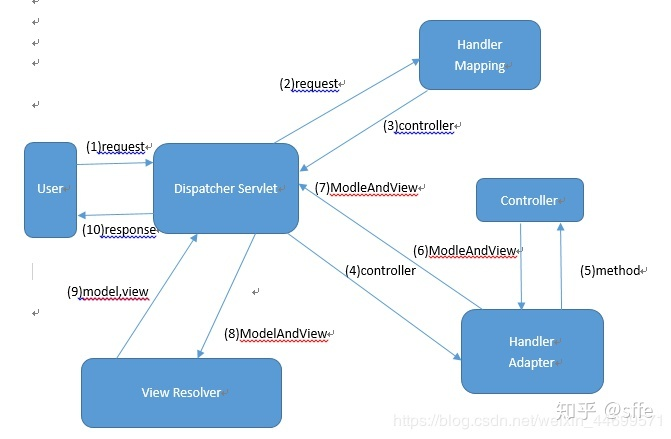
## springmvc工作流程图及介绍



（1）客户端通过url发送请求

（2-3）核心控制器Dispatcher Servlet接收到请求，通过系统或自定义的映射器配置找到对应的handler，并将url映射的控制器controller返回给核心控制器。

（4）通过核心控制器找到系统或默认的适配器

（5-7）由找到的适配器，调用实现对应接口的处理器，并将结果返回给适配器，结果中包含数据模型和视图对象，再由适配器返回给核心控制器

（8-9）核心控制器将获取的数据和视图结合的对象传递给视图解析器，获取解析得到的结果，并由视图解析器响应给核心控制器

（10）核心控制器将结果返回给客户端

## spring mvc 和 struts 的区别是什么？

一、拦截机制的不同

　　Struts2是类级别的拦截，每次请求就会创建一个Action，和Spring整合时Struts2的ActionBean注入作用域是原型模式prototype，然后通过setter，getter吧request数据注入到属性。Struts2中，一个Action对应一个request，response上下文，在接收参数时，可以通过属性接收，这说明属性参数是让多个方法共享的。Struts2中Action的一个方法可以对应一个url，而其类属性却被所有方法共享，这也就无法用注解或其他方式标识其所属方法了，只能设计为多例。

　　SpringMVC是方法级别的拦截，一个方法对应一个Request上下文，所以方法直接基本上是独立的，独享request，response数据。而每个方法同时又何一个url对应，参数的传递是直接注入到方法中的，是方法所独有的。处理结果通过ModeMap返回给框架。在Spring整合时，SpringMVC的Controller Bean默认单例模式Singleton，所以默认对所有的请求，只会创建一个Controller，有应为没有共享的属性，所以是线程安全的，如果要改变默认的作用域，需要添加@Scope注解修改。

　　Struts2有自己的拦截Interceptor机制，SpringMVC这是用的是独立的Aop方式，这样导致Struts2的配置文件量还是比SpringMVC大。

二、底层框架的不同

　　Struts2采用Filter（StrutsPrepareAndExecuteFilter）实现，SpringMVC（DispatcherServlet）则采用Servlet实现。Filter在容器启动之后即初始化；服务停止以后坠毁，晚于Servlet。Servlet在是在调用时初始化，先于Filter调用，服务停止后销毁。

三、性能方面

　　Struts2是类级别的拦截，每次请求对应实例一个新的Action，需要加载所有的属性值注入，SpringMVC实现了零配置，由于SpringMVC基于方法的拦截，有加载一次单例模式bean注入。所以，SpringMVC开发效率和性能高于Struts2。

四、配置方面

　　spring MVC和Spring是无缝的。从这个项目的管理和安全上也比Struts2高。

附注：拦截器与过滤器的区别 ：

拦截器是基于java的反射机制的，而过滤器是基于函数回调。

拦截器不依赖与servlet容器，过滤器依赖与servlet容器。

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。

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。

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

## 为什么要使用 spring？

1.方便解耦，便于开发（Spring就是一个大工厂，可以将所有对象的创建和依赖关系维护都交给spring管理）

2.spring支持aop编程（spring提供面向切面编程，可以很方便的实现对程序进行权限拦截和运行监控等功能）

3.声明式事务的支持（通过配置就完成对事务的支持，不需要手动编程）

4.方便程序的测试，spring 对junit4支持，可以通过注解方便的测试spring 程序

5.方便集成各种优秀的框架（）

6.降低javaEE API的使用难度（Spring 对javaEE开发中非常难用的一些API 例如JDBC,javaMail,远程调用等，都提供了封装，是这些API应用难度大大降低）

## 解释一下什么是 aop？

<https://www.cnblogs.com/songanwei/p/9417343.html>

在运行时，动态地将代码切入到类的指定方法、指定位置上的编程思想就是面向切面的编程。

Spring的AOP的原理：

1. AOP相关的概念

1） Aspect ：切面，切入系统的一个切面。比如事务管理是一个切面，权限管理也是一个切面；

2） Join point ：连接点，也就是可以进行横向切入的位置；

3） Advice ：通知，切面在某个连接点执行的操作(分为: Before advice , After returning advice , After throwing advice , After (finally) advice , Around advice )；

4） Pointcut ：切点，符合切点表达式的连接点，也就是真正被切入的地方；

2. AOP 的实现原理

AOP分为静态AOP和动态AOP。

静态AOP是指AspectJ实现的AOP，他是将切面代码直接编译到Java类文件中。

动态AOP是指将切面代码进行动态织入实现的AOP。

Spring的AOP为动态AOP，实现的技术为： JDK提供的动态代理技术 和 CGLIB(动态字节码增强技术) 。尽管实现技术不一样，但 都是基于代理模式 ， 都是生成一个代理对象 。

#### 1) JDK动态代理

主要使用到 InvocationHandler 接口和 Proxy.newProxyInstance() 方法。

 JDK动态代理要求被代理实现一个接口，只有接口中的方法才能够被代理 。

其方法是将被代理对象注入到一个中间对象，而中间对象实现InvocationHandler接口，

在实现该接口时，可以在 被代理对象调用它的方法时，在调用的前后插入一些代码。

而 Proxy.newProxyInstance() 能够利用中间对象来生产代理对象。

插入的代码就是切面代码。所以使用JDK动态代理可以实现AOP。

#### 2）CGLIB （code generate libary）

字节码生成技术实现AOP，其实就是继承被代理对象，然后Override需要被代理的方法，在覆盖该方法时，自然是可以插入我们自己的代码的。

因为需要Override被代理对象的方法，所以自然CGLIB技术实现AOP时，就 必须要求需要被代理的方法不能是final方法，因为final方法不能被子类覆盖 。

Spring中，如果被代理对象实现了接口，那么就使用JDK的动态代理技术，反之则使用CGLIB来实现AOP，所以 Spring默认是使用JDK的动态代理技术实现AOP的 。

## <aop:aspect>与<aop:advisor>的区别

<https://www.jianshu.com/p/40f79da0cdef>

< aop:advisor>大多用于事务管理。

< aop:aspect>大多用于日志，缓存。

可以看出，< aop:advisor>和< aop:aspect>其实都是将通知和切面进行了封装，原理基本上是一样的，只是使用的方式不同而已。

## Aop注解有哪些？

<https://blog.csdn.net/yhl_jxy/article/details/78815636>

@Aspect标识该类为切面类

@Pointcut定义切点，conference()只是一个标识，无所谓是什么，

方法中内容本身也是空的，使用该切点的地方直接通过标识conference()引用切点表达式。

@Pointcut("execution(\* com.lanhuigu.spring.ConferenceServiceImpl.conference(..))")

@Before("conference()")定义通知

/\*\*

\* Jdk代理：基于接口的代理，一定是基于接口，会生成目标对象的接口的子对象。

\* Cglib代理：基于类的代理，不需要基于接口，会生成目标对象的子对象。

\*

\* 1. 注解@EnableAspectJAutoProxy开启代理;

\*

\* 2. 如果属性proxyTargetClass默认为false, 表示使用jdk动态代理织入增强;

\*

\* 3. 如果属性proxyTargetClass设置为true，表示使用Cglib动态代理技术织入增强;

\*

\* 4. 如果属性proxyTargetClass设置为false，但是目标类没有声明接口，

\* Spring aop还是会使用Cglib动态代理，也就是说非接口的类要生成代理都用Cglib。

\*/

@Configuration

@EnableAspectJAutoProxy(proxyTargetClass = true)

@ComponentScan("com.lanhuigu")

public class AppConfig {

}

## 解释一下什么是 ioc？

　　IOC（控制反转）：全称为：Inverse of Control。从字面上理解就是控制反转了，将对在自身对象中的一个内置对象的控制反转，反转后不再由自己本身的对象进行控制这个内置对象的创建，而是由第三方系统去控制这个内置对象的创建。

