面试宝典

一, 微服务相关

1.谈谈你对Restful的理解

RESTful(Resource Representational State Transfer 资源表现层状态转化)是一种架构的风格，符合这种风格的架构就是RESTful架构。具体来说, 就是HTTP协议里面，四个表示操作方式的动词：GET、POST、PUT、DELETE。它们分别对应四种基本操作：GET用来获取资源，POST用来新建资源（也可以用于更新资源），PUT用来更新资源，DELETE用来删除资源。

（1）每一个URI代表一种资源；

（2）客户端和服务器之间，传递这种资源的某种表现层；

（3）客户端通过四个HTTP动词，对服务器端资源进行操作，实现"表现层状态转化"。

- 实例

```
保存
        传统：http://localhost:8080/user/save
        REST：http://localhost:8080/user                                    POST    执行保存

更新
        传统：http://localhost:8080/user/update?id=1
        REST：http://localhost:8080/user/1                                  PUT     执行更新
1就是id

删除
        传统：http://localhost:8080/user/delete?id=1
        REST：http://localhost:8080/user/1                                  DELETE  执行删除

查询
        传统：http://localhost:8080/user/findAll
        REST：http://localhost:8080/user                                    GET     查所有

        传统：http://localhost:8080/user/findById?id=1
        REST：http://localhost:8080/user/1                                  GET     根据id查1
个
```

2.谈谈对微服务的理解

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。

功能	SOA	微服务

从开发角度来说: 微服务微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。

3.微服务和SOA区别

SOA和微服务的确是一脉相承的，大神Martin Fowler提出来这一概念可以说把SOA的理念继续升华，精进了一步。其核心思想是在应用开发领域，使用一系列微小服务来实现单个应用的方式途径，或者说微服务的目的是有效的拆分应用，实现敏捷开发和部署 ，可以是使用不同的编程语言编写。而SOA可能包含的意义更泛一些，更不准确一些。

功能	SOA	微服务
组件大小	大块业务逻辑	单独任务或小块业务逻辑
耦合	通常松耦合	总是松耦合
公司架构	任何类型	小型、专注于功能交叉团队
管理	着重中央管理	着重分散管理
目标	确保应用能够交互操作	执行新功能、快速拓展开发团队

4.谈谈对SpringBoot的认识

Spring Boot 是 Spring 社区较新的一个项目。该项目的目的是帮助开发者更容易的创建基于 Spring 的应用程序和服务，让更多人的人更快的对 Spring 进行入门体验，为 Spring 生态系统提供了一种固定的、约定优于配置风格的框架。

Spring Boot 具有如下特性：

（1）为基于 Spring 的开发提供更快的入门体验

（2）开箱即用，没有代码生成，也无需 XML 配置。同时也可以修改默认值来满足特定的需求。

（3）提供了一些大型项目中常见的非功能性特性，如嵌入式服务器、安全、指标，健康检测、外部配置等。

（4）Spring Boot 并不是不对 Spring 功能上的增强，而是提供了一种快速使用 Spring 的方式。

5.SpringBoot的原理

5.1POM文件

- 我们的项目里面的pom文件,它的父工程是spring-boot-starter-parent

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.4.RELEASE</version>
    <relativePath/>
</parent>
```

- spring-boot-starter-parent的pom文件, 它的父工程是org.springframework.boot

```
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-dependencies</artifactId>
        <version>2.0.4.RELEASE</version>
        <relativePath>../../spring-boot-dependencies</relativePath>
    </parent>
```

- org.springframework.boot的pom文件

```
<properties>
        <activemq.version>5.15.4</activemq.version>
        <antlr2.version>2.7.7</antlr2.version>
        <appengine-sdk.version>1.9.64</appengine-sdk.version>
        <artemis.version>2.4.0</artemis.version>
        <aspectj.version>1.8.13</aspectj.version>
        <assertj.version>3.9.1</assertj.version>
        <atomikos.version>4.0.6</atomikos.version>
        <bitronix.version>2.1.4</bitronix.version>
         ...
</properties>
<dependencyManagement>
    <dependencies>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot</artifactId>
                <version>2.0.4.RELEASE</version>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-test</artifactId>
                <version>2.0.4.RELEASE</version>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-test-autoconfigure</artifactId>
                <version>2.0.4.RELEASE</version>
            </dependency>
    </dependencies>
 <dependencyManagement>
```

结论: org.springframework.boot是Spring Boot的版本仲裁中心；它来管理Spring Boot应用里面的所有依赖版本.也就意味着我们导入依赖默认是不需要写版本的,SpringBoot已经帮我们排除了常见的jar包冲突.（当然如果没有在dependencies里面管理的依赖,还是需要声明版本号的）

5.2启动器

- 依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

spring-boot-starter-web帮我们导入了web模块正常运行所依赖的组件；
spring-boot-starter：spring-boot场景启动器；
Spring Boot将所有的功能场景都抽取出来，做成一个个的starters（启动器），只需要在项目里面引入这些starter相关场景的所有依赖都会导入进来。要用什么功能就导入什么场景的启动器

5.3主程序类，主入口类

- 代码

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}

//说明:
@SpringBootApplication: Spring Boot应用标注在某个类上说明这个类是SpringBoot的主配置类，SpringBoot就应该运行这个类的main方法来启动SpringBoot应用；
```

- 我们的Application类有的有一个注解: @SpringBootApplication

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
        @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
        @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class) })
public @interface SpringBootApplication {
    ....

}

//说明:
@SpringBootConfiguration:Spring Boot的配置类；标注在某个类上，表示这是一个Spring Boot的配置类；
@Configuration:配置当前类为配置类(作用等同配置文件)
@EnableAutoConfiguration：开启自动配置功能；以前我们需要配置的东西，Spring Boot帮我们自动配置；告诉SpringBoot开启自动配置功能；这样自动配置才能生效；
```

- 注解SpringBootApplication上有一个元注解@EnableAutoConfiguration,这就是起自动装配的作用

```
@SuppressWarnings("deprecation")
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@AutoConfigurationPackage
@Import(EnableAutoConfigurationImportSelector.class)
public @interface EnableAutoConfiguration {
  ...

}

//说明:
@AutoConfigurationPackage：自动配置包
@Import(AutoConfigurationPackages.Registrar.class)：Spring的底层注解@Import，给容器中导入一个组件；
```

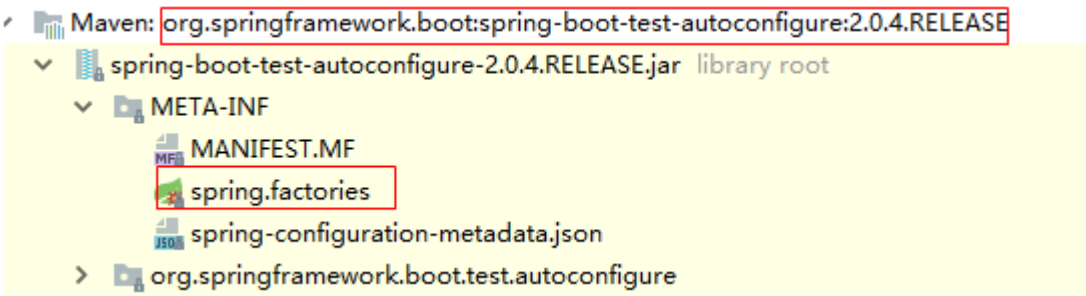
- 注解EnableAutoConfiguration上导入了一个类@Import(EnableAutoConfigurationImportSelector.class)

```
@Deprecated
public class EnableAutoConfigurationImportSelector
        extends AutoConfigurationImportSelector {

}
```

- EnableAutoConfigurationImportSelector的父类AutoConfigurationImportSelector

```
    public class AutoConfigurationImportSelector
        implements DeferredImportSelector, BeanClassLoaderAware, ResourceLoaderAware,
        BeanFactoryAware, EnvironmentAware, Ordered {
    @Override
    public String[] selectImports(AnnotationMetadata annotationMetadata) {
        ...
        try {
            AutoConfigurationMetadata autoConfigurationMetadata =
AutoConfigurationMetadataLoader
                    .loadMetadata(this.beanClassLoader);
            AnnotationAttributes attributes = getAttributes(annotationMetadata);
            List<String> configurations = getCandidateConfigurations(annotationMetadata,
                    attributes);
            ....
        }
    }

    //获得候选的配置
    protected List<String> getCandidateConfigurations(AnnotationMetadata metadata,
            AnnotationAttributes attributes) {
        List<String> configurations = SpringFactoriesLoader.loadFactoryNames(
                getSpringFactoriesLoaderFactoryClass(), getBeanClassLoader());
        Assert.notEmpty(configurations,
                "No auto configuration classes found in META-INF/spring.factories. If you "
                        + "are using a custom packaging, make sure that file is correct.");
        return configurations;
    }
```

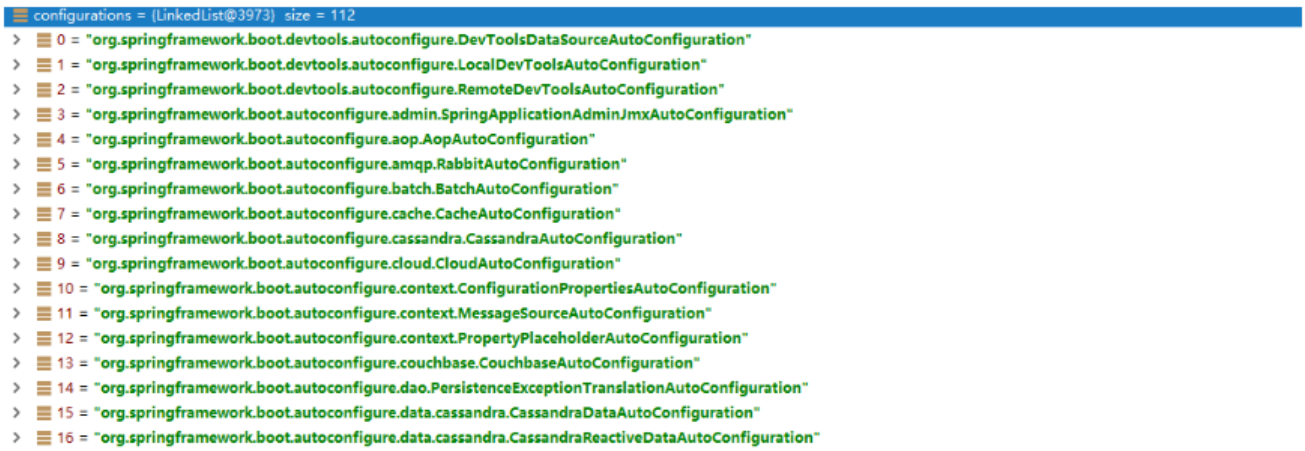
其中，SpringFactoriesLoader.loadFactoryNames 方法的作用就是从META-INF/spring.factories文件中读取指定类对应的类名称列表



META-INF/spring.factories的配置文件

