博客

2020年6月9日 9:40

Spring
https://blog.csdn.net/likun557/article/details/106045522?utm_medium=distribute.pc_feed.none-task-blog-alirecmd-19.nonecase&depth_1-utm_source=distribute.pc_feed.none-task-blog-alirecmd-19.nonecase&request_id=
Spring 事件

什么是Spring框架

2020年4月7日 8:06

Spring是一个轻量级的IoC和AOP容器框架。是为Java应用程序提供基础性服务的一套框架，目的是用于简化企业应用程序的开发，它使得开发者只需要关心业务需求。常见的配置方式有三种：基于XML的配置、基于注解的配置、基于Java的配置。

它是很多模块的集合，使用这些模块可以很方便地协助我们进行开发。这些模块是：核心容器、数据访问/集成,、Web、AOP（面向切面编程）、工具、消息和测试模块。比如：Core Container 中的 Core 组件是Spring 所有组件的核心，Beans 组件和 Context 组件是实现IOC和依赖注入的基础，AOP组件用来实现面向切面编程

IoC（Inverse of Control:控制反转）是一种**设计思想**，就是 **将原本在程序中手动创建对象的控制权，交由Spring框架来管理。** IoC 在其他语言中也有应用，并非 Spring 特有。 **IoC 容器是 Spring 用来实现 IoC 的载体， IoC 容器实际上就是个Map（key，value）,Map 中存放的是各种对象。**
将对象之间的相互依赖关系交给 IoC 容器来管理，并由 IoC 容器完成对象的注入。这样可以很大程度上简化应用的开发，把应用从复杂的依赖关系中解放出来。 **IoC 容器就像是一个工厂一样，当我们需要创建一个对象的时候，只需要配置好配置文件/注解即可，完全不用考虑对象是如何被创建出来的。** 在实际项目中一个 Service 类可能有几百甚至上千个类作为它的底层，假如我们需要实例化这个 Service，你可能要每次都要搞清这个 Service 所有底层类的构造函数，这可能会把人逼疯。如果利用 IoC 的话，你只需要配置好，然后在需要的地方引用就行了，这大大增加了项目的可维护性且降低了开发难度。

AOP(Aspect-Oriented Programming:面向切面编程)能够将那些与业务无关，**却为业务模块所共同调用的逻辑或责任（例如事务处理、日志管理、权限控制等）封装起来**，便于**减少系统的重复代码**，**降低模块间的耦合度**，并**有利于未来的可拓展性和可维护性。**
Spring AOP就是基于动态代理的，如果要代理的对象，实现了某个接口，那么Spring AOP会使用**JDK Proxy**，去创建代理对象，而对于没有实现接口的对象，就无法使用 JDK Proxy 去进行代理了，这时候Spring AOP会使用**Cglib** ，这时候Spring AOP会使用 **Cglib** 生成一个被代理对象的子类来作为代理，如下图所示：

- singleton : 唯一 bean 实例，Spring 中的 bean 默认都是单例的。
- prototype : 每次请求都会创建一个新的 bean 实例。
- request : 每一次HTTP请求都会产生一个新的bean，该bean仅在当前HTTP request内有效。
- session : 每一次HTTP请求都会产生一个新的 bean，该bean仅在当前 HTTP session 内有效。
- global-session： 全局session作用域，仅仅在基于portlet的web应用中才有意义，Spring5已经没有了。Portlet是能够生成语义代码(例如：HTML)片段的小型Java Web插件。它们基

于portlet容器，可以像servlet一样处理HTTP请求。但是，与 servlet 不同，每个 portlet 都有不同的会话

4、请解释下 Spring 框架中的 IOC?

（1）IOC就是控制反转，是指创建对象的控制权的转移，以前创建对象的主动权和时机是由自己把控的，而现在这种权力转移到Spring容器中，并由容器根据配置文件去创建实例和管理各个实例之间的依赖关系，对象与对象之间松散耦合，也利于功能的复用。DI依赖注入，和控制反转是同一个概念的不同角度的描述，即 应用程序在运行时依赖IoC容器来动态注入对象需要的外部资源。

（2）最直观的表达就是，IOC让对象的创建不用去new了，可以由spring自动生产，使用java的反射机制，根据配置文件在运行时动态的去创建对象以及管理对象，并调用对象的方法的。

（3）Spring的IOC有三种注入方式： 构造器注入、setter方法注入、根据注解注入。

5、BeanFactory 和 ApplicationContext 有什么区别?

BeanFactory 可以理解为含有 bean 集合的工厂类。BeanFactory 包含了种 bean 的定义，以
便在接收到客户端请求时将对应的 bean 实例化。
BeanFactory 还能在实例化对象的时生成协作类之间的关系。此举将 bean 自身与 bean 客户端
的配置中解放出来。BeanFactory 还包含了 bean 生命周期的控制，调用客户端的初始化方法
（initialization methods）和销毁方法（destruction methods）。
从表面上看，application context 如同 bean factory 一样具有 bean 定义、bean 关联关系的
设置，根据请求分发 bean 的功能。但 application context 在此基础上还提供了其他的功能。
1.提供了支持国际化的文本消息
2.统一的资源文件读取方式
3.已在监听器中注册的 bean 的事件
以下是三种较常见的 ApplicationContext 实现方式：
1、ClassPathXmlApplicationContext：从 classpath 的 XML 配置文件中读取上下文，并生成
上下文定义。应用程序上下文从程序环境变量中取得。
ApplicationContext context = new
ClassPathXmlApplicationContext(“application.xml”);
2、FileSystemXmlApplicationContext：由文件系统中的 XML 配置文件读取上下文。
ApplicationContext context = new
FileSystemXmlApplicationContext(“application.xml”);
3、XmlWebApplicationContext：由 Web 应用的 XML 文件读取上下文。

重要的Spring模块

2020年4月7日 8:06

Spring Core：基础，可以说Spring其他所有的功能都依赖于该类库。主要提供IOC和DI功能。
Spring Aspects：该模块为与AspectJ的集成提供支持。
Spring AOP：提供面向切面的编程实现。
Spring JDBC：Java数据库连接。
Spring JMS：Java消息服务。
Spring ORM：用于支持Hibernate等ORM工具。
Spring Web：为创建Web应用程序提供支持。
Spring Test：提供了对JUnit和TestNG测试的支持。
Spring MVC：提供面向Web应用的Model-View-Controller实现。

Spring的优点

2020年4月7日 8:39

（1）spring属于低侵入式设计，代码的污染极低；

（2）spring的DI机制将对象之间的依赖关系交由框架处理，减低组件的耦合性；

（3）Spring提供了AOP技术，支持将一些通用任务，如安全、事务、日志、权限等进行集中式管理，从而提供更好的复用。

（4）spring对于主流的应用框架提供了集成支持

BeanFactory和ApplicationContext有什么区别?

2020年4月7日 9:00

BeanFactory和ApplicationContext是Spring的两大核心接口，都可以当做Spring的容器。其中ApplicationContext是BeanFactory的子接口。

（1）BeanFactory：是Spring里面最底层的接口，包含了各种Bean的定义，读取bean配置文档，管理bean的加载、实例化，控制bean的生命周期，维护bean之间的依赖关系。ApplicationContext接口作为BeanFactory的派生，除了提供BeanFactory所具有的功能外，还提供了更完整的框架功能：

①继承MessageSource，因此支持国际化。

②统一的资源文件访问方式。

③提供在监听器中注册bean的事件。

④同时加载多个配置文件。

⑤载入多个（有继承关系）上下文，使得每一个上下文都专注于一个特定的层次，比如应用的web层。

（2）①BeanFactroy采用的是延迟加载形式来注入Bean的，即只有在使用到某个Bean时(调用getBean())，才对该Bean进行加载实例化。这样，我们就不能发现一些存在的Spring的配置问题。如果Bean的某一个属性没有注入，BeanFacotry加载后，直至第一次使用调用getBean方法才会抛出异常。

②ApplicationContext，它是在容器启动时，一次性创建了所有的Bean。这样，在容器启动时，我们就可以发现Spring中存在的配置错误，这样有利于检查所依赖属性是否注入。 ApplicationContext启动后预载入所有的单实例Bean，通过预载入单实例bean ,确保当你需要的时候，你就不用等待，因为它们已经创建好了。

③相对于基本的BeanFactory，ApplicationContext 唯一的不足是占用内存空间。当应用程序配置Bean较多时，程序启动较慢。

（3）BeanFactory通常以编程的方式被创建，ApplicationContext还能以声明的方式创建，如使用ContextLoader。

（4）BeanFactory和ApplicationContext都支持BeanPostProcessor、BeanFactoryPostProcessor的使用，但两者之间的区别是：BeanFactory需要手动注册，而ApplicationContext则是自动注册。

有哪些常用的 Context?

最常被使用的 ApplicationContext 接口实现：

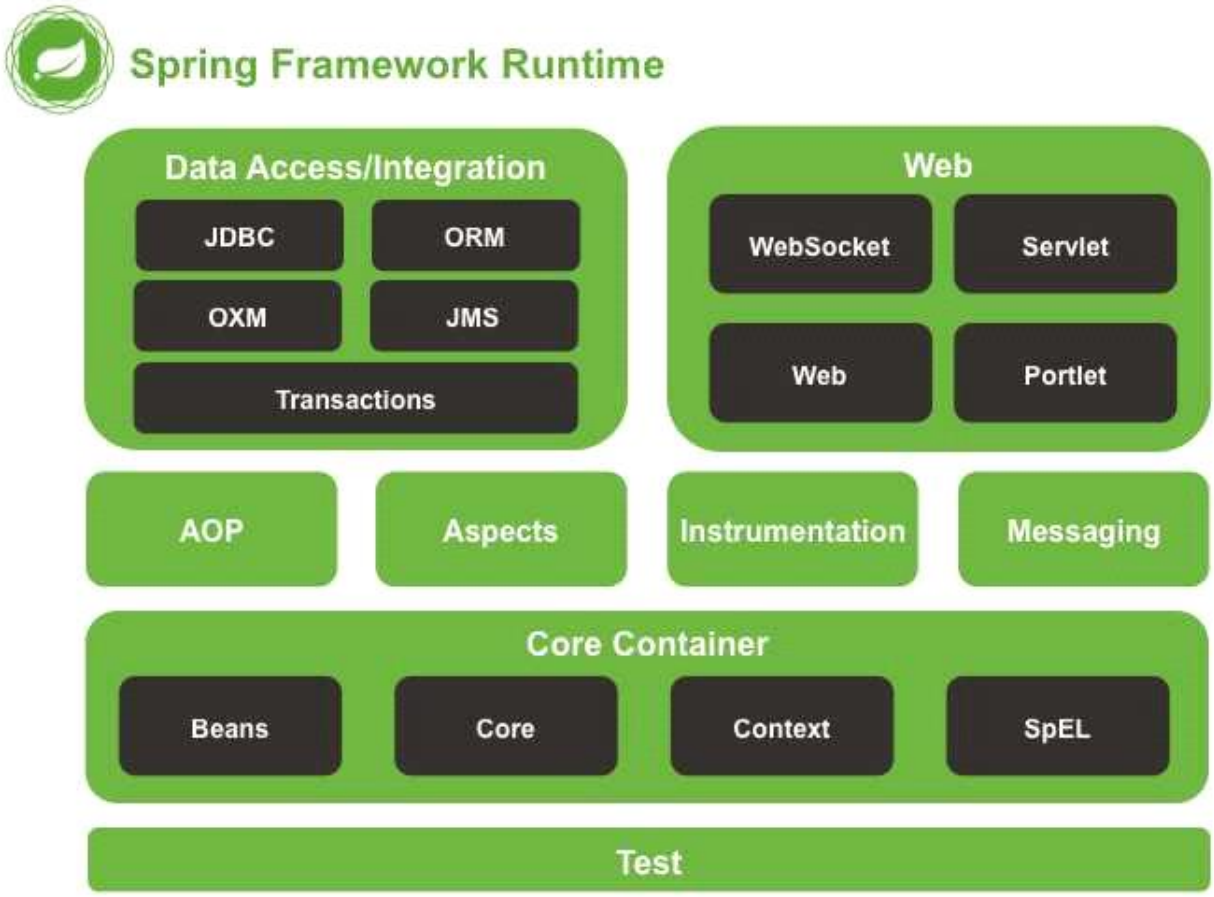
- FileSystemXmlApplicationContext：该容器从 XML 文件中加载已被定义的 bean。在这里，你需要提供给构造器 XML 文件的完整路径
- ClassPathXmlApplicationContext：该容器从 XML 文件中加载已被定义的 bean。在这里，你不需要提供 XML 文件的完整路径，只需正确配置 CLASSPATH 环境变量即可，因为，容器会从 CLASSPATH 中搜索 bean 配置文件。
- WebXmlApplicationContext：该容器会在一个 web 应用程序的范围内加载在 XML 文件中已被定义的 bean。

来自 <https://www.cnblogs.com/yuxiang1/p/11157563.html>

IOC容器

2020年4月2日 14:28

IOC:控制反转，依赖注入
BeanFactory:
Spring Framework 由组成大约 20 个模块的 features 组成。这些模块分为 Core Container，Data Access/Integration， Web，AOP(Aspect Oriented Programming)，Instrumentation，Messaging 和 Test，如下图所示。



bean配置：@Configuration，@Bean，@Import和@DependsOn 注释
@Component
然后可以用@Component、@Controller、@Service、@Repository注解来标注需要由Spring IoC容器进行对象托管的类。这几个注解没有本质区别，只不过@Controller通常用于控制器，@Service通常用于业务逻辑类，@Repository通常用于仓储类（例如我们的DAO实现类），普通的类用@Component来标注。

来自 <https://www.cnblogs.com/yuxiang1/p/11157563.html>

@Required 注解

这个注解表明bean的属性必须在配置的时候设置，通过一个bean定义的显式的属性值或通过自动装配，若@Required注解的bean属性未被设置，容器将抛出

BeanInitializationException。

@Autowired 注解

@Autowired 注解提供了更细粒度的控制，包括在何处以及如何完成自动装配。它的用法和@Required一样，修饰setter方法、构造器、属性或者具有任意名称和/或多个参数的PN方法。

@Qualifier 注解

当有多个相同类型的bean却只有一个需要自动装配时，将@Qualifier 注解和@Autowire 注解结合使用以消除这种混淆，指定需要装配的确切的bean。

来自 <https://www.cnblogs.com/yuxiang1/p/11157563.html>

Spring中的bean的作用域有哪些

2020年4月7日 8:13

1.singleton：唯一bean实例，Spring中的bean默认都是单例的。
2.prototype：每次请求都会创建一个新的bean实例。
3.request：每一次HTTP请求都会产生一个新的bean，该bean仅在当前HTTP request内有效。
4.session：每一次HTTP请求都会产生一个新的bean，该bean仅在当前HTTP session内有效。

Spring 配置 Bean 实例化

2020年4月13日 13:56

Bean 注入属性有哪几种方式

Spring中的单例bean的线程安全问题了解吗?

2020年4月7日 8:17

大部分时候我们并没有在系统中使用多线程，所以很少有人会关注这个问题。单例bean存在线程问题，主要是因为当多个线程操作同一个对象的时候，对这个对象的非静态成员变量的写操作会存在线程安全问题。

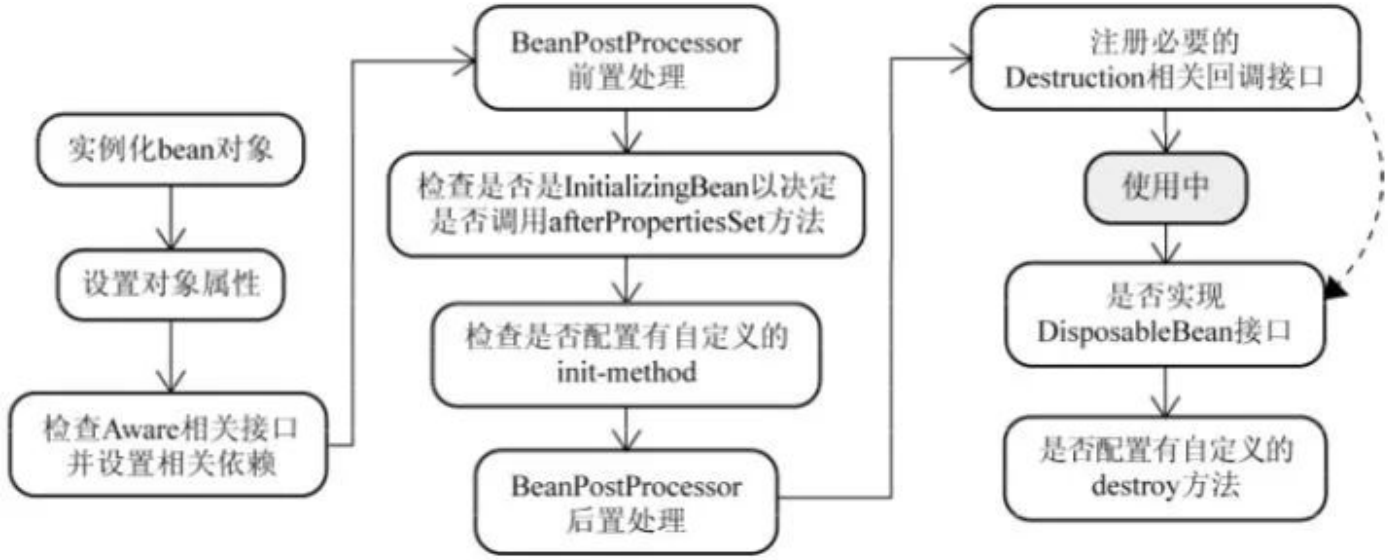
有两种常见的解决方案：

1.在bean对象中尽量避免定义可变的成员变量（不太现实）。

2.在类中定义一个ThreadLocal成员变量，将需要的可变成员变量保存在ThreadLocal中（推荐的一种方式）。

Spring中的bean生命周期

2020年4月7日 8:21



首先说一下Servlet的生命周期：实例化，初始init，接收请求service，销毁destroy；

Spring上下文中的Bean生命周期也类似，如下：

（1）实例化Bean：

对于BeanFactory容器，当客户向容器请求一个尚未初始化的bean时，或初始化bean的时候需要注入另一个尚未初始化的依赖时，容器就会调用createBean进行实例化。对于ApplicationContext容器，当容器启动结束后，通过获取BeanDefinition对象中的信息，实例化所有的bean。

（2）设置对象属性（依赖注入）：

实例化后的对象被封装在BeanWrapper对象中，紧接着，Spring根据BeanDefinition中的信息 以及 通过BeanWrapper提供的设置属性的接口完成依赖注入。

