J2EE 课件视频

2020年12月26日 13:39

U1 Spring框架核心

- 软件模块化
  - 把一个程序分割成一些不同的部分， 可以在某种程度上减少它的复杂性。
  - Model-View-Controller将软件用户界面和业务逻辑分离
  - 每一层承担特定的职能， 高层依赖于低层
  - 多层体系结构的优点
    - 结构简单， 便于不同技能的程序员分工负责不同的层
    - 便于测试， 每一层都可以独立测试
    - 变更可控， 可以把代码的变更控制在一层之内， 不会影响其他的层

  Model-View-Controller将软件用户界面和业务逻辑分离，每一层承担特定的职能， 高层依赖于低层
  视图层（View）：提供给用户的操作界面，是程序的外壳
  控制器层（Controller）：提供界面调用的API， 组装和解析视图层的数据（Spring MVC+Restful API）
  服务层（Service）：负责程序的业务逻辑（Bean）
  数据访问层（DAO）：负责获取数据，组装成对象模型（Bean+Redis）
  映射层（Mapper）：负责对象模型与关系数据库的映射（Bean+Mybatis）
  实体层（Entity）：数据的载体
- Spring5.0 概览
  - 大怪兽型应用，Spring Cloud，微服务体系结构应用，服务路由，分布式网络结构
    传统阻塞式Servlet技术栈和非阻塞式Reactive技术栈
    阻塞式Servlet技术栈：服务器端针对每个请求开启一个线程来处理，并行处理，处理完才释放，达到上限后请求被阻塞
    非阻塞式Reactive技术栈：异步回调方式，不用独立线程处理请求，把结果通过消息或回调
    分布式网络结构
- Spring 容器
  - Spring容器来负责创建对象并把对象关联起来提供服务。
  - 容器提供了公共服务
  - 容器依赖于配置信息
  - Spring框架的配置：XML文件，Java代码，注解，SpringBoot
  - Spring Bean对象的注解
    - @Component
      - id - Bean对象的名称， id属性名称理论上可以任意命名， 默认为类名且首字母小写
      - scope - singleton（默认值）， prototype， request， session, global-session
    - @Controller, @Service， @Repository与@Component含义相同， 分别用于标识Controller层， Service层， DAO层的Bean对象
  - Spring拥有两种类型的容器
    - BeanFactory
      - BeanFactory负责读取bean配置信息， 管理bean的加载， 实例化， 维护bean之间的依赖关系， 负责bean的生命周期， 每次获取对象时才会创建对象
      - BeanFactory类结构
        - BeanFactory: 主要的方法是getBean(StringbeanName)， 从容器中返回特定名称的Bean。
        - ListableBeanFactory： 定义了访问容器中Bean 基本信息的若干方法， 如查看Bean 的个数、 获取某一类型Bean 的配置名、 查看容器中是否包括某一Bean 等方法；
        - HierarchicalBeanFactory： 父子级联IoC 容器的接口， 子容器可以通过接口方法访问父容器；
        - ConfigurableBeanFactory： 增强了IoC 容器的可定制性， 它定义了设置类装载器、 属性编辑器、 容器初始化后置处理器等方法；
        - AutowireCapableBeanFactory： 定义了将容器中的Bean 按某种规则（如按名字匹配、 按类型匹配等）进行自动装配的方法；
        - SingletonBeanRegistry： 定义了允许在运行期间向容器注册单实例Bean 的方法；
        - BeanDefinitionRegistry： Spring 配置文件中每一个<bean>节点元素在Spring 容器里都通过一个BeanDefinition 对象表示， 它描述了Bean 的配置信息。 而BeanDefinitionRegistry 接口提供了向容器手工注册BeanDefinition 对象的方法。
    - ApplicationContext
      - ApplicationContext由BeanFactory派生而来， 同时也继承了容器的高级功能，如： MessageSource（国际化资源接口）、 ResourceLoader（资源加载接口）、ApplicationEventPublisher（应用事件发布接口） 等， 提供了更多面向实际应用的功能。 在容器启动时就会创建所有的对象
      - ApplicationContext类结构
        - ApplicationEventPublisher：让容器拥有发布应用上下文事件的功能，包括容器启动事件、关闭事件等。实现了ApplicationListener 事件监听接口的Bean 可以接收到容器事件， 并对事件进行响应处理。在ApplicationContext 抽象实现类AbstractApplicationContext 中，我们可以发现存在一个ApplicationEventMulticaster，它负责保存所有监听器，以便在容器产生上下文事件时通知这些事件监听 者。
        - MessageSource：为应用提供i18n 国际化消息访问的功能；
        - ResourcePatternResolver： 所有ApplicationContext 实现类都实现了类似于PathMatchingResourcePatternResolver的功能，可以通过带前缀的Ant 风格的资源文件路径装载Spring 的配置文件。
        - LifeCycle：该接口是Spring 2.0 加入的，该接口提供了start()和stop()两个方法，主要用于控制异步处理过程。在具体使用时，该接口同时被ApplicationContext 实现及具体Bean 实现，ApplicationContext 会将start/stop 的信息传递给容器中所有实现了该接口的Bean，以达到管理和控制JMX、任务调度等目的。
        - ConfigurableApplicationContext 扩展于ApplicationContext，它新增加了两个主要的方法： refresh()和close()，让ApplicationContext 具有启动、刷新和关闭应用上下文的能力。在应用上下文关闭的情况下调用refresh()即可启动应用上下文，在已经启动的状态下，调用 refresh()则清除缓存并重新装载配置信息，而调用close()则可关闭应用上下文。这些接口方法为容器的控制管理带来了便利.

