# 

# 框架

# mvc思想

MVC并不是一个啥新技术，它就是一种设计模式或者说是一种编程思想，使用MVC的目的是将M和V的实现代码分离。

M是指模型层也就是业务逻辑层，V是指视图层也就是用户界面，C则是控制层也就是控制器。它是在上世纪八十年代发明的一种软件设计模式，至今已被广泛使用

## 1视图

刚刚提到视图，视图呢详细说的话就是是用户看到并与之交互的界面（它可以包括一些可以显示数据信息的页面，或者展示形式。例如jsp，html，asp，php）

MVC一个大的好处是它能为你的应用程序处理很多不同的视图。在视图中其实没有真正的处理发生，不管这些数据是联机存储的还是一个雇员列表，作为视图来讲，它只是作为一种输出数据并允许用户操纵的方式。

## 2模型

还有模型，模型表示企业数据和业务规则（可以说就是后端接口，用于业务处理）。在MVC的三个部件中，模型拥有最多的处理任务。被模型返回的数据是中立的，就是说模型与数据格式无关，这样一个模型能为多个视图提供数据。由于应用于模型的代码只需写一次就可以被多个视图重用，所以减少了代码的重复性。

3控制器

最后还有个控制器，控制器接受用户的输入并调用模型和视图去完成用户的需求（接受客户发送的请求，根据请求调用所对应的接口，然后模型业务处理后返回的数据，由控制器决定调用那个View展示）

## 总结

通过我们上面说的就可以了解到mvc的整个处理过程，MVC的处理过程就是，首先控制器接收用户的请求，并决定应该调用哪个模型来进行处理，然后模型用业务逻辑来处理用户的请求并返回数据，最后控制器用相应的视图格式化模型返回的数据，并通过表示层呈现给用户。

## Mvc缺点

MVC的缺点是由于它没有明确的定义，所以完全理解MVC并不是很容易。使用MVC需要精心的计划，由于它的内部原理比较复杂，所以需要花费一些时间去思考。

根据我个人经验，由于我们将一个应用程序分成了三个部件，所以使用MVC同时也意味着你将要管理比以前更多的文件，这一点是显而易见的。这样好像我们的工作量增加了，但是请记住这比起它所能带给我们的好处是不值一提。

# Model

model层就是写数据库的增删改查操作是跟数据库打交道

我用过 的model层框架有:

Jdbc,mybatis ,hibernate, jpa

## Jdbc

jDBC（Java DataBase Connectivity,java数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问，它由一组用Java语言编写的类和接口组成。JDBC提供了一种基准，据此可以构建更高级的工具和接口，使数据库开发人员能够编写数据库应用程序，但是这个东西的话现在基本没人用，（他问为什么:）jdbc的灵活就比较差，搭建也比较麻烦，没有mybatis搭建方便，也没有mybatis灵活（如果问为啥,就把介绍mybatis）

(如果不问就说下面的)

因为好久都没有用过了现在只能记得它访问数据库的过程：

***访问数据库的过程***

主要分为六大步骤：

1.加载sql驱动 （关键词：class.forName）

2.使用DriverManager获取数据库连接 （关键词：DriverManager.getConnection）

3.使用Connecttion来创建一个Statement对象 Statement对象用来执行SQL语句 （关键词：con.createStatement）

4.执行SQL语句 （关键词：execute）

5.操作结果集 （关键词：while）

6.回收数据库资源 （关键词：close）

## Mybatis

mybatis是一个持久层框架，使用sql语句操作的是数据库，底层都是JDBC。

MyBatis 是支持定制化 SQL、存储过程以及高级映射的优秀的持久

层框架。MyBatis 避免了几乎所有的 JDBC 代码和手工设置参数以

及抽取结果集。MyBatis 使用简单的 XML 或注解来配置和映射基本

实体，将接口和 Java 的普通的 Java对像 映射成数据库中的记录。

mybatis很灵活:

mybatis让程序员将主要精力放在sql上，通过mytabis提供的映射方式，自动生成满足需要的sql语句，不满足的话也可以自己去写sqL

## hibernate

***hibernate 简介：***  
hibernate是一个开源框架，它是对象关联关系映射的框架，它对JDBC做了轻量级的封装，而我们java程序员可以使用面向对象的思想来操纵数据库。  
***hibernate核心接口***  
session：负责被持久化对象CRUD操作  
sessionFactory:负责初始化hibernate，创建session对象  
configuration:负责配置并启动hibernate，创建SessionFactory  
Transaction:负责事物相关的操作  
Query和Criteria接口：负责执行各种数据库查询

***hibernate中session中的常用方法***

save():保存

saveOrUpdate()保存或者修改

delete()删除

update()修改

get()根据唯一标示获取对象 立即加载

load()根据唯一标示获得对象 ，延迟加载

***hibernate运行原理***

1.通过configuration加载了hibernate.cfg.xml文件。

2.加载后创建sessionFactory.(sessionFactory是线程安全的)

3.然后通过sessionFactory获得session。（创建会话，相当于jdbc中的connection 是线程不安全的）

4.通过session操作数据库，最后通过transaction来进行事物的控制.

## Jpa

JPA即Java持久化API，JPA的出现主要是为了简化持久层（Dao层）开发以及整合ORM技术，结束Hibernate、TopLink(套铺赁克)、JDO等ORM框架各自为营的局面。

***Spring Data JPA的核心接口***

l Repository(瑞胖丝特瑞): 所有接口的父接口，而且是一个空接口，目的是为了统一所有Repository的类型，让组件扫描的时候能进行识别。

l CrudRepository：是Repository的子接口，提供CRUD(增删改查)的功能。

l Paging(培经)AndSorting(消停)Repository：是CrudRepository的子接口，添加分页和排序的功能。

l JpaRepository：是PagingAndSortingRepository的子接口，增加了一些实用的功能，例如批量操作。

l JpaSpecification(丝拍谁非K鑫)Executor(A克丝Q特儿)：用来做负责查询的接口。

l Specification(丝拍谁非K鑫)：是Spring Data JPA提供的一个查询规范，要做复杂的查询，只需围绕这个规范来设置查询条件即可。

***JPA中常用的注解***

@Entity【嗯忒忒】 标识当前类为实体类

@Table 配置与实体类相对应的数据库表信息(如表名)

@Id 标识当前字段为主键

@Generated(杰呢瑞忒德)Value 配置主键生成策略

@Column(靠乐木) 配置属性对应的数据库字段信息(如字段名)

@ManyToOne【美图玩】 配置一对多关系

@ManyToMany【没图没】 配置多对多关系

@Transient这个注解表示忽略这个属性，那么建表的时候就不会在数据库表中生成一个对应的字段了！

# controller

## servlet

Servlet是sun公司提供的一门用于开发动态web资源的技术。  
Sun公司在其API中提供了一个servlet接口，用户若想用发一个动态web资源(即开发一个Java程序向浏览器输出数据)，需要完成以下2个步骤：  
　　1、编写一个Java类，实现servlet接口。  
　　2、把开发好的Java类部署到web服务器中。  
　　按照一种约定俗成的称呼习惯，通常我们也把实现了servlet接口的java程序，称之为Servlet