（3）处理Aware接口：

接着，Spring会检测该对象是否实现了xxxAware接口，并将相关的xxxAware实例注入给Bean：

①如果这个Bean已经实现了BeanNameAware接口，会调用它实现的setBeanName(String beanId)方法，此处传递的就是Spring配置文件中Bean的id值；

②如果这个Bean已经实现了BeanFactoryAware接口，会调用它实现的setBeanFactory()方法，传递的是Spring工厂自身。

③如果这个Bean已经实现了ApplicationContextAware接口，会调用setApplicationContext(ApplicationContext)方法，传入Spring上下文；

（4）BeanPostProcessor：

如果想对Bean进行一些自定义的处理，那么可以让Bean实现了BeanPostProcessor接口，那将会调用postProcessBeforeInitialization(Object obj, String s)方法。

（5）InitializingBean 与 init-method：

如果Bean在Spring配置文件中配置了 init-method 属性，则会自动调用其配置的初始化方法。

（6）如果这个Bean实现了BeanPostProcessor接口，将会调用postProcessAfterInitialization(Object obj, String s)方法；由于这个方法是在Bean初始化结束时调用的，所以可以被应用于内存或缓存技术；

以上几个步骤完成后，Bean就已经被正确创建了，之后就可以使用这个Bean了。

（7）DisposableBean：

当Bean不再需要时，会经过清理阶段，如果Bean实现了DisposableBean这个接口，会调用其实现的destroy()方法；

（8）destroy-method：

Spring框架中用到了哪些设计模式

2020年4月7日 8:28

1.工厂设计模式：Spring使用工厂模式通过BeanFactory和ApplicationContext创建bean对象。
2.代理设计模式：Spring AOP功能的实现。
3.单例设计模式：Spring中的bean默认都是单例的。
4.模板方法模式：Spring中的jdbcTemplate、hibernateTemplate等以Template结尾的对数据库操作的类，它们就使用到了模板模式。
5.包装器设计模式：我们的项目需要连接多个数据库，而且不同的客户在每次访问中根据需要会去访问不同的数据库。这种模式让我们可以根据客户的需求能够动态切换不同的数据源。
6.观察者模式：Spring事件驱动模型就是观察者模式很经典的一个应用。
7.适配器模式：Spring AOP的增强或通知（Advice）使用到了适配器模式、Spring MVC中也是用到了适配器模式适配Controller

来自 <https://www.cnblogs.com/yanggb/p/11004887.html>

@Component和@Bean的区别是什么

2020年4月7日 8:31

1.作用对象不同。@Component注解作用于类，而@Bean注解作用于方法。
2.@Component注解通常是通过类路径扫描来自动侦测以及自动装配到Spring容器中（我们可以使用@ComponentScan注解定义要扫描的路径）。@Bean注解通常是在标有该注解的方法中定义产生这个bean，告诉Spring这是某个类的实例，当我需要用它的时候还给我。
3.@Bean注解比@Component注解的自定义性更强，而且很多地方只能通过@Bean注解来注册bean。比如当引用第三方库的类需要装配到Spring容器的时候，就只能通过@Bean注解来实现。

将一个类声明为Spring的bean的注解有哪些?

2020年4月7日 8:33

我们一般使用@Autowired注解去自动装配bean。而想要把一个类标识为可以用@Autowired注解自动装配的bean，可以采用以下的注解实现：

1.@Component注解。通用的注解，可标注任意类为Spring组件。如果一个Bean不知道属于哪一个层，可以使用@Component注解标注。

2.@Repository注解。对应持久层，即Dao层，主要用于数据库相关操作。

3.@Service注解。对应服务层，即Service层，主要涉及一些复杂的逻辑，需要用到Dao层（注入）。

4.@Controller注解。对应Spring MVC的控制层，即Controller层，主要用于接受用户请求并调用Service层的方法返回数据给前端页面。

来自 <https://www.cnblogs.com/yanggb/p/11004887.html>

Spring IOC和AOP

2020年4月7日 8:07

AOP:
OOP面向对象，允许开发者定义纵向的关系，但并适用于定义横向的关系，导致了大量代码的重复，而不利于各个模块的重用。

AOP，一般称为面向切面，作为面向对象的一种补充，用于将那些与业务无关，但却对多个对象产生影响的公共行为和逻辑，抽取并封装为一个可重用的模块，这个模块被命名为“切面”（Aspect），减少系统中的重复代码，降低了模块间的耦合度，同时提高了系统的可维护性。可用于权限认证、日志、事务处理。

AOP实现的关键在于 代理模式，AOP代理主要分为静态代理和动态代理。静态代理的代表为AspectJ；动态代理则以Spring AOP为代表。

（1）AspectJ是静态代理的增强，所谓静态代理，就是AOP框架会在编译阶段生成AOP代理类，因此也称为编译时增强，他会在编译阶段将AspectJ(切面)织入到Java字节码中，运行的时候就是增强之后的AOP对象。

（2）Spring AOP使用的动态代理，所谓的动态代理就是说AOP框架不会去修改字节码，而是每次运行时在内存中临时为方法生成一个AOP对象，这个AOP对象包含了目标对象的全部方法，并且在特定的切点做了增强处理，并回调原对象的方法。
Spring AOP中的动态代理主要有两种方式，JDK动态代理和CGLIB动态代理：

①JDK动态代理只提供接口的代理，不支持类的代理。核心InvocationHandler接口和Proxy类，InvocationHandler 通过invoke()方法反射来调用目标类中的代码，动态地将横切逻辑和业务编织在一起；接着，Proxy利用 InvocationHandler动态创建一个符合某一接口的的实例，生成目标类的代理对象。

②如果代理类没有实现 InvocationHandler 接口，那么Spring AOP会选择使用CGLIB来动态代理目标类。CGLIB（Code Generation Library），是一个代码生成的类库，可以在运行时动态的生成指定类的一个子类对象，并覆盖其中特定方法并添加增强代码，从而实现AOP。CGLIB是通过继承的方式做的动态代理，因此如果某个类被标记为final，那么它是无法使用CGLIB做动态代理的。

（3）静态代理与动态代理区别在于生成AOP代理对象的时机不同，相对来说AspectJ的静态代理方式具有更好的性能，但是AspectJ需要特定的编译器进行处理，而Spring AOP则无需特定的编译器处理。

Spring的IoC理解：

（1）IOC就是控制反转，是指创建对象的控制权的转移，以前创建对象的主动权和时机是由自己把控的，而现在这种权力转移到Spring容器中，并由容器根据配置文件去创建实例和管理各个实例之间的依赖关系，对象与对象之间松散耦合，也利于功能的复用。DI依赖注入，和控制反转是同一个概念的不同角度的描述，即 应用程序在运行时依赖IoC容器来动态注入对象需要的外部资源。

（2）最直观的表达就是，IOC让对象的创建不用去new了，可以由spring自动生产，使用java的反射机制，根据配置文件在运行时动态的去创建对象以及管理对象，并调用对象的方法的。

（3）Spring的IOC有三种注入方式： 构造器注入、setter方法注入、根据注解注入。

7.Spring中的自动装配有哪些限制?

答:

- 如果使用了构造器注入或者setter注入，那么将覆盖自动装配的依赖关系。
- 基本数据类型的值、字符串字面量、类字面量无法使用自动装配来注入。
- 优先考虑使用显式的装配来进行更精确的依赖注入而不是使用自动装配。

来自 <https://www.cnblogs.com/yuxiang1/p/11157563.html>

你如何理解AOP中的连接点（Joinpoint）、切点（Pointcut）、增强（Advice）、引介（Introduction）、织入（Weaving）、切面（Aspect）这些概念?

答： a. 连接点（Joinpoint）：程序执行的某个特定位置（如：某个方法调用前、调用后，方法抛出异常后）。一个类或一段程序代码拥有一些具有边界性质的特定点，这些代码中的特定点就是连接点。Spring仅支持方法的连接点。 b. 切点（Pointcut）：如果连接点相当于数据中的记录，那么切点相当于查询条件，一个切点可以匹配多个连接点。Spring AOP的规则解析引擎负责解析切点所设定的查询条件，找到对应的连接点。 c. 增强（Advice）：增强是织入到目标类连接点上的一段程序代码。Spring提供的增强接口都是带方位名的，如：BeforeAdvice、AfterReturningAdvice、ThrowsAdvice等。很多资料上将增强译为“通知”，这明显是个词不达意的翻译，让很多程序员困惑了许久。

说明： Advice在国内的很多书面资料中都被翻译成”通知”，但是很显然这个翻译无法表达其本质，有少量的读物上将这个词翻译为”增强”，这个翻译是对Advice较为准确的诠释，我们通过AOP将横切关注功能加到原有的业务逻辑上，这就是对原有业务逻辑的一种增强，这种增强可以是前置增强、后置增强、返回后增强、抛异常时增强和包围型增强。 d. 引介（Introduction）：引介是一种特殊的增强，它为类添加一些属性和方法。这样，即使一个业务类原本没有实现某个接口，通过引介功能，可以动态的为该业务类添加接口的实现逻辑，让业务类成为这个接口的实现类。 e. 织入（Weaving）：织入是将增强添加到目标类具体连接点上的过程，AOP有三种织入方式：①编译期织入：需要特殊的Java编译期（例如AspectJ的ajc）；②装载期织入：要求使用特殊的类加载器，在装载类的时候对类进行增强；③运行时织入：在运行时

为目标类生成代理实现增强。Spring采用了动态代理的方式实现了运行时织入，而AspectJ采用了编译期织入和装载期织入的方式。 f. 切面（Aspect）：切面是由切点和增强（引介）组成的，它包括了对横切关注功能的定义，也包括了对连接点的定义。

来自 <https://www.cnblogs.com/yuxiang1/p/11157563.html>

哪种依赖注入方式你建议使用，构造器注入，还是 Setter方法注入?

来自 <https://www.cnblogs.com/yuxiang1/p/11157563.html>

Spring AOP和AspectJ AOP有什么区别

2020年4月7日 8:09

Spring AOP是属于运行时增强，而AspectJ是编译时增强。Spring AOP基于代理（Proxying），而AspectJ基于字节码操作（Bytecode Manipulation）。
Spring AOP已经集成了AspectJ，AspectJ应该算得上是Java生态系统中最完整的AOP框架了。AspectJ相比于Spring AOP功能更加强大，但是Spring AOP相对来说更简单。
如果我们的切面比较少，那么两者性能差异不大。但是，当切面太多的话，最好选择AspectJ，它比SpringAOP快很多。

Spring事务管理的方式有几种

2020年4月7日 8:34

1.编程式事务：在代码中硬编码（不推荐使用）。
2.声明式事务：在配置文件中配置（推荐使用），分为基于XML的声明式事务和基于注解的声明式事务。

来自 <https://www.cnblogs.com/yanggb/p/11004887.html>

38.你更倾向用那种事务管理类型?

大多数Spring框架的用户选择声明式事务管理，因为它对应用代码的影响最小，因此更符合一个无侵入的轻量级容器的思想。声明式事务管理要优于编程式事务管理，虽然比编程式事务管理（这种方式允许你通过代码控制事务）少了一点灵活性。

来自 <https://www.cnblogs.com/yuxiang1/p/11157563.html>

Spring事务中的隔离级别有哪几种

2020年4月7日 8:35

在TransactionDefinition接口中定义了五个表示隔离级别的常量：
ISOLATION_DEFAULT：使用后端数据库默认的隔离级别，Mysql默认采用的REPEATABLE_READ隔离级别；Oracle默认采用的READ_COMMITTED隔离级别。
ISOLATION_READ_UNCOMMITTED：最低的隔离级别，允许读取尚未提交的数据变更，可能会导致脏读、幻读或不可重复读。
ISOLATION_READ_COMMITTED：允许读取并发事务已经提交的数据，可以阻止脏读，但是幻读或不可重复读仍有可能发生
ISOLATION_REPEATABLE_READ：对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，可以阻止脏读和不可重复读，但幻读仍有可能发生。
ISOLATION_SERIALIZABLE：最高的隔离级别，完全服从ACID的隔离级别。所有的事务依次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，该级别可以防止脏读、不可重复读以及幻读。但是这将严重影响程序的性能。通常情况下也不会用到该级别。

来自 <https://www.cnblogs.com/yanggb/p/11004887.html>

Spring事务

2020年4月7日 8:36

Spring 是如何管理事务的?

spring的事务声明有两种方式，编程式和声明式。spring主要是通过“声明式事务”的方式对事务进行管理，即在配置文件中进行声明，通过AOP将事务切面切入程序，最大的好处是大大减少了代码量。

Spring事务中有哪几种事务传播行为

在TransactionDefinition接口中定义了八个表示事务传播行为的常量。
支持当前事务的情况：
PROPAGATION_REQUIRED：如果当前存在事务，则加入该事务；如果当前没有事务，则创建一个新的事务。
PROPAGATION_SUPPORTS：如果当前存在事务，则加入该事务；如果当前没有事务，则以非事务的方式继续运行。
PROPAGATION_MANDATORY：如果当前存在事务，则加入该事务；如果当前没有事务，则抛出异常。（mandatory：强制性）。
不支持当前事务的情况：
PROPAGATION_REQUIRES_NEW：创建一个新的事务，如果当前存在事务，则把当前事务挂起。
PROPAGATION_NOT_SUPPORTED：以非事务方式运行，如果当前存在事务，则把当前事务挂起。
PROPAGATION_NEVER：以非事务方式运行，如果当前存在事务，则抛出异常。
其他情况：
PROPAGATION_NESTED：如果当前存在事务，则创建一个事务作为当前事务的嵌套事务来运行；如果当前没有事务，则该取值等价于PROPAGATION_REQUIRED。

来自 <https://www.cnblogs.com/yanggb/p/11004887.html>

Spring 核心类

2020年4月13日 13:33

Spring使用

2020年4月13日 13:58

Spring 中如何更高效的使用 JDBC

在 Spring 中如何实现时间处理

哪种依赖注入方式你建议使用，构造器注入，还是 Setter 方法注入

在 **Spring 中如何注入一个 Java 集合**

你可以在 Spring 中注入一个 null 和一个空字符串吗?

1. 什么是基于 Java 的 Spring 注解配置? 给一些注解的例子
2. 你更倾向用那种事务管理类型?
3. Bean 的调用方式有哪些?
4. Spring MVC 里面拦截器是怎么写的结合springboot
5. 当一个方法向 AJAX 返回特殊对象，譬如 Object、List 等，需要做什么处理
6. 如何使用 Spring MVC 完成 JSON 操作
7. 开发中主要使用 Spring 的什么技术 ?
8. 介绍一下 Spring MVC 常用的一些注解
9. Spring 框架的事务管理有哪些优点
10. Spring 如何处理线程并发问题?
11. 核心容器（应用上下文）模块的理解?
12. 为什么说 Spring 是一个容器?
13. Spring 的优点?
14. Spring 框架中的单例 Beans 是线程安全的么?
15. Spring 框架中有哪些不同类型的事件?
16. 什么是 Spring 的内部 Bean?
17. 自动装配有哪些局限性 ?
18. 在 Spring AOP 中，关注点和横切关注的区别是什么?
19. 如何给 Spring 容器提供配置元数据?
20. 哪些是重要的 Bean 生命周期方法? 你能重载它们吗
21. 讲下 Spring MVC 的执行流程
22. Spring MVC 的控制器是不是单例模式,如果是,有什么问题,怎么解决?
23. Spring 中循环注入的方式?

请介绍一下设计模式在 Spring 框架中的使用

2020年4月13日 13:59

设计模式作为工作学习中的枕边书，却时常处于勤说不用的尴尬境地，也不是我们时常忘记，只是一直没有记忆。
今天，螃蟹就设计模式的内在价值做一番探讨，并以spring为例进行讲解，只有领略了其设计的思想理念，才能在工作学习中运用到“无形”。
Spring作为业界的经典框架，无论是在架构设计方面，还是在代码编写方面，都堪称行内典范。好了，话不多说，开始今天的内容。
spring中常用的设计模式达到九种，我们一一举例：
第一种：简单工厂
又叫做静态工厂方法（StaticFactory Method）模式，但不属于23种GOF设计模式之一。
简单工厂模式的实质是由一个工厂类根据传入的参数，动态决定应该创建哪一个产品类。
spring中的BeanFactory就是简单工厂模式的体现，根据传入一个唯一的标识来获得bean对象，但是否是在传入参数后创建还是传入参数前创建这个要根据具体情况来定。如下配置，就是在 HelloItxxz 类中创建一个 itxxzBean。

```
<beans>
 <bean id="singletonBean" class="com.itxxz.HelloItxxz">
 <constructor-arg>
 <value>Hello! 这是singletonBean!value>
 </constructor-arg>
 </ bean>

 <bean id="itxxzBean" class="com.itxxz.HelloItxxz"
 singleton="false">
 <constructor-arg>
 <value>Hello! 这是itxxzBean! value>
 </constructor-arg>
 </bean>

</beans>
```

第二种：工厂方法（Factory Method）
通常由应用程序直接使用new创建新的对象，为了将对象的创建和使用相分离，采用工厂模式,即应用程序将对象的创建及初始化职责交给工厂对象。
一般情况下,应用程序有自己的工厂对象来创建bean.如果将应用程序自己的工厂对象交给Spring管理,那么Spring管理的就不是普通的bean,而是工厂Bean。
螃蟹就以工厂方法中的静态方法为例讲解一下：

```
import java.util.Random;
```

```
public class StaticFactoryBean {
 public static Integer createRandom() {
 return new Integer(new Random().nextInt());
 }
}
```

建一个config.xm配置文件，将其纳入Spring容器来管理,需要通过factory-method指定静态方法名称

```
<bean id="random"
class="example.chapter3.StaticFactoryBean"
factory-method="createRandom" //createRandom方法必须是static的,才能找到
scope="prototype"
/>
```

测试:

```
public static void main(String[] args) {
 //调用getBean()时,返回随机数.如果没有指定factory-method,会返回StaticFactoryBean的实例,即返回工厂Bean的实例
 XmlBeanFactory factory = new XmlBeanFactory(new
ClassPathResource("config.xml"));
 System.out.println("我是IT学习者创建的实例:"+factory.getBean("random").toString());
}
```

第三种：单例模式（Singleton）

保证一个类仅有一个实例，并提供一个访问它的全局访问点。

spring中的单例模式完成了后半句话，即提供了全局的访问点BeanFactory。但没有从构造器级别去控制单例，这是因为spring管理的是是任意的java对象。

核心提示点：Spring下默认的bean均为singleton，可以通过singleton= “true|false” 或者scope= “？” 来指定

第四种：适配器（Adapter）

在Spring的Aop中，使用的Advice（通知）来增强被代理类的功能。Spring实现这一AOP功能的原理就使用代理模式

（1、JDK动态代理。2、CGLib字节码生成技术代理）。对类进行方法级别的切面增强，即，生成被代理类的代理类， 并在代理类的方法前，设置拦截器，通过执行拦截器重的内容增强了代理方法的功能，实现的面向切面编程。

