步

当一个异步过程调用发出后，调用者不能立刻得到结果。实际处理这个调用的部件在完成后，通过状态、通知和回调来通知调用者。

（择取一个小纸条上面有我的号码，等到排到我这一号时由柜台的人通知我轮到我去办理业务了；）

##### 阻塞

 阻塞调用是指调用结果返回之前，当前线程会被挂起。函数只有在得到结果之后才会返回。  
（只能等待，什么都不能做）

##### 非阻塞

 非阻塞和阻塞的概念相对应，指在不能立刻得到结果之前，该函数不会阻塞当前线程，而会立刻返回。

（可以一边排队一边玩手机）

##### 同步阻塞形式

（就是你专心排队，什么别的事都不做）

如果这个线程在等待当前函数返回时，没有执行其他消息处理，而是处于挂起等待状态，那这种情况就叫做同步阻塞；

##### 异步阻塞形式

**（**只需要等大堂经理通知，在这段时间里他不能离开银行做其它的事情，**）**

如果这个线程在等待当前函数返回时，仍在执行其他消息处理，那这种情况就叫做同步非阻塞；

##### 同步非阻塞形式

（一边打着电话一边还需要抬头看到底队伍排到你了没有）

##### 异步非阻塞形式

（只需要等大堂经理通知，期间可以离开银行随意外出玩手机抽烟等）

##### wait()、notify()、notifyAll()

三个方法是 java.lang.Object 的 final native 方法

wait()：导致线程进入等待状态，直到它被其他线程通过notify()或者notifyAll唤醒，该方法只能在同步方法中调用。

notify()：随机选择一个在该对象上调用wait方法的线程，解除其阻塞状态，该方法只能在同步方法或同步块内部调用。

notifyAll()：解除所有那些在该对象上调用wait方法的线程的阻塞状态，同样该方法只能在同步方法或同步块内部调用。

调用这三个方法中任意一个，当前线程必须是锁的持有者，如果不是会抛出一个 IllegalMonitorStateException 异常

##### wait() 与 Thread.sleep(long time) 的区别

wait() 方法进入等待状态时会释放同步锁，而 sleep() 方法不会释放同步锁。

wait() 方法使实体所处线程暂停执行，从而使对象进入等待状态，直到被 notify() 方法通知或者 wait() 的等待的时间到。

sleep() 方法使持有的线程暂停运行，从而使线程进入休眠状态，直到用 interrupt 方法来打断他的休眠或者 sleep 的休眠的时间到。

##### 多线程创建的方式

继承Thread类，重写run()方法。

实现Runnable接口，

使用ExecutorSerivce、Callable、Future实现有返回值结果的多线程（JDK1.5）

##### 线程池

##### 线程的生命周期

就绪、运行、睡眠、等待、阻塞、死亡

##### 如何停止一个线程

**线程正常执行完毕，正常结束**

调用stop方法

调用interrupt

##### 什么是死锁，如何解除死锁

两个或以上的线程在等待对方执行完毕后才能继续往下执行的时候就发生了死锁。结果就是所以进程无线等待中。

避免死锁的方式：指定获取锁的顺序，并强制线程按照指定的顺序获取锁。

##### 创建和启动线程的方式

创建:实现thread类,或继承runnable接口

开启线程:run()

启动线程:strat()

start()方法被用来启动新创建的线程，而且start()内部调用了run()方法，这和直接调用run()方法的效果不一样。当你调用run()方法的时候，只会是在原来的线程中调用，没有新的线程启动，start()方法才会启动新线程。

# Linux

##### 如何查看服务器异常日志信息

通过cd命令，进入服务器日志文件所存放的目录

使用命令：tail -f log/production.log 查看新追加到文档中的信息，这样可以实时看到自己的访问日志，便于自己调试程序和查看错误

# JavaWeb

##### Jsp和servlet的区别

1、.jsp经编译后就变成了servlet(jsp本质就是servlet,jvm只能识别java的类，不能识别jsp代码，web容器将jsp的代码编译成jvm能够识别的java类)。

2、.jsp更擅长表现于页面显示，servlet更擅长于逻辑控制。

3.、setvlet中没有内置对象，jsp中的内置对象都是必须通过HttpServletRequest对象，HttpServletResponse对象及HttpServlet对象得到。

4.、jsp是servlet的一种简化，使用jsp只需要完成程序员需用输出到客户端的内容，jsp中的java脚本如何镶嵌到一个类中，由jsp容器完成，而servlet则是个完整的java类，这个类的service方法用于生成对客户端的响应。

##### 如何实现防盗链

防盗链用referer请求头

编写一个名为DownManagerServlet的Servlet程序，这个Servlet程序负责提供下载内容，但它要求下载请求必须通过本站的下载页面链接进来，否则将请求转发给本站的下载说明页。

##### JDBC工作原理

加载JDBC驱动

获得连接对象connection

创建statement对

执行sql语句

释放资源