Servlet的运行过程

Servlet程序是由WEB服务器调用，web服务器收到客户端的Servlet访问请求后：  
　　①Web服务器首先检查是否已经装载并创建了该Servlet的实例对象。如果是，则直接执行第④步，否则，执行第②步。  
　　②装载并创建该Servlet的一个实例对象。   
　　③调用Servlet实例对象的init()方法。  
　　④创建一个用于封装HTTP请求消息的HttpServletRequest对象和一个代表HTTP响应消息的HttpServletResponse对象，然后调用Servlet的service()方法并将请求和响应对象作为参数传递进去。  
　　⑤WEB应用程序被停止或重新启动之前，Servlet引擎将卸载Servlet，并在卸载之前调用Servlet的destroy()方法。

 Servlet的优点

1、  是mvc的基础，其他的框架比如struts1，struts2，webwork都是从servlet基础上发展过来的。所以掌握servlet是掌握mvc的关键。

2、  Servlet把最底层的api暴漏给程序员，使程序员更能清楚的了解mvc的各个特点。

3、  程序员可以对servlet进行封装。Struts2就是从servlet中封装以后得到的结果。

4、  市场上任何一个mvc的框架都是servlet发展过来的，所以要想学好struts2这个框架，了解servlet的运行机制很关键。

2   Servlet的缺点

1、  每写一个servlet在web.xml中都要做相应的配置。如果有多很servlet，会

导致web.xml内容过于繁多。

2、  这样的结构不利于分组开发。

3、  在servlet中，doGet方法和doPost方法有HttpServletRequest和HttpServletResponse参数。这两个参数与容器相关，如果想在servlet中作单元测试，则必须初始化这两个参数。

4、  如果一个servlet中有很多个方法，则必须采用传递参数的形式，分解到每一个方法中。

## springmvc

在web模型中，MVC是一种很流行的框架，通过把Model，View，Controller分离，把较为复杂的web应用分成逻辑清晰的几部分，是为了简化开发，减少出错。还是为了组内开发人员之间的配合。总之就是一种分层工作的办法。

springMVC，是spring的一个子框架，当然拥有spring的特性，如依赖注入，正因为这一点，通过简单的配置，就能够让SpringMVC框架和Spring容器一起使用。

***springMVC的运行原理***

1. 客户端请求提交到DispatcherServlet(第四排客车Servlet)

   2. 由DispatcherServlet控制器查询一个或多个HandlerMapping(憨的了卖屏)，找到处理请求的handler(憨的了)。

   3. DispatcherServlet将请求转发给到handler。

   4. 通过HandlerAdapter(憨的了额搭铺特)执行handler调用业务逻辑处理后，返回ModelAndView。

   5. DispatcherServlet查询一个或多个ViewResolver()视图解析器，将model(查询得到的数据)渲染到views(前端页面)。

   6.将views结果显示到客户端。

***SpringMVC框架的优点***：

1）SpringMVC框架提供了一整套完善的组件。

不同于以前框架只提供接口，然后让使用者自己根据接口写实现类来实现功能，SpringMVC提供了一整套完善的组件，可以直接使用。这就相当于以前的框架只给你提供设计图纸，你需要根据设计图纸自己把工具造出来，而现在SpringMVC直接给你一套现成的工具箱，里面有你用得到的各种工具。

而且使用SpringMVC框架提供这些工具也巨简单，通过一些简单的注解就搞定了，不需要编写繁琐冗长的代码

2）SpringMVC是以强大的Spring容器为基础的框架。

我们都使用过Spring容器，那你一定知道Spring容器的对象托管功能有多么的好用。它不需要你自己去创建和释放对象，Spring容器帮你全搞定! 同样地，能够放入Spring容器里面的这个SpringMVC工具箱，里面各种工具类的管理也是不需要操心的。而这省心省力的优点，是其他框架都没有的

3）框架的配置简单又不失灵活性。

SpringMVC框架的常用配置项就已经涵盖了项目中80%的配置需求，简单的项目甚至可以做到零配置，拿过来就用。

4）代码的可重用性很高。

翻译成大白话就是，一个成熟的，上线稳定运行过的SpringMVC项目，经过简单修改，甚至不用修改，就可以作为另一个新项目开发的基础。这样能在开发新项目时省很多力气。

5）可扩展性好。

如果SpringMVC框架提供的工具不能100%满足你的需求，你需要定制开发，SpringMVC框架同样也提供了相应的接口，可以进行工具的升级。这样你在遇到强力精英怪的时候，照样能用SpringMVC框架打败它。

springboot

Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化Spring应用初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。Spring Boot其实就是一个整合很多可插拔的组件（框架），内嵌了使用工具（比如内嵌了Tomcat、Jetty等），方便开发人员快速搭建和开发的一个框架。

Spring Boot使创建独立的、基于生产级Spring的应用程序变得很容易，您可以“直接运行”这些应用程序。 我们对Spring平台和第三方库有自己的见解，这样您就可以轻松入门了。大多数Spring引导应用程序只需要很少的Spring配置

创建Spring boot的两种方式：

1、spring boot官网：<https://start.spring.io/>

2、使用Idea创建spring boot项目

springboot特点：

1 快速搭建spring 框架 2 约定大于配置 3 开箱即用 4 嵌入了web容器

Spring 最初最核心的两大核心功能 Spring Ioc 和 Spring Aop 成就了 Spring，

Spring 在这两大核心的功能上不断的发展，才有了 Spring 事务、Spring Mvc 等

一系列伟大的产品，最终成就了 Spring 帝国，到了后期 Spring 几乎可以解决企

业开发中的所有问题。

Spring Boot 是在强大的 Spring 帝国生态基础上面发展而来，发明 Spring Boot 不

是为了取代 Spring ,是为了让人们更容易的使用 Spring

# springcloud分布式框架

### spring-cloud-eureka：

spring-cloud-eureka是spring-cloud-nettfix微服务套件中的一部分， 它基于nettfix-

eureka做了二次封装，主要负责微服务架构中的服务治理功能。

服务治理一般都会有两个功能：服务注册、服务发现。通常会有一个注册中心，每个服务单

元向注册中心登记自己信息，比如提供的服务，ip, 端口以及一些附近加信息等。注册中心

会将新的服务实例发送给其它依赖此服务的实例。

核心注解@EnableEurekaClient

### springcloud-Feign:

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也是运行在消费者端的，使用 Ribbon 进行负载均衡，所以 Feign 直接内置了 Ribbon。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。

核心注解：@EnableFeignClients//声明fegin

### Spring Cloud Ribbon:

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。核心：

启动类中注入bean

@Bean

@LoadBalanced //声明 ribbon （并对resttemmplate进行负载均衡 调用的时候直接写服务名就行 不用写ip和端口）

RestTemplate restTemplate() {

return new RestTemplate();// RestTemplate对httpclient发送请求 进行封装了

}

### Spring Cloud Hystrix：

Hystrix中文是豪猪的意思，在Spring Cloud中防雪崩的利器就是Spring Cloud Hystrix，Spring Cloud Hystrix是基于Netflix Hystrix实现的。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

Hystrix能干啥

l 服务熔断

当Hystrix Command请求后端服务失败数量超过一定比例(默认50%), 断路器会切换到开路状态(Open). 这时所有请求会直接失败而不会发送到后端服务. 断路器保持在开路状态一段时间后(默认5秒), 自动切换到半开路状态(HALF-OPEN).