- Spring Bean的生命周期从创建容器开始， 到容器销毁Bean为止。
  - Bean级生命周期接口
    - BeanNameAware
    - BeanFactoryAware
    - ApplicationContextAware
    - IntializingBean
      - void afterPropertiesSet() throws Exception；属性赋值完成之后调用
    - DisposableBean
      - void destroy() throws Exception；关闭容器时调用

    只影响一个Bean的接口
  - 容器级生命周期接口
    - InstantiationAwareBeanPostProcessorAdapter
      - Object postProcessBeforeInstantiation (Class<?> beanClass, String beanName) throws BeansException；在Bean对象实例化前调用
      - boolean postProcessAfterInstantiation(Object bean, String beanName) throws BeansException；在Bean对象实例化后调用
      - PropertyValues postProcessPropertyValues(PropertyValues pvs, PropertyDescriptor[] pds, Object bean, String beanName) throws BeansException；在（通过配置） 设置某个属性前调用
    - BeanPostProcessor
      - Object postProcessBeforeInitialization(Object o, String s) throws BeansException;实例化完成前
      - Object postProcessAfterInitialization(Object o, String s) throws BeansException;全部实例化完成以后调用该方法

    影响多个Bean的接口
  - Bean的生命周期
  - Aware

BeanNameAware	BeanFactoryAware	ApplicationContext Aware
void setBeanName(String beanName) 待对象实例化并设置属性之后调用该方法 设置BeanName	viod setBeanFactory (BeanFactory var1) throws BeansException 待调用setBeanName之后调用该方法设置BeanFactory	void setApplicationContext (ApplicationContext context) throws BeansException 获得ApplicationContext

  -
- 控制反转（IoC）
- 控制反转是指Bean对象之间的依赖不由它们自己管理， 而是由Spring容器负责管理对象之间的依赖
- Spring容器采用依赖注入（DI） 的方式实现控制反转：XML，JAVA，注解
  - 注解方式：@Autowired时， 首先在容器中查询对应类型的bean
    - 如果查询结果刚好为一个， 就将该bean装配给@Autowired指定的数据
    - 如果查询的结果不止一个， 那么@Autowired会根据变量的名称来查找。
  - @Autowired 标注在 Setter 方法上，构造方法上，属性上