　　DI（依赖注入）：全称为Dependency Injection，意思自身对象中的内置对象是通过注入的方式进行创建。

那么IOC和DI这两者又是什么关系呢？

IOC就是一种软件设计思想，DI是这种软件设计思想的一个实现。而Spring中的核心机制就是DI。

## spring 有哪些主要模块？

Spring有七大功能模块，分别是Spring Core，AOP，ORM，DAO，MVC，WEB，Context。

1，Spring Core

Core模块是Spring的核心类库，Spring的所有功能都依赖于该类库，Core主要实现IOC功能，Sprign的所有功能都是借助IOC实现的。

2，AOP

AOP模块是Spring的AOP库，提供了AOP（拦截器）机制，并提供常用的拦截器，供用户自定义和配置。

3，ORM

Spring 的ORM模块提供对常用的ORM框架的管理和辅助支持，Spring支持常用的Hibernate，ibtas，jdao等框架的支持，Spring本身并不对ORM进行实现，仅对常见的ORM框架进行封装，并对其进行管理

4，DAO模块

Spring 提供对JDBC的支持，对JDBC进行封装，允许JDBC使用Spring资源，并能统一管理JDBC事物，并不对JDBC进行实现。（执行sql语句）

5，WEB模块

WEB模块提供对常见框架如Struts1，WEBWORK（Struts 2），JSF的支持，Spring能够管理这些框架，将Spring的资源注入给框架，也能在这些框架的前后插入拦截器。

6，Context模块

Context模块提供框架式的Bean访问方式，其他程序可以通过Context访问Spring的Bean资源，相当于资源注入。

7，MVC模块

WEB MVC模块为Spring提供了一套轻量级的MVC实现，在Spring的开发中，我们既可以用Struts也可以用Spring自己的MVC框架，相对于Struts，Spring自己的MVC框架更加简洁和方便。

## spring 常用的注入方式有哪些？

常用的注入方式主要有三种：构造方法注入，setter注入，基于注解的注入。

1、构造方法注入：

<!-- 注册userService -->

<bean id="userService" class="com.lyu.spring.service.impl.UserService">

<constructor-arg ref="userDaoJdbc"></constructor-arg>

</bean>

<!-- 注册jdbc实现的dao -->

<bean id="userDaoJdbc" class="com.lyu.spring.dao.impl.UserDaoJdbc"></bean>

2、setter注入

<!-- 注册userService -->

<bean id="userService" class="com.lyu.spring.service.impl.UserService">

<!-- 写法一 -->

<!-- <property name="UserDao" ref="userDaoMyBatis"></property> -->

<!-- 写法二 -->

<property name="userDao" ref="userDaoMyBatis"></property>

</bean>

<!-- 注册mybatis实现的dao -->

<bean id="userDaoMyBatis" class="com.lyu.spring.dao.impl.UserDaoMyBatis"></bean>

3、注解注入

在介绍注解注入的方式前，先简单了解bean的一个属性autowire，autowire主要有三个属性值：constructor，byName，byType。

constructor：通过构造方法进行自动注入，spring会匹配与构造方法参数类型一致的bean进行注入，如果有一个多参数的构造方法，一个只有一个参数的构造方法，在容器中查找到多个匹配多参数构造方法的bean，那么spring会优先将bean注入到多参数的构造方法中。

byName：被注入bean的id名必须与set方法后半截匹配，并且id名称的第一个单词首字母必须小写，这一点与手动set注入有点不同。

byType：查找所有的set方法，将符合符合参数类型的bean注入。

下面进入正题：注解方式注册bean，注入依赖

主要有四种注解可以注册bean，每种注解可以任意使用，只是语义上有所差异：

@Component：可以用于注册所有bean

@Repository：主要用于注册dao层的bean

@Controller：主要用于注册控制层的bean

@Service：主要用于注册服务层的bean

描述依赖关系主要有两种：

1、@Resource：java的注解，默认以byName的方式去匹配与属性名相同的bean的id，如果没有找到就会以byType的方式查找，如果byType查找到多个的话，使用@Qualifier注解（spring注解）指定某个具体名称的bean。

2、@Autowired：spring注解，默认是以byType的方式去匹配与属性名相同的bean的id，如果没有找到，就通过byName的方式去查找。

## spring 中的 bean 是线程安全的吗？

**Spring容器中的Bean本身不具备线程安全的特性**，但是具体还是要结合具体scope的Bean去研究。

Spring 的 bean 作用域（scope）类型

1、singleton:单例，默认作用域。

2、prototype:原型，每次创建一个新对象。

3、request:请求，每次Http请求创建一个新对象，适用于WebApplicationContext环境下。

4、session:会话，同一个会话共享一个实例，不同会话使用不用的实例。

5、global-session:全局会话，所有会话共享一个实例。

**原型Bean**

对于原型Bean,每次创建一个新对象，也就是线程之间并不存在Bean共享，自然是不会有线程安全的问题。

**单例Bean**

对于单例Bean,所有线程都共享一个单例实例Bean,因此是存在资源的竞争。

如果单例Bean,是一个无状态Bean，也就是线程中的操作不会对Bean的成员执行查询以外的操作，那么这个单例Bean是线程安全的。比如Spring mvc 的 Controller、Service、Dao等，这些Bean大多是无状态的，只关注于方法本身。

对于有状态的bean，Spring官方提供的bean，一般提供了通过ThreadLocal去解决线程安全的方法，比如RequestContextHolder、TransactionSynchronizationManager、LocaleContextHolder等。

使用ThreadLocal的好处

使得多线程场景下，多个线程对这个单例Bean的成员变量并不存在资源的竞争，因为ThreadLocal为每个线程保存线程私有的数据。这是一种以空间换时间的方式。

当然也可以通过加锁的方法来解决线程安全，这种以时间换空间的场景在高并发场景下显然是不实际的。

RequestContextHolder介绍：

<https://www.jianshu.com/p/80165b7743cf>

## spring 支持几种 bean 的作用域？

Spring 的 bean 作用域（scope）类型

1、singleton:单例，默认作用域。

2、prototype:原型，每次创建一个新对象。

3、request:请求，每次Http请求创建一个新对象，适用于WebApplicationContext环境下。

4、session:会话，同一个会话共享一个实例，不同会话使用不用的实例。

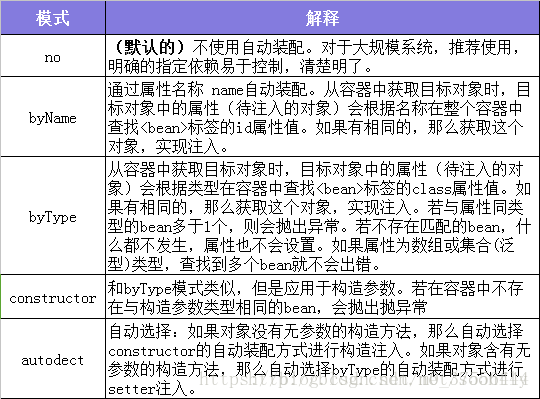
5、global-session:全局会话，所有会话共享一个实例。

## spring 自动装配 bean 有哪些方式？

<https://blog.csdn.net/m0_37556444/article/details/83108929>

1、xml方式

autowire属性可以设置为5种值：



2、注解方式

直接在类定义中使用@Autowired或@Resource来装配bean。

在使用注解装配之前，首先要开启注解装配的方式，在配置文件中加上下面这句话

<context:annotation-config>

使用@Autowired注解

可用于属性、构造函数、setter方法上。

使用@Autowired自动装配时，容器中只能有一个适合的Bean待选，否则的话，spring会抛出异常。(因为@Autowired默认是使用byType的方式装配)

如果在应用上下文当中找不到相应的bean去自动装配，那么spring也会抛出异常（NoSuchBeanDefinitionException）。

如果想避免这种情况发生，而且需要装配的属性也不是必须要装配的话，可以使用如下代码来使用注解：

@Autowired(required=false)

**@Qualifier注解**

刚才提到，如果在容器中出现了两个适合的bean，就会出错。怎么解决呢？这个时候可以使用@Qualifier注解指定一个Bean来装配，这样就不会报异常了。@Qualifier注解采用的是byName的方式。

**@Value注解**   
在使用注解自动装配的过程当中，如果想要自动装配基本类型的或者是字面值常量的参数的话，可以是用@Value注解

## spring 事务实现方式有哪些？

<https://www.cnblogs.com/WJ-163/p/6035462.html>

1.aspectJ AOP实现事务：

<bean id="dataSourceTransactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dateSource"></property>

</bean>

<tx:advice id="stockAdvice" transaction-manager="dataSourceTransactionManager">

<tx:attributes>

<tx:method name="by\*" isolation="DEFAULT" propagation="REQUIRED" rollback-for="MyExepction"/>

<tx:method name="select\*" isolation="DEFAULT" propagation="REQUIRED" read-only="true"></tx:method>

/tx:attributes>

</tx:advice>

<aop:config>

<aop:pointcut id="exAdvice" expression="execution(\* \*..service.\*.\*(..))"></aop:pointcut>

<aop:advisor advice-ref="stockAdvice" pointcut-ref="exAdvice"></aop:advisor>

</aop:config>

2.事务代理工厂Bean实现事务：

<bean id="tproxy" class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">

<property name="transactionManager" ref="dataSourceTransactionManager"></property>&lt;!&ndash;写的是事务&ndash;&gt;