l 服务降级

Fallback相当于是降级操作, 我们可以实现一个fallback方法, 当请求后端服务出现异常或者服务熔断后的时候, 直接调用本地fallback方法，告知后面的请求服务不可用了，不要再来了。

l 服务容错

l 线程和信号隔离

在Hystrix中, 主要通过线程池来实现资源隔离. 通常在使用的时候我们会根据调用的远程服务划分出多个线程池.比如说，一个服务调用两外两个服务，你如果调用两个服务都用一个线程池，那么如果一个服务卡在哪里，资源没被释放。后面的请求又来了，导致后面的请求都卡在哪里等待，导致你依赖的A服务把你卡在哪里，耗尽了资源，也导致了你另外一个B服务也不可用了。这时如果依赖隔离，某一个服务调用A B两个服务，如果这时我有100个线程可用，我给A服务分配50个，给B服务分配50个，这样就算A服务挂了，我的B服务依然可以用。

l 请求缓存

比如一个请求过来请求我userId=1的数据，你后面的请求也过来请求同样的数据，这时我不会继续走原来的那条请求链路了，而是把第一次请求缓存过了，把第一次的请求结果返回给后面的请求。

l 请求合并

我依赖于某一个服务，我要调用N次，我发了N条请求发然后拿到一堆结果，这时候我们可以把多个请求合并成一个请求，这样我们只需调用一次远程服务，提升了效率。

l 近乎实时的服务监控

### Hystrix 客户端核心：

@EnableHystrix// 熔断 启动类中

对你要熔断的方法

// 此注解标识是要熔断 fallbackMethod 熔断后进行执行的方法

@HystrixCommand(fallbackMethod = "errorMethod",commandProperties = {

@HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds",value = "3000")

}) //对此方法进行熔断

Hystrix 服务端：

启动类

@EnableHystrixDashboard

启动类中

/\* 提供监控平台的路径\*/

@Bean

public ServletRegistrationBean getServlet() {

HystrixMetricsStreamServlet streamServlet = new HystrixMetricsStreamServlet();

ServletRegistrationBean registrationBean = new ServletRegistrationBean(streamServlet);

registrationBean.setLoadOnStartup(1);

registrationBean.addUrlMappings("/hystrix.stream");

registrationBean.setName("HystrixMetricsStreamServlet");

return registrationBean;

### Spring Cloud zull：

Zuul是Netflix开源的微服务网关，他可以和Eureka,Ribbon,Hystrix等组件配合使用。Zuul组件的核心是一系列的过滤器，这些过滤器可以完成以下功能：

身份认证和安全:

识别每一个资源的验证要求，并拒绝那些不符的请求审查与监控：

***动态路由：***

动态将请求路由到不同后端集群

***压力测试：***

逐渐增加指向集群的流量，以了解性能

***负载分配：***

为每一种负载类型分配对应容量，并弃用超出限定值的请求

***静态响应处理：***

边缘位置进行响应，避免转发到内部集群

# shiro 安全框架

* Apache Shiro是一个强大而灵活的开源安全框架，它能够干净利落地处理身份认证、授权、企业会话管理和加密。相比spring security框架更简单灵活，spring security对spring依赖较强。shiro可以实现web系统、c/s、分布式等系统权限管理。

## **功能**

1. 登录认证
2. 权限认证
3. 加密

## 优点

1控制在线人数和同时在线问题

2所有的后端管理框架都可以使用

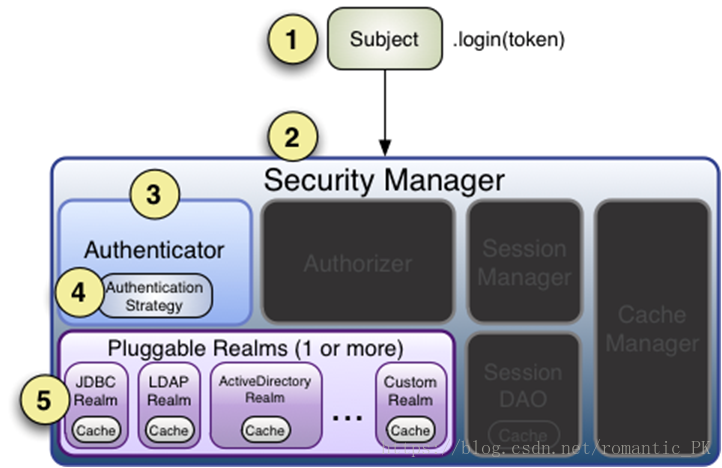
## **shiro的初步理解**

1. 默认基于cookie加session的模式完成登录认证.
2. 也可以重写一些方法和接口改成jwt的token验证.
3. 基于缓存的安全认证框架，他可以使用第三方缓存如redis.

## shiro的组成部分

1. Subject：主体，即当前操作用户
2. 2.(瑟Q弱tei)SecurityManager：安全管理器，它是Shiro框架的核心，典型的Facade模式，Shiro通过SecurityManager来管理内部组件实例，并通过它来提供安全管理的各种服务
3. Realm：领域，充当了Shiro与应用安全数据间的“桥梁”或者“连接器”。也就是说，当对用户执行认证（登录）和授权（访问控制）验证时，Shiro会从应用配置的Realm中查找用户及其权限信息
4. Authenticator(奥森特tei 特r)：认证器，AuthenticationStrategy(死抓特zhei)如果存在多个realm，则按着具体的策略进行登录控制，例如：如果有一个realm成功即可登录、必须所有realm都成功才能登录等
5. Authorizer（奥斯ruai zer）：授权器，决定subject能拥有什么样角色或者权限。
6. SessionManager：session管理器，创建和管理用户session。通过设置这个管理器，shiro可以在任何环境下使用session。
7. (卡诗)CacheManager：缓存管理器，可以减少不必要的后台访问。提高应用效率，增加用户体验。
8. Cryptography（抠铺淘客fei）：Shiro的api大幅度简化java api中繁琐的密码加密。

## **shrio的认证流程**



subject(主体)请求认证，调用subject.login(token)

SecurityManager (安全管理器)执行认证

SecurityManager通过ModularRealmAuthenticator进行认证。

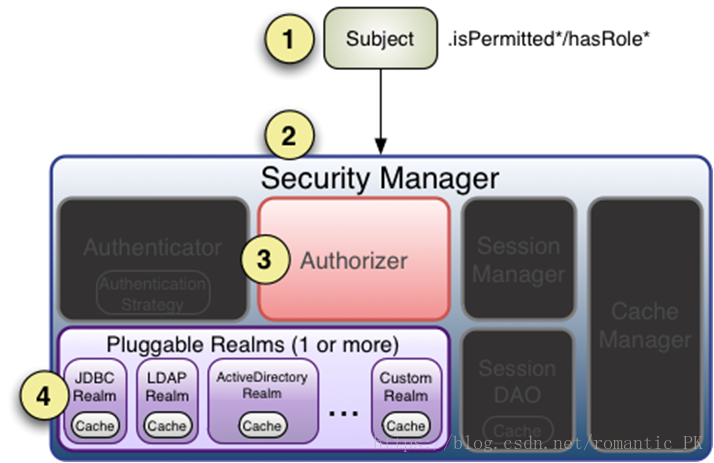
ModularRealmAuthenticator将token传给realm，realm根据token中用户信息从数据库查询用户信息（包括身份和凭证）

realm如果查询不到用户给ModularRealmAuthenticator返回null，ModularRealmAuthenticator抛出异常（用户不存在）

realm如果查询到用户给ModularRealmAuthenticator返回AuthenticationInfo(认证信息)

ModularRealmAuthenticator拿着AuthenticationInfo(认证信息)去进行凭证（密码）比对。如果一致则认证通过，如果不致抛出异常（凭证错误）。

## **Shiro授权流程：**



对subject进行授权，调用方法isPermitted("")或者hasRole("")