```
# AutoConfigureWebMvc auto-configuration imports
org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureWebMvc=\
org.springframework.boot.autoconfigure.context.MessageSourceAutoConfiguration,\
org.springframework.boot.autoconfigure.freemarker.FreeMarkerAutoConfiguration,\
org.springframework.boot.autoconfigure.groovy.template.GroovyTemplateAutoConfiguration,\
org.springframework.boot.autoconfigure.gson.GsonAutoConfiguration,\
org.springframework.boot.autoconfigure.hateoas.HypermediaAutoConfiguration,\
org.springframework.boot.autoconfigure.http.HttpMessageConvertersAutoConfiguration,\
org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration,\
org.springframework.boot.autoconfigure.jsonb.JsonbAutoConfiguration,\
org.springframework.boot.autoconfigure.mustache.MustacheAutoConfiguration,\
org.springframework.boot.autoconfigure.thymeleaf.ThymeleafAutoConfiguration,\
org.springframework.boot.autoconfigure.validation.ValidationAutoConfiguration,\
org.springframework.boot.autoconfigure.web.servlet.error.ErrorMvcAutoConfiguration,\
org.springframework.boot.autoconfigure.web.servlet.WebMvcAutoConfiguration
```

加载META-INF/spring.factories之后, 解析获得 List configurations的值:



结论:Spring Boot在启动的时候从类路径下的META-INF/spring.factories中获取EnableAutoConfiguration指定的值，将
这些值作为自动配置类导入到容器中，自动配置类就生效，帮我们进行自动配置工作；以前我们需要自己配置的东西，自动配置类都帮我们；J2EE的整体整合解决方案和自动配置都在spring-boot-autoconfigure-2.0.4.RELEASE.jar；

6.SpringCloud相关

6.1谈谈你对SpringCloud的认识

SpringCloud，是基于SpringBoot提供的一套微服务解决方案，包括服务注册与发现，配置中心，全链路监控，服务网关，负载均衡，熔断器等组件，除了基于NetFlix的开源组件做高度抽象封装之外，还有一些选型中立的开源组件。

SpringCloud利用SpringBoot的开发便利性巧妙地简化了分布式系统基础设施的开发，SpringCloud为开发人员提供了快速构建分布式系统的一些工具，包括配置管理、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,它们都可以用SpringBoot的开发风格做到一键启动和部署。

简单来说:SpringCloud分布式微服务架构下的一站式解决方案，是各个微服务架构落地技术的集合体.

6.2SpringCloud主要框架和版本

Spring Boot	Dubbo Spring Cloud	Spring Cloud

6.2.1SpringCloud主要框架

- 服务发现——Netflix Eureka
- 服务调用——Netflix Feign
- 熔断器——Netflix Hystrix
- 服务网关——Netflix Zuul
- 分布式配置——Spring Cloud Config
- 消息总线 —— Spring Cloud Bus

6.2.2SpringCloud版本

SpringCloud由于是一系列框架组合，为了避免与包含的自框架版本产生混淆，采用伦敦地铁站的名称作为版本名，形式为版本名+里程碑号。 M9为第9个里程碑版本。 以下是SpringBoot与Spring Cloud版本的对照表:

Spring Boot	Spring Cloud
1.2.x	Angel版本
1.3.x	Brixton版本
1.4.x	Camden版本
1.5.x	Dalston版本、Edgware版本
2.0.x	Finchley版本

6.3Dubbo和Spring Cloud的区别

	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
熔断器	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
.....	.....	.....

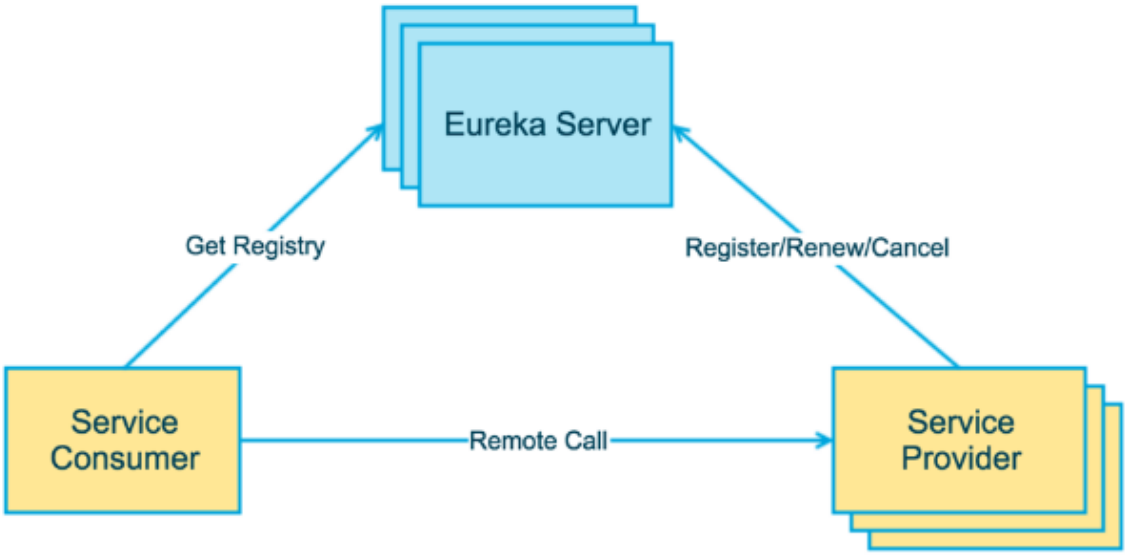
- Dubbo只是实现了服务治理，而Spring Cloud下面有17个子项目（可能还会新增）分别覆盖了微服务架构下的方方面面，服务治理只是其中的一个方面，
- SpringCloud抛弃了Dubbo的RPC通信，采用的是基于HTTP的REST方式。 严格来说，这两种方式各有优劣。虽然从一定程度上来说，后者牺牲了服务调用的性能，但也避免了上面提到的原生RPC带来的问题。而且REST

相比RPC更为灵活，服务提供方和调用方的依赖只依靠一纸契约，不存在代码级别的强依赖，这在强调快速演化的微服务环境下，显得更加合适。

- DUBBO停止了5年左右的更新，2017.7重启了; SpringCloud自从出现后社区活跃度特别高;

6.4Zookeeper和Eureka运行机制

- Eureka



Eureka 采用了 C-S 的设计架构。 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

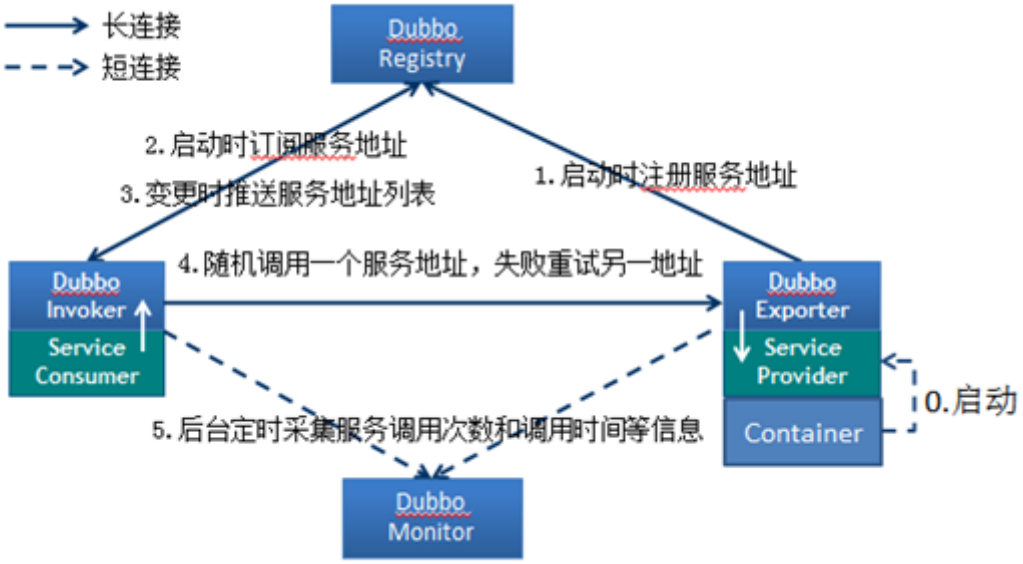
EurekaClient是一个Java客户端，用于简化Eureka Server的交互，客户端同时也具备一个内置的、使用轮询(round-robin)负载算法的负载均衡器。在应用启动后，将会向Eureka Server发送心跳(默认周期为30秒)。如果Eureka Server在多个心跳周期内没有接收到某个节点的心跳，EurekaServer将会从服务注册表中把这个服务节点移除（默认90秒）

三大角色:Eureka Server 提供服务注册和发现

Service Provider服务提供方将自身服务注册到Eureka，从而使服务消费方能够找到

Service Consumer服务消费方从Eureka获取注册服务列表，从而能够消费服务

- Zookeeper



服务容器负责启动，加载，运行服务提供者。

服务提供者在启动时，向注册中心注册自己提供的服务。

服务消费者在启动时，向注册中心订阅自己所需的服务。

注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。

服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。

服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

6.5Zookeeper和Eureka的区别

6.5.1CAP

- C:Consistency (一致性)

  在分布式系统中的所有数据备份，在同一时刻是否同样的值。（等同于所有节点访问同一份最新的数据副本）

- A:Availability（可用性):

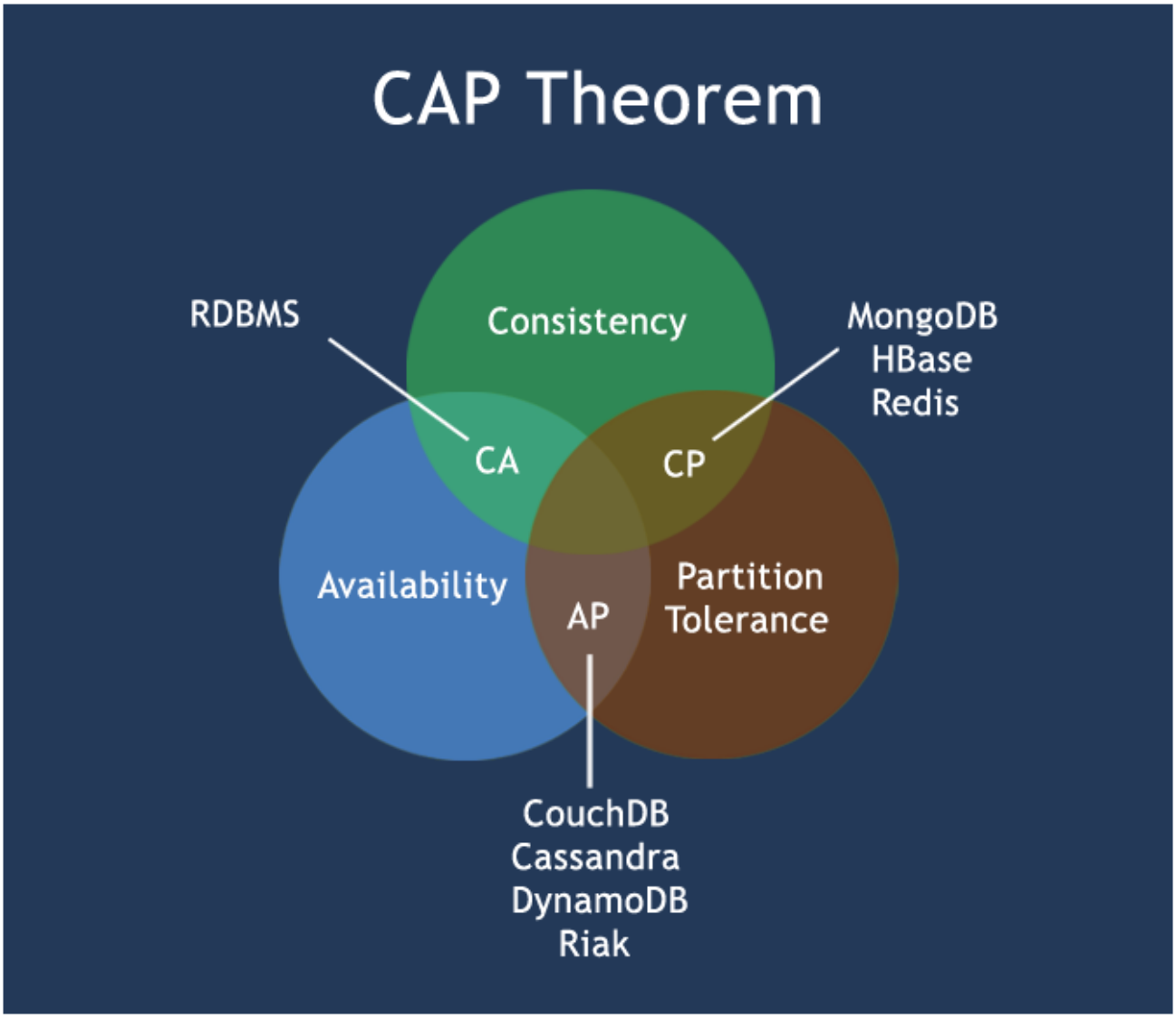
  在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求。（对数据更新具备高可用性）

- P:Partition tolerance（分区容错性）

  以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在C和A之间做出选择。

  分布式系统不可避免的出现了多个系统通过网络协同工作的场景，结点之间难免会出现网络中断、网延延迟等现
  象，这种现象一旦出现就导致数据被分散在不同的结点上，这就是网络分区。

CAP理论的核心是：一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求，最多只能同时较好的满足两个。因此，根据 CAP 原理将分布式系统分成了满足 CA 原则、满足 CP 原则和满足 AP 原则三大类：CA - 单点集群，满足一致性，可用性的系统，通常在可扩展性上不太强大。CP - 满足一致性，分区容忍必的系统，通常性能不是特别高。AP - 满足可用性，分区容忍性的系统，通常可能对一致性要求低一些。



CAP理论就是说在分布式存储系统中，最多只能实现上面的两点。而由于当前的网络硬件肯定会出现延迟丢包等问题，所以分区容错性是我们必须需要实现的。所以我们只能在一致性和可用性之间进行权衡.

CA 传统Oracle数据库

AP 大多数网站架构的选择

CP Redis、Mongodb

6.5.2Zookeeper和Eureka的区别

- 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AP

Eureka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：

1. 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
2. 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)
3. 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。

6.6你在项目中使用SpringCloud的哪些组件