<property name="target" ref="byStockService"></property>&lt;!&ndash;要进行事务的类&ndash;&gt;

<property name="transactionAttributes">

<props>&lt;!&ndash;key写的是service层要增强的方法&ndash;&gt;

&lt;!&ndash; 事务的隔离级别，后面逗号后面是异常类，用于回滚数据&ndash;&gt;

<prop key="ByStock">ISOLATION\_DEFAULT,PROPAGATION\_REQUIRED,-MyExepction</prop>

</props>

</property>

</bean>

3.注解方式实现事务：

<tx:annotation-driven transaction-manager="dataSourceTransactionManager"/>

在需要进行事务的方法上增加一个注解“@Transactional(rollbackFor = MyExepction.class )”

## 说一下 spring 的事务隔离？

<https://www.cnblogs.com/zhishan/p/3195219.html>

### 传播行为

事务的第一个方面是传播行为。传播行为定义关于客户端和被调用方法的事务边界。Spring定义了7中传播行为。

### 隔离级别

声明式事务的第二个方面是隔离级别。隔离级别定义一个事务可能受其他并发事务活动活动影响的程度。另一种考虑一个事务的隔离级别的方式，是把它想象为那个事务对于事物处理数据的自私程度。

在一个典型的应用程序中，多个事务同时运行，经常会为了完成他们的工作而操作同一个数据。并发虽然是必需的，但是会导致一下问题：

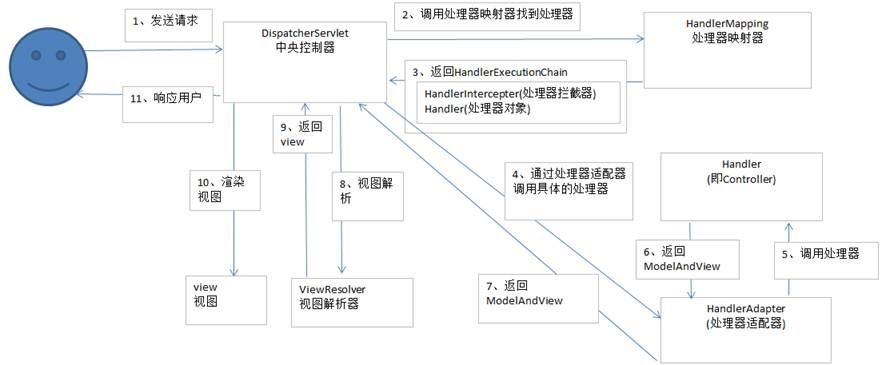
* 脏读（Dirty read）-- 脏读发生在一个事务读取了被另一个事务改写但尚未提交的数据时。如果这些改变在稍后被回滚了，那么第一个事务读取的数据就会是无效的。
* 不可重复读（Nonrepeatable read）-- 不可重复读发生在一个事务执行相同的查询两次或两次以上，但每次查询结果都不相同时。这通常是由于另一个并发事务在两次查询之间更新了数据。
* 幻影读（Phantom reads）-- 幻影读和不可重复读相似。当一个事务（T1）读取几行记录后，另一个并发事务（T2）插入了一些记录时，幻影读就发生了。在后来的查询中，第一个事务（T1）就会发现一些原来没有的额外记录。
* 在理想状态下，事务之间将完全隔离，从而可以防止这些问题发生。然而，完全隔离会影响性能，因为隔离经常牵扯到锁定在数据库中的记录（而且有时是锁定完整的数据表）。侵占性的锁定会阻碍并发，要求事务相互等待来完成工作。
* 考虑到完全隔离会影响性能，而且并不是所有应用程序都要求完全隔离，所以有时可以在事务隔离方面灵活处理。因此，就会有好几个隔离级别。

|  |  |
| --- | --- |
| ISOLATION\_DEFAULT | 使用后端数据库默认的隔离级别。 |
| ISOLATION\_READ\_UNCOMMITTED | 允许读取尚未提交的更改。可能导致脏读、幻影读或不可重复读。 |
| ISOLATION\_READ\_COMMITTED | 允许从已经提交的并发事务读取。可防止脏读，但幻影读和不可重复读仍可能会发生。 |
| ISOLATION\_REPEATABLE\_READ | 对相同字段的多次读取的结果是一致的，除非数据被当前事务本身改变。可防止脏读和不可重复读，但幻影读仍可能发生。 |
| ISOLATION\_SERIALIZABLE | 完全服从ACID的隔离级别，确保不发生脏读、不可重复读和幻影读。这在所有隔离级别中也是最慢的，因为它通常是通过完全锁定当前事务所涉及的数据表来完成的。 |

<https://blog.csdn.net/baidu_37107022/article/details/77481670>

### MySQL的默认隔离级别就是Repeatable,Oracle默认Read committed，最高级别Serializable

## 说一下 spring mvc 运行流程？



流程   
1、用户发送请求至前端控制器DispatcherServlet   
2、DispatcherServlet收到请求调用HandlerMapping处理器映射器。   
3、处理器映射器找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。   
4、DispatcherServlet调用HandlerAdapter处理器适配器   
5、HandlerAdapter经过适配调用具体的处理器(Controller，也叫后端控制器)。   
6、Controller执行完成返回ModelAndView   
7、HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet   
8、DispatcherServlet将ModelAndView传给ViewReslover视图解析器   
9、ViewReslover解析后返回具体View   
10、DispatcherServlet根据View进行渲染视图（即将模型数据填充至视图中）。   
11、DispatcherServlet响应用户

## spring mvc 有哪些组件？

<https://www.jianshu.com/p/a728acd817b7>

<https://blog.csdn.net/qq_36827957/article/details/82964346>

### HandlerMapping

我们可以看到HandlerMapping接口中只定义了一个方法，就是通过request找到HandlerExecutionChain，而HandlerExecutionChain包装了一个Handler和一组Interceptors。

### HandlerAdapter

之所以需要HandlerAdapter是因为Spring MVC没有对Handler做任何规定，它可以是类，可以是方法，也可以是任何其他东西，我们可以看到Handler的类型是Object，这样会非常灵活。但是怎么让任意类型的Handler处理固定格式的请求呢？没错，就是使用适配器，每种Handler都要有对应的HandlerAdapter才能处理请求。

### HandlerExceptionResolver

在处理请求的过程中，难免会出现异常，HandlerExceptionResolver就是专门来处理异常的组件，它能根据异常设置ModelAndView，然后交给render进行渲染。我们来看下HandlerExceptionResolver的接口定义

public interface HandlerExceptionResolver {

ModelAndView resolveException(

HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex);

}

从上面的分析中我们可以知道HandlerExceptionResolver只能处理页面渲染之前的异常，页面渲染过程中的异常，它是不能处理的，这时可以让容器跳转到指定的错误页面来处理异常。

### ViewResolver

ViewResolver用来将String类型的视图名和本地化信息Local解析成View类型的视图，其接口定义如下

public interface ViewResolver {

View resolveViewName(String viewName, Locale locale) throws Exception;

}

### RequestToViewNameTranslator

ViewResolver是根据viewName查找View，但有的Handler处理完后并没有设置View也没有设置viewName，这时就需要RequestToViewNameTranslator从request中找到默认的View了。我们来看下RequestToViewNameTranslator的接口定义

public interface RequestToViewNameTranslator {

String getViewName(HttpServletRequest request) throws Exception;

}

### LocaleResolver

解析视图需要两个参数：一是视图名，另一个是Locale。视图名是处理器返回的，Locale是从哪里来的？这就是LocaleResolver要做的事情。LocaleResolver用于从request解析出Locale，Locale就是zh-cn之类，表示一个区域，有了这个就可以对不同区域的用户显示不同的结果。SpringMVC主要有两个地方用到了Locale：一是ViewResolver视图解析的时候；二是用到国际化资源或者主题的时候。

### ThemeResolver

用于解析主题。SpringMVC中一个主题对应一个properties文件，里面存放着跟当前主题相关的所有资源、如图片、css样式等。SpringMVC的主题也支持国际化，同一个主题不同区域也可以显示不同的风格。SpringMVC中跟主题相关的类有 ThemeResolver、ThemeSource和Theme。主题是通过一系列资源来具体体现的，要得到一个主题的资源，首先要得到资源的名称，这是ThemeResolver的工作。然后通过主题名称找到对应的主题（可以理解为一个配置）文件，这是ThemeSource的工作。最后从主题中获取资源就可以了。

### MultipartResolver

    用于处理上传请求。处理方法是将普通的request包装成MultipartHttpServletRequest，后者可以直接调用getFile方法获取File，如果上传多个文件，还可以调用getFileMap得到FileName->File结构的Map。此组件中一共有三个方法，作用分别是判断是不是上传请求，将request包装成MultipartHttpServletRequest、处理完后清理上传过程中产生的临时资源。