SecurityManager执行授权，通过ModularRealmAuthorizer执行授权

ModularRealmAuthorizer执行realm（自定义的CustomRealm）从数据库查询权限数据调用realm的授权方法：doGetAuthorizationInfo

realm从数据库查询权限数据，返回ModularRealmAuthorizer

ModularRealmAuthorizer调用PermissionResolver进行权限串比对

如果比对后，isPermitted中"permission串"在realm查询到权限数据中，说明用户访问permission串有权限，否则没有权限，抛出异常。

# View

我用过的前台框架有：jquery bootstrap layui vue（钩子函数 属性 指令 路由 组件） elementui

## jquery学习总结

　jQuery的基本设计和主要用法，就是"选择某个网页元素，然后对其进行某种操作"。这是它区别于其他函数库的根本特点。  
　　使用jQuery的第一步，往往就是将一个选择表达式，放进构造函数jQuery()（简写为$），然后得到被选中的元素。

***选择表达式可以是CSS选择器：***

 $(document)//选择整个文档对象   
 $('#myId')//选择ID为myId的网页元素   
 $('div.myClass')//选择class为myClass的div元素   
 $('input[name=first]')//选择name属性等于first的input元素

***也可以是jQuery特有的表达式：***

 $('a:first')//选择网页中第一个a元素   
 $('tr:odd')//选择表格的奇数行   
 $('#myForm :input')//选择表单中的input元素   
 $('div:visible') //选择可见的div元素   
 $('div:gt(2)')//选择所有的div元素，除了前三个   
 $('div:animated')//选择当前处于动画状态的div元素

***元素的操作：取值和赋值：***

　操作网页元素，最常见的需求是取得它们的值，或者对它们进行赋值。  
　　jQuery使用同一个函数，来完成取值（getter）和赋值（setter）。到底是取值还是赋值，由函数的参数决定。

 $('h1').html(); //html()没有参数，表示取出h1的值  
 $('h1').html('Hello'); //html()有参数Hello，表示对h1进行赋值   
常见的取值和赋值函数如下：

1. .html() 返回或设置被选元素的内容 (inner HTML)     
2. .text() 取出或设置text内容       
3. .attr() 取出或设置某个属性的值     
4. .width() 取出或设置某个元素的宽度     
5. .height() 取出或设置某个元素的高度   
6. .val() 取出或设置html内容 取出某个表单元素的值   
　　需要注意的是，如果结果集包含多个元素，那么赋值的时候，将对其中所有的元素赋值；取值的时候，则是只取出第一个元素的值（.text()例外，它取出所有元素的text内容）

***元素的操作：移动***

.insertAfter()，把div元素移动p元素后面：

$('div').insertAfter('p');  
.after()，把p元素加到div元素前面：

 $('p').after('div');  
使用这种模式的操作方法，一共有四对

1. .insertAfter()和.after()：在现存元素的外部，从后面插入元素   
2. .insertBefore()和.before()：在现存元素的外部，从前面插入元素   
3. .appendTo()和.append()：在现存元素的内部，从后面插入元素   
4. .prependTo()和.prepend()  ：在现存元素的内部，从前面插入元素

***jQuery主要支持以下事件：***

 .blur() 表单元素失去焦点。   
 .change() 表单元素的值发生变化   
 .click() 鼠标单击   
 .dblclick() 鼠标双击   
 .focus() 表单元素获得焦点   
 .focusin() 子元素获得焦点   
 .focusout() 子元素失去焦点   
 .hover() 同时为mouseenter和mouseleave事件指定处理函数   
 .keydown() 按下键盘（长时间按键，只返回一个事件）   
 .keypress() 按下键盘（长时间按键，将返回多个事件）   
 .keyup() 松开键盘   
 .load() 元素加载完毕   
 .mousedown() 按下鼠标   
 .mouseenter() 鼠标进入（进入子元素不触发）   
 .mouseleave() 鼠标离开（离开子元素不触发）   
 .mousemove() 鼠标在元素内部移动   
 .mouseout() 鼠标离开（离开子元素也触发）   
 .mouseover() 鼠标进入（进入子元素也触发）   
 .mouseup() 松开鼠标   
 .ready() DOM加载完成   
 .resize() 浏览器窗口的大小发生改变   
 .scroll() 滚动条的位置发生变化   
 .select() 用户选中文本框中的内容   
 .submit() 用户递交表单   
 .toggle() 根据鼠标点击的次数，依次运行多个函数   
 .unload()    
 用户离开页面

# Java基础

## 数据类型 （八大数据类型）

1、数值型（6个）

（1-1）、整数型（4个）

字节型 byte 1个字节

短整型 short 2个字节

整型 int 4个字节

长整型 long 8个字节

（1-2）、浮点型（小数 2个）

单精度浮点型 float 4个字节

双精度浮点型 double 8个字节

2、字符型 char 2个字节

3、布尔类型 boolean 值为：true false 1个字节

**引用数据类型（3个）**

数组array 接口interface 类class

## ****容器（list 、set 、map）****

1、List 接口是有序可重复的：

Linkedlist  链表结构：链式存储，读慢，写快；

Arraylist 顺序结构：顺序存储，读快，写改慢；

如果涉及到堆栈，队列等操作，应该考虑用List，对于需要快速插入，删除元素，应该使用LinkedList，如果需要快速随机访问元素，应该使用ArrayList。

****2、Set 接口是无序不可重复：****

****Set是一种不包含重复的元素的Collection，即任意的两个元素e1和e2都有e1.equals(e2)=false，Set最多有一个null元素。需要equals()方法确保对象唯一性；****

**HashSet**: 为快速查找而设计,可以认为是基于**哈希表**的实现。 存入HashSet的元素**必须定义hashCode().**

**LinkedHashSet:** 具有HashSet的查询速度且内部使用链表维护元素的顺序（按插入的顺序或最近最少使用顺序）

TreeSet：保持次序的Set，底层为树结构（红黑树）。使用它可以从Set提取有序的序列。元素必须实现Comparable接口或者在构造TreeSet时传入Comparator参数。

3、Map基于Key-value的存储格式

HashMap  基于散列表的实现（它取代了HashTable, HashTable是过时的类）。

LinkedHashMap 类似与HashMap， 但是迭代遍历时，取得“键值对”的顺序是其插入的顺序或最近最少使用的顺序(如果在构造LiskedHashMap传入参数accessOrder=true)。比HashMap慢一点，但是在迭代访问时更快，因为它使用链表维护内部的次序。

TreeMap 基于红黑树的实现。查看key 或 key-value是，它们会被排序（次序由Comparator或Comparable决定）

## 包装类（封装类、file、date、string、exception、stream）

1、基本数据类型的包装类：

byte-->Byte

Short-->Short

Int-->Integer

long-->Long

float-->Float

double-->Double

boolean-->Boolean

char-->Character

2、File:

文件和目录路径名的抽象表达形式，一个File对象可以代表一个文件或者目录：

* 可以实现获取文件和目录属性等功能
* 可以实现对文件和目录的创建、删除等功能

常用方法有：

* createNewFile()  //有则不创建 返回false，没有则创建 返回true
* delete()             //删除成功 返回true ,失败返回false
* getAbsoluteFile()  //返回绝对路径
* getPath()           //返回相对路径
* getName()          //返回文件名
* isFile()              //是否为文件  boolean
* length              //长度
* mkdir()             //创建单层目录
* mkdirs()           //创建多级目录
* delete()           //只能删除空目录
* list()      //返回一个字符串数组，命名由此抽象路径名表示的目录中的文件和 目录。
* listFiles()         //返回一个抽象路径名数组，表示由该抽象路径名表示的目录 中的文件和目录。