服务发现组件Eureka, 服务间的调用 Feign,熔断器Hystrix, 微服务网关Zuul,集中配置组件SpringCloudConfig, 消息总线组件SpringCloudBus

6.7你在项目中哪个业务场景使用到微服务间的调用

交友微服务. 需要结合自己的业务来回答.

6.8解释雪崩效应，为什么要使用熔断器

在微服务架构中通常会有多个服务层调用，基础服务的故障可能会导致级联故障，进而造成整个系统不可用的情况，这种现象被称为服务雪崩效应。服务雪崩效应是一种因“服务提供者”的不可用导致“服务消费者”的不可用,并将不可用逐渐放大的过程。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。

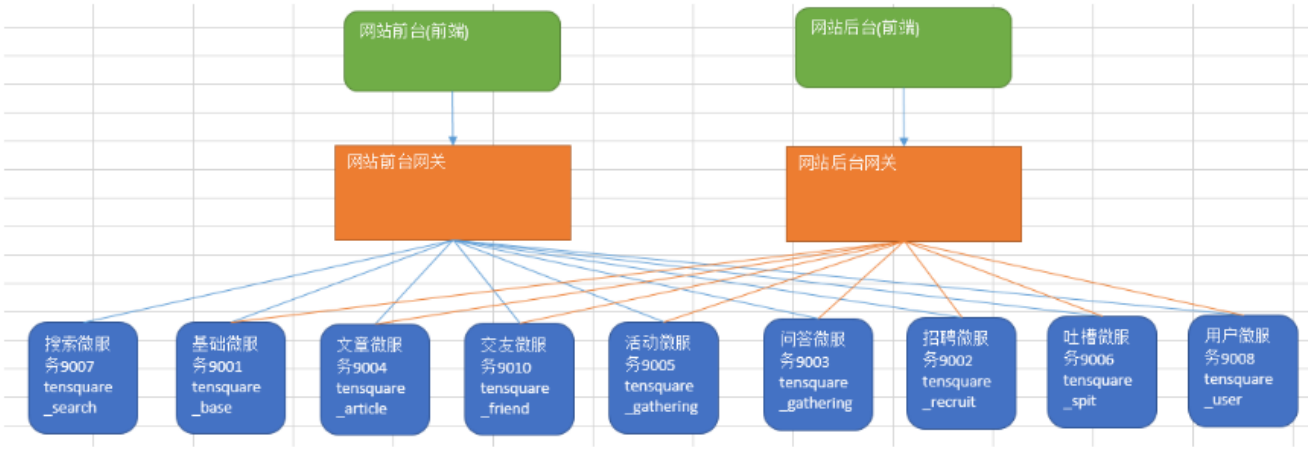
“断路器”本身是一种开关装置，当某个服务单元发生故障之后，通过断路器的故障监控（类似熔断保险丝），向调用方返回一个符合预期的、可处理的备选响应（FallBack），而不是长时间的等待或者抛出调用方无法处理的异常，这样就保证了服务调用方的线程不会被长时间、不必要地占用，从而避免了故障在分布式系统中的蔓延，乃至雪崩。

6.9为什么使用微服务网关

Zuul是Netflix开源的微服务网关, Zuul组件的核心是一系列的过滤器.

使用网关的目的: 微服务工程统一入口，方便前端调用, 身份认证和安全, 集中处理权限问题 ,压力测试(了解性能), 动态将请求路由到不同后端集群,负载分配.

附上十次方微服务网关图解(方便理解)



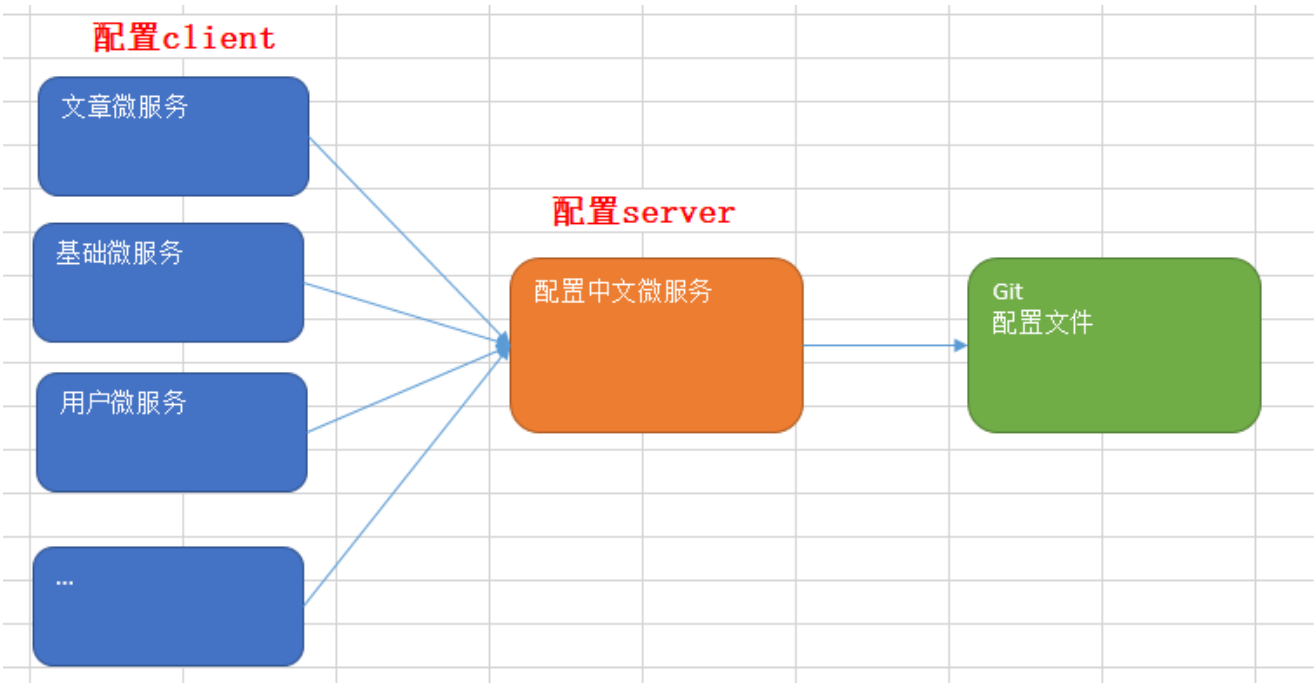
6.10为什么使用集中配置管理

在分布式系统中，由于服务数量巨多，为了方便服务配置文件统一管理，实时更新，所以需要分布式配置中心组件。在Spring Cloud中，有分布式配置中心组件spring cloudconfig ，它支持配置服务放在配置服务的内存中（即本地），也支持放在远程Git仓库中。在spring cloud config 组件中，分两个角色，一是config server，二是configclient。

Config Server是一个可横向扩展、集中式的配置服务器，它用于集中管理应用程序各个环境下的配置，默认使用Git存储配置文件内容，也可以使用SVN存储，或者是本地文件存储。

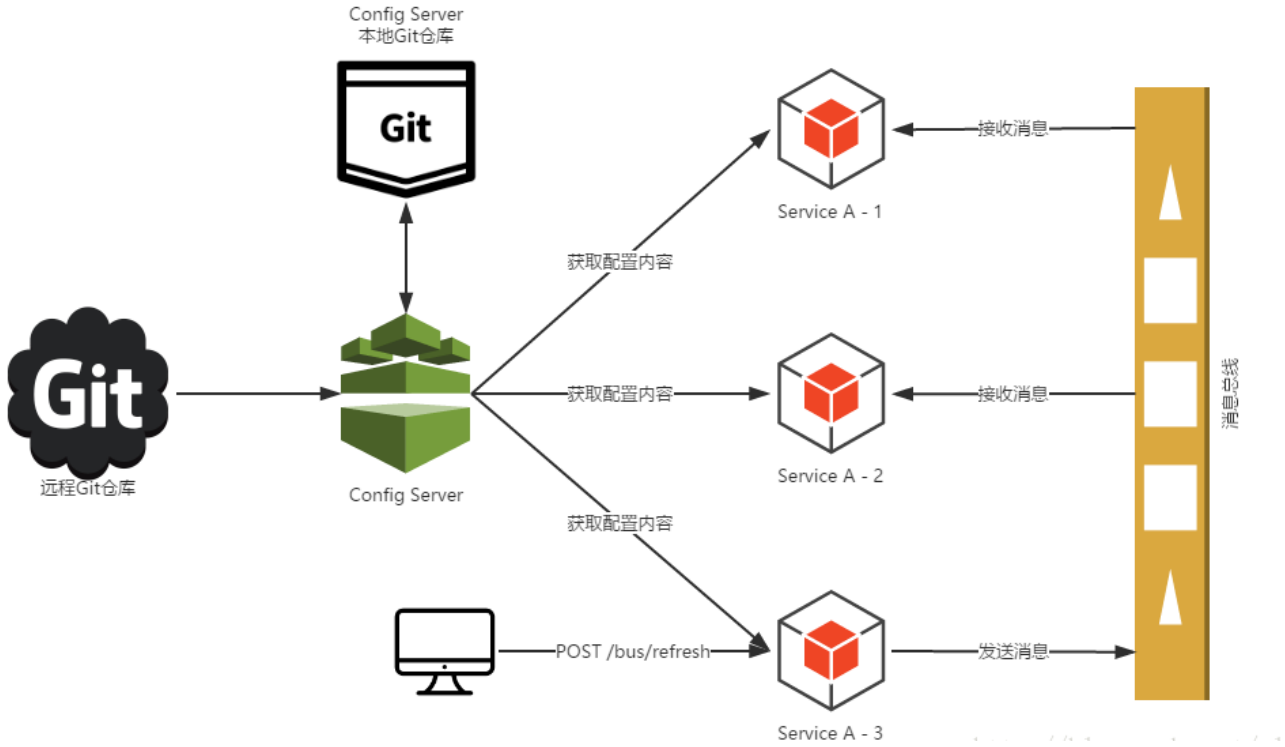
Config Client是Config Server的客户端，用于操作存储在Config Server中的配置内容。微服务在启动时会请求Config Server获取配置文件的内容，请求到后再启动容器。

附上十次方集中配置管理 图解(方便理解)



6.11为什么使用消息总线Spring Cloud Bus

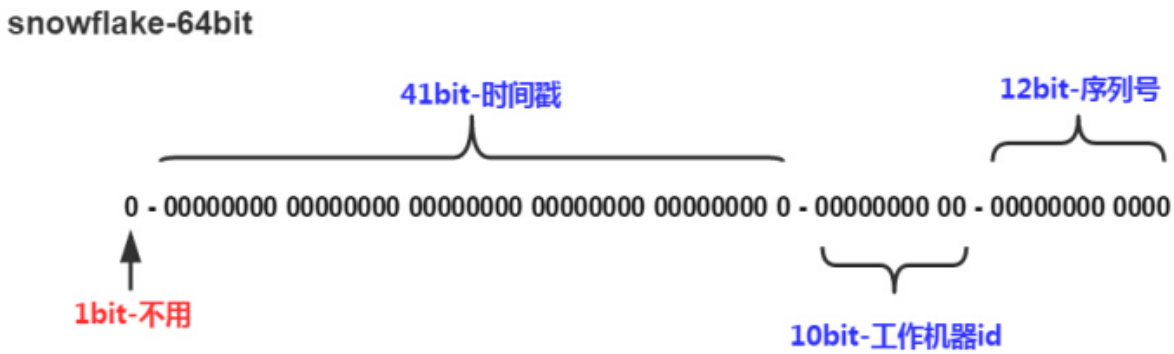
我们如果要去更新所有微服务的配置，在不重启的情况下去更新配置，只能依靠spring cloud config了，但是，是我们要一个服务一个服务的发送post请求，我们就可以使用Spring Cloud Bus避免挨个挨个的向服务发送Post请求来告知服务，只发一次,所有的微服务的配置就会更新. 其实就是发布订阅模型.



二,分布式,高并发,负载均衡相关

1,你工作中，分布式项目中如何解决ID生成问题？

我们采用的是开源的twitter(非官方中文惯称：推特.是国外的一个网站，是一个社交网络及微博客服务) 的snowflake算法。 效率较高，经测试，snowflake每秒能够产生26万ID左右，完全满足需要。



2.集群与分布式的区别

- 相同点：

  分布式和集群都是需要有很多节点服务器通过网络协同工作完成整体的任务目标。

- 不同点：

  分布式是指将业务系统进行拆分，即分布式的每一个节点都是实现不同的功能。而集群每个节点做的是同一件事情。

每个人都有不同的分工，一起协作干一件事，叫做“分布式”

每个划桨人干的都是一样的活，叫做集群

现代乐队这样图就是分布式啦

古代乐队的图就属于集群

3.如何解决网站高并发问题

1. 使用缓存, 可以大量减少与数据库的交互，提高性能。
2. 优化数据库查询语句
3. 能使用静态页面的地方尽量使用，减少容器的解析（尽量将动态内容生成静态html来显示）。
4. 服务器集群解决单台的瓶颈问题
5. 读写分离 (双机热备功能。第一台数据库服务器，是对外提供增删改业务的生产服务器；第二台数据库服务器，主要进行读的操作.)

4.负载均衡的种类有哪些？

1）一种是通过硬件来进行解决，常见的硬件有NetScaler、F5、Radware和Array等商用的负载均衡器，但是它们是比较昂贵的

2）一种是通过软件来进行解决的，常见的软件有LVS、Nginx、apache等,它们是基于Linux系统并且开源的负载均衡策略.

5.谈谈你对SOA架构的理解

SOA是Service-OrientedArchitecture的首字母简称，它是一种支持面向服务的架构样式。从服务、基于服务开发和服务的结果来看，面向服务是一种思考方式。其实SOA架构更多应用于互联网项目开发。

随着互联网的发展，网站应用的规模不断扩大，常规的垂直应用架构已无法应对，分布式服务架构以及流动计算架构势在必行，迫切需一个治理系统确保架构有条不紊的演进。

6.怎么实现远程通信

远程通信:简单来说，就是一个系统去调用另一个系统中的数据.

- Webservice的方式：

  优点：跨语言跨平台

  缺点：它是基于soap协议的，使用http+xml的方式进行数据传输，http是应用层协议，传输效率不是很高，而且xml的解析也比 较费时，所以项目内部进行通信的时候，不建议使用Websservice的方式

- restful形式的服务：

  优点：restful本身就是http，使用的是http+json的方式进行数据传输，因为json数据本身是非常简洁的，所以它比webservice的 传输效率更高；手机app端一般都使用该方法，其他很多项目也是用这种方式

  缺点：如果服务太多的话，会出现服务之间调用关系混乱，此时就需要治理服务

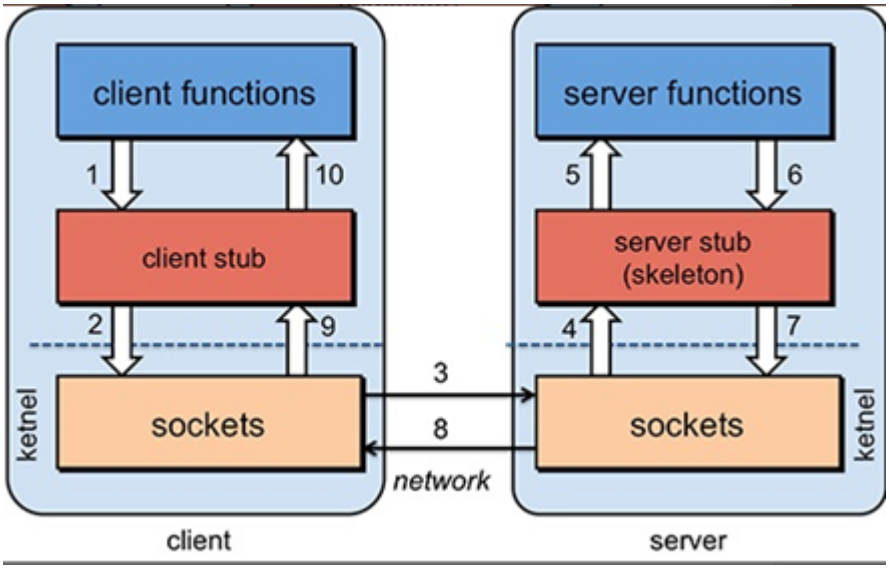
- Dubbo：

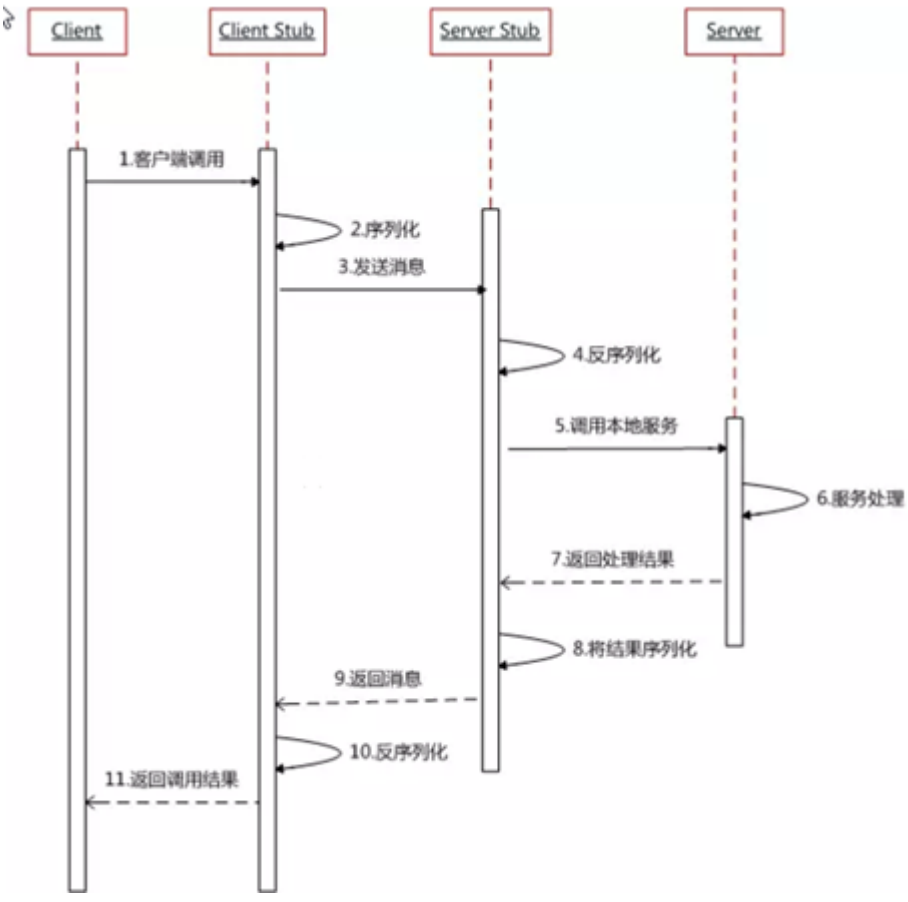
使用的是RPC方式进行远程调用，直接使用的socket进行通信，传输效率高，并且可以统计出系统 之间的调用关系和调用次数系统分布式SAO系统的内部通信推荐使用dubbo

7.dubbo框架

7.1RPC

RPC【Remote Procedure Call】是指远程过程调用，是一种进程间通信方式，他是一种技术的思想，而不是规范。它允许程序调用另一个地址空间（通常是共享网络的另一台机器上）的过程或函数，而不用程序员显式编码这个远程调用的细节。即程序员无论是调用本地的还是远程的函数，本质上编写的调用代码基本相同。





一次完整的RPC调用流程（同步调用，异步另说）如下：

1）服务消费方（client）调用以本地调用方式调用服务;

2）client stub接收到调用后负责将方法、参数等组装成能够进行网络传输的消息体；

3）client stub找到服务地址，并将消息发送到服务端；

4）server stub收到消息后进行解码；

5）server stub根据解码结果调用本地的服务；

6）本地服务执行并将结果返回给server stub；

7）server stub将返回结果打包成消息并发送至消费方；

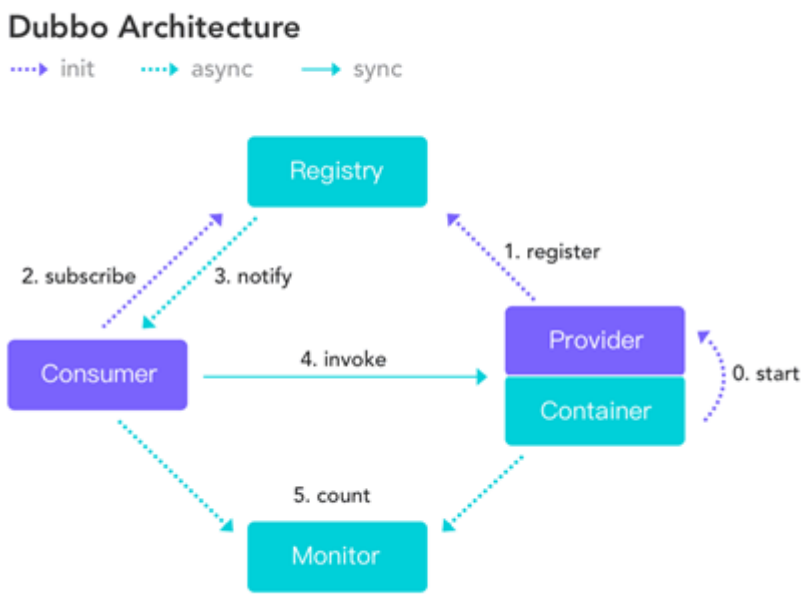
8）client stub接收到消息，并进行解码；

9）服务消费方得到最终结果。

RPC框架的目标就是要2~8这些步骤都封装起来，这些细节对用户来说是透明的，不可见的

7.2谈谈你对dubbo的理解

Apache Dubbo (incubating是一款高性能、轻量级的开源JavaRPC框架，它提供了三大核心能力：面向接口的远程方法调用，智能容错和负载均衡，以及服务自动注册和发现。



组件:

- 服务提供者（Provider）：暴露服务的服务提供方，服务提供者在启动时，向注册中心注册自己提供的服务。
- 服务消费者（Consumer）: 调用远程服务的服务消费方，服务消费者在启动时，向注册中心订阅自己所需的服务，服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。
- 注册中心（Registry）：注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者

- 监控中心（Monitor）：服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心

调用关系说明

- 服务容器负责启动，加载，运行服务提供者。
- 服务提供者在启动时，向注册中心注册自己提供的服务。
- 服务消费者在启动时，向注册中心订阅自己所需的服务。
- 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。
- 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。
- 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

7.3dubbo服务开发流程

第一步：要在系统中使用dubbo应该先搭建一个注册中心，一般推荐使用zookeeper。

第二步：有了注册中心然后是发布服务，发布服务需要使用spring容器和dubbo标签来发布服务。并且发布服务时需要指定注册中心的位置。

第三步：服务发布之后就是调用服务。一般调用服务也是使用spring容器和dubbo标签来引用服务，这样就可以在客户端的容器中生成一个服务的代理对象，在action或者Controller中直接调用service的方法即可。

7.4Dubbo挂机了怎么处理

同问1: Dubbo中zookeeper做注册中心，如果注册中心集群都挂掉，发布者和订阅者之间还能通信么?

同问2: zookeeper挂了还能不能调用服务

1. 监控中心宕掉不影响使用，只是丢失部分采样数据
2. 数据库宕掉后，注册中心仍能通过缓存提供服务列表查询，但不能注册新服务
3. 注册中心对等集群，任意一台宕掉后，将自动切换到另一台
4. 注册中心全部宕掉后，服务提供者和服务消费者仍能通过本地缓存通讯
5. 服务提供者无状态，任意一台宕掉后，不影响使用
6. 服务提供者全部宕掉后，服务消费者应用将无法使用，并无限次重连等待服务提供者恢复

7.5dubbo+zookeeper如何解决不同系统间调用的安全性问题

Dubbo通过Token令牌防止用户绕过注册中心直连，然后在注册中心上管理授权。

- 可以全局设置开启令牌验证