### FlashMapManager

<https://www.cnblogs.com/damens/p/6460652.html>

    用来管理FlashMap的，FlashMap主要用在redirect中传递参数。

retrieveAndUpdate这个方法是用来恢复参数的，对于恢复过的和超时的参数将都会被删除掉。

saveOutputFlashMap这个方法是用来保存参数的。

## @Autowired 的作用是什么？

@Autowired与@Resource都可以用来装配bean. 都可以写在字段上,或写在setter方法上。

@Autowired默认按类型装配（这个注解是属业spring的），默认情况下必须要求依赖对象必须存在，如果要允许null值，可以设置它的required属性为false，如：@Autowired(required=false) ，如果我们想使用名称装配可以结合@Qualifier注解进行使用。

## @Autowired和@Resource的区别？

@Autowired//默认按type注入  
@Qualifier("cusInfoService")//一般作为@Autowired()的修饰用  
@Resource(name="cusInfoService")//默认按name注入，可以通过name和type属性进行选择性注入

一般@Autowired和@Qualifier一起用，@Resource单独用。

## 什么是 spring boot？

**SpringBoot是一个框架，一种全新的编程规范，他的产生简化了框架的使用，所谓简化是指简化了Spring众多框架中所需的大量且繁琐的配置文件，所以 SpringBoot是一个服务于框架的框架，服务范围是简化配置文件。**

## 为什么要用 spring boot？

简化XML的配置；

内置嵌入HTTP服务器（Tomcat（默认）、Jetty），降低了对环境的要求；

最终以Java应用程序进行执行（SpringBoot项目中没有Web.xml），运行中应用状态的监控；

独立运行

简化配置

自动配置

无代码生成和XML配置

应用监控

上手容易

## spring boot 核心配置文件是什么？

Spring Boot项目使用一个全局的配置文件application.properties或者是application.yml，在resources目录下或者类路径下的/config下，一般我们放到resources下。在这个配置文件中你可以做一些服务器与Spring 的相关配置以及日志打印等等(在这个配置文件中可以作大量的配置)。

虽然Spring Boot 已经为我们做了很多的配置，但是如果在相关的项目中你仍然需要xml 文件做一些额外的配置，那么Spring Boot 也是支持的。你可以在入口类通过@ImportResource 进行xml 配置文件的导入并且支持对多个xml 文件的配置。

@ImportResource({"classpath:\*.xml","classpath:\*\*.xml"})

**bootstrap (.yml 或者 .properties)**

**application (.yml 或者 .properties)**

Spring Cloud 构建于 Spring Boot 之上，在 Spring Boot 中有两种上下文，一种是 bootstrap, 另外一种是 application, bootstrap 是应用程序的父上下文，也就是说 **bootstrap 加载优先于 applicaton**。bootstrap 主要用于从额外的资源来加载配置信息，还可以在本地外部配置文件中解密属性。这两个上下文共用一个环境，它是任何Spring应用程序的外部属性的来源。bootstrap 里面的属性会优先加载，它们默认也不能被本地相同配置覆盖。

因此，对比 application 配置文件，bootstrap 配置文件具有以下几个特性。

boostrap 由父 ApplicationContext 加载，比 applicaton 优先加载

boostrap 里面的属性不能被覆盖

## spring boot 配置文件有哪几种类型？它们有什么区别？

<https://blog.csdn.net/qq_33524158/article/details/79600434>

SpringBoot使用一个全局的配置文件，配置文件名是固定的；

•application.properties

•application.yml

配置文件的作用：修改SpringBoot自动配置的默认值；SpringBoot在底层都给我们自动配置好；

## spring boot 有哪些方式可以实现热部署？

<https://www.jb51.net/article/112473.htm>

主要是devtools

## Spring Boot 的核心注解是哪个？它主要由哪几个注解组成的？

启动类上面的注解是@SpringBootApplication，它也是 Spring Boot 的核心注解，主要组合包含了以下 3 个注解：

@SpringBootConfiguration：组合了 @Configuration 注解，实现配置文件的功能。

@EnableAutoConfiguration：打开自动配置的功能，也可以关闭某个自动配置的选项，如关闭数据源自动配置功能： @SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })。

@ComponentScan：Spring组件扫描

## 开启 Spring Boot 特性有哪几种方式？

1）继承spring-boot-starter-parent项目

2）导入spring-boot-dependencies项目依赖

## Spring Boot 需要独立的容器运行吗？

可以不需要，内置了 Tomcat/ Jetty 等容器。

## Spring Boot自动配置原理？

Spring Boot的自动配置注解是@EnableAutoConfiguration， 从上面的@Import的类可以找到下面自动加载自动配置的映射。

<https://blog.csdn.net/u014745069/article/details/83820511> 这一篇讲的比较好

## 如何理解 Spring Boot 中的 Starters？

Starters可以理解为启动器，它包含了一系列可以集成到应用里面的依赖包，你可以一站式集成 Spring 及其他技术，而不需要到处找示例代码和依赖包。如你想使用 Spring JPA 访问数据库，只要加入 spring-boot-starter-data-jpa 启动器依赖就能使用了。

Starters包含了许多项目中需要用到的依赖，它们能快速持续的运行，都是一系列得到支持的管理传递性依赖。

## 如何在 Spring Boot 启动的时候运行一些特定的代码？

可以实现接口 ApplicationRunner 或者 CommandLineRunner，这两个接口实现方式一样，它们都只提供了一个 run 方法

## Spring Boot 有哪几种读取配置的方式？

Spring Boot 可以通过 @PropertySource,@Value,@Environment, @ConfigurationProperties 来绑定变量。

## Spring Boot 支持哪些日志框架？推荐和默认的日志框架是哪个？

Spring Boot 支持 Java Util Logging, Log4j2, Lockback 作为日志框架，如果你使用 Starters 启动器，Spring Boot 将使用 Logback 作为默认日志框架。

## Spring Boot 配置加载顺序？

1. 开发者工具 `Devtools` 全局配置参数；  
   2、单元测试上的 `@TestPropertySource` 注解指定的参数；  
   3、单元测试上的 `@SpringBootTest` 注解指定的参数；  
   4、命令行指定的参数，如 `java -jar springboot.jar --name="Java技术栈"`；  
   5、命令行中的 `SPRING\_APPLICATION\_JSON` 指定参数, 如 `java -Dspring.application.json='{"name":"Java技术栈"}' -jar springboot.jar`  
   6、`ServletConfig` 初始化参数；  
   7、`ServletContext` 初始化参数；  
   8、JNDI参数（如 `java:comp/env/spring.application.json`）；  
   9、Java系统参数（来源：`System.getProperties()`）；  
   10、操作系统环境变量参数；  
   11、`RandomValuePropertySource` 随机数，仅匹配：`ramdom.\*`；  
   **12、JAR包外面的配置文件参数（`application-{profile}.properties（YAML）`）  
   13、JAR包里面的配置文件参数（`application-{profile}.properties（YAML）`）  
   14、JAR包外面的配置文件参数（`application.properties（YAML）`）  
   15、JAR包里面的配置文件参数（`application.properties（YAML）`）**  
   16、`@Configuration`配置文件上 `@PropertySource` 注解加载的参数；  
   17、默认参数（通过 `SpringApplication.setDefaultProperties` 指定）；

## Spring Boot 可以兼容老 Spring 项目吗，如何做？

可以兼容，使用 @ImportResource 注解导入老 Spring 项目配置文件。

## 保护 Spring Boot 应用有哪些方法？

<https://blog.csdn.net/zl1zl2zl3/article/details/83716770>

## Spring Boot、Spring MVC 和 Spring 有什么区别？

风马牛不相及，问这种问题的都是傻缺。

## 为什么我们需要 spring-boot-maven-plugin?

spring-boot-maven-plugin 提供了一些像 jar 一样打包或者运行应用程序的命令。

spring-boot:run 运行你的 SpringBooty 应用程序。

spring-boot：repackage 重新打包你的 jar 包或者是 war 包使其可执行

spring-boot：start 和 spring-boot：stop 管理 Spring Boot 应用程序的生命周期（也可以说是为了集成测试）。

spring-boot:build-info 生成执行器可以使用的构造信息。

## 什么是 spring cloud？

Spring Cloud是一个微服务框架，相比Dubbo等RPC框架, **Spring Cloud提供的全套的分布式系统解决方案**。

      Spring Cloud对微服务基础框架Netflix的多个开源组件进行了封装，同时又实现了和云端平台以及和Spring Boot开发框架的集成。

      Spring Cloud为微服务架构开发涉及的**配置管理，服务治理，熔断机制，智能路由，微代理，控制总线，一次性token，全局一致性锁，leader选举，分布式session，集群状态**管理等操作提供了一种简单的开发方式。