File不能访问文件内容；

1. date /SimpleDateFormat 类:

Date: 是用来表示一个时间、日期的类;

DateFormat :完成字符串和时间对象的转换；  
SimpleDateFormat:完成字符串和时间对象的转换

常用方法：

Date() 获取当前时间

Date(long time) | 获取指定时间戳对应的时间 |

void setTime(long time) | 使用指定时间戳设置一个日期对象 |

long getTime() | 获取一个指定日期对象的时间戳 |

equals(Object obj) | 判断是否与另外一个日期相同 |

boolean before(Date other) | 判断是否在另外一个日期之前 |

boolean after(Date other) | 判断是否在另外一个日期之后 |

SimpleDateFormat: 是一个用来格式化日期的类：

1. 可以将一个Date格式化为指定格式的字符串

2. 可以将一个自定格式的字符串解析为一个Date

3、yyyy(年)、yy(年)、MM(月)、dd(日)、HH(时24)、hh(时12)、mm(分)、ss(秒)

常用方法 | 描述 |

> | SimpleDateFormat(String pattern) | 使用指定的格式来实例化一个对象 |

> | String format(Date date) | 将一个日期格式化为指定的字符串 |

> | Date parse(String str) | 将一个指定格式的字符串解析为一个日期 |

## String类：

所属的jar包是java.lang包

String是一个非常特殊的引用数据类型 可以像基本数据类型一样 创建 赋值

string特性：String的不可变特性；

体现在两个地方：长度及内容

长度-->final修饰的数组 数组长度本身不变 final修饰数组 的地址也不变；

内容-->private修饰的属性 不能在类的外部访问

在String类中包含一个数组

private final char[] value;//存储String中的每一个字符

final最终的不可改变的-->地址不让改变；数组的长度本身不可变；

private 私有的 当前类中才可访问-->数组中的内容也不能改变；

String 常用方法

## Calendar类

> 是一个用来操作日期的类。提供了若干个对一个日期进行操作的方法。

> > 常用静态常量，用来描述一些字段

> > YEAR: 年 > > MONTH: 月（注意：月份从0开始）> > DAY\_OF\_MONTH: 日

> > HOUR: 时（12小时制）> > HOUR\_OF\_DAY: 时（24小时制）> > MINUTE: 分

> > SECOND: 秒

> | 方法 | 描述 |

> getInstance() 获取一个当前日期的Calenadar对象

> get(int filed) 获取指定ID的值

> set(int field, int value) 设置指定ID的值

> set(int year, int month, int date) 设置年、月、日

> set(int year, int month, int date, int hourOfDay, int minute, int day) 设置年、月、日、时、分、秒

> add(int filed, int amount) | 对某个id的值进行增（如果想减，将值修改成负数即可）

> setTime(Date date) 通过一个Date对象，对一个日期进行赋值

> getTime() 通过一个日期对象，获取一个Date对象

> boolean before(Object other) 判断是否在另外一个日期之前

> boolean after(Object other) 判断是否在另外一个日期之后

> boolean equals(Object other) 判断是否和另外一个日期相同

## 枚举

> 是jdk1.5之后新增的特性

> 是一种数据类型。class、interface， 是用关键字\*\*enum\*\*来修饰

> ```java

> 例如：[访问权限修饰符public] enum 枚举名字 {

枚举中的元素

}

> 枚举一般情况下是用来描述一些取值范围有限的数据。例如：星期几

## Random类

> 关于随机数的生成：

> ​ 随机数的生成，是由一个随机种子，带入到一个固定的随机数算法 中，生成一个数字序列。

> ​ 如果随机种子相同，产生的随机数列也相同。

BigInteger / BigDecimal 类

> 用来处理非常大的数字的基本运算

1. Java中运行时异常和非运行时异常的区别
2. 运行时异常:

RuntimeException,非受控异常

程序运行过程中出现的异常，一般情况下都是由于逻辑判断错误出现的异常。

IndexOutOfBoundsException 索引越界异常0

ArrayIndexOutOfBoundsException 数组下标越界异常

StringIndexOutOfBoundsException 字符串下标越界异常

ArithmeticException：数学计算异常

NullPointerException：空指针异常

1. 非运行时异常:

受控异常，在程序编译过程中出现的异常，该异常必须解决才能继续执行。

* + - 1. IOException、Io流异常
      2. FileNotFoundException：文件未找到异常
      3. EOFException：读写文件尾异常
      4. MalformedURLException：URL格式错误异常
      5. SocketException：Socket异常
      6. SQLException sql异常

ParseException 日期格式转换异常

## Error 错误

是在程序运行过程中出现的无法使用代码的解决的错误，我们称之为error；最常见的为：内存溢出outOfMemoryError

# Exception

运行时异常：

在代码运行过程中出现的异常我们称之为运行时异常，也叫做非受控异常

## Stream

一种声明的方式处理数据。

Stream API可以极大提高Java程序员的生产力，让程序员写出高效率、干净、简洁的代码。

这种风格将要处理的元素集合看作一种流， 流在管道中传输， 并且可以在管道的节点上进行处理， 比如筛选， 排序，聚合等。

## 什么是 Stream？

Stream（流）是一个来自数据源的元素队列并支持聚合操作

<strong元素队列< strong="">元素是特定类型的对象，形成一个队列。 Java中的Stream并不会存储元素，而是按需计算。

数据源 流的来源。 可以是集合，数组，I/O channel， 产生器generator 等。

聚合操作 类似SQL语句一样的操作， 比如filter, map, reduce, find, match, sorted等。

和以前的Collection操作不同， Stream操作还有两个基础的特征：

Pipelining: 中间操作都会返回流对象本身。 这样多个操作可以串联成一个管道， 如同流式风格（fluent style）。 这样做可以对操作进行优化， 比如延迟执行(laziness)和短路( short-circuiting)。

内部迭代： 以前对集合遍历都是通过Iterator或者For-Each的方式, 显式的在集合外部进行迭代， 这叫做外部迭代。 Stream提供了内部迭代的方式， 通过访问者模式(Visitor)实现。

## 生成流两种方式：

**stream()** − 为集合创建串行流。

**parallelStream()** − 为集合创建并行流。

forEach:

Stream 提供了新的方法 'forEach' 来迭代流中的每个数据。

Map:

map 方法用于映射每个元素到对应的结果

Filter:

filter 方法用于通过设置的条件过滤出元素

Limit:

limit 方法用于获取指定数量的流

Sorted:

sorted 方法用于对流进行排序

并行（parallel）程序:

parallelStream 是流并行处理程序的代替方法

Collectors:

Collectors 类实现了很多归约操作，例如将流转换成集合和聚合元素。Collectors 可用于返回列表或字符串：

统计:

另外，一些产生统计结果的收集器也非常有用。它们主要用于int、double、long等基本类型上，它们可以用来产生类似如下的统计结果。

## 控制语句 （if for while foreach 迭代器）

## 反射 reflect

3.反射 自定义注解

java反射

Java反射机制

对于任意一个类，都能知道这个类的所有属性和方法；

对于任意一个对象，都能够调用他的任意一个方法和属性，

这种动态获取信息以及动态调用对象方法的功能，

我们称为java语言的反射机制。

1 类.class

2 Class.forName(" 类的全路径")

3 new 对象 对象.getClass();

class对象.getDeclaredField() 获取所有的属性

class对象.getFields() 获取公共的属性

类对象.getDeclaredMethod("方法的名称") 得到一个具体的方法

getAnnotation 获取注解 额内特森 和自定义注解联合使用

//获取方法签名

joinPoint.getSignature()