Adapter类接口：Target

```
public interface AdvisorAdapter {

boolean supportsAdvice(Advice advice);
```

```
 MethodInterceptor getInterceptor(Advisor advisor);

}
```

MethodBeforeAdviceAdapter类，Adapter

```
class MethodBeforeAdviceAdapter implements AdvisorAdapter, Serializable {

 public boolean supportsAdvice(Advice advice) {
 return (advice instanceof MethodBeforeAdvice);
 }

 public MethodInterceptor getInterceptor(Advisor advisor) {
 MethodBeforeAdvice advice = (MethodBeforeAdvice) advisor.getAdvice();
 return new MethodBeforeAdviceInterceptor(advice);
 }

}
```

第五种：包装器（Decorator）

在我们的项目中遇到这样一个问题：我们的项目需要连接多个数据库，而且不同的客户在每次访问中根据需要会去访问不同的数据库。我们以往在spring和hibernate框架中总是配置一个数据源，因而sessionFactory的dataSource属性总是指向这个数据源并且恒定不变，所有DAO在使用sessionFactory的时候都是通过这个数据源访问数据库。但是现在，由于项目的需要，我们的DAO在访问sessionFactory的时候都不得不在多个数据源中不断切换，问题就出现了：如何让sessionFactory在执行数据持久化的时候，根据客户的需求能够动态切换不同的数据源？我们能不能在spring的框架下通过少量修改得到解决？是否有什么设计模式可以利用呢？

首先想到在spring的applicationContext中配置所有的dataSource。这些dataSource可能是各种不同类型的，比如不同的数据库：Oracle、SQL Server、MySQL等，也可能是不同的数据源：比如apache 提供的org.apache.commons.dbcp.BasicDataSource、spring提供的org.springframework.jndi.JndiObjectFactoryBean等。然后sessionFactory根据客户的每次请求，将dataSource属性设置成不同的数据源，以到达切换数据源的目的。

spring中用到的包装器模式在类名上有两种表现：一种是类名中含有Wrapper，另一种是类名中含有Decorator。

基本上都是动态地给一个对象添加一些额外的职责。

第六种：代理（Proxy）

为其他对象提供一种代理以控制对这个对象的访问。

从结构上来看和Decorator模式类似，但Proxy是控制，更像是一种对功能的限制，而Decorator是增加职责。

spring的Proxy模式在aop中有体现，比如JdkDynamicAopProxy和Cglib2AopProxy。

第七种：观察者（Observer）

定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。

spring中Observer模式常用的地方是listener的实现。如ApplicationListener。

第八种：策略（Strategy）

定义一系列的算法，把它们一个个封装起来，并且使它们可相互替换。本模式使得算法可独立于使用它的客户而变化。

spring中在实例化对象的时候用到Strategy模式

在SimpleInstantiationStrategy中有如下代码说明了策略模式的使用情况：

```
// Don't override the class with CGLIB if no overrides.
if (beanDefinition.getMethodOverrides().isEmpty()) {
    Constructor constructorToUse = (Constructor) beanDefinition.resolvedConstructorOrFactoryMethod;
    if (constructorToUse == null) {
        Class clazz = beanDefinition.getBeanClass();
        if (clazz.isInterface()) {
            throw new BeanInstantiationException(clazz, "Specified class is an interface");
        }
        try {
            constructorToUse = clazz.getDeclaredConstructor((Class[]) null);
            beanDefinition.resolvedConstructorOrFactoryMethod = constructorToUse;
        }
        catch (Exception ex) {
            throw new BeanInstantiationException(clazz, "No default constructor found", ex);
        }
    }
    return BeanUtils.instantiateClass(constructorToUse, null);
}
else {
    // Must generate CGLIB subclass.
    return instantiateWithMethodInjection(beanDefinition, beanName, owner)头条 @牛旦教育IT课堂
}
```

第九种：模板方法（Template Method）

定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。Template Method使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。

Template Method模式一般是需要继承的。这里想要探讨另一种对Template Method的理解。spring中的JdbcTemplate，在用这个类时并不想去继承这个类，因为这个类的方法太多，但是我们还是想用到JdbcTemplate已有的稳定的、公用的数据库连接，那么我们怎么办呢？我们可以把变化的东西抽出来作为一个参数传入JdbcTemplate的方法中。但是变化的东西是一段代码，而且这段代码会用到JdbcTemplate中的变量。怎么办？那我们就用回调对象吧。在这个回调对象中定义一个操纵JdbcTemplate中变量的方法，我们去实现这个方法，就把变化的东西集中到这里了。然后我们再传入这个回调对象到JdbcTemplate，从而完成了调用。这可能是Template Method不需要继承的另一种实现方式吧。

以下是一个具体的例子：

JdbcTemplate中的execute方法

```
public Object execute(ConnectionCallback action) throws DataAccessException {
    Assert.notNull(action, "Callback object must not be null");

    Connection con = DataSourceUtils.getConnection(getDataSource());
    try {
        Connection conToUse = con;
        if (this.nativeJdbcExtractor != null) {
            // Extract native JDBC Connection, castable to OracleConnection or the like.
            conToUse = this.nativeJdbcExtractor.getNativeConnection(con);
        }
        else {
            // Create close-suppressing Connection proxy, also preparing returned Statements.
            conToUse = createConnectionProxy(con);
        }
        return action.doInConnection(conToUse);
    }
    catch (SQLException ex) {
        // Release Connection early, to avoid potential connection pool deadlock
        // in the case when the exception translator hasn't been initialized yet.
        DataSourceUtils.releaseConnection(con, getDataSource());
        con = null;
        throw getExceptionTranslator().translate("ConnectionCallback", getSql(action), ex);
    }
    finally {
        DataSourceUtils.releaseConnection(con, getDataSource());
    }
}
```


JdbcTemplate执行execute方法

```
jdbcTemplate.execute(new ConnectionCallback() {
    public Object doInConnection(Connection con) throws SQLException, DataAccessException {
        // Do the insert
        PreparedStatement ps = null;
        try {
            ps = con.prepareStatement(getInsertString());
            setParameterValues(ps, values, null);
            ps.executeUpdate();
        } finally {
            JdbcUtils.closeStatement(ps);
        }
        //Get the key
        Statement keyStmt = null;
        ResultSet rs = null;
        HashMap keys = new HashMap(1);
        try {
            keyStmt = con.createStatement();
            rs = keyStmt.executeQuery(keyQuery);
            if (rs.next()) {
                long key = rs.getLong(1);
                keys.put(getGeneratedKeyNames()[0], key);
                keyHolder.getKeyList().add(keys);
            }
        } finally {
            JdbcUtils.closeResultSet(rs);
            JdbcUtils.closeStatement(keyStmt);
        }
        return null;
    }
});
```


以上为Spring中所应用到的9种设计模式，各位牛友，知道还有其它模式吗？请亮出来，让大家一起长进吧。

来自 <https://blog.csdn.net/u011277123/article/details/89001592>

Spring Boot

2020年4月13日 14:23

Spring Boot 自动配置的原理

Spring Boot 读取配置文件的方式

Ribbon 和 Feign 的区别

为什么要用 Spring Boot

- Spring Boot 的核心配置文件有哪几个？它们的区别是什么?

Spring Boot 的核心配置文件是 application 和 bootstrap 配置文件。

application 配置文件这个容易理解，主要用于 Spring Boot 项目的自动化配置。

bootstrap 配置文件有以下几个应用场景。

使用 Spring Cloud Config 配置中心时，

这时需要在 bootstrap 配置文件中添加连接到配置中心的配置属性来加载外部配置中心的配置信息；

一些固定的不能被覆盖的属性；

一些加密/解密的场景；

Spring Boot 的核心注解是哪个？它主要由哪几个注解组成的?

1.SpringBoot的核心注解是@SpringBootApplication由以下3个注解组成：

@SpringBootConfiguration：它组合了Configuration注解实现了 配置文件的功能。

@EnableAutoConfiguration：打开自动配置功能，也可以关闭某个指定的自动配置选项

如关闭数据源自动配置功能： @SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })。

@ComponentScan：Spring扫描组件。

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?

springboot不需要独立的容器就知可以运行，因为在springboot工程发布的道jar文件里已经包含了tomcat的jar文件。springboot运行的时专候，会创建tomcat对象，实现web服务功能。也可以将属springboot发布成war文件，放到tomcat里运行

- 运行 Spring Boot 有哪几种方式?

spring-boot的启动方式主要有三种:

1. 运行带有main方法类
2. 通过命令行 java -jar 的方式

3. 通过spring-boot-plugin的方式

- 你如何理解 Spring Boot 中的 Starters?

- 如何在 Spring Boot 启动的时候运行一些特定的代码?
  Spring Boot：项目启动时如何执行特定方法
  王晓(Java) 2018-11-06 13:14:42 8899 收藏 9
  展开

  在平时的开发中可能遇到这样的问题，在springboot 容器启动之后执行特定的方法或者类。

  Springboot给我们提供了两种“开机启动”某些方法的方式：ApplicationRunner和CommandLineRunner。这两种方法提供的目的是为了满足，在项目启动的时候立刻执行某些方法。他们都是在SpringApplication 执行之后开始执行的。

  这两个接口中有一个run方法，我们只需要实现这个方法即可。这两个接口的不同之处在于：ApplicationRunner中run方法的参数为ApplicationArguments，而CommandLineRunner接口中run方法的参数为String数组。下面通过两个简单的例子，来看一下这两个接口的实现。

  1.CommandLineRunner：

```
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class MyCommandLineRunner implements CommandLineRunner{
   @Override
   public void run(String... var1) throws Exception{
      System.out.println("This will be execute when the project was started!");
   }
}
```

  2.ApplicationRunner：

```
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Component;

@Component
public class MyApplicationRunner implements ApplicationRunner {
   @Override
   public void run(ApplicationArguments var1) throws Exception{
      System.out.println("MyApplicationRunner class will be execute when the project was started!");
   }
}
```

  这两种方式的实现都很简单，直接实现了相应的接口就可以了。记得在类上加@Component注

解。

如果想要指定启动方法执行的顺序，可以通过实现org.springframework.core.Ordered接口或者使用org.springframework.core.annotation.Order注解来实现。

1.Ordered接口：

```
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.core.Ordered;
import org.springframework.stereotype.Component;

@Component
public class MyApplicationRunner implements ApplicationRunner,Ordered{
   @Override
   public int getOrder(){
      return 1;//通过设置这里的数字来知道指定顺序
   }

   @Override
   public void run(ApplicationArguments var1) throws Exception{
      System.out.println("MyApplicationRunner1!");
   }
}
```

2.Order注解实现方式：

```
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.core.Ordered;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

@Component
@Order(value = 1)
public class MyApplicationRunner implements ApplicationRunner{

   @Override
   public void run(ApplicationArguments var1) throws Exception{
      System.out.println("MyApplicationRunner1!");
   }
}
```

————————————————————

- Spring Boot 有哪几种读取配置的方式?

  1.使用@Value注解

  使用@Value注解加载单个属性值

  如果在yaml或者properteis中存在配置：

- Spring Boot 实现热部署有哪几种方式?

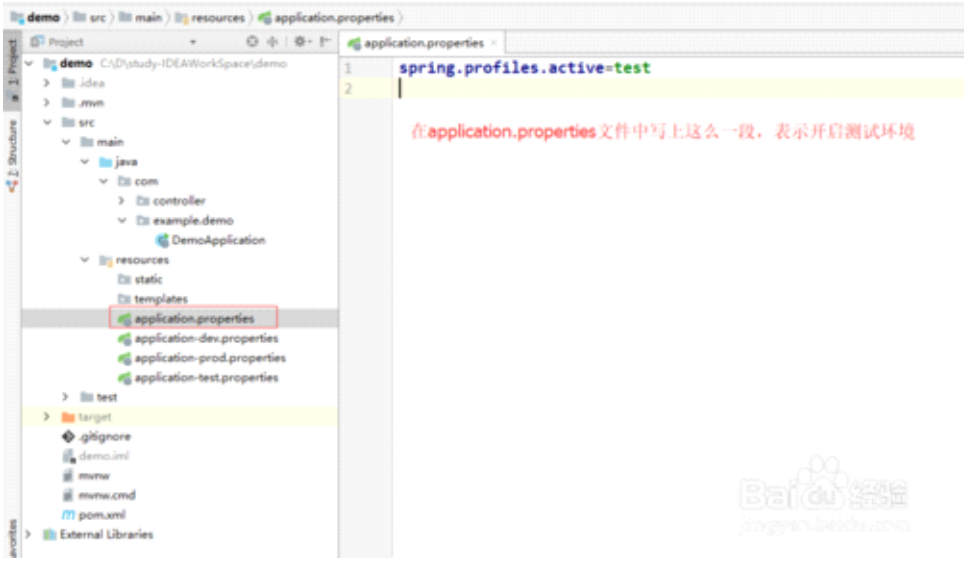
- Spring Boot 多套不同环境如何配置?
  众所周知再开发过程中，从**开发-测试-上线**，至少也得有3个环境，然而每个环境的配置都不一样，例如数据库配置、Redis配置、等各种配置。如果在打包环节来一个一个进行修改配置的话，非常容易出错。
  对于多环境配置，也有很多的构建工具，而他们的原理基本上也是通过配置多个不同环境的配置文件，进行区分打包。SpringBoot当然也支持。
  springboot 提供多环境配置的机制，让开发者**灵活**根据需求而**切换不同的配置环境。**
  如果不会创建SpringBootde 工程可以参考：以下链接
  35创建一个入门springboot项目（controller层）

工具/原料

- IDEA 全称IntelliJ IDEA
- SpringBoot

方法/步骤

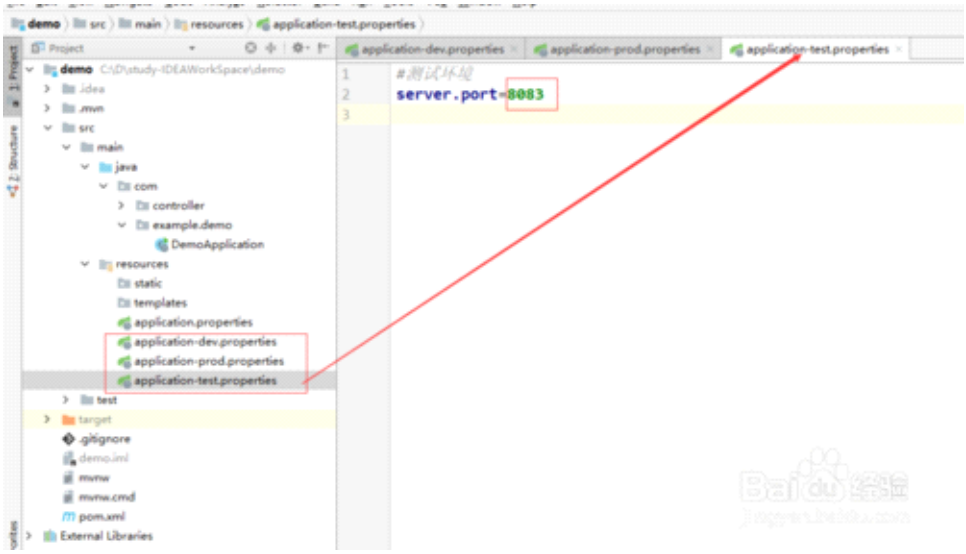
1. 1
   首先我们再SpringBoot的项目中
   resources文件夹下创建三个以properties为后缀的文件
   application-dev.properties：开发环境
   application-test.properties：测试环境
   application-prod.properties：生产环境
2. 2
   在application.properties文件中添加：
   spring.profiles.active=test
   （表示开启测试环境）



3. 3

   然后我们分别将：开发环境，生产环境，测试环境

   配置文件中的端口号改为：8081，8082，8083



4. 4

   然后我们运行项目：观察打印日志发现端口已经是：8083

   正是我们配置的测试环境端口

5. 5

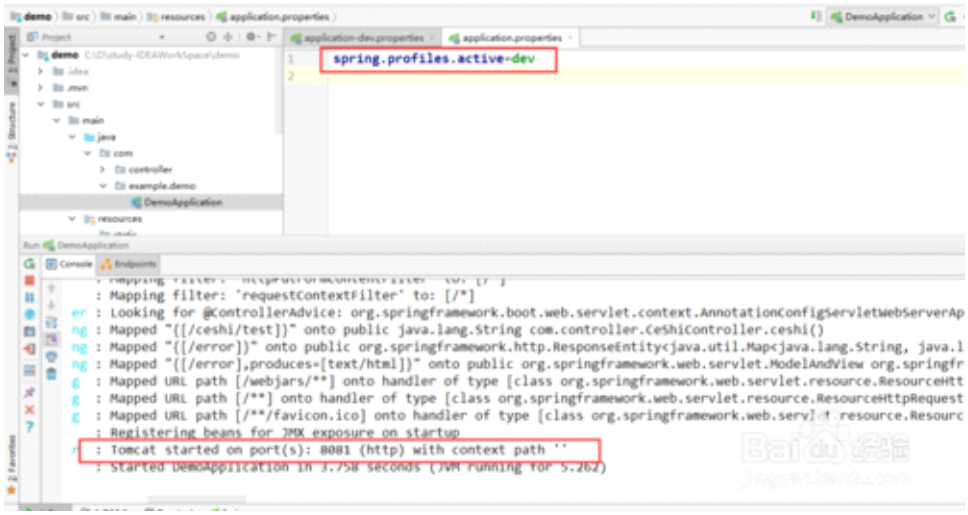
   同理我们将第2步的重复

   在application.properties文件中添加：

spring.profiles.active=dev

（表示开启开发环境）

然后运行，发现端口改变为开发环境端口



6. 6

   在Spring Boot中多环境配置文件名必须满足：

   application-{profile}.properties的固定格式，

   其中**{profile}**对应你的环境标识

   例如：

   application-**dev**.properties：开发环境

   application-**test**.properties：测试环境

   application-**prod**.properties：生产环境

   application.properyies通过spring.profiles.active来具体激活一个或者多个配置文件，如果没有指定任何profile的配置文件的话，spring boot默认会启动application-default.properties。

7. 7

   而哪个配置文件运行：

   spring.profiles.active=test

   就会加载application-test.properties配置文件内容

8. 8

   在此一定要注意：

   profile的配置文件可以按照application.properyies的放置位置一样，放于以下四个位置，

   1.当前目录的 “/config”的子目录下

   2.当前目录下

   3.classpath根目录的“/config”包下

   4.classpath的根目录下

   END

- Spring Boot 可以兼容老 Spring 项目吗，如何做?