U2 开发环境介绍

- IDEA
- MAVEN

  Apache 下的一个纯 Java 开发的开源项目。基于项目对象模型（缩写： POM）管理一个项目的构建、依赖、报告和文档等步骤。
  - Maven是一个构建工具，实现自动化构建。
  - Maven是跨平台的，对外提供一致的操作接口。
  - Maven是一个依赖管理工具和项目信息管理工具。它还提供了中央仓库,能帮我们自动下载构件。
  - Maven是一个标准，对于项目目录结构、测试用例命名方式等内容都有既定的规则。
  - Maven提倡使用一个共同的标准目录结构
- Maven POM
  - POM(Project Object Model，项目对象模型) 是一个XML文件，包含了项目的基本信息，用于描述项目如何构建，声明项目依赖，等等。
  - Maven 会在当前目录中查找 并读取POM文件，获取所需的配置信息，然后执行目标。
  - POM 中可以指定以下配置：
    - 项目依赖
    - 插件
    - 执行目标
    - 项目构建 profile
    - 项目版本
    - 项目开发者列表
    - 相关邮件列表信息
- Maven 的三个标准生命周期：
  - clean：项目构建前的清理工作
  - default(或 build)：构建的核心部分(编译，测试，打包，安装，部署等等)
    - validate 验证项目是否正确且所有必须信息是可用的
    - compile 源代码编译在此阶段完成
    - test 使用适当的单元测试框架（例如JUnit）运行测试。
    - package 创建JAR/WAR包如在 pom.xml 中定义提及的包
    - verify 对集成测试的结果进行检查，以保证质量达标
    - install 安装打包的项目到本地仓库，以供其他项目使用
    - deploy 拷贝最终的工程包到远程仓库中，以共享给其他开发人员和工程
  - site：生成项目报告，发布站点

- 父（Super） POM
  - 父（Super） POM是 Maven 默认的 POM。所有的 POM 都继承自一个父 POM（无论是否显式定义了这个父 POM）。父 POM 包含了一些可以被继承的默认设置
  - 可以用Show Effective POM看到最终有效的POM定义
- Maven 插件

  Maven 实际上是一个依赖插件执行的框架，每个任务实际上是由插件完成。
  - clean 构建之后清理目标文件。删除目标目录。
  - compiler 编译 Java 源文件。
  - surefile 运行 JUnit 单元测试。创建测试报告。
  - jar 从当前工程中构建 JAR 文件。
  - war 从当前工程中构建 WAR 文件。
  - javadoc 为工程生成 Javadoc。
  - antrun 从构建过程的任意一个阶段中运行一个 ant 任务的集合。

U3 SpringBoot

- SpringBoot的作用的是方便开发独立的应用程序
  - 内嵌Tomcat、 Jetty或Undertow
  - 采用Starter POM简化Maven的配置
  - 大量采用约定简化Spring的配置
  - 提供产品级的运行监控功能
- Starter

  没有Starter之前：在Maven中引入使用的库；引入使用的库所依赖的库；在xxx.xml中配置一些属性信息；反复的调试直到可以正常运行

  有了Starter：只需要引入一个Starter；starter会把所有用到的依赖都给包含进来，避免了开发者自己去引入依赖所带来的麻烦
- SpringBoot配置
  - @SpringBootApplication是一个复合注解

    SpringBoot使用一个全局的配置文件application.properties或者application.yml(或者是yaml)

    作用是修改SpringBoot自动配置的默认值；
- SpringBoot Actuator：健康检查；审计；统计；监控

U5 Web系统基础

- Web访问过程：统一资源定位系统（Uniform Resource Locator）是互联网上用于指定信息位置的表示方法。
- HTTP协议：超文本传输协议是一种用于分布式、协作式和超媒体信息系统的应用层协议。
  - 请求方法：OPTIONS，GET，POST，PUT，DELETE，CONNECT，TRACE
  - 返回状态码：200 OK，400 Bad Request，401 Unauthorized，403 Forbidden，404 Not Found，500 Internal Server Error ，503 Server Unavailable
- HTTPS协议：基于HTTP协议，通过SSL或TLS加密传输数据、验证对方身份以及数据完整性保护
  - 内容加密：采用混合加密技术，中间者无法直接查看明文内容
  - 验证身份：通过证书认证客户端访问的是自己的服务器
  - 保护数据完整性：防止传输的内容被中间人冒充或者篡改
- Servlet：运行在 Web 服务器或应用服务器上的程序，它可以收集来自网页表单的用户输入，呈现来自数据库的记录，动态生成网页。

  单实例，多线程的并发处理机制，通过线程池管理多线程
  - 降低资源消耗-->重复利用已经创建的线程
  - 提高响应速度-->当请求到达时而线程池中有空闲线程时候，可以直接从线程池获取线程,而不需要创建新的线程
  - 提高线程的可管理性-->使用线程池可以统一分配,调优和监控,例如可以根据系统的承受能力,调增线程池中工作线程的数目
- AJAX：利用JavaScript与服务器进行少量数据交换。 ajax可以使网页实现异步更新。即在不重新加载整个网页的情况下，对网页的某部分进行更新。
- 跨域访问问题（CORS）