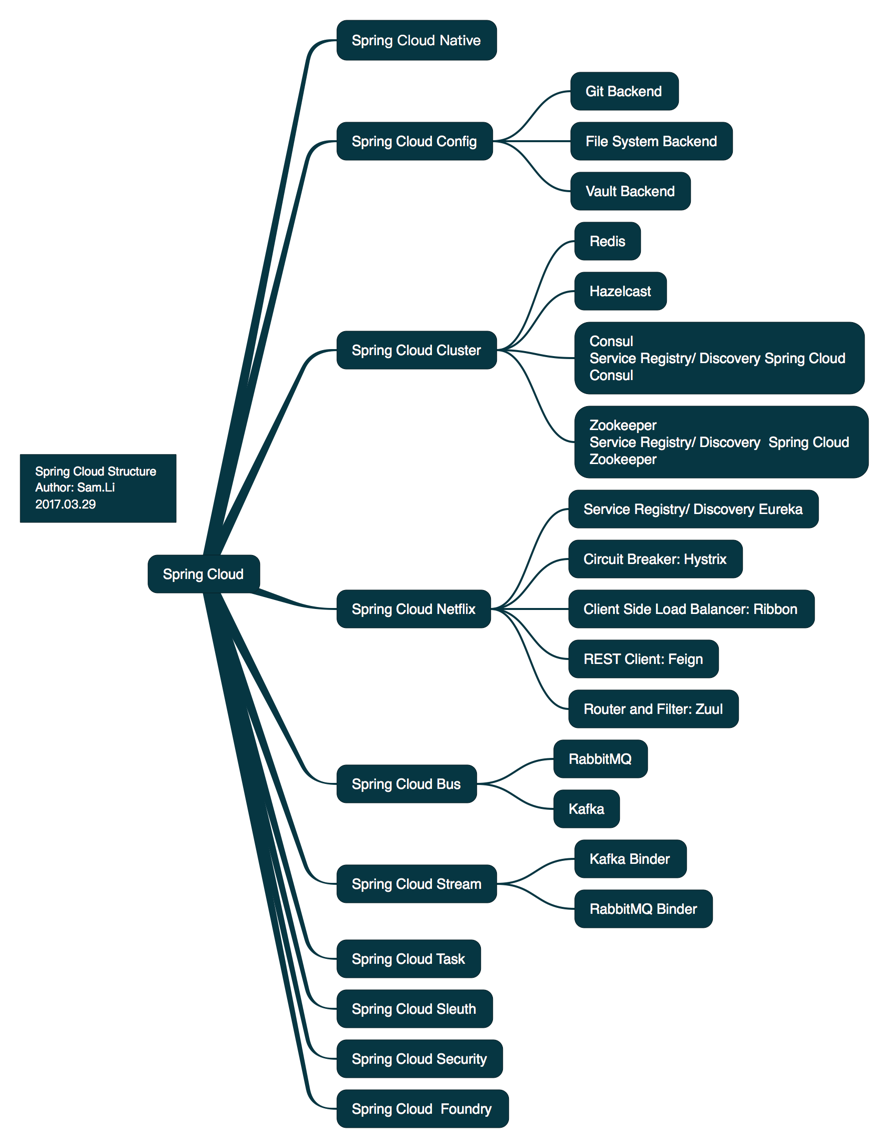
      Spring Cloud 为开发者提供了快速构建**分布式系统的工具**，开发者可以快速的启动服务或构建应用、同时能够快速和云平台资源进行对接。

## spring cloud 断路器的作用是什么？

<https://blog.csdn.net/u012702547/article/details/78004866>

## spring cloud 的核心组件有哪些？

<https://www.cnblogs.com/lexiaofei/p/6808152.html>

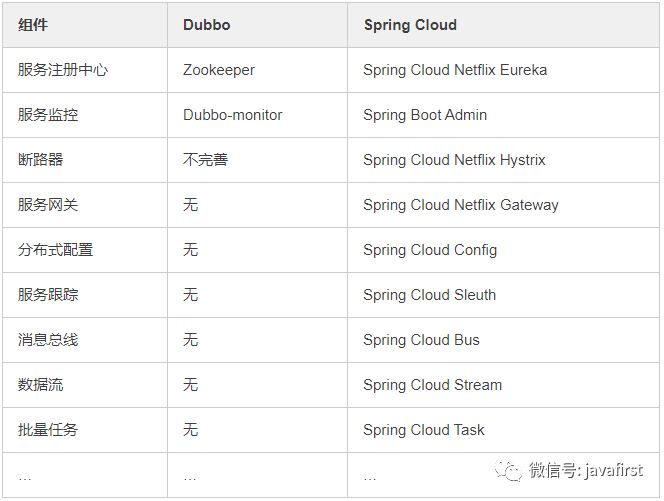


## **Dubbo 和 Spring Cloud 有什么区别？**

1）通信方式不同

Dubbo 使用的是 RPC 通信，而 Spring Cloud 使用的是 HTTP RESTFul 方式。

2）组成部分不同



## dubbo都支持什么协议，推荐用哪种？

dubbo://（推荐）

rmi://

hessian://

http://

webservice://

thrift://

memcached://

redis://

rest://

## Dubbo需要 Web 容器吗？

不需要，如果硬要用 Web 容器，只会增加复杂性，也浪费资源。

## Dubbo内置了哪几种服务容器？

Spring Container

Jetty Container

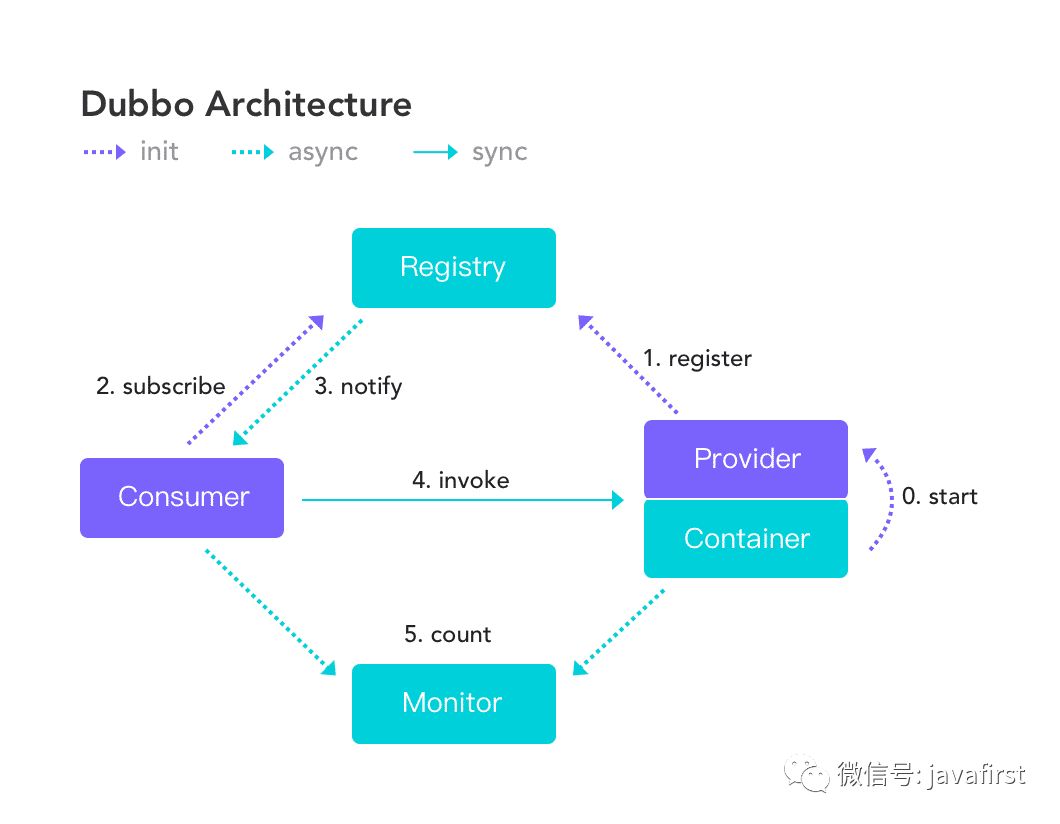
Log4j Container

Dubbo 的服务容器只是一个简单的 Main 方法，并加载一个简单的 Spring 容器，用于暴露服务。

## Dubbo里面有哪几种节点角色？



## **服务注册与发现的流程图？**



## Dubbo默认使用什么注册中心，还有别的选择吗？

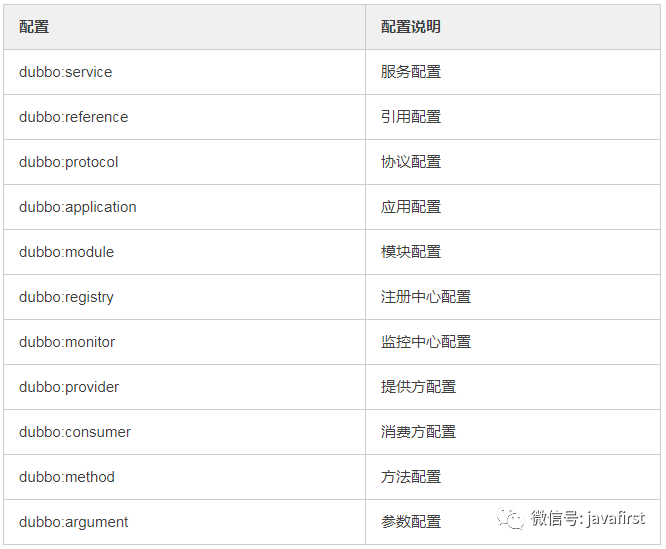
推荐使用 Zookeeper 作为注册中心，还有 Redis、Multicast、Simple 注册中心，但不推荐。