Spring Cloud

2020年4月13日 14:24

- 什么是 Spring Cloud?
  Spring Cloud是一个微服务框架的规范
  介绍一下 Spring Cloud 常用的组件?
  Spring Cloud Netflix有哪些组件呢?

  eureka (提供服务注册与发现功能)
  ribbon（提供负载均衡功能）
  Feign（整合了ribbon和Hystrix，具有负载均衡和熔断限流等功能）
  Hystrix （提供了熔断限流，合并请求等功能）
  Zuul （提供了智能路由的功能）
  Hystrix Dashboard （提供了服务监控的功能，提供了数据监控和友好的图形化界面）
  Hystrix Turbine （Hystrix Turbine将每个服务Hystrix Dashboard数据进行了整合。也是监控系统的功能）
  spring cloud config （提供了统一配置的功能）
  Spring Cloud Bus （提供了配置实时更新的功能）
- Spring Cloud 如何实现服务注册的?
- 什么是负载均衡？有什么作用?
  负载均衡是建立在现有网络结构之上，它提供了一种廉价有效透明的方法扩展网络设备和服务器的带宽、增加吞吐量、加强网络数据处理能力、提高网络的灵活性和可用性。

- 什么是服务熔断?
  熔断机制是应对雪崩效应的一种微服务链路保护机制。当扇出链路的某个微服务不可用或者响应时间太长时，会进行服务降级，进而熔断该节点微服务的调用，快速返回“错误”的响应信息。当检测到该节点微服务调用响应正常后恢复调用链路。

  在SpringCloud框架里熔断机制通过Hystrix实现，Hystrix会监控微服务间调用的状况，当失败的调用到一定阈值，缺省是5秒内调用20次，如果失败，就会启动熔断机制。熔断机制的注解是@HystrixCommand。

  服务降级，一般是从整体负荷考虑。就是当某个服务熔断之后，服务器将不再被调用，此时客户端可以自己准备一个本地的fallback回调，返回一个缺省值。这样做，虽然水平下降，但好歹可用，比直接挂掉强。
  请介绍一下 Ribbon 的主要作用?

Spring模块

2020年5月6日 21:30

3. Spring由哪些模块组成?

以下是Spring 框架的基本模块：

- Core module
- Bean module
- Context module
- Expression Language module
- JDBC module
- ORM module
- OXM module
- Java Messaging Service(JMS) module
- Transaction module
- Web module
- Web-Servlet module
- Web-Struts module
- Web-Portlet module

来自 <https://www.cnblogs.com/wxd0108/p/5465517.html>

Spring Boot参数验证

2021年3月25日 16:50

JSR提供的校验注解:

- @Null 被注释的元素必须为 null
- @NotNull 被注释的元素必须不为 null
- @AssertTrue 被注释的元素必须为 true
- @AssertFalse 被注释的元素必须为 false
- @Min(value) 被注释的元素必须是一个数字，其值必须大于等于指定的最小值
- @Max(value) 被注释的元素必须是一个数字，其值必须小于等于指定的最大值
- @DecimalMin(value) 被注释的元素必须是一个数字，其值必须大于等于指定的最小值
- @DecimalMax(value) 被注释的元素必须是一个数字，其值必须小于等于指定的最大值
- @Size(max=, min=) 被注释的元素的大小必须在指定的范围内
- @Digits (integer, fraction) 被注释的元素必须是一个数字，其值必须在可接受的范围内
- @Past 被注释的元素必须是一个过去的日期
- @Future 被注释的元素必须是一个将来的日期
- @Pattern(regex=,flag=) 被注释的元素必须符合指定的正则表达式

Hibernate Validator提供的校验注解:

- @NotBlank(message =) 验证字符串非null，且长度必须大于0
- @Email 被注释的元素必须是电子邮箱地址
- @Length(min=,max=) 被注释的字符串的大小必须在指定的范围内
- @NotEmpty 被注释的字符串的必须非空
- @Range(min=,max=,message=) 被注释的元素必须在合适的范围内

来自 <https://github.com/Snailclimb/springboot-guide/blob/master/docs/advanced/spring-bean-validation.md>

Mybatis是什么

2020年4月13日 11:31

MyBatis

1、什么是 MyBatis?

答：MyBatis 是一个可以自定义 SQL、存储过程和高级映射的持久层框架。

2、讲下 MyBatis 的缓存

答：MyBatis 的缓存分为一级缓存和二级缓存,一级缓存放在 session 里面,默认就有,二级缓存放在它的命名空间里,默认是不打开的,使用二级缓存属性类需要实现 Serializable 序列化接口(可用来保存对象的状态),可在它的映射文件中配置<cache/>

3、Mybatis 是如何进行分页的？分页插件的原理是什么?

答：

1）Mybatis 使用 RowBounds 对象进行分页，也可以直接编写 sql 实现分页，也可以使用 Mybatis 的分页插件。

2）分页插件的原理：实现 Mybatis 提供的接口，实现自定义插件，在插件的拦截方法内拦截待执行的 sql，然后重写 sql。

举例：select from student，拦截 sql 后重写为：select t. from （select from student）t limit 0，10

4、简述 Mybatis 的插件运行原理，以及如何编写一个插件?

答：

1）Mybatis 仅可以编写针对
ParameterHandler、ResultSetHandler、StatementHandler、
Executor 这 4 种接口的插件，Mybatis 通过动态代理，为需要拦截的接口生成代理对象以实现接口方法拦截功能，每当执行这 4 种接口对象的方法时，就会进入拦截方法，具体就是 InvocationHandler 的 invoke()方法，当然，只会拦截那些你指定需要拦截的方法。

2）实现 Mybatis 的 Interceptor 接口并复写 intercept()方法，然后在给插件编写注解，指定
要拦截哪一个接口的哪些方法即可，记住，别忘了在配置文件中配置你编写的插件。

5、Mybatis 动态 sql 是做什么的？都有哪些动态 sql？能简述一下动态 sql 的执行原理不?

答：

1）Mybatis 动态 sql 可以让我们在 Xml 映射文件内，以标签的形式编写动态 sql，完成逻辑判断和动态拼接 sql 的功能。

2）Mybatis 提供了 9 种动态 sql 标签：
trim|where|set|foreach|if|choose|when|otherwise|bind。

3）其执行原理为，使用 OGNL 从 sql 参数对象中计算表达式的值，根据表达式的值动态拼接 sql，以此来完成动态 sql 的功能。

6、#{}和${}的区别是什么?

答：

1）#{}是预编译处理，${}是字符串替换。
2）Mybatis 在处理#{}时，会将 sql 中的#{}替换为?号，调用 PreparedStatement 的 set 方法
来赋值；
3）Mybatis 在处理${}时，就是把${}替换成变量的值。
4）使用#{}可以有效的防止 SQL 注入，提高系统安全性。

7、为什么说 Mybatis 是半自动 ORM 映射工具？它与全自动的区别在哪里？

答：Hibernate 属于全自动 ORM 映射工具，使用 Hibernate 查询关联对象或者关联集合对象
时，可以根据对象关系模型直接获取，所以它是全自动的。而 Mybatis 在查询关联对象或关联集合对象时，需要手动编写 sql 来完成，所以，称之为半自动 ORM 映射工具。

8、Mybatis 是否支持延迟加载？如果支持，它的实现原理是什么？

答：
1）Mybatis 仅支持 association 关联对象和 collection 关联集合对象的延迟加载，association
指的就是一对一，collection 指的就是一对多查询。在 Mybatis 配置文件中，可以配置是否启用延迟加载 lazyLoadingEnabled=true|false。
2）它的原理是，使用 CGLIB 创建目标对象的代理对象，当调用目标方法时，进入拦截器方法，比如调用 a.getB().getName()，拦截器 invoke()方法发现 a.getB()是 null 值，那么就会单
独发送事先保存好的查询关联 B 对象的 sql，把 B 查询上来，然后调用 a.setB(b)，于是 a 的对象 b 属性就有值了，接着完成 a.getB().getName()方法的调用。这就是延迟加载的基本原理。

9、MyBatis 与 Hibernate 有哪些不同？

答：
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。
总之，按照用户的需求在有限的资源环境下只要能做出维护性、扩展性良好的软件架构都

是好架构，所以框架只有适合才是最好。

10、MyBatis 的好处是什么?

答：

1）MyBatis 把 sql 语句从 Java 源程序中独立出来，放在单独的 XML 文件中编写，给程序的维护带来了很大便利。

2）MyBatis 封装了底层 JDBC API 的调用细节，并能自动将结果集转换成 Java Bean 对象，大大简化了 Java 数据库编程的重复工作。

3）因为 MyBatis 需要程序员自己去编写 sql 语句，程序员可以结合数据库自身的特点灵活控制 sql 语句，因此能够实现比 Hibernate 等全自动 orm 框架更高的查询效率，能够完成复杂查询。

11、简述 Mybatis 的 Xml 映射文件和 Mybatis 内部数据结构之间的映射关系?

答：Mybatis 将所有 Xml 配置信息都封装到 All-In-One 重量级对象 Configuration 内部。在

Xml 映射文件中，<parameterMap>标签会被解析为 ParameterMap 对象，其每个子元素会

被解析为 ParameterMapping 对象。<resultMap>标签会被解析为 ResultMap 对象，其每个子

元素会被解析为 ResultMapping 对象。每一个

<select>、<insert>、<update>、<delete>标签

均会被解析为 MappedStatement 对象，标签内的 sql 会被解析为 BoundSql 对象。

12、什么是 MyBatis 的接口绑定,有什么好处?

答：接口映射就是在 MyBatis 中任意定义接口,然后把接口里面的方法和 SQL 语句绑定,我们直接调用接口方法就可以,这样比起原来了 SqlSession 提供的方法我们可以有更加灵活的选择和设置.

13、接口绑定有几种实现方式,分别是怎么实现的?

答：接口绑定有两种实现方式,一种是通过注解绑定,就是在接口的方法上面加上 @Select@Update 等注解里面包含 Sql 语句来绑定,另外一种就是通过 xml 里面写 SQL 来绑定,在这种情况下,要指定 xml 映射文件里面的 namespace 必须为接口的全路径名.

14、什么情况下用注解绑定,什么情况下用 xml 绑定?

答：当 Sql 语句比较简单时候,用注解绑定；当 SQL 语句比较复杂时候,用 xml 绑定,一般用 xml 绑定的比较多

15、MyBatis 实现一对一有几种方式?具体怎么操作的?

答：有联合查询和嵌套查询,联合查询是几个表联合查询,只查询一次,通过在 resultMap 里面配置 association 节点配置一对一的类就可以完成;嵌套查询是先查一个表,根据这个表里面的结果的外键 id,去再另外一个表里面查询数据,也是通过 association 配置,但另外一个表的查询通过 select 属性配置。*

来自 <https://blog.51cto.com/14442094/2423256>

2021年3月31日 11:17

面试了一些人，简历上都说自己熟悉 Spring Boot, 或者说正在学习 Spring Boot，一问他们时，都只停留在简单的使用阶段，很多东西都不清楚，也让我对面试者大失所望。下面，我给大家总结下有哪些 Spring Boot 的面试题，这是我经常拿来问面试者的，希望对你有帮助。

1、什么是 Spring Boot?

Spring Boot 是 Spring 开源组织下的子项目，是 Spring 组件一站式解决方案，主要是简化了使用 Spring 的难度，简省了繁重的配置，提供了各种启动器，开发者能快速上手。

更多 Spring Boot 详细介绍请看这篇文章《什么是Spring Boot?》。

2、为什么要用 Spring Boot?

Spring Boot 优点非常多，如：

- 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 集这么多优点于一身，还有理由不使用它呢?

3、Spring Boot 的核心配置文件有哪几个？它们的区别是什么?

Spring Boot 的核心配置文件是 application 和 bootstrap 配置文件。

application 配置文件这个容易理解，主要用于 Spring Boot 项目的自动化配置。

bootstrap 配置文件有以下几个应用场景。

- 使用 Spring Cloud Config 配置中心时，这时需要在 bootstrap 配置文件中添加连接到配置中心的配置属性来加载外部配置中心的配置信息；
- 一些固定的不能被覆盖的属性；
- 一些加密/解密的场景；

具体请看这篇文章《Spring Boot 核心配置文件详解》。

4、Spring Boot 的配置文件有哪几种格式？它们有什么区别?

.properties 和 .yml，它们的区别主要是书写格式不同。

1).properties

```
app.user.name = javastack
```

2).yml

```
app:
  user:
    name: javastack
```

另外，.yml 格式不支持 @PropertySource 注解导入配置。

5、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。

6、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

具体请参考这篇文章《Spring Boot开启的2种方式》。

7、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。

8、运行 Spring Boot 有哪几种方式?

1）打包用命令或者放到容器中运行

2）用 Maven/ Gradle 插件运行

3）直接执行 main 方法运行

9、Spring Boot 自动配置原理是什么?

注解 @EnableAutoConfiguration, @Configuration, @ConditionalOnClass 就是自动配置的核心，首先它得是一个配置文件，其次根据类路径下是否有这个类去自动配置。

具体看这篇文章《Spring Boot自动配置原理、实战》。

10、Spring Boot 的目录结构是怎样的?

```
cn
 +- javastack
    +- MyApplication.java
    |
    +- customer
    |   +- Customer.java
    |   +- CustomerController.java
    |   +- CustomerService.java
    |   +- CustomerRepository.java
    |
    +- order
        +- Order.java
        +- OrderController.java
        +- OrderService.java
        +- OrderRepository.java
```

这个目录结构是主流及推荐的做法，而在主入口类上加上 @SpringBootApplication 注解来开启 Spring Boot 的各项能力，如自动配置、组件扫描等。具体看这篇文章

《Spring Boot 主类及目录结构介绍》。

11、你如何理解 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。具体请看这篇文章《Spring Boot Starters启动器》。

12、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，具体请看这篇文章《Spring Boot Runner启动器》。

13、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，具体请看这篇文章《Spring Boot读取配置的几种方式》。

14、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，具体请看这篇文章《Spring Boot日志集成》。

15、SpringBoot 实现热部署有哪几种方式?

主要有两种方式：

- Spring Loaded
- Spring-boot-devtools

Spring-boot-devtools 使用方式可以参考这篇文章《Spring Boot实现热部署》。

16、你如何理解 Spring Boot 配置加载顺序?

在 Spring Boot 里面，可以使用以下几种方式来加载配置。

1）properties文件；

2）YAML文件；

3）系统环境变量；

4）命令行参数；

等等……

具体请看这篇文章《Spring Boot 配置加载顺序详解》。

17、Spring Boot 如何定义多套不同环境配置?

提供多套配置文件，如：

applcation.properties

application-dev.properties

application-test.properties

application-prod.properties

运行时指定具体的配置文件，具体请看这篇文章《Spring Boot Profile 不同环境配置》。

18、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。

19、保护 Spring Boot 应用有哪些方法?

- 在生产中使用HTTPS
- 使用Snyk检查你的依赖关系
- 升级到最新版本
- 启用CSRF保护
- 使用内容安全策略防止XSS攻击
- ...

更多请看这篇文章《10 种保护 Spring Boot 应用的绝佳方法》。

20、Spring Boot 2.X 有什么新特性？与 1.X 有什么区别?

- 配置变更
- JDK 版本升级
- 第三方类库升级
- 响应式 Spring 编程支持
- HTTP/2 支持
- 配置属性绑定
- 更多改进与加强...

来自 <https://segmentfault.com/a/1190000016686735>

2021年3月31日 11:40

一 Redis 持久化机制

1. Redis是一个支持持久化的内存数据库，通过持久化机制把内存中的数据同步到硬盘文件来保证数据持久化。当Redis重启后通过把硬盘文件重新加载到内存，就能达到恢复数据的目的。
2. 实现：单独创建fork()一个子进程，将当前父进程的数据库数据复制到子进程的内存中，然后由子进程写入到临时文件中，持久化的过程结束了，再用这个临时文件替换上次的快照文件，然后子进程退出，内存释放。
3. RDB是Redis默认的持久化方式。按照一定的时间周期策略把内存的数据以快照的形式保存到硬盘的二进制文件。即Snapshot快照存储，对应产生的数据文件为dump.rdb，通过配置文件中的save参数来定义快照的周期。（快照可以是其所表示的数据的一个副本，也可以是数据的一个复制品。）
4. AOF：Redis会将每一个收到的写命令都通过Write函数追加到文件最后，类似于MySQL的binlog。当Redis重启是会通过重新执行文件中保存的写命令来在内存中重建整个数据库的内容。
5. 当两种方式同时开启时，数据恢复Redis会优先选择AOF恢复。

二 缓存雪崩、缓存穿透、缓存预热、缓存更新、缓存降级等问题

1. 一、缓存雪崩
2. 缓存雪崩我们可以简单的理解为：由于原有缓存失效，新缓存未到期间
3. (例如：我们设置缓存时采用了相同的过期时间，在同一时刻出现大面积的缓存过期)，所有原本应该访问缓存的请求都去查询数据库了，而对数据库CPU和内存造成巨大压力，严重的会造成数据库宕机。从而形成一系列连锁反应，造成整个系统崩溃。
4. 解决办法：
5. 大多数系统设计者考虑用加锁（最多的解决方案）或者队列的方式保证来保证不会有大量的线程对数据库一次性进行读写，从而避免失效时大量的并发请求落到底层存储系统上。还有一个简单方案就时讲缓存失效时间分散开。
6. 二、缓存穿透
7. 缓存穿透是指用户查询数据，在数据库没有，自然在缓存中也不会有。这样就导致用户查询的时候，在缓存中找不到，每次都要去数据库再查询一遍，然后返回空（相当于进行了两次无用的查询）。这样请求就绕过缓存直接查数据库，这也是经常提的缓存命中率问题。

8. 解决办法;
9. 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。
10. 另外也有一个更为简单粗暴的方法，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。通过这个直接设置的默认值存放到缓存，这样第二次到缓冲中获取就有值了，而不会继续访问数据库，这种办法最简单粗暴。
11. 5TB的硬盘上放满了数据，请写一个算法将这些数据进行排重。如果这些数据是一些32bit大小的数据该如何解决？如果是64bit的呢?
12. 对于空间的利用到达了一种极致，那就是Bitmap和布隆过滤器(Bloom Filter)。
13. Bitmap： 典型的就是哈希表
14. 缺点是，Bitmap对于每个元素只能记录1bit信息，如果还想完成额外的功能，恐怕只能靠牺牲更多的空间、时间来完成了。
15. 布隆过滤器（推荐）
16. 就是引入了k(k>1)k(k>1)个相互独立的哈希函数，保证在给定的空间、误判率下，完成元素判重的过程。
17. 它的优点是空间效率和查询时间都远远超过一般的算法，缺点是有一定的误识别率和删除困难。
18. Bloom-Filter算法的核心思想就是利用多个不同的Hash函数来解决“冲突”。
19. Hash存在一个冲突（碰撞）的问题，用同一个Hash得到的两个URL的值有可能相同。为了减少冲突，我们可以多引入几个Hash，如果通过其中的一个Hash值我们得出某元素不在集合中，那么该元素肯定不在集合中。只有在所有的Hash函数告诉我们该元素在集合中时，才能确定该元素存在于集合中。这便是Bloom-Filter的基本思想。
20. Bloom-Filter一般用于在大数据量的集合中判定某元素是否存在。
21. 受提醒补充：缓存穿透与缓存击穿的区别
22. 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。
23. 解决方案;在访问key之前，采用SETNX（set if not exists）来设置另一个短期key来锁住当前key的访问，访问结束再删除该短期key。
24. 增：给一个我公司处理的案例：背景双机拿token，token在存一份到redis，保证系统在token过期时都只有一个线程去获取token;线上环境有两台机器，故使用分布式锁实现。
25. 三、缓存预热
26. 缓存预热这个应该是一个比较常见的概念，相信很多小伙伴都应该可以很容易的理解，缓存预热就是系统上线后，将相关的缓存数据直接加载到缓存系统。这样就可以避免在用户请求的时候，先查询数据库，然后再将数据缓存的问题！用户直接查询事先被预热的缓存数据!
27. 解决思路：
28. 1、直接写个缓存刷新页面，上线时手工操作下；