  允许浏览器向跨源服务器，发出HTTPRequest请求。
  - 预检请求：浏览器先询问服务器，当前网页所在的域名是否在服务器的许可名单之中，以及可以使用哪些HTTP动词和头信息字段。只有得到肯定答复，浏览器才会发出正式的HTTPRequest请求，否则就报错。
  - 服务器收到"预检"请求以后，检查了Origin、 Access-Control-Request-Method和Access-Control-Request-Headers字段以后，确认允许跨源请求，就可以做出回应。
  - 服务器通过了"预检"请求，以后每次浏览器正常的CORS请求，会有一个Origin头信息字段。服务器的回应，也都会有一个Access-Control-Allow-Origin头信息字段。

U6 SpringMVC和RestFul API

- SpringMVC：Spring MVC中的Controller负责接收HTTP Request：交给View生成HTML页面 （传统的方式）；直接把数据写入到Http Response中（Restful）
- RestFul API

  REST的最基本特征是面向资源
  - Resource – 资源：资源是构成应用系统的所有的基本概念
  - Representational – 表述：REST的资源可以用XML， JSON，纯文本等多种方式表述
  - State Transfer – 状态转移：客户端通过HTTP的GET/POST/PUT/DELETE/PATCH对资源进行操作，实现资源的状态转移
    - POST 创建一个新的资源并返回新资源
    - GET 查询特定资源
    - PUT 修改资源的全部属性并返回修改后的资源
    - PATCH 修改资源的特定属性并返回修改后的资源
    - DELETE 删除资源
  - RestFul Controller的注解

    @RestController：与@Controller标签相同，用于标注在类定义前面，使得类会被认定为Controller对象；用于告知Spring容器，该类所有方法的返回值需要以JSON格式写到Response的Body内。
  - SpringMVC的HTTP请求方法注解
    - @GetMapping POST
    - @PostMapping GET
    - @PutMapping PUT

        - @PatchMapping PATCH
        - @DeleteMapping DELETE
        - @RequestMapping 可用于以上五种请求，需在method属性中指定

    JSON（JavaScript Object Notation）
- WebMvcConfigurer：跨域问题的解决方案
- SpringBootTest

    使用测试驱动的开发方式开发系统
    - 创建一个Bean对象
    - 增加一些方法
    - 写这些方法的测试方法
    - 验证所写的方法能通过测试

    测试的主要方法：白盒测试，黑盒测试

    不同层次的测试：界面测试，集成测试，单元测试
    - 使用测试驱动的开发方式开发系统，黑盒测试（测试代码功能，输如输出），白盒测试（测试代码，Jacoco，覆盖面）

      不同层次的测试：Controller层单元测试，Dao层单元测试与Service层单元测试

      单元测试，切片测试，集成测试，界面测试。期末集成测试

      单元测试：最基本的模型对象，pojo，VO

      切片测试，集成测试：切片测试可把某一部分切开

U7 Spring的AOP

- AOP是什么：面向方面编程

    术语：
    - Advise 通知：要做什么，什么时候做
    - JointPoint 连接点：系统中要插入的点
    - PointCut 切点：一个或多个JointPoint插入连接点的地方
    - Aspect 切面：前三个元素合起来
    - Target 对象：要掺入的对象

    两种方式：AspectJ，现有框架

    五种Advice
    - @Before – 在Joint Point执行之前
    - @After – 在Joint Point执行之后，无论是否成功执行
    - @AfterReturning – 在Joint Point成功执行之后
    - @AfterThrowing – 在Joint Point抛出异常之后
    - @Around – 在Joint Point运行之前和之后
- Spring的AOP实现机制：动态代理机制实现AOP，动态代理对象根据Aspect定义掺入
- 定义Pointcut
    - args() Join point是方法，且参数是某些特定类型
    - @args() Join point是方法，且其参数用特定注解标注
    - execution() Join point是方法，其方法名称满足特定条件
    - @annotation Joint point用特定注解标注
- 定义Aspect：@Aspect
- AOP：权限日志紧密耦合到系统；解决耦合，定义到独立地方，编译时耦合