### 注解 annotation

自定义注解

Java 注解(Annotation)又称 Java 标注,

是 JDK5.0 引入的一种注释机制。

自定义注解（）

关键字

@interface

元注解

@Target(ElementType.METHOD) //声明注解用的地方 （类上 方法上 属性上 ）

@Retention(RetentionPolicy.RUNTIME) //声明注解的生命周期

## jdk 新特性 lamda表达式（接口的新实现方式） stream流操作

2. jdk 1.8新特性

lamdba表达式 （其实就是函数接口的实现 ） jdk 提供哪些函数接口

stream流

1 创建流的方式

list.stream()

Arrays.stream();

Stream.of();

无限流Stream.iterate Stream.generate

2 流的中间操作

list.stream().filter(） 过滤数据

stream.limit() 截断流 指定输出多少条数据

stream.distinct() 去重

stream.skip() 跳过元素 返回一个扔掉了前n个元素的流。若流中元素不足n个，则返回一 个空，与limit（n）互补。

3 结束操作

allMatch--检查是否匹配所有元素

anyMatch--检查是否至少匹配一个元素

noneMatch--检查是否没有匹配的元素

findFirst--返回第一个元素

findAny--返回当前流中的任意元素

count--返回流中元素的总个数

max--返回流中最大值

min--返回流中最小值

## 多线程 thread 安全问题

1.多线程

1 多线程的创建方式

方式一：继承Thread类的方式

方式二：实现Runnable接口的方式

重写run方法 （run方法为多线程的方法体） 启动用start方法

2 线程通常都有五种状态，创建、就绪、运行、阻塞和死亡。

创建状态。在生成线程对象，并没有调用该对象的start方法，这是线程处于创建状态。

就绪状态。当调用了线程对象的start方法之后，该线程就进入了就绪状态，但是此时线程调度程序还没有把该线程设置为当前线程，此时处于就绪状态。在线程运行之后，从等待或者睡眠中回来之后，也会处于就绪状态。

运行状态。线程调度程序将处于就绪状态的线程设置为当前线程，此时线程就进入了运行状态，开始运行run函数当中的代码。

阻塞状态。线程正在运行的时候，被暂停，通常是为了等待某个时间的发生(比如说某项资源就绪)之后再继续运行。sleep,suspend，wait等方法都可以导致线程阻塞。

死亡状态。如果一个线程的run方法执行结束或者调用stop方法后，该线程就会死亡。对于已经死亡的线程，无法再使用start方法令其进入就绪

3 安全问题

当多个线程同时执行，它们之间也是相互抢占资源来执行，抢占资源的过程中会导致出现非法数据，这种现象会出现多线程的并发安全问题。

通过同步代码块的方式将对象锁起来。用synchronized来将代码限制起来

4 什么是死锁？

死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系 统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。是操作系统层面的一个错误，是进程死锁的简称

5 线程和进程的区别？

简而言之，进程是程序运行和资源分配的基本单位，一个程序至少有一个进程，一个进程至少有一个线程。进程在执行过程中拥有独立的内存单元，而多个线程共享内存资源，减少切换次数，从而效率更高。线程是进程的一个实体，是cpu调度和分派的基本单位，是比程序更小的能独立运行的基本单位。同一进程中的多个线程之间可以并发执行。

你们项目中哪些地方用过 （不用行不行）

多线程定时发送邮件 多线程poi导出

做poi导出，我们同样需要使用反射和自定义注解，但不同的是，这时候自定义注解添加的位置成了属性，作用是为了得到需要导出的字段以及相应的数据。

最初，我只是做了一个简单的Excel表，使用XSSFWorkbook，生成sheet页、行、列，随之放入数据，最后写入Excel中，关流即可。这样写并没有什么问题，开发和测试都一切顺利。但是到了上线阶段数据量上去之后就不适用了。

后来我想到了生成多个excel文件多个sheet页并将其做成一个压缩包，这样再多的数据量也不会有事了。我先是尝试了使用流的方式进行操作。首先创建了ServletOutputStrea流和ZipOutputStream流，在此，我们需要确定每个sheet页数据的条数和sheet页的个数，以此来确定excel文件的个数，然后使用HSSFWorkbook通过循环，生成sheet页、行、列，随后放入数据，生成excel，依次写入zip压缩流下载完成。

再之后，需求发生了变化，需要做成多线程的poi导出，（此时先介绍多线程）这时，我之前做的那种思路的zip压缩流就不能再用了，因为它无法加到多线程中去。我换了一个思路，先在服务器上创建了一个临时文件夹，通过多线程去生成多个的excel文件，将其放到临时文件夹中。这些都完成之后，通过FileInputStream流依次读取临时文件夹中的数据，再通过ZipEntry来进行写入，最后下载zip压缩包，关流。至此，多线程poi导出便写完了。

（springboot创建定时器 只需要在启动类上加@EnableScheduling 然后在需要定时的方法上加 @Scheduled(cron = "0 00 08 ? \* \*")）

我们需要在定时任务的执行体里面添加一个分布式锁，目的是为了防止多台服务器集群同时触发我们的定时任务造成的邮件重复问题，这个分布式锁 我们用到了redis中的getset方法，

redis的getset方法redis中string类型下的 它会存进去新的值并且返回我们原来的值，原来没有值的话返回null；

通过getset方法把我们的当前时间作为key存到我们的redis中，并且给设置一个过期时间，之后判断getset的返回值是否为空，不为空那我们就给他直接返回，并且提示一下我们的任务已经执行过了

为空的话我们就先去查询我们数据库中今天过生日的用户，（之后创建一个线程池来发送邮箱 线程池核心线程数根据自己的用户数量和服务器配置来均衡一个中间点，我们这用到了三个核心线程数）。

循环我们的用户数据，创建我们的多线程穿过去几个参数1.我们邮件的标题2.邮件的内容3.邮件接收者，之后再我们的线程执行体中（run方法中）去调用一下我们邮件的工具类，工具类中配置号我们发送方的邮箱号，设置我们的发送方的邮箱类型（smtp.qq.com）配置发件人的邮箱密码 （密码是我们在账号设置中设置的smtp的密匙）启动用statr方法