29. 2、数据量不大，可以在项目启动的时候自动进行加载；
30. 3、定时刷新缓存；
31. 四、缓存更新
32. 除了缓存服务器自带的缓存失效策略之外（Redis默认的有6中策略可供选择），我们还可以根据具体的业务需求进行自定义的缓存淘汰，常见的策略有两种：
33. （1）定时去清理过期的缓存；
34. （2）当有用户请求过来时，再判断这个请求所用到的缓存是否过期，过期的话就去底层系统得到新数据并更新缓存。
35. 两者各有优劣，第一种的缺点是维护大量缓存的key是比较麻烦的，第二种的缺点就是每次用户请求过来都要判断缓存失效，逻辑相对比较复杂！具体用哪种方案，大家可以根据自己的应用场景来权衡。
36. 五、缓存降级
37. 当访问量剧增、服务出现问题（如响应时间慢或不响应）或非核心服务影响到核心流程的性能时，仍然需要保证服务还是可用的，即使是有损服务。系统可以根据一些关键数据进行自动降级，也可以配置开关实现人工降级。
38. 降级的最终目的是保证核心服务可用，即使是有损的。而且有些服务是无法降级的（如加入购物车、结算）。
39. 以参考日志级别设置预案：
40. （1）一般：比如有些服务偶尔因为网络抖动或者服务正在上线而超时，可以自动降级；
41. （2）警告：有些服务在一段时间内成功率有波动（如在95~100%之间），可以自动降级或人工降级，并发送告警；
42. （3）错误：比如可用率低于90%，或者数据库连接池被打爆了，或者访问量突然猛增到系统能承受的最大阀值，此时可以根据情况自动降级或者人工降级；
43. （4）严重错误：比如因为特殊原因数据错误了，此时需要紧急人工降级。
44. 服务降级的目的，是为了防止Redis服务故障，导致数据库跟着一起发生雪崩问题。因此，对于不重要的缓存数据，可以采取服务降级策略，例如一个比较常见的做法就是，Redis出现问题，不去数据库查询，而是直接返回默认值给用户。

三 热点数据和冷数据是什么

1. 热点数据，缓存才有价值
2. 对于冷数据而言，大部分数据可能还没有再次访问到就已经被挤出内存，不仅占用内存，而且价值不大。频繁修改的数据，看情况考虑使用缓存
3. 对于上面两个例子，寿星列表、导航信息都存在一个特点，就是信息修改频率不高，读取通常非常高的场景。
4. 对于热点数据，比如我们的某IM产品，生日祝福模块，当天的寿星列表，缓存以后可能读取数十万次。再举个例子，某导航产品，我们将导航信息，缓存以后可能读取数百万次。
5. **数据更新前至少读取两次，**缓存才有意义。这个是最基本的策略，如果缓存还没有起作用就失效了，那就没有太大价值了。

6. 那存不存在，修改频率很高，但是又不得不考虑缓存的场景呢？有！比如，这个读取接口对数据库的压力很大，但是又是热点数据，这个时候就需要考虑通过缓存手段，减少数据库的压力，比如我们的某助手产品的，点赞数，收藏数，分享数等是非常典型的热点数据，但是又不断变化，此时就需要将数据同步保存到Redis缓存，减少数据库压力。

四 Memcache与Redis的区别都有哪些?

1. 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，redis可以持久化其数据
2. 2)、数据支持类型 memcached所有的值均是简单的字符串，redis作为其替代者，支持更为丰富的数据类型 ，提供list，set，zset，hash等数据结构的存储
3. 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。
4. 4). value 值大小不同：Redis 最大可以达到 512M；memcache 只有 1mb。
5. 5）redis的速度比memcached快很多
6. 6）Redis支持数据的备份，即master-slave模式的数据备份。

五 单线程的redis为什么这么快

1. (一)纯内存操作
2. (二)单线程操作，避免了频繁的上下文切换
3. (三)采用了非阻塞I/O多路复用机制

六 redis的数据类型，以及每种数据类型的使用场景

1. 常见五种
2. (一)String
3. 这个其实没啥好说的，最常规的set/get操作，value可以是String也可以是数字。一般做一些复杂的计数功能的缓存。
4. (二)hash
5. 这里value存放的是结构化的对象，比较方便的就是操作其中的某个字段。博主在做单点登录的时候，就是用这种数据结构存储用户信息，以cookieId作为key，设置30分钟为缓存过期时间，能很好的模拟出类似session的效果。
6. (三)list
7. 使用List的数据结构，可以做简单的消息队列的功能。另外还有一个就是，可以利用lrange命令，做基于redis的分页功能，性能极佳，用户体验好。本人还用一个场景，很合适—取行情信息。就也是个生产者和消费者的场景。LIST可以很好的完成排队，先进

先出的原则。

8. (四)set
9. 因为set堆放的是一堆不重复值的集合。所以可以做全局去重的功能。为什么不用JVM自带的Set进行去重？因为我们的系统一般都是集群部署，使用JVM自带的Set，比较麻烦，难道为了一个做一个全局去重，再起一个公共服务，太麻烦了。
10. 另外，就是利用交集、并集、差集等操作，可以计算共同喜好，全部的喜好，自己独有的喜好等功能。
11. (五)sorted set
12. sorted set多了一个权重参数score,集合中的元素能够按score进行排列。可以做排行榜应用，取TOP N操作。

七 Redis 内部结构

1. 1. dict 本质上是为了解决算法中的查找问题（Searching）是一个用于维护key和value映射关系的数据结构，与很多语言中的Map或dictionary类似。 本质上是为了解决算法中的查找问题（Searching）
2. 2. sds sds就等同于char * 它可以存储任意二进制数据，不能像C语言字符串那样以字符’\0’来标识字符串的结 束，因此它必然有个长度字段。
3. 3. skiplist （跳跃表） 跳表是一种实现起来很简单，单层多指针的链表，它查找效率很高，堪比优化过的二叉平衡树，且比平衡树的实现，
4. 4. quicklist
5. 5. ziplist 压缩表 ziplist是一个编码后的列表，是由一系列特殊编码的连续内存块组成的顺序型数据结构

八 redis的过期策略以及内存淘汰机制

1. redis采用的是定期删除+惰性删除策略。
2. 为什么不用定时删除策略?
3. 定时删除,用一个定时器来负责监视key,过期则自动删除。虽然内存及时释放，但是十分消耗CPU资源。在大并发请求下，CPU要将时间应用在处理请求，而不是删除key,因此没有采用这一策略.
4. 定期删除+惰性删除是如何工作的呢?
5. 定期删除，redis默认每个100ms检查，是否有过期的key,有过期key则删除。需要说明的是，redis不是每个100ms将所有的key检查一次，而是随机抽取进行检查(如果每隔100ms,全部key进行检查，redis岂不是卡死)。因此，如果只采用定期删除策略，会导致很多key到时间没有删除。
6. 于是，惰性删除派上用场。也就是说在你获取某个key的时候，redis会检查一下，这个key如果设置了过期时间那么是否过期了？如果过期了此时就会删除。
7. 采用定期删除+惰性删除就没其他问题了么?
8. 不是的，如果定期删除没删除key。然后你也没即时去请求key，也就是说惰性删除也没生效。这样，redis的内存会越来越高。那么就应该采用内存淘汰机制。

9. 在redis.conf中有一行配置 maxmemory-policy volatile-lru
10. 该配置就是配内存淘汰策略的(什么，你没配过？好好反省一下自己)
11. 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
12. 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
13. 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
14. allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰
15. allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰
16. no-enviction（驱逐）：禁止驱逐数据，新写入操作会报错
17. ps：如果没有设置 expire 的key, 不满足先决条件(prerequisites); 那么 volatile-lru, volatile-random 和 volatile-ttl 策略的行为, 和 noeviction(不删除) 基本上一致。

九 Redis 为什么是单线程的

1. 官方FAQ表示，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利用队列技术将并发访问变为串行访问
2. 1）绝大部分请求是纯粹的内存操作（非常快速）2）采用单线程,避免了不必要的上下文切换和竞争条件
3. 3）非阻塞IO优点：
4. 1.速度快，因为数据存在内存中，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)
5. 2. 支持丰富数据类型，支持string，list，set，sorted set，hash
6. 3.支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行
7. 4. 丰富的特性：可用于缓存，消息，按key设置过期时间，过期后将会自动删除如何解决redis的并发竞争key问题
8. 同时有多个子系统去set一个key。这个时候要注意什么呢？ 不推荐使用redis的事务机制。因为我们的生产环境，基本都是redis集群环境，做了数据分片操作。你一个事务中有涉及到多个key操作的时候，这多个key不一定都存储在同一个redis-server上。因此，redis的事务机制，十分鸡肋。
9. (1)如果对这个key操作，不要求顺序： 准备一个分布式锁，大家去抢锁，抢到锁就做set操作即可
10. (2)如果对这个key操作，要求顺序： 分布式锁+时间戳。 假设这会系统B先抢到锁，将key1设置为{valueB 3:05}。接下来系统A抢到锁，发现自己的valueA的时间戳早于缓存中的时间戳，那就不做set操作了。以此类推。
11. (3) 利用队列，将set方法变成串行访问也可以redis遇到高并发，如果保证读写key的一致性

12. 对redis的操作都是具有原子性的,是线程安全的操作,你不用考虑并发问题,redis内部已经帮你处理好并发的问题了。

十 Redis 集群方案应该怎么做？都有哪些方案?

1. 1.twemproxy，大概概念是，它类似于一个代理方式， 使用时在本需要连接 redis 的地方改为连接 twemproxy， 它会以一个代理的身份接收请求并使用一致性 hash 算法，将请求转接到具体 redis，将结果再返回 twemproxy。
2. 缺点： twemproxy 自身单端口实例的压力，使用一致性 hash 后，对 redis 节点数量改变时候的计算值的改变，数据无法自动移动到新的节点。
3. 2.codis，目前用的最多的集群方案，基本和 twemproxy 一致的效果，但它支持在 节点数量改变情况下，旧节点数据可恢复到新 hash 节点
4. 3.redis cluster3.0 自带的集群，特点在于他的分布式算法不是一致性 hash，而是 hash 槽的概念，以及自身支持节点设置从节点。具体看官方文档介绍。

十一 有没有尝试进行多机redis 的部署？如何保证数据一致的?

1. 主从复制，读写分离
2. 一类是主数据库（master）一类是从数据库（slave），主数据库可以进行读写操作，当发生写操作的时候自动将数据同步到从数据库，而从数据库一般是只读的，并接收主数据库同步过来的数据，一个主数据库可以有多个从数据库，而一个从数据库只能有一个主数据库

十二 对于大量的请求怎么样处理

1. redis是一个单线程程序，也就说同一时刻它只能处理一个客户端请求；
2. redis是通过IO多路复用（select，epoll, kqueue，依据不同的平台，采取不同的实现）来处理多个客户端请求的

十三 Redis 常见性能问题和解决方案?

1. (1) Master 最好不要做任何持久化工作，如 RDB 内存快照和 AOF 日志文件
2. (2) 如果数据比较重要，某个 Slave 开启 AOF 备份数据，策略设置为每秒同步一次
3. (3) 为了主从复制的速度和连接的稳定性， Master 和 Slave 最好在同一个局域网内
4. (4) 尽量避免在压力很大的主库上增加从库
5. (5) 主从复制不要用图状结构，用单向链表结构更为稳定，即： Master <- Slave1 <- Slave2 <-
6. Slave3…

十四 讲解下Redis线程模型

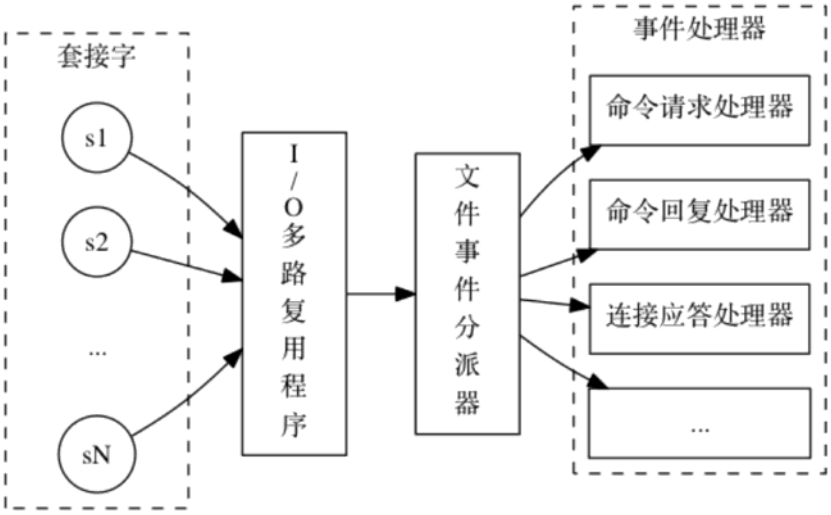


图 IMAGE_CONSTRUCT_OF_FILE_EVENT_HANDLER　文件事件处理器的四个组成部分

1. 文件事件处理器包括分别是套接字、 I/O 多路复用程序、 文件事件分派器（dispatcher）、 以及事件处理器。使用 I/O 多路复用程序来同时监听多个套接字，并根据套接字目前执行的任务来为套接字关联不同的事件处理器。当被监听的套接字准备好执行连接应答（accept）、读取（read）、写入（write）、关闭（close）等操作时， 与操作相对应的文件事件就会产生， 这时文件事件处理器就会调用套接字之前关联好的事件处理器来处理这些事件。
2. I/O 多路复用程序负责监听多个套接字， 并向文件事件分派器传送那些产生了事件的套接字。
3. 工作原理：
4. 1)I/O 多路复用程序负责监听多个套接字， 并向文件事件分派器传送那些产生了事件的套接字。
5. 尽管多个文件事件可能会并发地出现， 但 I/O 多路复用程序总是会将所有产生事件的套接字都入队到一个队列里面， 然后通过这个队列， 以有序（sequentially）、同步（synchronously）、每次一个套接字的方式向文件事件分派器传送套接字： 当上一个套接字产生的事件被处理完毕之后（该套接字为事件所关联的事件处理器执行完毕），I/O 多路复用程序才会继续向文件事件分派器传送下一个套接字。如果一个套接字又可读又可写的话， 那么服务器将先读套接字， 后写套接字。

十五 为什么Redis的操作是原子性的，怎么保证原子性的?

1. 对于Redis而言，命令的原子性指的是：一个操作的不可以再分，操作要么执行，要么不执行。

2. Redis的操作之所以是原子性的，是因为Redis是单线程的。
3. Redis本身提供的所有API都是原子操作，Redis中的事务其实是要保证批量操作的原子性。
4. 多个命令在并发中也是原子性的吗?
5. 不一定， 将get和set改成单命令操作，incr 。使用Redis的事务，或者使用Redis+Lua==的方式实现

十六 Redis事务

1. Redis事务功能是通过MULTI、EXEC、DISCARD和WATCH 四个原语实现的
2. Redis会将一个事务中的所有命令序列化，然后按顺序执行。
3. 1.redis 不支持回滚“Redis 在事务失败时不进行回滚，而是继续执行余下的命令”， 所以 Redis 的内部可以保持简单且快速。
4. 2.如果在一个事务中的命令出现错误，那么所有的命令都不会执行;
5. 3.如果在一个事务中出现运行错误，那么正确的命令会被执行。
6. 注：redis的discard只是结束本次事务,正确命令造成的影响仍然存在.
7. 1）MULTI命令用于开启一个事务，它总是返回OK。 MULTI执行之后，客户端可以继续向服务器发送任意多条命令，这些命令不会立即被执行，而是被放到一个队列中，当EXEC命令被调用时，所有队列中的命令才会被执行。
8. 2）EXEC：执行所有事务块内的命令。返回事务块内所有命令的返回值，按命令执行的先后顺序排列。 当操作被打断时，返回空值 nil 。
9. 3）通过调用DISCARD，客户端可以清空事务队列，并放弃执行事务， 并且客户端会从事务状态中退出。
10. 4）WATCH 命令可以为 Redis 事务提供 check-and-set （CAS）行为。 可以监控一个或多个键，一旦其中有一个键被修改（或删除），之后的事务就不会执行，监控一直持续到EXEC命令。

十七 Redis实现分布式锁

1. Redis为单进程单线程模式，采用队列模式将并发访问变成串行访问，且多客户端对Redis的连接并不存在竞争关系Redis中可以使用SETNX命令实现分布式锁。
2. 将 key 的值设为 value ，当且仅当 key 不存在。 若给定的 key 已经存在，则 SETNX 不做任何动作

3. 解锁：使用 del key 命令就能释放锁
4. 解决死锁：
5. 1）通过Redis中expire()给锁设定最大持有时间，如果超过，则Redis来帮我们释放锁。
6. 2） 使用 setnx key “当前系统时间+锁持有的时间”和getset key “当前系统时间+锁持有的时间”组合的命令就可以实现。

来自 <https://blog.csdn.net/weixin_41241676/article/details/115051943>

2021年3月31日 11:43

1 什么是kafka

Kafka是分布式发布-订阅消息系统，它最初是由LinkedIn公司开发的，之后成为Apache项目的一部分，Kafka是一个分布式，可划分的，冗余备份的持久性的日志服务，它主要用于处理流式数据。

2 为什么要使用 kafka，为什么要使用消息队列

缓冲和削峰：上游数据时有突发流量，下游可能扛不住，或者下游没有足够多的机器来保证冗余，kafka在中间可以起到一个缓冲的作用，把消息暂存在kafka中，下游服务就可以按照自己的节奏进行慢慢处理。