```
<!--随机token令牌，使用UUID生成-->
<dubbo:provider interface="com.foo.BarService" token="true" />
<!--固定token令牌，相当于密码-->
<dubbo:provider interface="com.foo.BarService" token="123456" />
```

- 也可在服务级别设置

```
<!--随机token令牌，使用UUID生成-->
<dubbo:service interface="com.foo.BarService" token="true" />
<!--固定token令牌，相当于密码-->
<dubbo:service interface="com.foo.BarService" token="123456" />
```

- 还可在协议级别设置

```
<!--随机token令牌，使用UUID生成-->
<dubbo:protocol name="dubbo" token="true" />
<!--固定token令牌，相当于密码-->
<dubbo:protocol name="dubbo" token="123456" />
```

7.6怎么配置重试次数和超时时间

- 配置重试次数(失败自动切换，当出现失败，重试其它服务器，但重试会带来更长延迟。可通过 retries="2" 来设置重试次数(不含第一次))

```
<dubbo:service retries="2" />
或者
<dubbo:reference retries="2" />
或者
<dubbo:reference>
    <dubbo:method name="findFoo" retries="2" />
</dubbo:reference>
```

- 配置超时时间

  由于网络或服务端不可靠，会导致调用出现一种不确定的中间状态（超时）。为了避免超时导致客户端资源（线程）挂起耗尽，必须设置超时时间。

1. Dubbo消费端

```
全局超时配置
<dubbo:consumer timeout="5000" />

指定接口以及特定方法超时配置
<dubbo:reference interface="com.foo.BarService" timeout="2000">
    <dubbo:method name="sayHello" timeout="3000" />
</dubbo:reference>
```

2. Dubbo服务端