    定义@Aspect定义切面，其中@Pointcut定义Controller层切点，条件在属性，通常注解放方法前面，方法为空，只为了定义切点

    Advise是一组方法，通过注解定义时间，内容为要做的事情，@Before，@After，@AfterReturning，@AfterThrowing，@Aroud

    切点定义：args（）方法的参数要特定的类型，@args（）方法的参数要加特定的注解，execution（）方法本身符合一定条件，@annotation（）方法前面加了特定注解

U8 对象关系映射、 MyBatis和数据库

- 对象关系映射
    - Hibernate基本上可以自动生成。其对数据库结构提供了较为完整的封装
        - 开发效率上Hibernate 比较快。
        - Hibernate自动生成的sql效果不理想
    - MyBatis是一个半自动化的对象-关系模型持久化框架。
        - 采用XML和标注把数据库记录映射成为Java对象，把JDBC、参数设置和结果处理代码从工程中移除。
        - MyBatis框架是以sql的开发方式，可以进行细粒度的优化 。
    - 一对一的关联映射，一对多的关联映射，多对多的关联映射
- MyBatis：MyBatis-Spring-Boot-Starter
    - 自动识别已存在的DataSource
    - 创建并登记一个SqlSessionFactory的对象
    - 用SqlSessionFactory创建并登记一个SqlSessionTemplate的对象
    - 自动扫描并创建Mapper对象，将它们与SqlSessionTemplate对象关联，并登记到Spring上下文中，以备将来注入带Bean中
- MyBatis的映射标记
    - insert：用于映射INSERT SQL语句
    - update：用于映射UPDATE SQL语句
    - delete：用于映射DELETE SQL语句
    - select：用于映射SELECT SQL语句
    - resultMap：用于把数据库的结果集映射成对象
    - sql: 可重用的SQL代码片段
    - parameter：用#{}来表示SQL中的参数

ResultMap：MyBatis 会自动创建一个 ResultMap，基于属性名来映射记录的列到JavaBean 的属性上。列名和属性名没有精确匹配，可以在 SELECT 语句中对列使用别名来匹配对象属性

- 动态sql：if，choose（when，otherwise），trim（where，set），foreach
- 配置信息

- MyBatis的事务
  - 事务：@Transaction
  - 事务传播设置
  - 事务并发的问题
    - 脏读：事务A读取了事务B更新的数据，然后B回滚操作，那么A读取到的数据是脏数据
    - 不可重复读：事务 A 多次读取同一数据，事务 B 在事务A多次读取的过程中，对数据作了更新并提交，导致事务A多次读取同一数据时，结果不一致。
    - 幻读：系统管理员A统计数据库中所有订单的数量，但是用户就在这个时候新提交了一个订单，当系统管理员A统计后发现还有一条订单没有统计，就好像发生了幻觉一样，这就叫幻读。
  - 事务隔离级别：默认使用MySQL（READ_COMMITED）

事务隔离级别	脏读	不可重复读	幻读
读未提交（read-uncommitted）	是	是	是
不可重复读（read-committed）	否	是	是
可重复读（repeatable-read）	否	否	是
串行化（serializable）	否	否	否

  - MVCC多版本并发控制
    - MVCC通过保存数据在某个时间点的快照来实现读的并发。这意味着一个事务无论运行多长时间，在同一个事务里能够看到数据一致的视图。根据事务开始的时间不同，同时也意味着在同一个时刻不同事务看到的相同表里的数据可能是不同的。
    - 每行数据都存在一个版本，每次数据更新时都更新该版本。
    - 修改时Copy出当前版本随意修改，各个事务之间无干扰。
    - 保存时比较版本号，如果成功（commit），则覆盖原记录；失败则放弃copy（rollback）
  - MVCC下InnoDB的增删查改的工作原理
    - 插入数据（insert）：记录的版本号即当前事务的版本号
    - 在更新操作的时候，采用的是先标记旧的那行记录为已删除，并且删除版本号是事务版本号，然后插入一行新的记录的方式。
    - 删除操作的时候，就把事务版本号作为删除版本号。
    - 查询操作，符合以下两个条件的记录才能被事务查询出来：
      - 删除版本号未指定或者大于当前事务版本号，即查询事务开启后确保读取的行未被删除。（即上述事务id为2的事务查询时，依然能读取到事务id为3所删除的数据行）
      - 创建版本号 小于或者等于 当前事务版本号 ，就是说记录创建是在当前事务中（等于的情况）或者在当前事务启动之前的其他事物进行的insert。