解耦和扩展性：项目开始的时候，并不能确定具体需求。消息队列可以作为一个接口层，解耦重要的业务流程。只需要遵守约定，针对数据编程即可获取扩展能力。

冗余：可以采用一对多的方式，一个生产者发布消息，可以被多个订阅topic的服务消费到，供多个毫无关联的业务使用。

健壮性：消息队列可以堆积请求，所以消费端业务即使短时间死掉，也不会影响主要业务的正常进行。

异步通信：很多时候，用户不想也不需要立即处理消息。消息队列提供了异步处理机制，允许用户把一个消息放入队列，但并不立即处理它。想向队列中放入多少消息就放多少，然后在需要的时候再去处理它们。

3.Kafka中的ISR、AR又代表什么？ISR的伸缩又指什么

ISR:In-Sync Replicas 副本同步队列
AR:Assigned Replicas 所有副本
ISR是由leader维护，follower从leader同步数据有一些延迟（包括延迟时间replica.lag.time.max.ms和延迟条数replica.lag.max.messages两个维度, 当前最新的版本0.10.x中只支持replica.lag.time.max.ms这个维度），任意一个超过阈值都会把follower剔除出ISR, 存入OSR（Outof-Sync Replicas）列表，新加入的follower也会先存放在OSR中。AR=ISR+OSR。

4.kafka中的broker 是干什么的

broker 是消息的代理，Producers往Brokers里面的指定Topic中写消息，Consumers从Brokers里面拉取指定Topic的消息，然后进行业务处理，broker在中间起到一个代理保存消息的中转站。

5.kafka中的 zookeeper 起到什么作用，可以不用zookeeper么

zookeeper 是一个分布式的协调组件，早期版本的kafka用zk做meta信息存储，consumer的消费状态，group的管理以及 offset的值。考虑到zk本身的一些因素以及整个架构较大概率存在单点问题，新版本中逐渐弱化了zookeeper的作用。新的consumer使用了kafka内部的group coordination协议，也减少了对zookeeper的依赖，

但是broker依然依赖于ZK，zookeeper 在kafka中还用来选举controller 和 检测broker是否存活等等。

6.kafka follower如何与leader同步数据

Kafka的复制机制既不是完全的同步复制，也不是单纯的异步复制。完全同步复制要求All Alive Follower都复制完，这条消息才会被认为commit，这种复制方式极大的影响了吞吐率。而异步复制方式下，Follower异步的从Leader复制数据，数据只要被Leader写入log就被认为已经commit，这种情况下，如果leader挂掉，会丢失数据，kafka使用ISR的方式很好的均衡了确保数据不丢失以及吞吐率。Follower可以批量的从Leader复制数据，而且Leader充分利用磁盘顺序读以及send file(zero copy)机制，这样极大的提高复制性能，内部批量写磁盘，大幅减少了Follower与Leader的消息量差。

7.什么情况下一个 broker 会从 isr中踢出去

leader会维护一个与其基本保持同步的Replica列表，该列表称为ISR(in-sync Replica)，每个Partition都会有一个ISR，而且是由leader动态维护 ，如果一个follower比一个leader落后太多，或者超过一定时间未发起数据复制请求，则leader将其重ISR中移除 。

8.kafka 为什么那么快

Cache Filesystem Cache PageCache缓存

顺序写 由于现代的操作系统提供了预读和写技术，磁盘的顺序写大多数情况下比随机写内存还要快。

Zero-copy 零拷技术减少拷贝次数

Batching of Messages 批量量处理。合并小的请求，然后以流的方式进行交互，直顶网络上限。

Pull 拉模式 使用拉模式进行消息的获取消费，与消费端处理能力相符。

9.kafka producer如何优化打入速度

增加线程

提高 batch.size

增加更多 producer 实例

增加 partition 数

设置 acks=-1 时，如果延迟增大：可以增大 num.replica.fetchers（follower 同步数据的线程数）来调解；

跨数据中心的传输：增加 socket 缓冲区设置以及 OS tcp 缓冲区设置。

10.kafka producer 打数据，ack 为 0， 1， -1 的时候代表啥， 设置 -1 的时候，什么情况下，leader 会认为一条消息 commit了

1（默认） 数据发送到Kafka后，经过leader成功接收消息的的确认，就算是发送成功了。在这种情况下，如果leader宕机了，则会丢失数据。

0 生产者将数据发送出去就不管了，不去等待任何返回。这种情况下数据传输效率最高，但是数据可靠性确是最低的。

-1 producer需要等待ISR中的所有follower都确认接收到数据后才算一次发送完成，可靠性最高。当ISR中所有Replica都向Leader发送ACK时，leader才commit，这时候producer才能认为一个请求中的消息都commit了。

11.kafka unclean 配置代表啥，会对 spark streaming 消费有什么影响

unclean.leader.election.enable 为true的话，意味着非ISR集合的broker 也可以参与选举，这样有可能就会丢数据，spark streaming在消费过程中拿到的 end offset 会突然变小，导致 spark streaming job挂掉。如果unclean.leader.election.enable参数设置为true，就有可能发生数据丢失和数据不一致的情况，Kafka的可靠性就会降低；而如果unclean.leader.election.enable参数设置为false，Kafka的可用性就会降低。

12.如果leader crash时，ISR为空怎么办

kafka在Broker端提供了一个配置参数：unclean.leader.election,这个参数有两个值：

true（默认）：允许不同步副本成为leader，由于不同步副本的消息较为滞后，此时成为leader，可能会出现消息不一致的情况。

false：不允许不同步副本成为leader，此时如果发生ISR列表为空，会一直等待旧leader恢复，降低了可用性。

13.kafka的message格式是什么样的

一个Kafka的Message由一个固定长度的header和一个变长的消息体body组成

header部分由一个字节的magic(文件格式)和四个字节的CRC32(用于判断body消息体是否正常)构成。

当magic的值为1的时候，会在magic和crc32之间多一个字节的数据：attributes(保存一些相关属性，

比如是否压缩、压缩格式等等);如果magic的值为0，那么不存在attributes属性

body是由N个字节构成的一个消息体，包含了具体的key/value消息

14.kafka中consumer group 是什么概念

同样是逻辑上的概念，是Kafka实现单播和广播两种消息模型的手段。同一个topic的数据，会广播给不同的group；同一个group中的worker，只有一个worker能拿到这个数据。换句话说，对于同一个topic，每个group都可以拿到同样的所有数据，但是数据进入group后只能被其中的一个worker消费。group内的worker可以使用多线程或多进程来实现，也可以将进程分散在多台机器上，worker的数量通常不超过partition的数量，且二者最好保持整数倍关系，因为Kafka在设计时假定了一个partition只能被一个worker消费（同一group内）。

15.Kafka中的消息是否会丢失和重复消费?

要确定Kafka的消息是否丢失或重复，从两个方面分析入手：消息发送和消息消费。

1、消息发送

Kafka消息发送有两种方式：同步（sync）和异步（async），默认是同步方式，可通过producer.type属性进行配置。Kafka通过配置request.required.acks属性来确认消息的生产：

0---表示不进行消息接收是否成功的确认；
1---表示当Leader接收成功时确认；
-1---表示Leader和Follower都接收成功时确认；

综上所述，有6种消息生产的情况，下面分情况来分析消息丢失的场景：

（1）acks=0，不和Kafka集群进行消息接收确认，则当网络异常、缓冲区满了等情况时，消息可能丢失；

（2）acks=1、同步模式下，只有Leader确认接收成功后但挂掉了，副本没有同步，数据可能丢失；

2、消息消费

Kafka消息消费有两个consumer接口，Low-level API和High-level API：

Low-level API：消费者自己维护offset等值，可以实现对Kafka的完全控制；

High-level API：封装了对parition和offset的管理，使用简单；

如果使用高级接口High-level API，可能存在一个问题就是当消息消费者从集群中把消息取出来、并提交了新的消息offset值后，还没来得及消费就挂掉了，那么下次再消费时之前没消费

成功的消息就“诡异”的消失了；

解决办法：

针对消息丢失：同步模式下，确认机制设置为-1，即让消息写入Leader和Follower之后再确认消息发送成功；异步模式下，为防止缓冲区满，可以在配置文件设置不限制阻塞超时时间，当缓冲区满时让生产者一直处于阻塞状态；

针对消息重复：将消息的唯一标识保存到外部介质中，每次消费时判断是否处理过即可。

消息重复消费及解决参考：https://www.javazhiyin.com/22910.html

16.为什么Kafka不支持读写分离？

在 Kafka 中，生产者写入消息、消费者读取消息的操作都是与 leader 副本进行交互的，从 而实现的是一种主写主读的生产消费模型。

Kafka 并不支持主写从读，因为主写从读有 2 个很明 显的缺点:

(1)数据一致性问题。数据从主节点转到从节点必然会有一个延时的时间窗口，这个时间窗口会导致主从节点之间的数据不一致。某一时刻，在主节点和从节点中 A 数据的值都为 X，之后将主节点中 A 的值修改为 Y，那么在这个变更通知到从节点之前，应用读取从节点中的 A 数据的值并不为最新的 Y，由此便产生了数据不一致的问题。

(2)延时问题。类似 Redis 这种组件，数据从写入主节点到同步至从节点中的过程需要经 历网络→主节点内存→网络→从节点内存这几个阶段，整个过程会耗费一定的时间。而在 Kafka 中，主从同步会比 Redis 更加耗时，它需要经历网络→主节点内存→主节点磁盘→网络→从节点内存→从节点磁盘这几个阶段。对延时敏感的应用而言，主写从读的功能并不太适用。

17.Kafka中是怎么体现消息顺序性的？

kafka每个partition中的消息在写入时都是有序的，消费时，每个partition只能被每一个group中的一个消费者消费，保证了消费时也是有序的。
整个topic不保证有序。如果为了保证topic整个有序，那么将partition调整为1.

18.消费者提交消费位移时提交的是当前消费到的最新消息的offset还是offset+1？

offset+1

19.kafka如何实现延迟队列？

Kafka并没有使用JDK自带的Timer或者DelayQueue来实现延迟的功能，而是基于时间轮自定义了一个用于实现延迟功能的定时器（SystemTimer）。JDK的Timer和DelayQueue插入和删除操作的平均时间复杂度为O(nlog(n))，并不能满足Kafka的高性能要求，而基于时间轮可以将插入

和删除操作的时间复杂度都降为O(1)。时间轮的应用并非Kafka独有，其应用场景还有很多，在Netty、Akka、Quartz、Zookeeper等组件中都存在时间轮的踪影。

底层使用数组实现，数组中的每个元素可以存放一个TimerTaskList对象。TimerTaskList是一个环形双向链表，在其中的链表项TimerTaskEntry中封装了真正的定时任务TimerTask.

Kafka中到底是怎么推进时间的呢？Kafka中的定时器借助了JDK中的DelayQueue来协助推进时间轮。具体做法是对于每个使用到的TimerTaskList都会加入到DelayQueue中。Kafka中的TimingWheel专门用来执行插入和删除TimerTaskEntry的操作，而DelayQueue专门负责时间推进的任务。再试想一下，DelayQueue中的第一个超时任务列表的expiration为200ms，第二个超时任务为840ms，这里获取DelayQueue的队头只需要O(1)的时间复杂度。如果采用每秒定时推进，那么获取到第一个超时的任务列表时执行的200次推进中有199次属于“空推进”，而获取到第二个超时任务时有需要执行639次“空推进”，这样会无故空耗机器的性能资源，这里采用DelayQueue来辅助以少量空间换时间，从而做到了“精准推进”。Kafka中的定时器真可谓是“知人善用”，用TimingWheel做最擅长的任务添加和删除操作，而用DelayQueue做最擅长的时间推进工作，相辅相成。

参考：https://blog.csdn.net/u013256816/article/details/80697456

20.Kafka中的事务是怎么实现的？

参考：https://blog.csdn.net/u013256816/article/details/89135417

21.Kafka中有那些地方需要选举？这些地方的选举策略又有哪些？

2021年3月31日 11:44

1、什么是微服务?

微服务架构是一种架构模式或者说是一种架构风格，它提倡将单一应用程序划分成一组小的服务，每个服务运行在其独立的自己的进程中，服务之间互相协调、互相配合，为用户提供最终价值。 服务之间采用轻量级的通信机制互相沟通（通常是基于HTTP的RESTful API）。每个服务都围绕着具体业务进行构建，并且能够被独立地部署到生产环境、类生产环境等。另外，应尽量避免统一的、集中式的服务管理机制，对具体的一个服务而言，应根据业务上下文，选择合适的语言、工具对其进行构建，可以有一个非常轻量级的集中式管理来协调这些服务，可以使用不同的语言来编写服务，也可以使用不同的数据存储。

从技术维度来说：

微服务化的核心就是将传统的一站式应用，根据业务拆分成一个一个的服务，彻底地去耦合，每一个微服务提供单个业务功能的服务，一个服务做一件事，从技术角度看就是一种小而独立的处理过程，类似进程概念，能够自行单独启动或销毁，拥有自己独立的数据库。

2、微服务之间是如何通讯的?

第一种：远程过程调用（Remote Procedure Invocation）

直接通过远程过程调用来访问别的service。

示例：REST、gRPC、Apache、Thrift

优点：

简单，常见。因为没有中间件代理，系统更简单

缺点：

只支持请求/响应的模式，不支持别的，比如通知、请求/异步响应、发布/订阅、发布/异步响应降低了可用性，因为客户端和服务端在请求过程中必须都是可用的

第二种：消息

使用异步消息来做服务间通信。服务间通过消息管道来交换消息，从而通信。

示例：Apache Kafka、RabbitMQ

优点:

把客户端和服务端解耦，更松耦合 提高可用性，因为消息中间件缓存了消息，直到消费者可以消费支持很多通信机制比如通知、请求/异步响应、发布/订阅、发布/异步响应

缺点:

消息中间件有额外的复杂性

3、springcloud 与dubbo有哪些区别?

相同点：SpringCloud 和Dubbo可以实现RPC远程调用框架，可以实现服务治理。

不同点:SpringCloud是一套目前比较网站微服务框架了，整合了分布式常用解决方案遇到了问题注册中心Eureka、负载均衡器Ribbon ，客户端调用工具Rest和Feign，分布式配置中心Config，服务保护Hystrix，网关Zuul Gateway ，服务链路Zipkin，消息总线Bus等。

Dubbo内部实现功能没有SpringCloud强大（全家桶），只是实现服务治理，缺少分布式配置中心、网关、链路、总线等，如果需要用到这些组件，需要整合其他框架。

表 Spring Cloud与Dubbo功能对比

功能名称	Dubbo	Spring Cloud
服务注册中心	ZooKeeper	Spring Cloud Netflix Eureka
服务调用方式	RPC	REST API
服务网关	无	Spring Cloud Netflix Zuul
断路器	不完善	Spring Cloud Netflix Hystrix
分布式配置	无	Spring Cloud Config
服务跟踪	无	Spring Cloud Sleuth
消息总线	无	Spring Cloud Bus
数据流	无	Spring Cloud Stream
批量任务	无	Spring Cloud Task

4、请谈谈对SpringBoot 和SpringCloud的理解

SpringBoot专注于快速方便的开发单个个体微服务。SpringCloud是关注全局的微服务协调整理治理框架，它将SpringBoot开发的一个个单体微服务整合并管理起来，为各个微服务之间提供，配置管理、服务发现、断路器、路由、微代理、事件总线、全局锁、决策竞选、分布式会话等等集成服务SpringBoot可以离开SpringCloud独立使用开发项目，但是SpringCloud离不开SpringBoot，属于依赖的关系.SpringBoot专注于快速、方便的开发单个微服务个体，SpringCloud关注全局的服务治理框架。Spring Boot可以离开Spring Cloud独立使用开发项目，但是Spring Cloud离不开Spring Boot，属于依赖的关系。

5、分布式系统面临的问题

复杂分布式体系结构中的应用程序有数十个依赖关系，每个依赖关系在某些时候将不可避免地失败。

服务雪崩多个微服务之间调用的时候，假设微服务A调用微服务B和微服务C，微服务B和微服务C又调用其它的微服务，这就是所谓的“扇出”。如果扇出的链路上某个微服务的调用响应时间过长或者不可用，对微服务A的调用就会占用越来越多的系统资源，进而引起系统崩溃，所谓的“雪崩效应”.对于高流量的应用来说，单一的后端依赖可能会导致所有服务器上的所有资源都在几秒钟内饱和。比失败更糟糕的是，这些应用程序还可能导致服务之间的延迟增加，备份队列，线程和其他系统资源紧张，导致整个系统发生更多的级联故障。这些都表示需要对故障和延迟进行隔离和管理，以便单个依赖关系的失败，不能取消整个应用程序或系统。

一般情况对于服务依赖的保护主要有以下三种解决方案：

熔断模式：这种模式主要是参考电路熔断，如果一条线路电压过高，保险丝会熔断，防止火灾。放到我们的系统中，如果某个目标服务调用慢或者有大量超时，此时，熔断该服务的调用，对于后续调用请求，不在继续调用目标服务，直接返回，快速释放资源。如果目标服务情况好转则恢复调用。隔离模式：这种模式就像对系统请求按类型划分成一个个小岛的一样，当某个小岛被火少光了，不会影响到其他的小岛。例如可以对不同类型的请求使用线程池来资源隔离，每种类型的请求互不影响，如果一种类型的请求线程资源耗尽，则对后续的该类型请求直接返回，不再调用后续资源。这种模式使用场景非常多，例如将一个服务拆开，对于重要的服务使用单独服务器来部署，再或者公司最近推广的多中心。限流模式：上述的熔断模式和隔

离模式都属于出错后的容错处理机制，而限流模式则可以称为预防模式。限流模式主要是提前对各个类型的请求设置最高的QPS阈值，若高于设置的阈值则对该请求直接返回，不再调用后续资源。这种模式不能解决服务依赖的问题，只能解决系统整体资源分配问题，因为没有被限流的请求依然有可能造成雪崩效应。

6、什么是服务熔断，什么是服务降级

服务熔断

熔断机制是应对雪崩效应的一种微服务链路保护机制。当扇出链路的某个微服务不可用或者响应时间太长时，会进行服务的降级，进而熔断该节点微服务的调用，快速返回”错误”的响应信息。当检测到该节点微服务调用响应正常后恢复调用链路。在SpringCloud框架里熔断机制通过Hystrix实现。Hystrix会监控微服务间调用的状况，当失败的调用到一定阈值，缺省是5秒内20次调用失败就会启动熔断机制。熔断机制的注解是@HystrixCommand。

Hystrix服务降级

其实就是线程池中单个线程障处理，防止单个线程请求时间太长，导致资源长期被占有而得不到释放，从而导致线程池被快速占用完，导致服务崩溃。

Hystrix能解决如下问题：

请求超时降级，线程资源不足降级，降级之后可以返回自定义数据线程池隔离降级，分布式服务可以针对不同的服务使用不同的线程池，从而互不影响自动触发降级与恢复实现请求缓存和请求合并7、微服务的优缺点分别是什么？说下你在项目开发中碰到的坑?