```
全局超时配置
<dubbo:provider timeout="5000" />

指定接口以及特定方法超时配置
<dubbo:provider interface="com.foo.BarService" timeout="2000">
    <dubbo:method name="sayHello" timeout="3000" />
</dubbo:provider>
```

3. 配置原则

   dubbo推荐在Provider上尽量多配置Consumer端属性

   作服务的提供者，比服务使用方更清楚服务性能参数，如调用的超时时间，合理的重试次数，等等
   在Provider配置后，Consumer不配置则会使用Provider的配置值，即Provider配置可以作为Consumer的缺省值。否则，Consumer会使用Consumer端的全局设置，这对于Provider不可控的，并且往往是不合理的

7.7集群下dubbo负载均衡

在集群负载均衡时，Dubbo 提供了多种均衡策略，缺省为 random 随机调用

- RandomLoadBalance

  随机，按权重设置随机概率。

  在一个截面上碰撞的概率高，但调用量越大分布越均匀，而且按概率使用权重后也比较均匀，有利于动态调整提供者权重。

- RoundRobinLoadBalance

  轮循，按公约后的权重设置轮循比率。

  存在慢的提供者累积请求的问题，比如：第二台机器很慢，但没挂，当请求调到第二台时就卡在那，久而久之，所有请求都卡在调到第二台上。

- LeastActiveLoadBalance

  最少活跃调用数，相同活跃数的随机，活跃数指调用前后计数差。

  使慢的提供者收到更少请求，因为越慢的提供者的调用前后计数差会越大。

- ConsistentHashLoadBalance

  一致性 Hash，相同参数的请求总是发到同一提供者。

  当某一台提供者挂时，原本发往该提供者的请求，基于虚拟节点，平摊到其它提供者，不会引起剧烈变动。算法参见：

配置

- 服务端服务级别

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

- 客户端服务级别

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

- 服务端方法级别

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

- 客户端方法级别

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

8.注册中心Zookeeper

8.1谈谈你对zookeeper的理解解

官方推荐使用 zookeeper 注册中心。注册中心负责服务地址的注册与查找，相当于目录服务，服务提供者和消费者只在启动时与注册中心交互，注册中心不转发请求，压力较小。

Zookeeper 是 ApacaheHadoop 的子项目，是一个树型的目录服务，支持变更推送，适合作为Dubbox 服务的注册中心，工业强度较高，可用于生产环境。

8.2zookeeper注册中心的作用?端口是多少

- 作用

  Zookeeper注册中心的作用主要就是注册和发现服务的作用。类似于房产中介的作用，在系统中并不参与服务的调用及数据的传输

- 端口

  2181：对cline端提供服务

  3888：选举leader使用

  2888：集群内机器通讯使用（Leader监听此端口）

8.3zookeeper在项目中怎么用的

作为dubbo的注册中心,暴露服务,然后消费方订阅服务用的

作为solr集群的调配中心,达到负载均衡的效果

8.4zookeeper存在什么缺陷

- 本身不是为高可用性设计，master撑不住高流量容易导致系统crash
- zookeeper的选举过程速度很慢(例如发生网络问题时，ZK集群需要开始选主，选主过程如果持续较长，应用都会抛异常。而且后续可能会出现follower不能及时跟上leader的情况。如果这个过程持续数十分钟，那么将会导致应用在这个期间内无法提供服务。影响是非常大的.)
- 难以避免数据的不一致
- zookeeper的性能是有限的

8.5zookeeper基本原理

ZooKeeper 是以 Fast Paxos 算法为基础的， Paxos 算法存在活锁的问题，即当有多个 proposer 交错提交时，有可能互相排斥导致没有一个 proposer 能提交成功，而 Fast Paxos 作了一些优化，通过选举产生一个 leader (领导者)，只有 leader 才能提交 proposer，具体 算法可见 Fast Paxos。因此，要想弄懂 ZooKeeper 首先得对 FastPaxos 有所了解。

ZooKeeper 的基本运转流程:1、选举 Leader。 2、同步数据。 3、选举 Leader 过程中算法有很多，但要达到的
选举标准是一致的。 4、 Leader 要具有最高的执行 ID，类似 root 权限。 5、集群中大多数的机器得到响应并 follow

9.Nginx

9.1Nginx作为反向代理为什么能够提升服务器性能

对于后端是动态服务来说，比如 Java 和 PHP。这类服务器(如 JBoss 和 tomcat) 的 IO 处理能力往往不高。Nginx 有个好处是它会把 Request 在读取完整之前进行缓冲，这样交给后端的就是一个完整的 HTTP请求，从而提高后端的效率，而不是断断续续的传递(互联网上连接速度一般比较慢)。 同样， Nginx 也可以把response 缓冲，同样也是减轻后端的压力。

9.2Nginx 的作用

- 反向代理

  是用来代理服务器的，代理我们要访问的目标服务器。 代理服务器接受请求，然后将请求转发给内部网络的服务器(集群化)， 并将从服务器上得到的结果返回给客户端，此时代理服务器对外就表现为一个服务器。

- 负载均衡

  多在高并发情况下需要使用。其原理就是将数据流量分摊到多个服务器执行，减轻每台服务器的压力，多台服务器(集群)共同完成工作任务，从而提高了数据的吞吐量。

- 动静分离

  Nginx提供的动静分离是指把动态请求和静态请求分离开，合适的服务器处理相应的请求，使整个服务器系统的性能、效率更高。Nginx可以根据配置对不同的请求做不同转发，这是动态分离的基础。静态请求对应的静态资源可以直接放在Nginx上做缓冲，更好的做法是放在相应的缓冲服务器上。动态请求由相应的后端服务器处理。

9.3怎么实现Nginx的高可用

使用Nginx+keepalived双机热备（主从模式）

Keepalived 是一种高性能的服务器高可用或热备解决方案，Keepalived 可以用来防止服务器单点故障的发生，通过配合 Nginx 可以实现 web 前端服务的高可用。

三,缓存相关

1.在项目中哪部分业务用到缓存?

文章,活动, 首页中广告、 搜索面板数据、 购物车等

2.如何提高缓存的利用率？

做数据的缓存时， 因为数据量很大， 而且缓存是把数据保存到内存中， 此时不可能把所有的数据都放到缓存中。 所以需要设置数据缓存的有效期， 当用户访问到非热点数据后， 此数据放到缓存中， 当缓存到期后就从缓存中删除， 而且长时间不会添加到缓存。 而热点数据一旦从缓存中删除会马上又添加到缓存。 这样可以提高缓存的利用率， 同时也减轻了数据库的压力。

3.项目中使用到了缓存， 那么如何实现缓存同步的？

只要使用了缓存就涉及到缓存同步的问题。 缓存同步其实就是当缓存的信息发生变化，也就是对后台对缓存的数据进行增、 删、 改操作后， 数据库中的数据发生了变化同时要把缓存中的数据对应删除即可。 当页面再次请求数据时， 缓存中不能命中就会从数据库中查询并且添加到缓存中， 即实现了缓存同步。

4.Redis

4.1谈谈你对Redis的理解

Redis是用C语言开发的一个开源的高性能键值对（key-value）数据库，官方提供测试数据，50个并发执行100000个请求,读的速度是110000次/s,写的速度是81000次/s ，且Redis通过提供多种键值数据类型来适应不同场景下的存储需求，目前为止Redis支持的键值数据类型如下：

- 字符串类型 string

- 散列类型 hash
- 列表类型 list
- 集合类型 set
- 有序集合类型 sortedset

项目中使用redis一般都是作为缓存来使用的，缓存的目的就是为了减轻数据库的压力提高存取的效率。

4.2redis的应用场景

- 缓存（数据查询、短连接、新闻内容、商品内容等等）
- 任务队列。（秒杀、抢购、12306等等）
- 数据过期处理（可以精确到毫秒)
- 聊天室的在线好友列表
- 应用排行榜
- 网站访问统计
- 分布式集群架构中的session分离

4.3redis优点/缺点

Redis 的优点

- 性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s 。
- 丰富的数据类型 – Redis支持string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和hash（哈希类型） 数据类型操作。
- 使用简单 – Redis的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象(可简单理解为redis已经实现好多种数据结构，程序员无需再去实现，直接使用即可)。
- 原子性 – Redis的所有操作都是原子性的，同时Redis还支持对几个操作全并后的原子性执行。
- 丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性。
- 支持数据的备份 – 即master-slave模式的数据备份(主从复制)。