- 查询优化
  - 索引

    MySQL的索引：
    - 主键索引： primary key ： 加速查找+约束（不为空且唯一）
    - 唯一索引： unique：加速查找+约束 （唯一）
    - 联合索引
      - primary key(id,name):联合主键索引
      - unique(id,name):联合唯一索引
      - index(id,name):联合普通索引
    - 全文索引fulltext :用于搜索很长一篇文章的时候，效果最好。

    InnoDB使用的是聚簇索引，将主键组织到一棵B+树中，而行数据就储存在叶子节点上。其中Id作为主索引， Name作为辅助索引
- Innodb和Myisam数据库引擎

U9 缓存机制

- 为什么需要缓存：操作系统在不同层数据传输需要的时间
- 不同类型的缓存技术

  前端JS缓存 只读的数据

  Web服务器/NGINX varnish缓存 固定的数据

  应用服务器/TOMCAT MyBatis和Hibernate缓存 修改不大，常用的数据

  分布式缓存/Redis 多台服务器共享缓存，会修改的数据

  ORM框架Mybatis缓存：

  一级缓存：单个事务；二级缓存：应用服务器上

  查询相同：StatementID相同+SQL相同+参数相同=》<K，V>存到HashMap

  Update后清空缓存

  一级缓存：SqlSession对象种Executor查找本地缓存

  二级缓存：Mapper级别的缓存，Namespace，基于一级缓存，SqlSession Executor前放了装饰器Cacheing Executor查找namespace缓存