设置发送人，接收人，标题，内容，这就实现了我们的邮箱发送

# javascript

js又称 javascript

javascript是一个脚本语言 基于对象和事件驱动的脚本语言

所谓事件驱动就是利用事件来触发消息 系统在对相应的消息进行处理的一个机制

js 数据类型分为3种 也就是 基本数据类型,复合数据类型和特殊数据类型

其中 基本数据类型又分为3种、

字符串类型 （string） ，数值类型（number）

布尔类型（boolean） true 和false

复合数据类型分为2种

数组 Array 和对象 object

特殊数据类型也分为2种

空 null 未定义 undefined

当一个数据我们不知道他是什么类型的时候可以用一个方法来检测他是个什么类型的数据这个方法就是 （typeof 定义的变量名）就可以得到这个数据的类型是什么

js原生选择器有

getElementById()、getElementsByName()、getElementsByTagName()

getElementById(id) 返回指定ID的对象 id是唯一标识返回的只有一个

getElementsByName(name) 返回name 属性是name值得元素 因为name可以有多个重复得 所以得到的可能是一个元素 也可能是一个数组

getElementsByTagName(tagname) 返回得是指定标签得元素 也可能有多个

jQuery选择器

基本选择器有 ID选择器、类选择器、标签选择器、通用选择器 还有复合选择器 就是多个选择器联合使用

ID选择器 的语法 $("#lastname") 就是取 id是lastname 的元素

类选择器 的语法 $(".intro")就是取 所有 class="intro" 的元素

标签选择器 的语法 $("p") 所有 <p> 元素

js 中的一些常用事件

事件就是用户对网页或浏览器做出的动作

通常来说,鼠标和键盘的动作称之为事件 还有页面事件

页面事件 一般分为2个事件

载入页面 onload

页面载入事件就是页面加载完成触发的一些东西

离开页面 onunload

离开页面事件就是页面关闭之后触发的一些东西

鼠标事件就有点多了 单机,双击, 按下,抬起,移上,移下 事件

单击事件 onclick 双击事件 ondblclick

鼠标按下事件 onmousedown 鼠标抬起事件 onmouseup

鼠标移上事件 onmouseover 鼠标移开事件 onmouseout

键盘事件 也就是 按下 抬起 按下并抬起

键盘按下事件 onkeydown 键盘抬起事件 onkeyup

键盘按下并抬起事件 onkeypress

还有表单事件

表单事件里有 提交 重置 获得焦点 失去焦点 内容改变事件

表单提交事件 onsubmit 表单重置事件 onreset

获得焦点事件 onfocus 失去焦点事件 onblur

内容改变事件 onchange

js 一些内置对象，方法

--math对象：

ceil（）向上取整 floor（）向下取整 abs（）绝对值

round（）四舍五入 random（）随机数 min() 最小值 max() 最大值

--string对象：

indexOf（）查找子字符串在字符串中首次出现的位置

lastIndexOf() 查找子字符串在字符串中最后一次出现的位置

split（）将字符串分割为数组

replace（）替换字符串

substr（）按下标和个数截取字符串

substring（）按下标截取字符串

toLowerCase() 将字符串转为小写

toUpperCase() 将字符串转为大写

--date对象：

getFullyear（）返回指定日期的年份

getMonth（）返回指定日期的月份

getDay（）返回指定日期星期中的第几天（0-6）

getDate（）返回指定日期月份中的第几天（1-31）

getHours() 返回指定日期对象的小时数

getMinutes() 返回指定日期对象的分钟数

getSeconds() 返回指定日期对象的秒数

getMilliseconds() 返回指定日期对象的毫秒数

toLocalestring（）根据当前区域设置并转化为字符串返回

toString（）将日期转化为字符串返回

--array对象：

concat（）将两个数组或多个数组拼接成一个新的数组

shift() 移除数组中第一个元素并返回该元素

unshifit() 向数组头部添加一个或多个元素，并返回数组的新长度

pop（）移除数组中最后一个元素并返回该元素

push（）向数组中添加一个元素并返回数组的长度

splice() 在任意位置添加或删除任意个元素

toString（）将数组以字符串的形式返回

join（）返回一个字符串，字符串是以指定的符号分隔

dom的一些常用操作

childNodes 返回html元素的子节点数组

firstChild 返回html元素的首个子节点

lastChild 返回html元素的最后一个子节点。

nextSibling 返回节点的下一个兄弟节点

parentNode 返回节点的父节点

previousSibling 返回节点的上一个兄弟节点

appendChild() 向元素添加新的子节点，并作为最后一个子节点。

cloneNode(是否深度克隆) 克隆元素

insertBefore(新节点,已有子节点) 在指定的已有的子节点之前插入新节点。

removeChild(已有子节点) 从元素中移除子节点

replaceChild(新节点,旧节点) 替换元素中的子节点

jsp

jsp 全称 JavaServer Pages是一种动态网页技术标准

jsp可以响应客户端发送的请求 并根据请求内容动态生成htnl xml或其他的

web网页 然后返回给请求者

--jsp的运行原理:

1、用户发送一个请求---->服务器(tomcat)

2、服务器根据请求的url(地址)----->找到相应的jsp文件

3、将找到的jsp文件转译成java源文件(servlet)

4、将转译出来的java源文件进行编译，得到一个可执行文件(.class)

5、执行.class文件并且告诉服务器(tomcat)

6、服务器返回一个响应信息------>用户

.Jsp的四个作用域、作用域范围、以及对应的作用域对象

page 当前页面 pageContext

request 当前请求 可以跨越多个页面 request

session 当前会话 多次请求 session

application 整个项目 整个服务应用 application

Jsp 9大隐式对象。

输入输出对象：

request 一次请求（对应的实体类是(HttpServletRequest)获取请求的信息(这是用户提交上来的信息)

同时还是作用域通讯对象在当前请求中，一直有效）

response 响应用户的信息 (对应的实体类是HttpServletResponse)

out输出对象（对应的实体类是(JspWriter)，一般输出一些调试信息

out.println(); 向页面输出一些信息

System.out.println();向控制台输出一些信息）

作用域通讯对象：

application 整个应用

（对应的实体类是(ServletContext)只要服务器不关闭，一直有效)

session 一次会话（对应的实体类是(HttpSession)

只要当前会话不关，一直有效）

pageContext 当前页面（对应的实体类是(PageContext)

只在本页面上有用(离开里本页面就没用了))

request 一次请求 在当前请求中，一直有效

servlet对象：

page (对应的实体类是(Page)相当于new了一个实例，类似于java中的this关键字）

config （对应的实体类是(ServletConfig)可以获取servlet初始化的一些信息，不太常用）

错误对象：

exception （对应的实体类是(Throwable) 专门处理错误）

SQL的分类 分为种

DDL(Data Definition Language)数据定义语言:create(创建)、alter(更改)、drop(删除)

DML(Data Manipulation Language)数据操作语言:insert(增加)、delete(删除)、update(修改)、select(查询)

TCL(Transaction Control Language)事务控制语言:commit(提交)、rollback(回滚)

DCL(Data Control Language)数据控制语言:grant(授权)、revoke(撤销)

DML数据操作语言

增加

insert into 表名 (字段1,字段2,字段3,...) value (值1,值2,值3,...);

insert into 表名 values (值1,值2,值3....);

values 所有的字段都要有值 value 这是 前面写了几个字段 后面跟着几个值

删除

delete from 表名 [where子句];

修改

update 表名 set 字段1=值1,字段2=值2,… [where子句];

document的内置对象

对象属性

document.title //设置文档标题等价于HTML的title标签

document.bgColor //设置页面背景色

document.fgColor //设置前景色(文本颜色)

document.linkColor //未点击过的链接颜色

document.alinkColor //激活链接(焦点在此链接上)的颜色

document.vlinkColor //已点击过的链接颜色

document.URL //设置URL属性从而在同一窗口打开另一网页

document.fileCreatedDate //文件建立日期，只读属性

document.fileModifiedDate //文件修改日期，只读属性

document.fileSize //文件大小，只读属性

document.cookie //设置和读出cookie

document.charset //设置字符集 简体中文:gb2312

———————————————————————

常用对象方法

document.write() //动态向页面写入内容

document.createElement(Tag) //创建一个html标签对象

document.getElementById(ID) //获得指定ID值的对象

document.getElementsByName(Name) //获得指定Name值的对象

document.body.appendChild(oTag)