- 支持数据的持久化 – 可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。(所以在对不同数据集进行高速读写时需要权衡内存，应为数据量不能大于硬件内存)

Redis 的缺点

- Redis不具备自动容错和恢复功能，主机从机的宕机都会导致前端部分读写请求失败，需要等待机器重启或者手动切换前端的IP才能恢复。
- 主机宕机，宕机前有部分数据未能及时同步到从机，切换IP后还会引入数据不一致的问题，降低了系统的可用性。
- Redis的主从复制采用全量复制，复制过程中主机会fork出一个子进程对内存做一份快照，并将子进程的内存快照保存为文件发送给从机，这一过程需要确保主机有足够多的空余内存。若快照文件较大，对集群的服务能力会产生较大的影响，而且复制过程是在从机新加入集群或者从机和主机网络断开重连时都会进行，也就是网络波动都会造成主机和从机间的一次全量的数据复制，这对实际的系统运营造成了不小的麻烦。
- Redis较难支持在线扩容，在集群容量达到上限时在线扩容会变得很复杂。为避免这一问题，运维人员在系统上线时必须确保有足够的空间，这对资源造成了很大的浪费。

4.4Redis是nosql 数据库,是否适合存储大数据?

Redis 是 nosql 数据库， 但是 redis 是 key-value 形式的 nosql 数据库， 数据是存储到内存中的， 适合于快速存取一般作为缓存使用。 所以不适合于大数据的存储。 并且 redis 是单线程的如果某个操作进行大数据的存储的话其他的进程都处于等待状态， 这样就降低了性能所以在 redis 中不适合于大数据的存储。 如果是类似商品评论这样的价值不高的大批量数据，我们的做法是采用 mongodb。

4.5Redis是个单线程程序,速度为什么会这么快

1. 完全基于内存，大部分请求都是纯粹的内存操作，非常快速，数据存储在内存中，例如HashMap，优势是查找和操作的时间复杂度 都是O(1)；
2. 数据结构简单，对数据操作也简单，Redis中的数据结构是专门进行设计的；
3. 采用单线程，避免了不必要的上下文切换和竞争条件，也不存在多进程或多线程导致的切换而消耗cpu，不用去考虑各种锁的问题，不存在加锁释放锁操作，没有因为可能出现死锁而导致的性能消耗；
4. 使用异步非阻塞IO；(重点说)

   备注:

   同步/异步:是否主动读写数据;阻塞/非阻塞:是否需要等待.

   同步：执行一个操作之后，等待结果，然后才继续执行后续的操作。

   异步：执行一个操作后，可以去执行其他的操作，然后等待通知再回来执行刚才没执行完的操作。

   阻塞：进程给CPU传达一个任务之后，一直等待CPU处理完成，然后才执行后面的操作。

   非阻塞：进程给CPU传达任务后，继续处理后续的操作，隔断时间再来询问之前的操作是否完成。这样的过程其实也叫轮询。
5. 底层自己构建了VM机制，因为一般系统调用系统函数的话会浪费一定的时间去移动和请求。

5.MongDB

5.1谈谈你对MongDB的理解

MongoDB 是一个跨平台的，面向文档的数据库，是当前 NoSQL 数据库产品中最热门 的一种。它介于关系数据库和非关系数据库之间，是非关系数据库当中功能最丰富，最像关系数据库的产品。它支持的数据结构非常松散，是类似 JSON 的 BSON 格式，因此可以存 储比较复杂的数据类型。

MongDB适合用于海量数据的访问效率提升

5.2在项目的哪些场景下使用MongoDB

吐槽 、文章评论, 贴吧等, 这类数据都有相同的特点

- 数据量大
- 写入操作频繁
- 价值较低

对于这样的数据，我们更适合使用MongoDB来实现数据的存储 .

5.3MongoDB和Redis区别?分别适用哪些场景?

比较指标	MongoDB	Redis	比较说明
实现语言	c++	c/c++	-
协议	BSON,自定义二进制	类telnet(TCP/IP)	-
性能	依赖内存,TPS较高	依赖内存,TPS非常高	Redis优于MongoDB(TPS 每秒事务处理量)
可操作性	丰富的数据表达,索引;最类似于关系型数据库,支持丰富的查询语句	数据丰富,较少的IO	MongoDB优于Redis
内存及存储	适合大数据量存储,依赖系统虚拟内存,采用镜像文件存储;内存占用率比较高,官方建议独立部署在64位系统	Redis2.0后支持虚拟内存特性(VM) 突破物理内存限制;数据可以设置时效性,类似于memcache	不同的应用场景,各有千秋
可用性	支持master-slave,replicatset(内部采用paxos选举算法,自动故障恢复),auto sharding机制,对客户端屏蔽了故障转移和切片机制	依赖客户端来实现分布式读写;主从复制时,每次从节点重新连接主节点都要依赖整个快照,无增量复制;不支持auto sharding,需要依赖程序设定一致性hash机制	MongoDB优于Redis；单点问题上,MongoDB应用简单,相对用户透明,Redis比较复杂,需要客户端主动解决.(MongoDB一般使用replicasets和sharding相结合,replicasets侧重高可用性以及高可靠,sharding侧重性能,水平扩展)
可靠性	从1.8版本后,采用binlog方式(类似Mysql) 支持持久化	依赖快照进行持久化;AOF增强可靠性;增强性的同时,影响访问性能	mongodb在启动时，专门初始化一个线程不断循环（除非应用crash掉），用于在一定时间周期内来从defer队列中获取要持久化的数据并写入到磁盘的journal(日志)和mongofile(数据)处，当然它不是在用户添加记录时就写到磁盘上
一致性	不支持事务,靠客户端保证	支持事务,比较脆,仅能保证事务中的操作按顺序执行	Redis优于MongoDB
数据分析	内置数据分析功能(mapreduce)	不支持	MongoDB优于Redis

比较指标	MongoDB	Redis	比较说明
应用场景	海量数据的访问效率提升	较小数据量的性能和运算	MongoDB优于Redis

6.项目中是如何使用缓存的

1. 使用Spring Data Redis
2. 使用SpringCache整合Redis, 相关的注解

   @Cacheable:使用这个注解的方法在执行后会缓存其返回结果。
   @CacheEvict:使用这个注解的方法在其执行前或执行后移除Spring Cache中的某些元素。

四,搜索相关

1.你们系统中搜索是怎么实现的

- 电商项目(eg: 品优购)

  我们使用的是 solr 作为全文检索服务器， 实现搜索功能。 我们在 solr 中配置跟业务相关的业务域， 从数据库中把相关的数据导入到索引库中。例如商品搜索功能就把商品表中的数据导入到索引库中。 然后使用 solr 实现商品搜索， 然后在页面中把搜索结果展示出来。我们使用的是 SpringDataSolr 框架来实现对 solr 的操作。

- 社交类型项目(eg: 十次方)

  我们使用的是Elasticsearch. 从数据库中把相关的数据导入到索引库中. 例如把文章表的数据导入到索引库里面.把文章的标题, 摘要进行分词,存储,进行索引. 然后在页面中把搜索结果展示出来, 我们使用的是Spring Data ElasticSearch 框架来实现对 ElasticSearch的操作

2.谈谈你对分词,索引,存储的理解

分词(tokenized): 是否分词，就看在搜索时候是要整体匹配还是单词匹配. 单词匹配就需要分词

- 适合进行分词的: 商品名称、商品描述等，这些内容用户要输入关键字搜索，由于搜索的内容格式大、内容多需要分词后将语汇单元建立索引
- 不适合进行分词的:商品id、订单号、身份证号等

索引(indexed): 是否进行索引, 就看在搜索时候是否需要被搜索到; 需要被搜索到, 就需要进行索引

- 适合进行索引的: 商品名称、商品描述分析后进行索引，订单号、身份证号不用分词但也要索引，这些将来都要作为查询条件。
- 不适合进行索引的: 图片路径、文件路径等，不用作为查询条件的不用索引。

存储(stored): 是否存储, 就看在搜索的结果页面上是否需要展示. 需要展示,就要存储

- 适合进行存储的：商品名称、订单号，凡是将来要从Document中获取的Field都要存储。

- 不适合进行存储的: 商品描述，内容较大不用存储。如果要向用户展示商品描述可以从系统的关系数据库中获取。

3.如何处理数据量大、 并发量高的搜索？

如果要搜索的内容数据量很大并且并发量很高的情况下， 一个 solr 服务是不能满足要求的， 所以此时需要 SolrCloud 来解决。 SolrCloud 也就是 solr 的分布式解决方案。 是zookeeper+solr 实现的。

如果使用ES, 就使用ES集群

4.如何实现索引库和数据库的同步？

- 方式一(solar): 通过solr连接数据库，实现数据的定时同步.
- 方式二(Elasticsearch): 使用 Logstash. Logstash 是一个开源的数据收集引擎，它具有备实时数据传输能力。它可以统一过滤来自不同源的数据，并按照开发者的制定的规范输出到目的地。Logstash 通过管道进行运作，管道有两个必需的元素，输入和输出，还有一个可选的元素，过滤器。我们编写配置文件,通过corn表达式, 定时的从mysql查询出数据作为输入, 把索引库作为输出.
- 方式三: 使用消息中间件, 在数据库数据修改之前，发送消息, 同步索引库

5.分词器

5.1你们项目如何实现索引库的分词的

在项目中搜索功能使用 solr 实现， 在 solr 中配置中文分析器， 我们使用 IKAnalyzer 来实现中文分词。 IK里面提供了两种算法 最少切分 最细切分. 如果是一些新的词汇， 为了查询的准确性， 我们会把新的关键词添加到 IKAnalyzer 的扩展词典中。

5.2IK分词器的原理