## Dubbo有哪几种配置方式？

1）Spring 配置方式

2）Java API 配置方式

## Dubbo 核心的配置有哪些？



## 在 Provider 上可以配置的 Consumer 端的属性有哪些？

1）timeout：方法调用超时

2）retries：失败重试次数，默认重试 2 次

3）loadbalance：负载均衡算法，默认随机

4）actives 消费者端，最大并发调用限制

## Dubbo启动时如果依赖的服务不可用会怎样？

Dubbo 缺省会在启动时检查依赖的服务是否可用，不可用时会抛出异常，阻止 Spring 初始化完成，默认 check="true"，可以通过 check="false" 关闭检查。

## Dubbo推荐使用什么序列化框架，你知道的还有哪些？

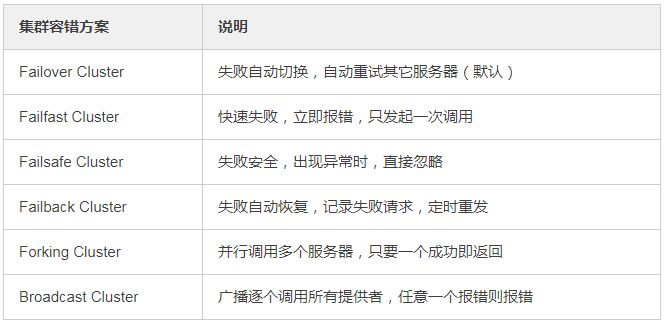
hessian是其默认的序列化协议

推荐使用Hessian序列化，还有Duddo、FastJson、Java自带序列化。

## Dubbo默认使用的是什么通信框架，还有别的选择吗？

Dubbo 默认使用 Netty 框架，也是推荐的选择，另外内容还集成有Mina、Grizzly。

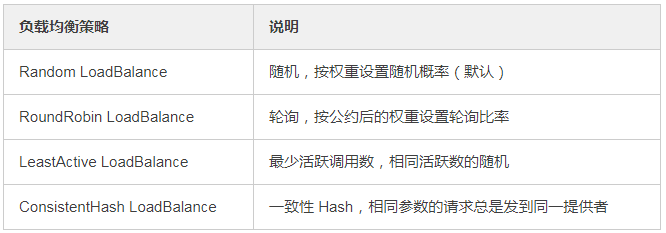
## Dubbo有哪几种集群容错方案，默认是哪种？



<dubbo:service cluster="failsafe" />

<dubbo:reference cluster="failsafe" />

## Dubbo有哪几种负载均衡策略，默认是哪种？



LeastActive LoadBalance的解释：

最少活跃调用数，相同活跃数的随机，活跃数指调用前后计数差。

使慢的提供者收到更少请求，因为越慢的提供者的调用前后计数差会越大。

**配置方法：**

服务端服务级别

<dubbo:service interface="..." loadbalance="roundrobin" />

客户端服务级别

<dubbo:reference interface="..." loadbalance="roundrobin" />

服务端方法级别

<dubbo:service interface="...">

<dubbo:method name="..." loadbalance="roundrobin"/>

</dubbo:service>

客户端方法级别

<dubbo:reference interface="...">

<dubbo:method name="..." loadbalance="roundrobin"/>

</dubbo:reference>

## 注册了多个同一样的服务，如果测试指定的某一个服务呢？

可以配置环境点对点直连，绕过注册中心，将以服务接口为单位，忽略注册中心的提供者列表。

## Dubbo支持服务多协议吗？

Dubbo 允许配置多协议，在不同服务上支持不同协议或者同一服务上同时支持多种协议。

## 当一个服务接口有多种实现时怎么做？

当一个接口有多种实现时，可以用 group 属性来分组，服务提供方和消费方都指定同一个 group 即可。

## 服务上线怎么兼容旧版本？

可以用版本号（version）过渡，多个不同版本的服务注册到注册中心，版本号不同的服务相互间不引用。这个和服务分组的概念有一点类似。

## Dubbo可以对结果进行缓存吗？

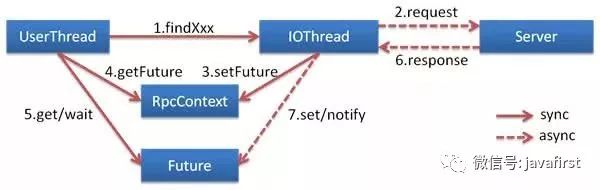
可以，Dubbo 提供了声明式缓存，用于加速热门数据的访问速度，以减少用户加缓存的工作量。

## Dubbo服务之间的调用是阻塞的吗？

默认是同步等待结果阻塞的，支持异步调用。

Dubbo 是基于 NIO 的非阻塞实现并行调用，客户端不需要启动多线程即可完成并行调用多个远程服务，相对多线程开销较小，异步调用会返回一个 Future 对象。

异步调用流程图如下。



## Dubbo telnet 命令能做什么？

dubbo 通过 telnet 命令来进行服务治理

<https://mp.weixin.qq.com/s?__biz=MzU5NTAzNjM0Mw==&mid=2247485628&idx=2&sn=b12f8aaaab4c99db014f18024edbe6d0&chksm=fe795964c90ed072773a8529a3aef6f5a1ebc0827cb639a5fc77cf4a3dc0594c151e41779078&scene=21#wechat_redirect>

## Dubbo支持服务降级吗？

Dubbo 2.2.0 以上版本支持。

服务降级：<https://blog.csdn.net/ityouknow/article/details/81230412>

## Dubbo如何优雅停机？

Dubbo 是通过 JDK 的 ShutdownHook 来完成优雅停机的，所以如果使用 kill -9 PID 等强制关闭指令，是不会执行优雅停机的，只有通过 kill PID 时，才会执行。

## § Dubbo完整的一次调用链路介绍；

<https://blog.csdn.net/icool_ali/article/details/80985130>

<https://www.jianshu.com/p/4fedd4946009>

## Dubbo Provider服务提供者要控制执行并发请求上限，具体怎么做？

在Provider配置的Consumer端属性：actives，消费者端，最大并发调用限制，即当Consumer对一个服务的并发调用到上限后，新调用会Wait直到超时

Provider上配置的Provider端属性： executes，一个服务提供者并行执行请求上限，即当Provider对一个服务的并发调用到上限后，新调用会Wait（Consumer可能到超时）。在方法上配置（dubbo:method ）则并发限制针对方法，在接口上配置（dubbo:service），则并发限制针对服务。

## § Dubbo启动的时候支持几种配置方式？

1. XML 配置文件方式

2. properties 配置文件方式

3. annotation 配置方式

4. API 配置方式

## § 了解几种消息中间件产品？各产品的优缺点介绍；

RabbitMQ、RocketMQ、ActiveMQ、Kafka

<https://blog.csdn.net/weixin_38229356/article/details/85057370>

## § 消息中间件如何保证消息的一致性和如何进行消息的重试机制？

(1)主动方应用先把消息发送给消息中间件，消息状态标记为待确认;

(2)消息中间件收到消息之后，把消息持久化到消息存储中，但并不向被动方应用投递消息；

(3)消息中间件返回消息持久化结果（成功，或者失效），主动方应用根据返回结果进行判断如何处理业务操作处理；

①失败：放弃业务操作处理，结束（必须向上层返回失败结果）

②成功：执行业务操作处理

(4)业务操作完成后，把业务操作结果（成功/失败）发送给消息中间件；

(5)消息中间件收到业务操作结果后，根据结果进行处理；

①失败：删除消息存储中的消息，结束；

②成功：更新消息存储中的消息状态为·待发送（可发送）·，紧接着执行消息投递；

(6)前面的正向流程都成功后，向被动方应用投递消息；

## § Spring Cloud熔断机制介绍；

<https://www.cnblogs.com/huangjuncong/p/9026949.html>

## § Spring Cloud对比下Dubbo，什么场景下该使用Spring Cloud？

<https://blog.csdn.net/kobejayandy/article/details/52078275>

## 服务提供者能实现失效踢出是什么原理？

服务失效踢出基于 Zookeeper 的临时节点原理。

## 如何解决服务调用链过长的问题？

Dubbo 可以使用 Pinpoint 和 Apache Skywalking(Incubator) 实现分布式服务追踪，当然还有其他很多方案。

## 服务读写推荐的容错策略是怎样的？

读操作建议使用 Failover 失败自动切换，默认重试两次其他服务器。

写操作建议使用 Failfast 快速失败，发一次调用失败就立即报错。

## Dubbo的管理控制台能做什么？

管理控制台主要包含：路由规则，动态配置，服务降级，访问控制，权重调整，负载均衡，等管理功能。

## 说说 Dubbo 服务暴露的过程

Dubbo 会在 Spring 实例化完 bean 之后，在刷新容器最后一步发布 ContextRefreshEvent 事件的时候，通知实现了 ApplicationListener 的 ServiceBean 类进行回调 onApplicationEvent 事件方法，Dubbo 会在这个方法中调用 ServiceBean 父类 ServiceConfig 的 export 方法，而该方法真正实现了服务的（异步或者非异步）发布。