优点

每个服务足够内聚，足够小，代码容易理解这样能聚焦一个指定的业务功能或业务需求开发简单、开发效率提高，一个服务可能就是专一的只干一件事。微服务能够被小团队单独开发，这个小团队是2到5人的开发人员组成。微服务是松耦合的，是有功能意义的服务，无论是在开发阶段或部署阶段都是独立的。微服务能使用不同的语言开发。易于和第三方集成，微服务允许容易且灵活的方式集成自动部署，通过持续集成工具，如Jenkins, Hudson, bamboo 。微服务易于被一个开发人员理解，修改和维护，这样小团队能够更关注自己的工作成果。无需通过合作才能体现价值。微服务允许你利用融合最新技术。微服务只是业务逻辑的代码，不会和HTML,CSS 或其他界面组件混合。每个微服务都有自己的存储能力，可以有自己的数据库。也可以有统一数据库。

缺点

开发人员要处理分布式系统的复杂性多服务运维难度，随着服务的增加，运维的压力也在增大系统部署依赖服务间通信成本数据一致性系统集成测试性能监控……

8、你所知道的微服务技术栈有哪些？请列举一二

服务开发Springboot、Spring、SpringMVC服务配置与管理Netflix公司的Archaius、阿里的Diamond等服务注册与发现Eureka、Consul、Zookeeper等服务调用Rest、RPC、gRPC服务熔断器Hystrix、Envoy等负载均衡Ribbon、Nginx等服务接口调用(客户端调用服务的简化工具)Feign等消息队列Kafka、RabbitMQ、ActiveMQ等服务配置中心管理SpringCloudConfig、Chef等服务路由（API网关）Zuul等服务监控Zabbix、Nagios、Metrics、Spectator等全链路追踪Zipkin，Brave、Dapper等服务部署Docker、OpenStack、Kubernetes等数据流操作开发包SpringCloud Stream（封装与Redis,Rabbit、Kafka等发送接收消息）事件消息总线Spring Cloud Bus9、什么是 Eureka服

务注册与发现
Eureka是Netflix的一个子模块，也是核心模块之一。Eureka是一个基于REST的服务，用于定位服务，以实现云端中间层服务发现和故障转移。服务注册与发现对于微服务架构来说是非常重要的，有了服务发现与注册，只需要使用服务的标识符，就可以访问到服务，而不需要修改服务调用的配置文件了。功能类似于dubbo的注册中心，比如Zookeeper。
10、Eureka的基本架构是什么?
Spring Cloud 封装了 Netflix 公司开发的 Eureka 模块来实现服务注册和发现(请对比Zookeeper)。
Eureka 采用了 C-S 的设计架构。Eureka Server 作为服务注册功能的服务器，它是服务注册中心。
而系统中的其他微服务，使用 Eureka 的客户端连接到 Eureka Server并维持心跳连接。这样系统的维护人员就可以通过 Eureka Server 来监控系统中各个微服务是否正常运行。SpringCloud 的一些其他模块（比如Zuul）就可以通过 Eureka Server 来发现系统中的其他微服务，并执行相关的逻辑。
Eureka包含两个组件： Eureka Server 和 Eureka Client
Eureka Server提供服务注册服务各个节点启动后，会在EurekaServer中进行注册，这样EurekaServer中的服务注册表中将会存储所有可用服务节点的信息，服务节点的信息可以在界面中直观的看到
EurekaClient是一个Java客户端用于简化Eureka Server的交互，客户端同时也具备一个内置的、使用轮询(round-robin)负载算法的负载均衡器。在应用启动后，将会向Eureka Server发送心跳(默认周期为30秒)。如果Eureka Server在多个心跳周期内没有接收到某个节点的心跳，EurekaServer将会从服务注册表中把这个服务节点移除（默认90秒）
11、作为服务注册中心，Eureka比Zookeeper好在哪里?
著名的CAP理论指出，一个分布式系统不可能同时满足C(一致性)、A(可用性)和P(分区容错性)。由于分区容错性P在是分布式系统中必须要保证的，因此我们只能在A和C之间进行权衡。
因此，Zookeeper 保证的是CP, Eureka 则是AP。
Zookeeper保证CP当向注册中心查询服务列表时，我们可以容忍注册中心返回的是几分钟以前的注册信息，但不能接受服务直接down掉不可用。也就是说，服务注册功能对可用性的要求要高于一致性。但是zk会出现这样一种情况，当master节点因为网络故障与其他节点失去联系时，剩余节点会重新进行leader选举。问题在于，选举leader的时间太长，30~120s,且选举期间整个zk集群都是不可用的，这就导致在选举期间注册服务瘫痪。在云部署的环境下，因网络问题使得zk集群失去master节点是较大概率会发生的事，虽然服务能够最终恢复，但是漫长的选举时间导致的注册长期不可用是不能容忍的。
Eureka保证APEureka看明白了这一点，因此在设计时就优先保证可用性。Eureka各个节点都是平等的，几个节点挂掉不会影响正常节点的工作，剩余的节点依然可以提供注册和查询服务。而Eureka的客户端在向某个Eureka注册或时如果发现连接失败，则会自动切换至其它节点，只要有一台Eureka还在，就能保证注册服务可用(保证可用性)，只不过查到的信息可能不是最新的(不保证强一致性)。
除此之外，Eureka还有一种自我保护机制，如果在15分钟内超过85%的节点都没有正常的心

跳，那么Eureka就认为客户端与注册中心出现了网络故障，此时会出现以下几种情况：
Eureka不再从注册列表中移除因为长时间没收到心跳而应该过期的服务Eureka仍然能够接受新服务的注册和查询请求，但是不会被同步到其它节点上(即保证当前节点依然可用)当网络稳定时，当前实例新的注册信息会被同步到其它节点中因此， Eureka可以很好的应对因网络故障导致部分节点失去联系的情况，而不会像zookeeper那样使整个注册服务瘫痪。

12、什么是 Ribbon负载均衡

Spring Cloud Ribbon是基于Netflix Ribbon实现的一套客户端 负载均衡的工具。
简单的说，Ribbon是Netflix发布的开源项目，主要功能是提供客户端的软件负载均衡算法，将Netflix的中间层服务连接在一起。Ribbon客户端组件提供一系列完善的配置项如连接超时，重试等。简单的说，就是在配置文件中列出Load Balancer（简称LB）后面所有的机器，Ribbon会自动的帮助你基于某种规则（如简单轮询，随机连接等）去连接这些机器。我们也很容易使用Ribbon实现自定义的负载均衡算法。

13、Ribbon负载均衡能干嘛?

LB（负载均衡LB，即负载均衡(Load Balance)，在微服务或分布式集群中经常用的一种应用。负载均衡简单的说就是将用户的请求平摊的分配到多个服务上，从而达到系统的HA。常见的负载均衡有软件Nginx，LVS，硬件 F5等。相应的在中间件，例如：dubbo和SpringCloud中均给我们提供了负载均衡，SpringCloud的负载均衡算法可以自定义。集中式LB即在服务的消费方和提供方之间使用独立的LB设施(可以是硬件，如F5, 也可以是软件，如nginx), 由该设施负责把访问请求通过某种策略转发至服务的提供方；进程内LB将LB逻辑集成到消费方，消费方从服务注册中心获知有哪些地址可用，然后自己再从这些地址中选择出一个合适的服务器。注意： Ribbon就属于进程内LB，它只是一个类库，集成于消费方进程，消费方通过它来获取到服务提供方的地址。

14、什么是 Feign 负载均衡

Feign是一个声明式WebService客户端。使用Feign能让编写Web Service客户端更加简单, 它的使用方法是定义一个接口，然后在上面添加注解，同时也支持JAX-RS标准的注解。Feign也支持可拔插式的编码器和解码器。Spring Cloud对Feign进行了封装，使其支持了Spring MVC标准注解和HttpMessageConverters。 Feign可以与Eureka和Ribbon组合使用以支持负载均衡。
Feign是一个声明式的Web服务客户端，使得编写Web服务客户端变得非常容易，只需要创建一个接口，然后在上面添加注解即可。

15、Feign 能干什么

Feign旨在使编写Java Http客户端变得更容易。
前面在使用Ribbon+RestTemplate时，利用RestTemplate对http请求的封装处理，形成了一套模版化的调用方法。但是在实际开发中，由于对服务依赖的调用可能不止一处，往往一个接口会被多处调用，所以通常都会针对每个微服务自行封装一些客户端类来包装这些依赖服务的调用。所以，Feign在此基础上做了进一步封装，由他来帮助我们定义和实现依赖服务接口的定义。在Feign的实现下，我们只需创建一个接口并使用注解的方式来配置它(以前是Dao接口上面标注Mapper注解,现在是一个微服务接口上面标注一个Feign注解即可)，即可完成对服务提供方的接口绑定，简化了使用Spring cloud Ribbon时，自动封装服务调用客户端的开发量。

Feign集成了Ribbon利用Ribbon维护了MicroServiceCloud-Dept的服务列表信息，并且通过轮询实现了客户端的负载均衡。而与Ribbon不同的是，通过feign只需要定义服务绑定接口且以声明式的方法，优雅而简单的实现了服务调用

Feign通过接口的方法调用Rest服务（之前是Ribbon+RestTemplate），该请求发送给Eureka服务器（http://MICROSERVICECLOUD-DEPT/dept/list）,通过Feign直接找到服务接口，由于在进行服务调用的时候融合了Ribbon技术，所以也支持负载均衡作用。

16、什么是 Hystrix断路器

Hystrix是一个用于处理分布式系统的延迟和容错的开源库，在分布式系统里，许多依赖不可避免的会调用失败，比如超时、异常等， Hystrix能够保证在一个依赖出问题的情况下，不会导致整体服务失败，避免级联故障，以提高分布式系统的弹性。

“断路器”本身是一种开关装置，当某个服务单元发生故障之后，通过断路器的故障监控（类似熔断保险丝），向调用方返回一个符合预期的、可处理的备选响应（FallBack），而不是长时间的等待或者抛出调用方无法处理的异常，这样就保证了服务调用方的线程不会被长时间、不必要地占用，从而避免了故障在分布式系统中的蔓延，乃至雪崩。

17、Hystrix断路器能干嘛?

服务降级整体资源快不够了，忍痛将某些服务先关掉，待渡过难关，再开启回来服务熔断熔断机制是应对雪崩效应的一种微服务链路保护机制。当扇出链路的某个微服务不可用或者响应时间太长时，会进行服务的降级，进而熔断该节点微服务的调用，快速返回”错误”的响应信息。当检测到该节点微服务调用响应正常后恢复调用链路。在SpringCloud框架里熔断机制通过Hystrix实现。Hystrix会监控微服务间调用的状况，当失败的调用到一定阈值，缺省是5秒内20次调用失败就会启动熔断机制。熔断机制的注解是@HystrixCommand。服务限流接近实时的监控除了隔离依赖服务的调用以外，Hystrix还提供了准实时的调用监控（Hystrix Dashboard），Hystrix会持续地记录所有通过Hystrix发起的请求的执行信息，并以统计报表和图形的形式展示给用户，包括每秒执行多少请求多少成功，多少失败等。Netflix通过hystrix-metrics-event-stream项目实现了对以上指标的监控。Spring Cloud也提供了Hystrix Dashboard的整合，对监控内容转化成可视化界面。18、什么是 zuul路由网关

Zuul 包含了对请求的路由和过滤两个最主要的功能：

其中路由功能负责将外部请求转发到具体的微服务实例上，是实现外部访问统一入口的基础而过滤器功能则负责对请求的处理过程进行干预，是实现请求校验、服务聚合等功能的基础.Zuul和Eureka进行整合，将Zuul自身注册为Eureka服务治理下的应用，同时从Eureka中获得其他微服务的消息，也即以后的访问微服务都是通过Zuul跳转后获得。

注意： Zuul服务最终还是会注册进Eureka

提供=代理+路由+过滤 三大功能

19、什么是SpringCloud Config分布式配置中心

SpringCloud Config为微服务架构中的微服务提供集中化的外部配置支持，配置服务器为各个不同微服务应用的所有环境提供了一个中心化的外部配置。

20、分布式配置中心能干嘛?

集中管理配置文件不同环境不同配置，动态化的配置更新，分环境部署比如dev/test/prod/beta/release运行期间动态调整配置，不再需要在每个服务部署的机器上编写配置文件，服务会向配置中心统一拉取配置自己的信息当配置发生变动时，服务不需要重启即

可感知到配置的变化并应用新的配置将配置信息以REST接口的形式暴露

来自 <https://baijiahao.baidu.com/s?id=1655329799036379323&wfr=spider&for=pc>

服务限流

2021年3月31日 11:47

1 TCP和UDP区别，应用场景

TCP：为应用层提供可靠的、面向连接的和基于流的服务。使用超时重传、数据确认等方式来确保数据包被正确地发送至目的端，因此服务是可靠的。使用TCP协议通信的双方必须先建立TCP连接，并在内核中为该连接维持一些必要的数据结构，比如连接的状态、读写缓冲区，以及诸多定时器等。当通信结束时，双方必须关闭连接以释放这些内核数据。TCP基于流。基于流的数据没有边界（长度）限制，它源源不断地从通信的一端流入另一端。发送端可以逐个地向数据流中写入数据，接收端也可以逐个地将它们读出。

缺点：因为TCP有确认机制、三次握手机制，这些也导致TCP容易被人利用，实现DOS、DDOS、CC等攻击

UDP：则与TCP想反，它为应用层提供不可靠、无连接和基于数据报的服务。“不可靠”意味着UDP协议无法保证数据从发送端正确地传送到目的端。如果数据在途中丢失，或者目的端通过数据校验发现数据错误而将其丢弃，则UDP协议只是简单地通知应用程序发送失败。因此使用UDP协议的应用程序通常要自己处理数据确认，超时重传等逻辑。UDP是无连接的，即通信双方不保持一个长久的联系，因此应用程序每次发送数据都要明确指定接收端的地址（IP地址等信息）。基于数据报的服务，是相对基于流的服务而言的。每个UDP数据报都有一个长度，接收端必须以该长度为最小单位将其所有内容依次读出，否则数据将被截断。

UDP优点：快比TCP稍安全

缺点：不可靠，不稳定，如果网络质量不好，很容易丢包。

什么时候该使用TCP： 当对网络通讯质量有要求的时候，比如：整个数据要准确无误的传递给对方，这往往用于一些要求可靠的应用，比如HTTP、HTTPS、FTP等传输文件的协议，POP、SMTP等邮件传输的协议。

在日常生活中，常见使用TCP协议的应用如下： 浏览器，用的HTTP FlashFXP，用的FTP Outlook，用的POP、SMTP Putty，用的Telnet、SSH QQ文件传输

什么时候使用UDP：当对网络通讯质量要求不高的时候，要求速度尽可能的快，这时就可以使用UDP。用UDP协议的应用如下： QQ语音 QQ视频 TFTP。

2 浏览器输入www.baidu.com后，网络过程

事件顺序：

(1) 浏览器获取输入的域名www.baidu.com

(2) 浏览器向DNS请求解析www.baidu.com的IP地址

(3) 域名系统DNS解析出百度服务器的IP地址 （详细介绍DNS）-通过网关出去

(4) 浏览器与该服务器建立TCP连接(默认端口号80)

(5) 浏览器发出HTTP请求，请求百度首页

(6) 服务器通过HTTP响应把首页文件发送给浏览器

(7) TCP连接释放

(8) 浏览器将首页文件进行解析，并将Web页显示给用户。

涉及到的协议:

(1) 应用层：HTTP(WWW访问协议)，DNS(域名解析服务)

DNS解析域名为目的IP，通过IP找到服务器路径，客户端向服务器发起HTTP会话，然后通过运输层TCP协议封装数据包，在TCP协议基础上进行传输。

(2) 传输层：TCP(为HTTP提供可靠的数据传输)，UDP(DNS使用UDP传输)，HTTP会话会被分成报文段，添加源、目的端口；TCP协议进行主要工作。

(3)网络层：IP(IP数据数据包传输和路由选择)，ICMP(提供网络传输过程中的差错检测)，ARP(将本机的默认网关IP地址映射成物理MAC地址)为数据包选择路由，IP协议进行主要工作，相邻结点的可靠传输，ARP协议将IP地址转成MAC地址。

简单理解: 域名解析 --> 发起TCP的3次握手 --> 建立TCP连接后发起http请求 --> 服务器响应http请求，浏览器得到html代码 --> 浏览器解析html代码，并请求html代码中的资源（如js、css、图片等） --> 浏览器对页面进行渲染呈现给用户。

3 三次握手和四次挥手过程

三次握手：

第一次握手：建立连接时，客户端发送SYN包到服务端，其中包含客户端的初始序号seq=x,并进入SYN_SENT(同步已发送)状态，等待服务器确认。（其中SYN=1，ACK=0，表示这是一个TCP连接请求数据报文；序号seq=x，表明数据传输数据时第一个数据字节的序号是X）.

第二次握手：服务器收到请求后，必须确认客户的数据包。同时自己也发送一个SYN包，即SYN+ACK包，此时服务器进入SYN_RECV（同步以收到）状态。（其中确认报文段中，标识位SYN=1,ACK=1，表示这是一个TCP连接响应数据报文，并包含服务器的初始序号seq(服务器)=y,以及服务器对客户端初始序号的确认号ack(服务器)=seq（客户端）+1=x+1）.