IK分词器的分词原理本质上是词典分词。先在内存中初始化一个词典，然后在分词过程中逐个读取字符，和字典中的字符相匹配，把文档中的所有的词语拆分出来的过程。

6.Solr

6.1谈谈你对solr的理解

Solr是一个开源搜索平台，用于构建搜索应用程序。 它建立在Lucene(全文搜索引擎)之上. Solr是一个可扩展的，可部署，搜索/存储引擎，优化搜索大量以文本为中心的数据。

6.2 Solr的原理

Solr是基于Lucene开发的全文检索服务器，而Lucene就是一套实现了全文检索的api，其本质就是一个全文检索的过程。全文检索就是把原始文档根据一定的规则拆分成若干个关键词，然后根据关键词创建索引，当查询时先查询索引找到对应的关键词，并根据关键词找到对应的文档，也就是查询结果，最终把查询结果展示给用户的过程。

Solr对外提供标准的http接口来实现对数据的索引的增删改查。在 Solr 中，用户通过向部署在servlet 容器中的 Solr Web 应用程序发送 HTTP 请求来启动索引和搜索。Solr 接受请求，确定要使用的适当 SolrRequestHandler，然后处理请求。通过 HTTP 以同样的方式返回响应。默认配置返回Solr 的标准 XML 响应，也可以配置Solr 的备用响应格式.

7.Solr和Elasticsearch的区别

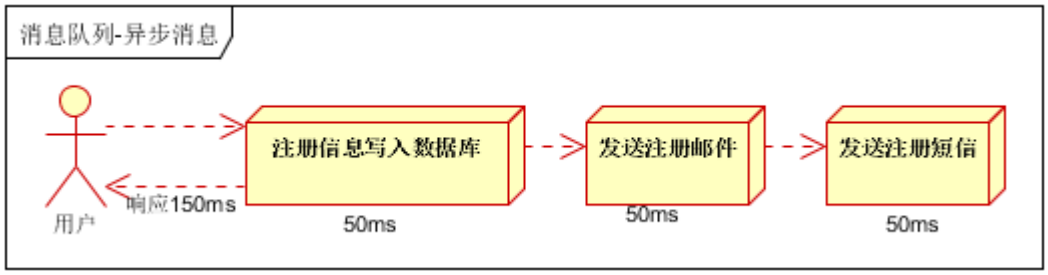
- Solr 利用 Zookeeper 进行分布式管理，而 Elasticsearch 自身带有分布式协调管理功能;
- Solr 支持更多格式的数据，而 Elasticsearch 仅支持json文件格式；
- Solr 官方提供的功能更多，而 Elasticsearch 本身更注重于核心功能，高级功能多有第三方插件提供；
- Solr 在传统的搜索应用中表现好于 Elasticsearch，但在处理实时搜索应用时效率明显低于 Elasticsearch。
- Solr 是传统搜索应用的有力解决方案，但 Elasticsearch 更适用于新兴的实时搜索应用。

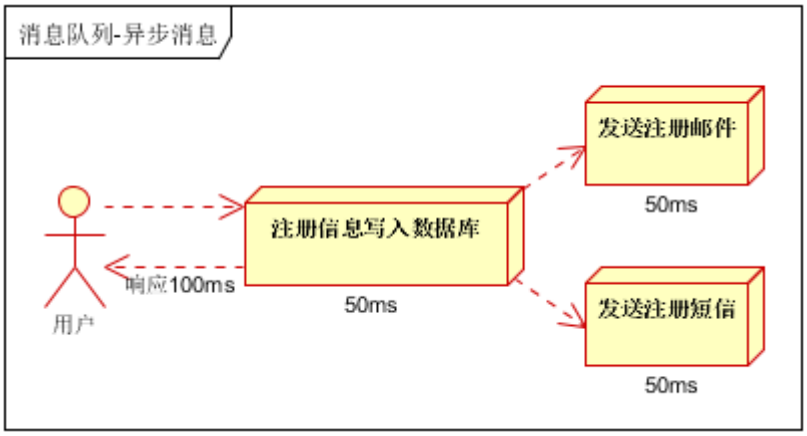
五,消息中间件相关

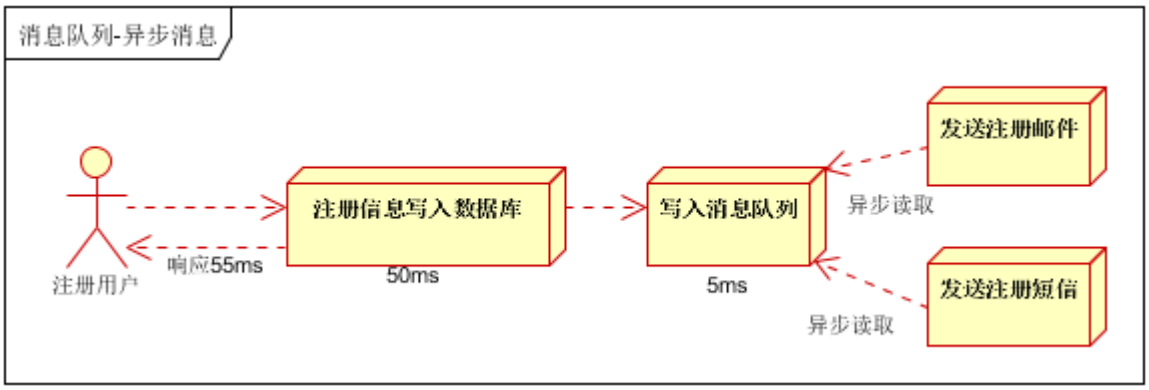
1.消息中间件应用场景

消息中间件是关注于数据的发送和接收，利用高效可靠的异步消息传递机制集成分布式系统。

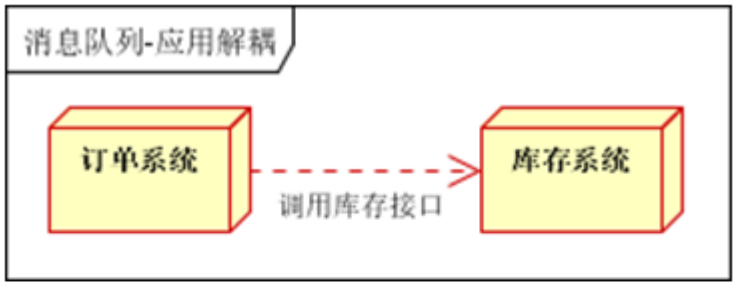
- 异步处理

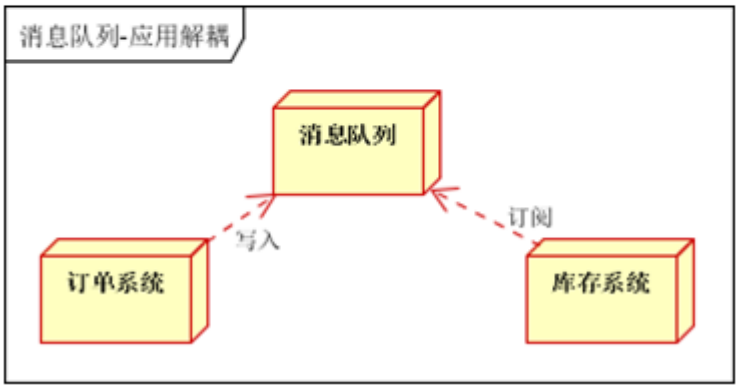




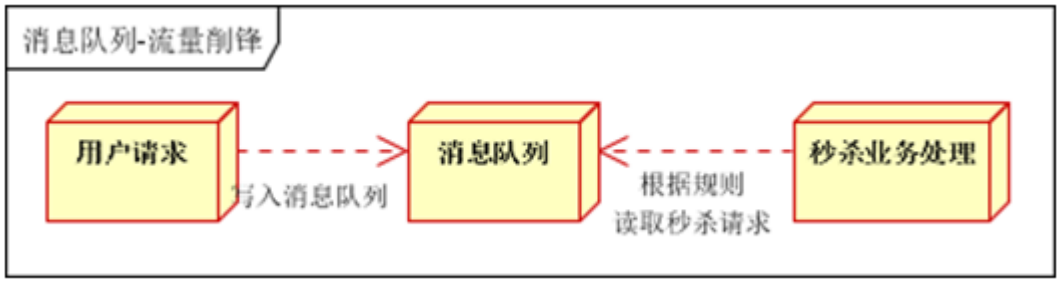


- 应用解耦





- 流量削峰



2.在你项目中的哪些业务使用到了消息中间件

- 应用场景 1： 当后台系统对商品数据进行添加、 删除、 修改后， 将会发送一个消息变化的消息， 此消息通过 topic 进行通信， 有多个消费端， :商品详情页面的静态页面会重新生成。
- 应用场景 2： 用户注册时， 向用户注册发送短信验证码， 采用 queue 方式通信。 消费端调用阿里大于短信接口进行短信的发送。
- 应用场景 3: 索引库数据和数据库进行同步.

3.Activemq

3.1Activemq 有几种消息通信方式？

使用 MQ 中间件有两种通信方式 queue 和 topic。

Queue 可以实现点到点之间的通信， 可以有多个 Producer 也可以有多个 Consumer， 但是消息只能被一个 Consumer 接收，一旦消息被消费后就没有了。

Topic 可以实现类似广播的通信方式， 可以有多个 Producer 和多个 Consumer， 一旦有Producer 发送消息后， 此消息可以被所有 Consumer 接收。

3.2 ActiveMQ的优点

1. 消息不丢失：服务器挂掉,消息会保存到mq中间件中，当消费者服务器恢复后就会重新发过去，消息不会丢失.
2. 异步处理：比如一个商城用户购买产品后,后台会去更新数据库,然后响应给客户端.如果在高并发的情况下，这样更新数据库响应客户端会变慢，可以使用mq消息队列的消费者进程中获取数据来进行异步写数据，由于消息对垒的服务处理速度远快于数据库，因此响应延迟能得到有效改善.

3.3AcitveMQ的作用和工作原理

Activemq的作用就是系统之间进行通信。当然可以使用其他方式进行系统间通信，如果使用Activemq的话可以对系统之间的调用进行解耦，实现系统间的异步通信。

工作原理就是生产者生产消息，把消息发送给activemq。Activemq接收到消息，然后查看有多少个消费者，然后把消息转发给消费者，此过程中生产者无需参与。消费者接收到消息后做相应的处理和生产者没有任何关系.

3.3 activeMQ如果数据提交不成功怎么办

Activemq有两种通信方式，点到点形式和发布订阅模式。

如果是点到点模式的话，如果消息发送不成功此消息默认会保存到activemq服务端,直到有消费者将其消费，所以此时消息是不会丢失的。

如果是发布订阅模式的通信方式，默认情况下只通知一次，如果接收不到此消息就没有了。这种场景只适用于对消息送达率要求不高的情况。如果要求消息必须送达不可以丢失的话，需要配置持久订阅。每个订阅端定义一个id，在订阅是向activemq注册。发布消息和接收消息时需要配置发送模式为持久化。此时如果客户端接收不到消息，消息会持久化到服务端，直到客户端正常接收后为止。

3.4ActiveMQ消息丢失如何解决

用持久化消息，或者非持久化消息及时处理不要堆积，或者启动事务，启动事务后，commit()方法会负责任的等待服务器的返回，也就不会关闭连接导致消息丢失了。

3.5MQ发送消息失败怎么办

可以设置重试机制，如果重试多次会出现重复消息，只需要保证数据的最终一致性即可，(利用幂等的理论，通过每次接收消息设置版本号来判断是否可以进行消费。并且在生产者和消费者两端需要记录消息消费流水号。辅以监控系统监控。再通过人工干预的手段保证最终一致.)

3.6 activemq如何确认消息发送成功

activemq通过应答模式来确认是否发送或者消息消费成功

4.RabbitMQ

4.1RabbitMQ 有几种消息通信方式？

RabbitMQ 有四种通讯方式：direct(直接模式)、fanout(分列模式)、topic(主题模式)、headers(头模式) 。

direct(直接模式): 它是完全匹配、单播的模式. 任何发送到Direct Exchange的消息都会被转发到RouteKey中指定的Queue。

fanout(分列模式): 每个发到 fanout 类型交换器的消息都会分到所有绑定的队列上去。fanout 交换器不处理路由键，只是简单的将队列绑定到交换器上，每个发送到交换器的消息都会被转发到与该交换器绑定的所有队列上。很像子网广播，每台子网内的主机都获得了一份复制的消息。fanout 类型转发消息是最快的。

JMS	AMQP

topic(主题模式): topic交换器通过模式匹配分配消息的路由键属性，将路由键和某个模式进行匹配，此时队列需要绑定到一个模式上。它将路由键和绑定键的字符串切分成单词，这些单词之间用点隔开。它同样也会识别两个通配符： 符号“#”和符号“*”。#匹配0个或多个单词，*匹配一个单词。

4.2 Activemq 和RabbitMQ 区别

	JMS	AMQP
定义	Java api	高级消息队列协议
跨语言	否	是
跨平台	否	是
Model	提供两种消息模型： （1）、Peer-2-Peer （2）、Pub/sub	提供了五种消息模型： （1）、direct exchange （2）、fanout exchange （3）、topic change （4）、headers exchange （5）、system exchange 本质来讲，后四种和JMS的pub/sub模型没有太大差别，仅是在路由机制上做了更详细的划分；
支持消息类型	多种消息类型： TextMessage MapMessage BytesMessage StreamMessage ObjectMessage Message （只有消息头和属性）	byte[] 当实际应用时，有复杂的消息，可以将消息序列化后发送。
综合评价	JMS 定义了JAVA API层面的标准；在java体系中，多个client均可以通过JMS进行交互，不需要应用修改代码，但是其对跨平台的支持较差；	AMQP定义了wire-level层的协议标准；天然具有跨平台、跨语言特性。
产品	ActiveMQ	RabbitMQ

4.3 如何确保消息正确地发送至RabbitMQ

RabbitMQ使用发送方确认模式，确保消息正确地发送到RabbitMQ。

发送方确认模式：将信道设置成confirm模式（发送方确认模式），则所有在信道上发布的消息都会被指派一个唯一的ID。一旦消息被投递到目的队列后，或者消息被写入磁盘后（可持久化的消息），信道会发送一个确认给生产者（包含消息唯一ID）。如果RabbitMQ发生内部错误从而导致消息丢失，会发送一条nack（not acknowledged，未确认）消息。

发送方确认模式是异步的，生产者应用程序在等待确认的同时，可以继续发送消息。当确认消息到达生产者应用程序，生产者应用程序的回调方法就会被触发来处理确认消息。

4.4 如何确保消息接收方消费了消息

接收方消息确认机制：消费者接收每一条消息后都必须进行确认（消息接收和消息确认是两个不同操作）。只有消费者确认了消息，RabbitMQ才能安全地把消息从队列中删除。

这里并没有用到超时机制，RabbitMQ仅通过Consumer的连接中断来确认是否需要重新发送消息。也就是说，只要连接不中断，RabbitMQ给了Consumer足够长的时间来处理消息。

下面罗列几种特殊情况：