## Dubbo的执行流程：

项目一启动，加载配置文件的时候，就会初始化，服务的提供方ServiceProvider就会向注册中心注册自己提供的服务，当消费者在启动时，就会向注册中心订阅自己所需要的服务，如果服务提供方有数据变更等，注册中心将基于长连接的形式推送变更数据给消费者。

默认使用Dubbo协议：

连接个数：单连接

连接方式：长连接

传输协议：TCP

传输方式：NIO异步传输

序列化：Hessian二进制序列化

适用范围：传入传出参数数据包较小（建议小于100K），消费者比提供者个数多，单一消费者无法压满提供者，尽量不要使用dubbo协议传输大文件或超大字符串

使用场景：常规远程服务方法调用

从上面的适用范围总结，dubbo适合小数据量大并发的服务调用，以及消费者机器远大于生产者机器数的情况，不适合传输大数据量的服务比如文件、视频等，除非请求量很低。

dubbo心跳机制

## dubbo的心跳机制：

目的：检测provider与consumer之间的connection连接是不是还连接着，如果连接断了，需要作出相应的处理。

原理：

provider：dubbo的心跳默认是在heartbeat（默认是60s）内如果没有接收到消息，就会发送心跳消息，如果连着3次（180s）没有收到心跳响应，provider会关闭channel。

consumer：dubbo的心跳默认是在60s内如果没有接收到消息，就会发送心跳消息，如果连着3次（180s）没有收到心跳响应，consumer会进行重连。

## mybatis 中 #{}和 ${}的区别是什么？

<https://www.cnblogs.com/baizhanshi/p/5778692.html>

1. #将传入的数据都当成一个字符串，会对自动传入的数据加一个双引号。如：order by #user\_id#，如果传入的值是111,那么解析成sql时的值为order by "111", 如果传入的值是id，则解析成的sql为order by "id".

2. $将传入的数据直接显示生成在sql中。如：order by $user\_id$，如果传入的值是111,那么解析成sql时的值为order by user\_id, 如果传入的值是id，则解析成的sql为order by id.

3. #方式能够很大程度防止sql注入。

4.$方式无法防止Sql注入。

5.$方式一般用于传入数据库对象，例如传入表名.

6.一般能用#的就别用$.

**MyBatis排序时使用order by 动态参数时需要注意，用$而不是#**

## mybatis 有几种分页方式？

<https://blog.csdn.net/chenbaige/article/details/70846902>

1. 先查出所有数据再截取需要的数据；
2. 用limit语句实现分页；
3. 用自定义拦截器Interceptor实现分页；

<https://blog.csdn.net/chenbaige/article/details/72084481>

自定义拦截器实现了拦截所有以**ByPage**结尾的查询语句，并且利用获取到的分页相关参数统一在sql语句后面加上limit分页的相关语句，一劳永逸。不再需要在每个语句中单独去配置分页相关的参数了。

首先我们看一下拦截器的具体实现，在这里我们需要拦截所有以ByPage结尾的所有查询语句，因此要使用该拦截器实现分页功能，那么再定义名称的时候需要满足它拦截的规则（以ByPage结尾）。

四、RowBounds实现分页

原理：通过RowBounds实现分页和通过数组方式分页原理差不多，都是一次获取所有符合条件的数据，然后在内存中对大数据进行操作，实现分页效果。只是数组分页需要我们自己去实现分页逻辑，这里更加简化而已。

存在问题：一次性从数据库获取的数据可能会很多，对内存的消耗很大，可能导致性能变差，甚至引发内存溢出。

适用场景：在数据量很大的情况下，建议还是适用拦截器实现分页效果。RowBounds建议在数据量相对较小的情况下使用。

简单介绍：这是代码实现上最简单的一种分页方式，只需要在dao层接口中要实现分页的方法中加入RowBounds参数，然后在service层通过offset（从第几行开始读取数据，默认值为0）和limit（要显示的记录条数，默认为java允许的最大整数：2147483647）两个参数构建出RowBounds对象，在调用dao层方法的时，将构造好的RowBounds传进去就能轻松实现分页效果了。

## RowBounds 是一次性查询全部结果吗？为什么？

<https://blog.csdn.net/qq924862077/article/details/52611848>

是的。

## mybatis 逻辑分页和物理分页的区别是什么？

总结：

1：逻辑分页 内存开销比较大,在数据量比较小的情况下效率比物理分页高;在数据量很大的情况下,内存开销过大,容易内存溢出,不建议使用

2：物理分页 内存开销比较小,在数据量比较小的情况下效率比逻辑分页还是低,在数据量很大的情况下,建议使用物理分页

## mybatis 是否支持延迟加载？延迟加载的原理是什么？

<https://blog.csdn.net/shfqbluestone/article/details/52888144>

Mybatis支持延迟加载。

延迟加载的原理是cglib动态代理。

## 说一下 mybatis 的一级缓存和二级缓存？

一级缓存：Mybatis 的一级缓存的作用域是 session，当 openSession()后，如果执行相同的SQL（相同语句和参数），Mybatis 不进行执行 SQL，而是从缓存中命中返回。

二级缓存：Mybatis 的二级缓存的作用域是一个 mapper 的 namespace，同一个 namespace 中查询 sql 可以从缓存中命中。二级缓存是可以跨 session 的.

## mybatis 和 hibernate 的区别有哪些？

        1、Mybatis和hibernate不同，它不完全是一个ORM框架，因为MyBatis需要程序员自己编写Sql语句，不过mybatis可以通过XML或注解方式灵活配置要运行的sql语句，并将java对象和sql语句映射生成最终执行的sql，最后将sql执行的结果再映射生成java对象。

        2、Mybatis学习门槛低，简单易学，程序员直接编写原生态sql，可严格控制sql执行性能，灵活度高，非常适合对关系数据模型要求不高的软件开发，例如互联网软件、企业运营类软件等，因为这类软件需求变化频繁，一但需求变化要求成果输出迅速。但是灵活的前提是mybatis无法做到数据库无关性，如果需要实现支持多种数据库的软件则需要自定义多套sql映射文件，工作量大。

        3、Hibernate对象/关系映射能力强，数据库无关性好，对于关系模型要求高的软件（例如需求固定的定制化软件）如果用hibernate开发可以节省很多代码，提高效率。但是Hibernate的缺点是学习门槛高，要精通门槛更高，而且怎么设计O/R映射，在性能和对象模型之间如何权衡，以及怎样用好Hibernate需要具有很强的经验和能力才行。

## mybatis 有哪些执行器（Executor）？

<https://www.jianshu.com/p/96ddaec4aea7>

**SimpleExecutor：**每执行一次update或select，就开启一个Statement对象，用完立刻关闭Statement对象。

**ReuseExecutor：**执行update或select，以sql作为key查找Statement对象，存在就使用，不存在就创建，用完后，不关闭Statement对象，而是放置于Map内，供下一次使用。简言之，就是重复使用Statement对象。

**BatchExecutor：**执行update（没有select，JDBC批处理不支持select），将所有sql都添加到批处理中（addBatch()），等待统一执行（executeBatch()），它缓存了多个Statement对象，每个Statement对象都是addBatch()完毕后，等待逐一执行executeBatch()批处理。与JDBC批处理相同。

作用范围：Executor的这些特点，都严格限制在SqlSession生命周期范围内。

## Mybatis中如何指定使用哪一种Executor执行器？

答：在Mybatis配置文件中，可以指定默认的ExecutorType执行器类型，也可以手动给DefaultSqlSessionFactory的创建SqlSession的方法传递ExecutorType类型参数。

## mybatis 分页插件的实现原理是什么？

<https://blog.csdn.net/itcats_cn/article/details/81586724>

## mybatis 如何编写一个自定义插件？

<http://www.cnblogs.com/drafire/p/9773071.html>

实现Interceptor接口

在plugins标签中配置plugin标签。

## redis 是什么？都有哪些使用场景？

Redis是内存数据库

作用：

* 解决应用服务器的cpu和内存压力
* 减少io的读操作，减轻io的压力
* 关系型数据库的扩展性不强，难以改变表结构

使用场景：

1. 配合关系型数据库做高速缓存
2. 缓存高频次访问的数据，降低数据库io
3. 分布式架构，做session共享
4. 可以持久化特定数据。
5. **利用zset类型可以存储排行榜**
6. **利用list的自然时间排序存储最新n个数据**

## redis 有哪些功能？

<https://blog.csdn.net/u010963948/article/details/79589710>

额，这个问题，就像在问，你的小JJ有哪些功能一样废话。

你可以拿来上厕所，也可以拿来玩，想怎么玩就怎么玩。

## mysql里有2000w数据，redis中只存20w的数据，如何保证redis中的数据都是热点数据？

<http://wiki.jikexueyuan.com/project/redis/data-elimination-mechanism.html>

redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略（回收策略）。

redis 提供 6种数据淘汰策略：

volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选**最近最少使用**的数据淘汰

volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选**将要过期**的数据淘汰

volatile-random：从已设置过期时间的数据集（server.db[i].expires）中**任意选择数据**淘汰

allkeys-lru：从数据集（server.db[i].dict）中挑选**最近最少使用**的数据淘汰

allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

no-enviction（驱逐）：禁止驱逐数据

LRU 数据淘汰机制是这样的：**在数据集中随机挑选几个键值对**，取出其中 lru 最大的键值对淘汰。所以，你会发现，Redis 并不是保证取得所有数据集中最近最少使用（LRU）的键值对，而只是随机挑选的几个键值对中的。