第三次握手：客户端收到服务器的SYN+ACK 包，向服务器发送一个序列号(seq=x+1)，确认号为ack=y+1,此包发送完毕，客户端和服务器进入ESTAB_LISHED（TCP连接成功)状态，完成三次握手。

为啥三次握手：假设客户端请求建立连接，发给服务器SYN包等待服务器确认，服务器收到确认后，如果是两次握手，假设服务器给客户端在第二次握手时发送数据，数据从服务器发出，服务器任务连接已经建立，但在发送数据的过程中数据丢失，客户端认为连接没有建立，会进行重传。假设每一次发送的数据一直在丢失，客户端一直SYN，服务器就会产生多个无效连接，占用资源，这个时候服务器可能会挂掉。这就是SYN的洪水攻击

第三次握手是为了防止：如果客户端迟迟没有收到服务器返回确认报文，这时会放弃连接，重新启动一条连接请求，但问题是：服务器不知道客户端没有收到，所以他会收到两个连接，浪费连接开销。如果每次都是这样，就会浪费多个连接开销。

四次挥手：

step1：第一次挥手

首先，客户端发送一个FIN，用来关闭客户端到服务器的数据传送，然后等待服务器的确认。其中终止标志位FIN=1，序列号seq=u。
step2：第二次挥手

服务器收到这个FIN，它发送一个ACK，确认ack为收到的序号加一。
step3：第三次挥手

关闭服务器到客户端的连接，发送一个FIN给客户端。
step4：第四次挥手

客户端收到FIN后，并发回一个ACK报文确认，并将确认序号seq设置为收到序号加一。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。

客户端发送FIN后，进入终止等待状态，服务器收到客户端连接释放报文段后，就立即给客户端发送确认，服务器就进入CLOSE_WAIT状态，此时TCP服务器进程就通知高层应用进程，因而从客户端到服务器的连接就释放了。此时是“半关闭状态”，即客户端不可以发送给服务器，服务器可以发送给客户端。
此时，如果服务器没有数据报发送给客户端，其应用程序就通知TCP释放连接，然后发送给客户端连接释放数据报，并等待确认。客户端发送确认后，进入TIME_WAIT状态，但是此时TCP连接还没有释放，然后经过等待计时器设置的2MSL后，才进入到CLOSE状态。

.为什么需要2MSL时间?
首先，MSL即Maximum Segment Lifetime，就是最大报文生存时间，是任何报文在网络上的存在的最长时间，超过这个时间报文将被丢弃。《TCP/IP详解》中是这样描述的：MSL是任何报文段被丢弃前在网络内的最长时间。RFC 793中规定MSL为2分钟，实际应用中常用的是30

秒、1分钟、2分钟等。

TCP的TIME_WAIT需要等待2MSL，当TCP的一端发起主动关闭，三次挥手完成后发送第四次挥手的ACK包后就进入这个状态，等待2MSL时间主要目的是：防止最后一个ACK包对方没有收到，那么对方在超时后将重发第三次握手的FIN包，主动关闭端接到重发的FIN包后可以再发一个ACK应答包。在TIME_WAIT状态时两端的端口不能使用，要等到2MSL时间结束才可以继续使用。当连接处于2MSL等待阶段时任何迟到的报文段都将被丢弃。

.为什么是四次挥手，而不是三次或是五次、六次?
双方关闭连接要经过双方都同意。所以，首先是客服端给服务器发送FIN，要求关闭连接，服务器收到后会发送一个ACK进行确认。服务器然后再发送一个FIN，客户端发送ACK确认，并进入TIME_WAIT状态。等待2MSL后自动关闭。

总结:
（1）为了保证客户端发送的最后一个ACK报文段能够到达服务器。即最后一个确认报文可能丢失，服务器会超时重传，然后服务器发送FIN请求关闭连接，客户端发送ACK确认。一个来回是两个报文生命周期。

如果没有等待时间，发送完确认报文段就立即释放连接的话，服务器就无法重传，因此也就收不到确认，就无法按步骤进入CLOSE状态，即必须收到确认才能close。
（2）防止已经失效的连接请求报文出现在连接中。经过2MSL，在这个连续持续的时间内，产生的所有报文段就可以都从网络消失。

4 DNS域名解析过程:
步骤一：当在浏览器中输入域名按下回车键后，浏览器会检查缓存中有没有这个域名对应的解析过的IP地址。如果缓存有，解析结束。浏览器缓存域名在大小和时间上都是有限制的。缓存时间可由TTL属性来设置缓存时间太长太短都不好，太长，会导致IP解析有变化，会导致域名不能正常解析，部分用户无法访问网站。缓存时间太短，用户每次都需要重新解析一次域名。

步骤二：如果用户的浏览器中缓存没有，浏览器会查找操作系统中是否有这个域名对应的DNS解析结果。其实操作系统中也会有一个域名解析的过程，在windows中可以通过c:\\windows\system32\drivers\etc\hosts文件来设置，你可以将任何域名解析到任何能够访问的IP地址。（黑客劫持域名）步骤一和步骤二都是由本机完成的。

步骤三：当机无法完成域名解析，就会真正请求域名服务器来解析这个域名了。我们怎样知道域名服务器？网络配置中都会有“DNS服务器地址”操作系统会把这个域名发送到设置中的LDNS，也就是本地的域名服务器。DNS通常都会提供给你本地互联网接入的一个DNS服务器。比如你在学校，那么这个DNS服务器一定在你们学校。WIndows中可由ipconfig查询这个地址。

步骤四：如果LDNS仍然没有解析到，就直接到Root Service域名解析器请求解析

不周五：根域名服务器返回给本地域名服务器一个所查询余的主域名服务器（gTLDServer）地址。gTLD是国际顶级域名服务器，如：.com/.cn/.org等，全球只有13台左右。

步骤六：本地域名服务器（Local DNS Server）再向上一步返回的gTLD服务器发送请求

步骤七：接收请求的gTLD服务器查找并返回此域名对应的Name Server域名服务器的地址，这个Name Server通常就是你注册的域名服务器（如你的域名供应商）

步骤八：Name Server域名服务器会查询存储的域名和IP的映射关系表，正常情况下都根据域名得到目标IP记录，连同一个TTL值返回给DNS Server域名服务器

步骤九：返回该域名对应的IP和TTL值，Local DNS Server会缓存这个域名和IP的对应关系，缓存的时间有TTL值控制。

步骤十：把解析的结果返回给用户，用户根据TTL值缓存在本地系统缓存中，域名解析过程结束。

关于DNS解析的TTL参数：

我们在配置DNS解析的时候，有一个参数常常容易忽略，就是DNS解析的TTL参数，Time To Live。TTL这个参数告诉本地DNS服务器，域名缓存的最长时间。用阿里云解析来举例，阿里云解析默认的TTL是10分钟，10分钟的含义是，本地DNS服务器对于域名的缓存时间是10分钟，10分钟之后，本地DNS服务器就会删除这条记录，删除之后，如果有用户访问这个域名，就要重复一遍上述复杂的流程。

5 HTTP协议

HTTP协议是Hyper Text Transfer Protocol（超文本传输协议）的缩写,是用于从万维网（WWW:World Wide Web ）服务器传输超文本到本地浏览器的传送协议。

HTTP是一个基于TCP/IP通信协议来传递数据（HTML 文件, 图片文件, 查询结果等）。

HTTP是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。

主要特点：

1、简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。

2、灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。

3.无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

4.无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意

味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。
5、支持B/S及C/S模式。
URI和URL的区别

URI，是uniform resource identifier，统一资源标识符，用来唯一的标识一个资源。

Web上可用的每种资源如HTML文档、图像、视频片段、程序等都是一个来URI来定位的
URI一般由三部组成：
①访问资源的命名机制
②存放资源的主机名
③资源自身的名称，由路径表示，着重强调于资源。

URL是uniform resource locator，统一资源定位器，它是一种具体的URI，即URL可以用来标识一个资源，而且还指明了如何locate这个资源。

URL是Internet上用来描述信息资源的字符串，主要用在各种WWW客户程序和服务器程序上，特别是著名的Mosaic。
采用URL可以用一种统一的格式来描述各种信息资源，包括文件、服务器的地址和目录等。URL一般由三部组成：
①协议(或称为服务方式)
②存有该资源的主机IP地址(有时也包括端口号)
③主机资源的具体地址。如目录和文件名等

URN，uniform resource name，统一资源命名，是通过名字来标识资源，比如mailto:java-net@java.sun.com。

URI是以一种抽象的，高层次概念定义统一资源标识，而URL和URN则是具体的资源标识的方式。URL和URN都是一种URI。笼统地说，每个 URL 都是 URI，但不一定每个 URI 都是 URL。这是因为 URI 还包括一个子类，即统一资源名称 (URN)，它命名资源但不指定如何定位资源。上面的 mailto、news 和 isbn URI 都是 URN 的示例。

在Java的URI中，一个URI实例可以代表绝对的，也可以是相对的，只要它符合URI的语法规则。而URL类则不仅符合语义，还包含了定位该资源的信息，因此它不能是相对的。
在Java类库中，URI类不包含任何访问资源的方法，它唯一的作用就是解析。
相反的是，URL类可以打开一个到达资源的流。
HTTP之请求消息Request

客户端发送一个HTTP请求到服务器的请求消息包括以下格式：

请求行（request line）、请求头部（header）、空行和请求数据四个部分组成。

Http请求消息结构.png

请求行以一个方法符号开头，以空格分开，后面跟着请求的URI和协议的版本。

Get请求例子，使用Charles抓取的request：

```
GET /562f25980001b1b106000338.jpg HTTP/1.1
Host    img.mukewang.com
User-Agent  Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.106 Safari/537.36
Accept  image/webp,image/*,*/*;q=0.8
Referer http://www.imooc.com/
Accept-Encoding gzip, deflate, sdch
Accept-Language zh-CN,zh;q=0.8
```

第一部分：请求行，用来说明请求类型,要访问的资源以及所使用的HTTP版本.

GET说明请求类型为GET,[/562f25980001b1b106000338.jpg]为要访问的资源，该行的最后一部分说明使用的是HTTP1.1版本。

第二部分：请求头部，紧接着请求行（即第一行）之后的部分，用来说明服务器要使用的附加信息

从第二行起为请求头部，HOST将指出请求的目的地.User-Agent,服务器端和客户端脚本都能访问它,它是浏览器类型检测逻辑的重要基础.该信息由你的浏览器来定义,并且在每个请求中自动发送等等

第三部分：空行，请求头部后面的空行是必须的

即使第四部分的请求数据为空，也必须有空行。

第四部分：请求数据也叫主体，可以添加任意的其他数据。

这个例子的请求数据为空。

POST请求例子，使用Charles抓取的request：

```
POST / HTTP1.1
Host:www.wrox.com
User-Agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.04506.648; .NET CLR 3.5.21022)
Content-Type:application/x-www-form-urlencoded
Content-Length:40
Connection: Keep-Alive

name=Professional%20Ajax&publisher=Wiley
```

第一部分：请求行，第一行明了是post请求，以及http1.1版本。
第二部分：请求头部，第二行至第六行。
第三部分：空行，第七行的空行。
第四部分：请求数据，第八行。
HTTP之响应消息Response

一般情况下，服务器接收并处理客户端发过来的请求后会返回一个HTTP的响应消息。

HTTP响应也由四个部分组成，分别是：状态行、消息报头、空行和响应正文。

http响应消息格式.jpg

例子

```
HTTP/1.1 200 OK
Date: Fri, 22 May 2009 06:07:21 GMT
Content-Type: text/html; charset=UTF-8

<html>
    <head></head>
    <body>
        <!--body goes here-->
    </body>
</html>
```

第一部分：状态行，由HTTP协议版本号， 状态码， 状态消息 三部分组成。

第一行为状态行，（HTTP/1.1）表明HTTP版本为1.1版本，状态码为200，状态消息为（ok）

第二部分：消息报头，用来说明客户端要使用的一些附加信息

第二行和第三行为消息报头，
Date:生成响应的日期和时间；Content-Type:指定了MIME类型的HTML(text/html),编码类型是UTF-8

第三部分：空行，消息报头后面的空行是必须的

第四部分：响应正文，服务器返回给客户端的文本信息。

空行后面的html部分为响应正文。
HTTP之状态码

状态代码有三位数字组成，第一个数字定义了响应的类别，共分五种类别:

1xx：指示信息--表示请求已接收，继续处理

2xx：成功--表示请求已被成功接收、理解、接受

3xx：重定向--要完成请求必须进行更进一步的操作

4xx：客户端错误--请求有语法错误或请求无法实现

5xx：服务器端错误--服务器未能实现合法的请求

以下是 HTTP 请求/响应的步骤：

1、客户端连接到Web服务器

一个HTTP客户端，通常是浏览器，与Web服务器的HTTP端口（默认为80）建立一个TCP套接字连接。例如，http://www.oakcms.cn。

2、发送HTTP请求

通过TCP套接字，客户端向Web服务器发送一个文本的请求报文，一个请求报文由请求行、请求头部、空行和请求数据4部分组成。

3、服务器接受请求并返回HTTP响应

Web服务器解析请求，定位请求资源。服务器将资源复本写到TCP套接字，由客户端读取。一个响应由状态行、响应头部、空行和响应数据4部分组成。

4、释放连接TCP连接

若connection 模式为close，则服务器主动关闭TCP连接，客户端被动关闭连接，释放TCP连接;
若connection 模式为keepalive，则该连接会保持一段时间，在该时间内可以继续接收请求;

5、客户端浏览器解析HTML内容

客户端浏览器首先解析状态行，查看表明请求是否成功的状态代码。然后解析每一个响应头，响应头告知以下为若干字节的HTML文档和文档的字符集。客户端浏览器读取响应数据HTML，根据HTML的语法对其进行格式化，并在浏览器窗口中显示。

例如：在浏览器地址栏键入URL，按下回车之后会经历以下流程：

1、浏览器向 DNS 服务器请求解析该 URL 中的域名所对应的 IP 地址;

2、解析出 IP 地址后，根据该 IP 地址和默认端口 80，和服务器建立TCP连接;

3、浏览器发出读取文件(URL 中域名后面部分对应的文件)的HTTP 请求，该请求报文作为 TCP 三次握手的第三个报文的数据发送给服务器;

4、服务器对浏览器请求作出响应，并把对应的 html 文本发送给浏览器;

5、释放 TCP连接;

6、浏览器将该 html 文本并显示内容;

GET和POST区别：

GET提交的数据会放在URL之后，以?分割URL和传输数据，参数之间以&相连，如EditPosts.aspx?name=test1&id=123456. POST方法是把提交的数据放在HTTP包的Body中.

GET提交的数据大小有限制（因为浏览器对URL的长度有限制），而POST方法提交的数据没有限制.

GET方式需要使用Request.QueryString来取得变量的值，而POST方式通过Request.Form来获取变量的值。

GET方式提交数据，会带来安全问题，比如一个登录页面，通过GET方式提交数据时，用户名和密码将出现在URL上，如果页面可以被缓存或者其他人可以访问这台机器，就可以从历史记录获得该用户的账号和密码.

HTTP协议【详解】——经典面试题

http请求由三部分组成，分别是：请求行、消息报头、请求正文

HTTP（超文本传输协议）是一个基于请求与响应模式的、无状态的、应用层的协议，常基于TCP的连接方式，HTTP1.1版本中给出一种持续连接的机制，绝大多数的Web开发，都是构建在HTTP协议之上的Web应用。

1、常用的HTTP方法有哪些?

GET： 用于请求访问已经被URI（统一资源标识符）识别的资源，可以通过URL传参给服务器。

POST：用于传输信息给服务器，主要功能与GET方法类似，但一般推荐使用POST方式。

PUT： 传输文件，报文主体中包含文件内容，保存到对应URI位置。

HEAD： 获得报文首部，与GET方法类似，只是不返回报文主体，一般用于验证URI是否有效。

DELETE：删除文件，与PUT方法相反，删除对应URI位置的文件。

OPTIONS：查询相应URI支持的HTTP方法。

2、GET方法与POST方法的区别

区别一：

get重点在从服务器上获取资源，post重点在向服务器发送数据;

区别二：

get传输数据是通过URL请求，以field（字段）= value的形式，置于URL后，并用"?"连接，多

个请求数据间用"&"连接，如http://127.0.0.1/Test/login.action?name=admin&password=admin，这个过程用户是可见的；

post传输数据通过Http的post机制，将字段与对应值封存在请求实体中发送给服务器，这个过程对用户是不可见的；

区别三：

Get传输的数据量小，因为受URL长度限制，但效率较高；

Post可以传输大量数据，所以上传文件时只能用Post方式；

区别四：

get是不安全的，因为URL是可见的，可能会泄露私密信息，如密码等；

post较get安全性较高；

区别五：

get方式只能支持ASCII字符，向服务器传的中文字符可能会乱码。

post支持标准字符集，可以正确传递中文字符。

3、HTTP请求报文与响应报文格式

请求报文包含三部分：

a、请求行：包含请求方法、URI、HTTP版本信息

b、请求首部字段

c、请求内容实体

响应报文包含三部分：

a、状态行：包含HTTP版本、状态码、状态码的原因短语

b、响应首部字段

c、响应内容实体

4、常见的HTTP相应状态码

返回的状态

1xx：指示信息--表示请求已接收，继续处理

2xx：成功--表示请求已被成功接收、理解、接受

3xx：重定向--要完成请求必须进行更进一步的操作

4xx：客户端错误--请求有语法错误或请求无法实现

5xx：服务器端错误--服务器未能实现合法的请求

200：请求被正常处理

204：请求被受理但没有资源可以返回

206：客户端只是请求资源的一部分，服务器只对请求的部分资源执行GET方法，相应报文中通过Content-Range指定范围的资源。

301：永久性重定向

302：临时重定向

303：与302状态码有相似功能，只是它希望客户端在请求一个URI的时候，能通过GET方法重定向到另一个URI上

304：发送附带条件的请求时，条件不满足时返回，与重定向无关

307：临时重定向，与302类似，只是强制要求使用POST方法

400：请求报文语法有误，服务器无法识别

401：请求需要认证

403：请求的对应资源禁止被访问

404：服务器无法找到对应资源

500：服务器内部错误

503：服务器正忙

5、HTTP1.1版本新特性

a、默认持久连接节省通信量，只要客户端服务端任意一端没有明确提出断开TCP连接，就一直保持连接，可以发送多次HTTP请求

b、管线化，客户端可以同时发出多个HTTP请求，而不用一个个等待响应

c、断点续传原理

6、常见HTTP首部字段

a、通用首部字段（请求报文与响应报文都会使用的首部字段）

Date：创建报文时间

Connection：连接的管理

Cache-Control：缓存的控制

Transfer-Encoding：报文主体的传输编码方式

b、请求首部字段（请求报文会使用的首部字段）

Host：请求资源所在服务器

Accept：可处理的媒体类型

Accept-Charset：可接收的字符集

Accept-Encoding：可接受的内容编码

Accept-Language：可接受的自然语言

c、响应首部字段（响应报文会使用的首部字段）

Accept-Ranges：可接受的字节范围

Location：令客户端重新定向到的URI

Server：HTTP服务器的安装信息

d、实体首部字段（请求报文与响应报文的的实体部分使用的首部字段）

Allow：资源可支持的HTTP方法

Content-Type：实体主类的类型

Content-Encoding：实体主体适用的编码方式

Content-Language：实体主体的自然语言

Content-Length：实体主体的的字节数

Content-Range：实体主体的位置范围，一般用于发出部分请求时使用

7、HTTP的缺点与HTTPS

a、通信使用明文不加密，内容可能被窃听

b、不验证通信方身份，可能遭到伪装

c、无法验证报文完整性，可能被篡改

HTTPS就是HTTP加上加密处理（一般是SSL安全通信线路）+认证+完整性保护

8、HTTP优化

利用负载均衡优化和加速HTTP应用

利用HTTP Cache来优化网站

————————————————————

原文链接：https://blog.csdn.net/cclliii/article/details/82932851