  缺点：Mybatis不同服务器有不同缓存，无法集群缓存；不能高效利用内存
- Redis

  Redis特性：支持数据的持久化，支持复杂数据结构，支持分布式部署，性能极高，支持事务

  常用五种类型：

  String，常用删除，更新较慢

  Hash，存对象，可以直接修改某一属性

  List，链表增加和删除

Set，集合，元素不能重复

Sorted List，排序集合，插入后即排序

缓存有效期：缓存时间长

淘汰策略：缓存满

缓存清理方法：定时过期占用资源，惰性过期你内存不友好，定期过期（定时扫描）

Redis：惰性过期+定期过期，随机抽取

缓存淘汰机制：报错，LRU最近最少使用，random，ttl，LFU最近不常使用，新数据不友好，定期衰减

Redis：随机n个排序

缓存雪崩和穿透：大批数据同时过期，每个数据增加随机值；缓存不存在，布隆过滤器/缓存空值

U10 消息的发送与接收

- 消息服务器作用：
  - 同步请求，异步处理
  - 服务中间栈，系统解耦
  - 削峰填谷
  - 提升性能，多个服务接受消息并处理
- 消息服务器概念：
  - TOPIC：主题，相当于逻辑地址
  - 生产者：发送消息，向TOPIC发送消息
  - 消费者：处理消息，从TOPIC获取消息
  - 生产者组-消费者组：事务类生产者
- 消息的发送与接收：
  - 同步消息：发送后处于等待状态，等待接收方响应
  - 异步消息：发送方发送完后不用等待接收方结果，提供回调函数，接收方消费成功后回调
  - 单向消息：不需要回调，发送后不管，写日志
- 消费类型：
  - 拉取型消费：消费者主动从服务器获取消息，获取后消费
  - 推送型消费：服务器推送消息给消费者（内部定时拉取实现）
  - 顺序消费：消息消费时按照发送顺序消费，只有一个消息队列，只有一个消费者
  - 并行消费：消息由多个消费者并行消费，不保证先发送的先消费
- 特殊消息：
  - 延时消息：生产者发送消息后，消费者特定时间后消费消息；不可指定任何时间点来消费，时间窗口有选择点；利用重发机制实现
  - 事务消息：发送消息和数据库事务关联到一起，在一个事务中发送消息时分为多个阶段：prepare消息，事务成功后提交commit动作，服务器将commit状态消息放入队列；通过回调函数查询生产者事务状况
- RocketMQ架构：
  - 生产者：生产者产生消息，通过Name Server获取所有Broker的topic；根据负载平衡原则选择一个Name Server结点保持长连接，定时查询topic，若Name Server不可用自动选择下一个；每个生产者跟关联的所有的组的broker产生连接保持心跳，每次发送成功后选择另一个Broker发送为了负载均衡
  - 消费者：消费者从Name Server上获取topic，保持长连接获取新的Topic，挂掉后选择下一个Name Server；获取Broker的topic路由信息，发送pull请求获取消息数据；单个消费者跟所有Broker保持长连接心跳，失去心跳后，跟同组其他消费者发送通知，将消息发送重发给同组的消费者
  - Name Server：注册中心；可架设多个；多个NameServer间互不通信，每台Broker设定所属Name Server，Name Server无法知道存在，Broker与Name Server保持连接心跳
  - Broker：负责消息存储，处理，分发；配置多个Broker，可配主从；主Broker之间隔离，通过Name Server协调；Broker启动时在NameServer注册，定时发送Topic路由信息，集中在NameServer上
  - Topic：消息队列模型中消息的逻辑分类，一个topic可以分布在多个Broker上，一个Broker上的topic子集称作分片，分片中消息的物理存储结构是队列，可设置每个分片多少个队列，队列是基本的消息处理方式
    Broker创建时就创建队列，创建多个队列的原因：队列是消息资源分配的最基本的资源；集群消费情况下topic中的队列平均分给多个消费者，需注意消费者小于队列数量，若大于则有没有分到队列；广播消费情况下，消费者分配所有队列
  - 消息：是消息队列中的载体，一个消息被分发给一个topic，topic相当于地址，物理上发给topic的一个队列中；消息可以指定topic；topic可以指定tag，用作过滤

启动顺序：最先启动NameServer，然后启动Broker选定注册到哪个NameServer上，然后启动生产者消费者；生产者启动后连到NameServer上获取Broker，轮流指定Broker；消费者启动后连到NameServer上得到Broker的路由，订阅Topic，分配Broker队列

U11 SPRING 响应式编程与WebFLUX

Servlet并发处理多请求：线程池

WebFlux并发处理多请求：事件循环

订阅/发布模式：订阅发布模式定义了一种一对多的依赖关系，让多个订阅者对象同时监听某一个主题对象。这个主题对象在自身状态变化时，会通知所有订阅者对象，使它们能够自动更新自己的状态。

Flux和Mono：Reactor 中的两个基本概念。Flux 表示的是包含 0 到 N 个元素的异步序列。在该序列中可以包含三种不同类型的消息通知：正常的包含元素的消息、序列结束的消息和序列出错的消息。当消息通知产生时，订阅者中对应的方法 onNext(), onComplete()和 onError()会被调用。Mono 表示的是包含 0 或者 1 个元素的异步序列。该序列中同样可以包含与 Flux 相同的三种类型的消息通知。Flux 和 Mono 之间可以进行转换。对一个 Flux 序列进行计数操作，得到的结果是一个 Mono对象。把两个 Mono 序列合并在一起，得到的是一个 Flux 对象。

U12 SPRING CLOUD GATEWAY

- 简介
  - API网关：通常我们如果有一个服务，会部署到多台服务器上，这些微服务如果都暴露给客户，是非常难以管理的，我们系统需要有一个唯一的出口， API网关是一个服务，是系统的唯一出口。 API网关封装了系统内部的微服务，为客户端提供一个定制的API。客户端只需要调用网关接口，就可以调用到实际的微服务，实际的服务对客户不可见，并且容易扩展服务。
  - SpringCloud Gateway是基于WebFlux框架实现的，而WebFlux框架底层则使用了高性能的Reactor模式通信框架Netty。
  - 不仅提供统一的路由方式，并且基于 Filter 链的方式提供了网关基本的功能
- 架构

SpringCloud Gateway 使用的Webflux中的reactor-netty响应式编程组件，底层使用了Netty通讯框架。

- 处理流程

  客户端向 Spring Cloud Gateway 发出请求。然后在 Gateway Handler Mapping 中找到与请求相匹配的路由，将其发送到 Gateway Web Handler。 Handler 再通过指定的过滤器链来将请求发送到我们实际的服务执行业务逻辑，然后返回。过滤器之间用两种颜色分开是因为过滤器可能会在发送代理请求之前（“pre”）或之后（“post”）执行业务逻辑。