Redis 数据集数据结构中保存了键值对过期时间的表，即 redisDb.expires，在使用 SET 命令的时候，就有一个键值对超时时间的选项。和 LRU 数据淘汰机制类似，TTL 数据淘汰机制是这样的：从过期时间 redisDB.expires 表中随机挑选几个键值对，取出其中 ttl 最大的键值对淘汰。同样你会发现，**Redis 并不是保证取得所有过期时间的表中最快过期的键值对，而只是随机挑选的几个键值对中的。**

无论是什么机制，都是从所有的键值对中挑选合适的淘汰。

使用原则：

1、如果数据呈现幂律分布，也就是一部分数据访问频率高，一部分数据访问频率低，则使用allkeys-lru  
 2、如果数据呈现平等分布，也就是所有的数据访问频率都相同，则使用allkeys-random

## redis 和 memecache 有什么区别？

1)、存储方式 Memecache把数据全部存在内存之中，断电后会挂掉，数据不能超过内存大小。 Redis有部份存在硬盘上，这样能保证数据的持久性。

2)、数据支持类型 Memcache对数据类型支持相对简单。 Redis有复杂的数据类型。

3)、使用底层模型不同 它们之间底层实现方式 以及与客户端之间通信的应用协议不一样。 Redis直接自己构建了VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求。

Redis比memcached效率高是因为单线程避免了多线程频繁的上下文切换。

## redis 为什么是单线程的？

**redis利用队列技术将并发访问变为串行访问**，消除了传统数据库串行控制的开销

redis的并发竞争问题如何解决?

　　　Redis为单进程单线程模式，**采用队列模式将并发访问变为串行访问**。Redis本身没有锁的概念，Redis对于多个客户端连接并不存在竞争，但是在Jedis客户端对Redis进行并发访问时会发生连接超时、数据转换错误、阻塞、客户端关闭连接等问题，这些问题均是

由于客户端连接混乱造成。对此有2种解决方法：

　　　1．客户端角度，为保证每个客户端间正常有序与Redis进行通信，对连接进行池化，同时对客户端读写Redis操作采用内部锁synchronized。

　　　2.服务器角度，利用setnx实现锁。

　　　注：对于第一种，需要应用程序自己处理资源的同步，可以使用的方法比较通俗，可以使用synchronized也可以使用lock；第二种需要用到Redis的setnx命令，但是需要注意一些问题。

## 什么是缓存穿透？怎么解决？

<https://baijiahao.baidu.com/s?id=1619572269435584821&wfr=spider&for=pc>

**缓存穿透，是指查询一个数据库一定不存在的数据。**

解决办法：如果从数据库查询的对象为空，也放入缓存，只是设定的缓存过期时间较短，比如设置为60秒。

**缓存雪崩，是指在某一个时间段，缓存集中过期失效。**

产生雪崩的原因之一，比如在写本文的时候，马上就要到双十二零点，很快就会迎来一波抢购，这波商品时间比较集中的放入了缓存，假设缓存一个小时。那么到了凌晨一点钟的时候，这批商品的缓存就都过期了。而对这批商品的访问查询，都落到了数据库上，对于数据库而言，就会产生周期性的压力波峰。

小编在做电商项目的时候，一般是采取不同分类商品，缓存不同周期。在同一分类中的商品，加上一个随机因子。这样能尽可能分散缓存过期时间，而且，热门类目的商品缓存时间长一些，冷门类目的商品缓存时间短一些，也能节省缓存服务的资源。

**缓存击穿，是指一个key非常热点，在不停的扛着大并发，大并发集中对这一个点进行访问，当这个key在失效的瞬间，持续的大并发就穿破缓存，直接请求数据库，就像在一个屏障上凿开了一个洞。**

小编在做电商项目的时候，把这货就成为“爆款”。

其实，大多数情况下这种爆款很难对数据库服务器造成压垮性的压力。达到这个级别的公司没有几家的。所以，务实主义的小编，对主打商品都是早早的做好了准备，让缓存永不过期。即便某些商品自己发酵成了爆款，也是直接设为永不过期就好了。

## redis 支持的数据类型有哪些？

String，list，set，hash，zset

## redis 支持的 java 客户端都有哪些？

Redisson,Jedis，lettuce等等，官方推荐使用Redisson。

## jedis 和 redisson 有哪些区别？

<https://blog.csdn.net/u014209205/article/details/82185826>

jedis简单封装了redis的命令；

redisson基于netty做了二次开发，很复杂。

## 怎么保证缓存和数据库数据的一致性？

<https://www.jianshu.com/p/a532962cb9e9>

## redis 持久化有几种方式？

<https://www.cnblogs.com/AndyAo/p/8135980.html>

创建快照文件或者AOF

## redis 怎么实现分布式锁？

<https://blog.csdn.net/liubenlong007/article/details/53782934>

<https://xiaozhuanlan.com/topic/4672859130>

为了确保分布式锁可用，我们至少要确保锁的实现同时满足以下四个条件：

1. 互斥性。在任意时刻，只有一个客户端能持有锁。
2. 不会发生死锁。即使有一个客户端在持有锁的期间崩溃而没有主动解锁，也能保证后续其他客户端能加锁。
3. 具有容错性。只要大部分的Redis节点正常运行，客户端就可以加锁和解锁。
4. 解铃还须系铃人。加锁和解锁必须是同一个客户端，客户端自己不能把别人加的锁给解了。

加锁：jedis.set()

解锁：jedis.eval()

## redis 分布式锁有什么缺陷？

<https://www.cnblogs.com/gxyandwmm/p/9588383.html>

<http://wemedia.ifeng.com/88987490/wemedia.shtml>

## redis 如何做内存优化？

<https://www.cnblogs.com/jandison/p/6902396.html>

## redis 淘汰策略有哪些？

<https://blog.csdn.net/ligupeng7929/article/details/79603060>

noeviction: 不删除策略, 达到最大内存限制时, 如果需要更多内存, 直接返回错误信息。 大多数写命令都会导致占用更多的内存(有极少数会例外, 如 DEL )。

* allkeys-lru: 所有key通用; 优先删除最近最少使用(less recently used ,LRU) 的 key。
* volatile-lru: 只限于设置了 expire 的部分; 优先删除最近最少使用(less recently used ,LRU) 的 key。
* allkeys-random: 所有key通用; 随机删除一部分 key。
* volatile-random: 只限于设置了 expire 的部分; 随机删除一部分 key。
* volatile-ttl: 只限于设置了 expire 的部分; 优先删除剩余时间(time to live,TTL) 短的key。

## redis 常见的性能问题有哪些？该如何解决？

* Master最好不要做任何持久化工作，包括内存快照和AOF日志文件，特别是不要启用内存快照做持久化。
* 如果数据比较关键，某个Slave开启AOF备份数据，策略为每秒同步一次。
* 为了主从复制的速度和连接的稳定性，Slave和Master最好在同一个局域网内。
* 尽量避免在压力较大的主库上增加从库
* 为了Master的稳定性，主从复制不要用图状结构，用单向链表结构更稳定，即主从关系为：Master<–Slave1<–Slave2<–Slave3…….，这样的结构也方便解决单点故障问题，实现Slave对Master的替换，也即，如果Master挂了，可以立马启用Slave1做Master，其他不变。

## § Redis为什么这么快？redis采用多线程会有哪些问题？

1、完全基于内存，绝大部分请求是纯粹的内存操作，非常快速。数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O(1)；

2、数据结构简单，对数据操作也简单，Redis中的数据结构是专门进行设计的；

3、采用单线程，避免了不必要的上下文切换和竞争条件，也不存在多进程或者多线程导致的切换而消耗 CPU，不用去考虑各种锁的问题，不存在加锁释放锁操作，没有因为可能出现死锁而导致的性能消耗；

4、使用多路I/O复用模型，非阻塞IO；

5、使用底层模型不同，它们之间底层实现方式以及与客户端

因为Redis是基于内存的操作，CPU不是Redis的瓶颈，Redis的瓶颈最有可能是机器内存的大小或者网络带宽。既然单线程容易实现，而且CPU不会成为瓶颈，那就顺理成章地采用单线程的方案了（毕竟采用多线程会有很多麻烦！）。

## § Redis支持哪几种数据结构；

String、list、set、hash、zset、

## § Redis跳跃表的问题；

## § Redis单进程单线程的Redis如何能够高并发?

IO多路复用

## § Redis如何使用Redis实现分布式锁？

Setnx命令

## § Redis分布式锁操作的原子性，Redis内部是如何实现的？