———————————————————————

body-主体子对象

document.body //指定文档主体的开始和结束等价于body>/body>

document.body.bgColor //设置或获取对象后面的背景颜色

document.body.link //未点击过的链接颜色

document.body.alink //激活链接(焦点在此链接上)的颜色

document.body.vlink //已点击过的链接颜色

document.body.text //文本色

document.body.innerText //设置body>…/body>之间的文本

document.body.innerHTML //设置body>…/body>之间的HTML代码

document.body.topMargin //页面上边距

document.body.leftMargin //页面左边距

document.body.rightMargin //页面右边距

document.body.bottomMargin //页面下边距

document.body.background //背景图片

document.body.appendChild(oTag) //动态生成一个HTML对象

常用对象事件

document.body.οnclick=”func()” //鼠标指针单击对象是触发

document.body.οnmοuseοver=”func()” //鼠标指针移到对象时触发

document.body.οnmοuseοut=”func()” //鼠标指针移出对象时触发

———————————————————————

location-位置子对象

document.location.hash // #号后的部分

document.location.host // 域名+端口号

document.location.hostname // 域名

document.location.href // 完整URL

document.location.pathname // 目录部分

document.location.port // 端口号

document.location.protocol // 网络协议(http:)

document.location.search // ?号后的部分

documeny.location.reload() //刷新网页

document.location.reload(URL) //打开新的网页

document.location.assign(URL) //打开新的网页

document.location.replace(URL) //打开新的网页

———————————————————————

selection-选区子对象

document.selection

———————————————————————

images集合(页面中的图象)

a)通过集合引用

document.images //对应页面上的img标签

document.images.length //对应页面上img标签的个数

document.images[0] //第1个img标签

document.images[i] //第i-1个img标签

b)通过nane属性直接引用

img name=”oImage”

document.images.oImage //document.images.name属性

c)引用图片的src属性

document.images.oImage.src //document.images.name属性.src

d)创建一个图象

var oImage

oImage = new Image()

document.images.oImage.src=”1.jpg”

同时在页面上建立一个img /标签与之对应就可以显示

———————————————————————-

forms集合(页面中的表单)

a)通过集合引用

document.forms //对应页面上的form标签

document.forms.length //对应页面上/formform标签的个数

document.forms[0] //第1个/formform标签

document.forms[i] //第i-1个/formform标签

document.forms[i].length //第i-1个/formform中的控件数

document.forms[i].elements[j] //第i-1个/formform中第j-1个控件

b)通过标签name属性直接引用

/formform name=”Myform”>input name=”myctrl”/>/form

document.Myform.myctrl //document.表单名.控件名

c)访问表单的属性

document.forms[i].name //对应form name>属性

document.forms[i].action //对应/formform action>属性

document.forms[i].encoding //对应/formform enctype>属性

document.forms[i].target //对应/formform target>属性

document.forms[i].appendChild(oTag) //动态插入一个控件

document.all.oDiv //引用图层oDiv

document.all.oDiv.style.display=” //图层设置为可视

document.all.oDiv.style.display=”none” //图层设置为隐藏

document.getElementId(”oDiv”) //通过getElementId引用对象

document.getElementId(”oDiv”).style=”

document.getElementId(”oDiv”).display=”none”

/\*document.all表示document中所有对象的集合

只有ie支持此属性，因此也用来判断浏览器的种类\*/

图层对象的4个属性

document.getElementById(”ID”).innerText //动态输出文本

document.getElementById(”ID”).innerHTML //动态输出HTML

document.getElementById(”ID”).outerText //同innerText

document.getElementById(”ID”).outerHTML //同innerHTML

# 架构

架构

docker

nginx（负载均衡）

springboot

什么是springboot？(微服务)

Spring Boot 就是一些库的集合，框架的粘合剂，主要做接口开发

为所有 Spring 开发提供一个更快更广泛的入门体验，快速搭建spring 框架

零配置。无冗余代码生成和XML 强制配置，遵循“约定大于配置” 。

集成了大量常用的第三方库，提供了几乎可以零配置的开箱即用的能力

嵌入了web容器

spirngboot环境搭建

前提条件 jdk1.8及以上 maven项目

pom文件 声明是springboot项目 声明jdk 版本 引入启动器

创建启动类 @springbootapplication

@SpringBootApplication? : 是Sprnig Boot项目的核心注解，目的是开启自动配置

包含以下三个注解：

1.@Configuration： 用于定义一个配置类

2.@EnableAutoConfiguration ：Spring Boot会自动根据你jar包的依赖来自动配置项目。

3.@ComponentScan： 告诉Spring 哪个packages 的用注解标识的类

springcloud

SpringCloud是分布式微服务架构的一站式解决方案，是一个全家桶式的技术栈，包含了很多组件。

（eureka，zuul，feign，ribbon，hystric）

Eureka(优瑞卡)是什么

Eureka是服务注册中心。Spring Cloud 封装了 Netflix 公司开发的 Eureka 模块来实现服务注册和发现。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。

Eureka两大组件(Eureka Server和EurekaClient)

Eureka Server提供服务注册功能，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）

Eureka自我保护机制

默认情况下，如果Eureka Server在一定时间内没有接收到某个微服务实例的心跳，

RabbitMQ 消息队列

通常我们谈到消息队列服务, 会有三个概念： 发消息者、消息队列、收消息者。RabbitMQ 在这个基本概念之上, 多做了一层抽象, 在发消息者和队列之间, 加入了交换器 (Exchange)。这样发消息者和消息队列就没有直接联系，转而变成发消息者把消息发给交换器，交换器根据调度策略再把消息转发给消息队列。

消息生产者并没有直接将消息发送给消息队列，而是通过建立与Exchange的Channel，将消息发送给Exchange。Exchange根据路由规则，将消息转发给指定的消息队列。消息队列储存消息，等待消费者取出消息。消费者通过建立与消息队列相连的Channel,从消息队列中获取消息。

Exchange消息调度策略

调度策略是指Exchange在收到生产者发送的消息后依据什么规则把消息转发到一个或多个队列中保存

1. Fanout (订阅模式)

交换器将接受到的消息分发给有绑定关系的所有消息队列队列（不论Binding Key和Routing Key是什么）。类似于子网广播，子网内的每台主机都获得了一份复制的消息。Fanout交换机转发消息是最快的。

2. Direct(路由模式)

精确匹配：当消息的Routing Key与 Exchange和Queue 之间的Binding Key完全匹配，如果匹配成功，将消息分发到该Queue。只有当Routing Key和Binding Key完全匹配的时候，消息队列才可以获取消息。Direct是Exchange的默认模式。

3.Topic(通配符模式)

按照正则表达式模糊匹配：用消息的Routing Key与 Exchange和Queue 之间的Binding Key进行模糊匹配，如果匹配成功，将消息分发到该Queue。

消息确认机制

在实际应用中，可能会发生消费者收到Queue中的消息，但没有处理完成就宕机（或出现其他意外）的情况，这种情况下就可能会导致消息丢失。为了避免这种情况发生，我们可以要求消费者在消费完消息后发送一个回执给RabbitMQ，RabbitMQ收到消息回执（Message acknowledgment）后才将该消息从Queue中移除。

死信队列

在我们的业务中有时候需要等待一段时间再去处理某些业务，我们可以使用死信队列，当定义队列时指定了x-dead-letter-exchange(死信队列)死信交换机中的消息会在过期时被重新分配到其他正常交换机中，在被分配到队列，根据此特性我们可以完成延时操作。

分布式事务seata解决的