- 如果消费者接收到消息，在确认之前断开了连接或取消订阅，RabbitMQ会认为消息没有被分发，然后重新分发给下一个订阅的消费者。（可能存在消息重复消费的隐患，需要根据bizId去重）
- 如果消费者接收到消息却没有确认消息，连接也未断开，则RabbitMQ认为该消费者繁忙，将不会给该消费者分发更多的消息。

4.5如何避免消息重复投递或重复消费

在消息生产时，MQ内部针对每条生产者发送的消息生成一个inner-msg-id，作为去重和幂等的依据（消息投递失败并重传），避免重复的消息进入队列；在消息消费时，要求消息体中必须要有一个bizId（对于同一业务全局唯一，如支付ID、订单ID、帖子ID等）作为去重和幂等的依据，避免同一条消息被重复消费。

4.6如何确保消息不丢失

消息持久化的前提是：将交换器/队列的durable属性设置为true，表示交换器/队列是持久交换器/队列，在服务器崩溃或重启之后不需要重新创建交换器/队列（交换器/队列会自动创建）。

如果消息想要从Rabbit崩溃中恢复，那么消息必须：

- 在消息发布前，通过把它的 “投递模式” 选项设置为2（持久）来把消息标记成持久化
- 将消息发送到持久交换器
- 消息到达持久队列

RabbitMQ确保持久性消息能从服务器重启中恢复的方式是，将它们写入磁盘上的一个持久化日志文件，当发布一条持久性消息到持久交换器上时，Rabbit会在消息提交到日志文件后才发送响应（如果消息路由到了非持久队列，它会自动从持久化日志中移除）。一旦消费者从持久队列中消费了一条持久化消息，RabbitMQ会在持久化日志中把这条消息标记为等待垃圾收集。如果持久化消息在被消费之前RabbitMQ重启，那么Rabbit会自动重建交换器和队列（以及绑定），并重播持久化日志文件中的消息到合适的队列或者交换器上。

4.7消息基于什么传输

由于TCP连接的创建和销毁开销较大，且并发数受系统资源限制，会造成性能瓶颈。RabbitMQ使用信道的方式来传输数据。信道是建立在真实的TCP连接内的虚拟连接，且每条TCP连接上的信道数量没有限制。

4.8消息怎么路由

从概念上来说，消息路由必须有三部分：交换器、路由、绑定。生产者把消息发布到交换器上；绑定决定了消息如何从路由器路由到特定的队列；消息最终到达队列，并被消费者接收。

1. 消息发布到交换器时，消息将拥有一个路由键（routing key），在消息创建时设定。
2. 通过队列路由键，可以把队列绑定到交换器上。
3. 消息到达交换器后，RabbitMQ会将消息的路由键与队列的路由键进行匹配（针对不同的交换器有不同的路由规则）。如果能够匹配到队列，则消息会投递到相应队列中；如果不能匹配到任何队列，消息将进入 “黑洞”。

常用的交换器主要分为一下三种：

- direct：如果路由键完全匹配，消息就被投递到相应的队列
- fanout：如果交换器收到消息，将会广播到所有绑定的队列上
- topic：可以使来自不同源头的消息能够到达同一个队列。 使用topic交换器时，可以使用通配符，比如：“*” 匹配特定位置的任意文本， “.” 把路由键分为了几部分，“#” 匹配所有规则等。特别注意：发往topic交换器的消息不能随意的设置选择键（routing_key），必须是由"."隔开的一系列的标识符组成。

六,登录,认证机制和加密

算法	特点	有效破解方式	破解难度	其它

1.项目里面有没有使用第三方登录?

使用过. 第三方登录是基于OAuth协议的. 我们使用的是微信扫码登陆. 先去微信开放平台进行账号的注册, 进行开发者资质认证,创建网站应用等待审核, 审核成功后会得到AppID和 AppSecret. 下面是我们项目中使用微信扫码登陆流程

1. 查阅微信开发者文档, 在网站页面生成一个二维码
2. 通过微信扫描二维码, 第三方发起微信授权登录请求，微信用户允许授权第三方应用后，微信会拉起应用或重定向到第三方网站，并且带上授权临时票据code参数；
3. 通过code参数加上AppID和AppSecret等，通过API换取access_token；
4. 通过access_token进行接口调用，获取用户基本数据资源或帮助用户实现基本操作。

2.之前的项目有没有和安卓,ios等客户端对接, 用户状态怎么处理的?

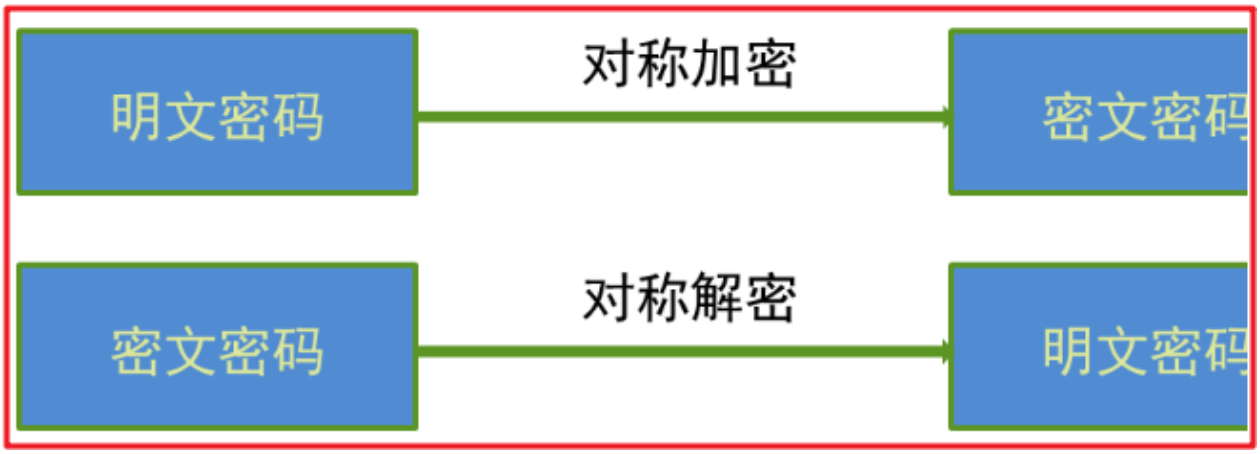
对接过,我们采取的是token. token说白了就是在服务端通过密码和加密算法获得到的一个字符串，我们项目中采取的是JJWT签发token的, 并且设置了过期时间,为7天. 也就是说7天需要再重新登录一次. token的工作流程:

1. 客户端使用用户名跟密码请求登录
2. 服务端收到请求，去验证用户名与密码
3. 验证成功后，服务端会签发一个 Token，再把这个 Token 发送给客户端
4. 客户端(安卓,ios)收到 Token 以后可以把它存储起来，比如放在 Cookie 里
5. 客户端每次向服务端请求资源的时候需要带着服务端签发的 Token
6. 服务端收到请求，然后去验证客户端请求里面带着的 Token，如果验证成功，就向客户端返回请求的数据

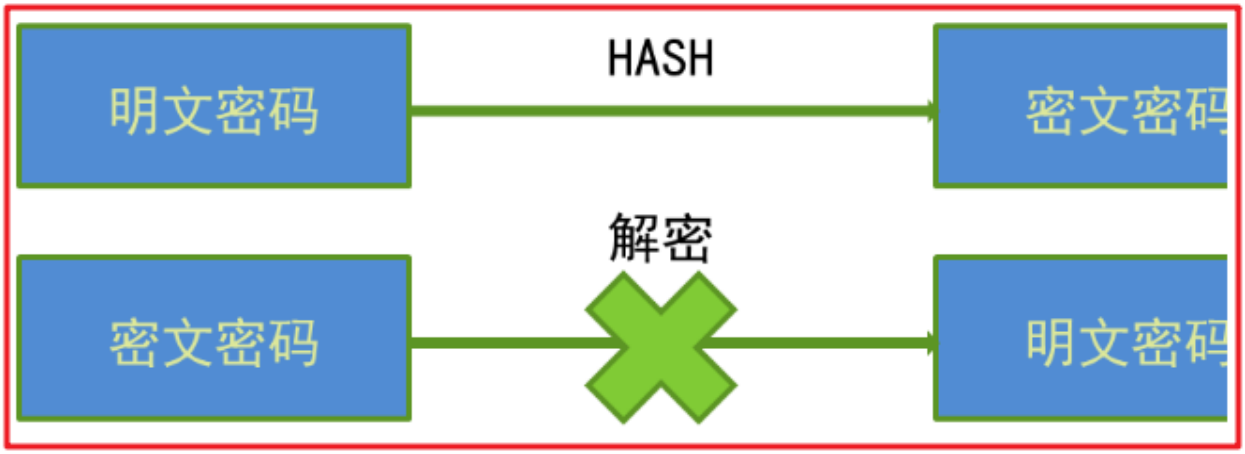
3.用户的密码采取的是什么加密方式? 常见的加密方式有哪些

算法	特点	有效破解方式	破解难度	其它
明文保存	实现简单	无需破解	简单	
对称加密	可以解密出明文	获取密钥	中	需要确保密钥不泄露
单向HASH	不可解密	碰撞、彩虹表	中	
特殊HASH	不可解密	碰撞、彩虹表	中	需要确保“盐”不泄露
Pbkdf2	不可解密	无	难	需要设定合理的参数

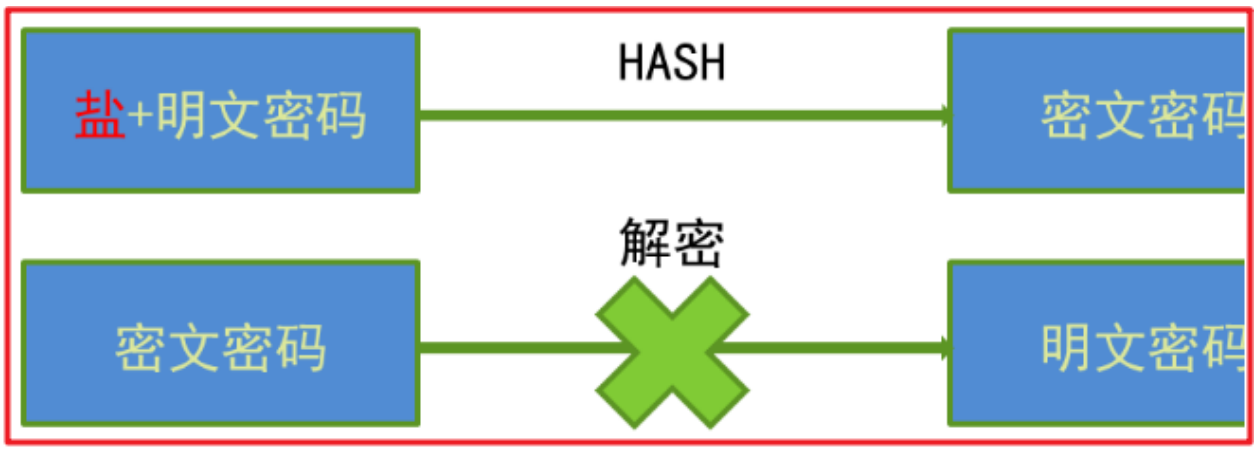
1. 使用对称加密算法来保存，比如3DES、AES等算法，使用这种方式加密是可以通过解密来还原出原始密码的，当然前提条件是需要获取到密钥。不过既然大量的用户信息已经泄露了，密钥很可能也会泄露，当然可以将一般数据和密钥分开存储、分开管理，但要完全保护好密钥也是一件非常复杂的事情，所以这种方式并不是很好的方式。



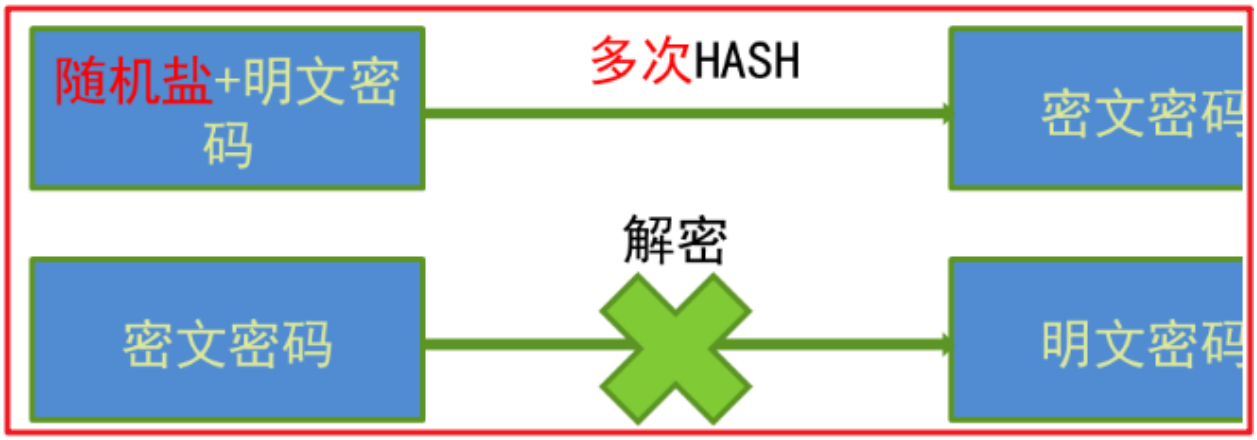
2. 使用MD5、SHA1等单向HASH算法保护密码，使用这些算法后，无法通过计算还原出原始密码，而且实现比较简单，因此很多互联网公司都采用这种方式保存用户密码，曾经这种方式也是比较安全的方式，但随着彩虹表技术的兴起，可以建立彩虹表进行查表破解，目前这种方式已经很不安全了。



3. 特殊的单向HASH算法，由于单向HASH算法在保护密码方面不再安全，于是有些公司在单向HASH算法基础上进行了加盐、多次HASH等扩展，这些方式可以在一定程度上增加破解难度，对于加了“固定盐”的HASH算法，需要保护“盐”不能泄露，这就会遇到“保护对称密钥”一样的问题，一旦“盐”泄露，根据“盐”重新建立彩虹表可以进行破解，对于多次HASH，也只是增加了破解的时间，并没有本质上的提升。



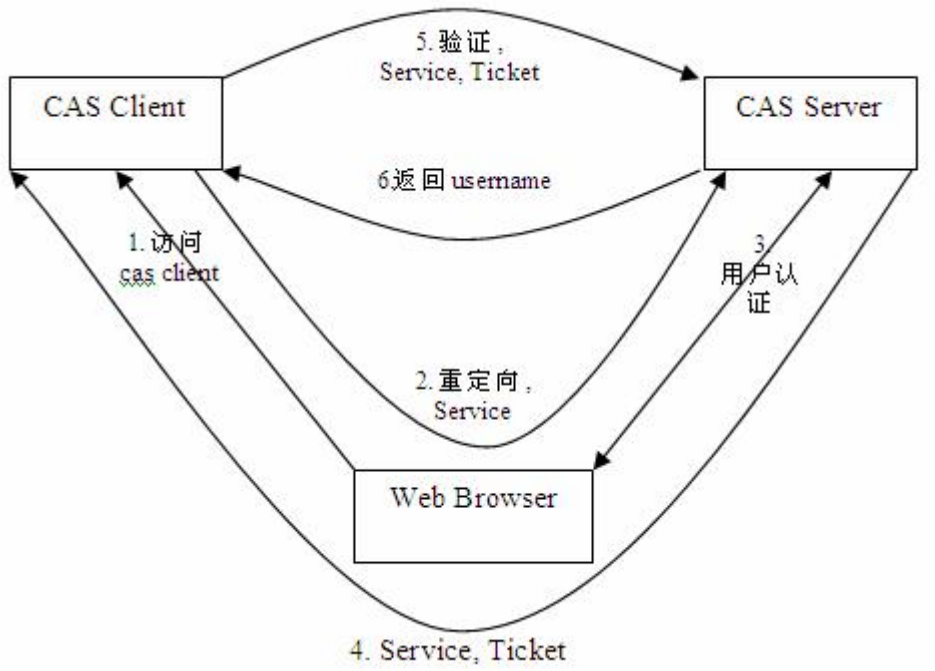
4. PBKDF2算法，该算法原理大致相当于在HASH算法基础上增加随机盐，并进行多次HASH运算，随机盐使得彩虹表的建表难度大幅增加，而多次HASH也使得建表和破解的难度都大幅增加。使用PBKDF2算法时，HASH算法一般选用sha1或者sha256，随机盐的长度一般不能少于8字节，HASH次数至少也要1000次，这样安全性才足够高。一次密码验证过程进行1000次HASH运算，对服务器来说可能只需要1ms，但对于破解者来说计算成本增加了1000倍，而至少8字节随机盐，更是把建表难度提升了N个数量级，使得大批量的破解密码几乎不可行，该算法也是美国国家标准与技术研究院推荐使用的算法。



5. bcrypt、scrypt等算法，这两种算法也可以有效抵御彩虹表，使用这两种算法时也需要指定相应的参数，使破解难度增加。

4.CAS

4.1 CAS工作流程



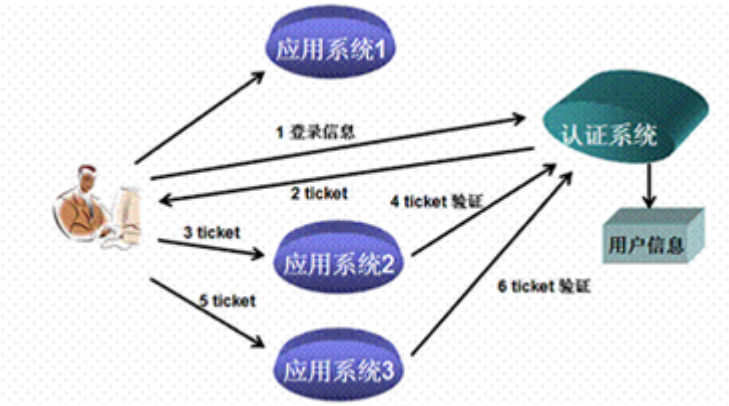
1. 访问服务：SSO客户端发送请求访问应用系统提供的服务资源。
2. 定向认证：SSO客户端会重定向用户请求到SSO服务器。
3. 用户认证：用户身份认证。
4. 发放票据：SSO服务器会产生一个随机的Service Ticket。
5. 验证票据：SSO服务器验证票据Service Ticket的合法性，验证通过后，允许客户端访问服务。
6. 传输用户信息：SSO服务器验证票据通过后，传输用户认证结果信息给客户端。

4.2 单点登录是怎么实现的

当用户第一次访问应用系统的时候，因为还没有登录，会被引导到认证系统中进行登录；

根据用户提供的登录信息，认证系统进行身份校验，如果通过校验，应该返回给用户一个认证的凭据－－ticket；

用户再访问别的应用的时候，就会将这个ticket带上，作为自己认证的凭据，应用系统接受到请求之后会把ticket送到认证系统进行校验，检查ticket的合法性。如果通过校验，用户就可以在不用再次登录的情况下访问应用系统2和应用系统3了。



4.3 如果一个用户在两台电脑登录，会有什么问题？怎么解决

出现的问题: 会存在同一个用户登录两次，存在两个token的情况。

解决: 应当在用户以同一个账号进行登录时，可以登录成功，并把上一次登录的信息清除掉，即强制退出。也就是说，数据库中只存放最后登录的账号信息

实现步骤: 用到LoginListenner监听类、login登录方法以及在web.xml中配置监听类.

1. 记录用户的sessionid
2. 登录的时候删除本机以外的sessionid
3. 另一台的sessionid不存在自然就退出登录了

5.如果cookie被禁用了怎么办

1. 经常被使用的一种技术叫做URL重写，就是把session id直接附加在URL路径的后面。
2. 还有一种技术叫做表单隐藏字段。就是服务器会自动修改表单，添加一个隐藏字段，以便在表单提交时能够把session id传递回服务器

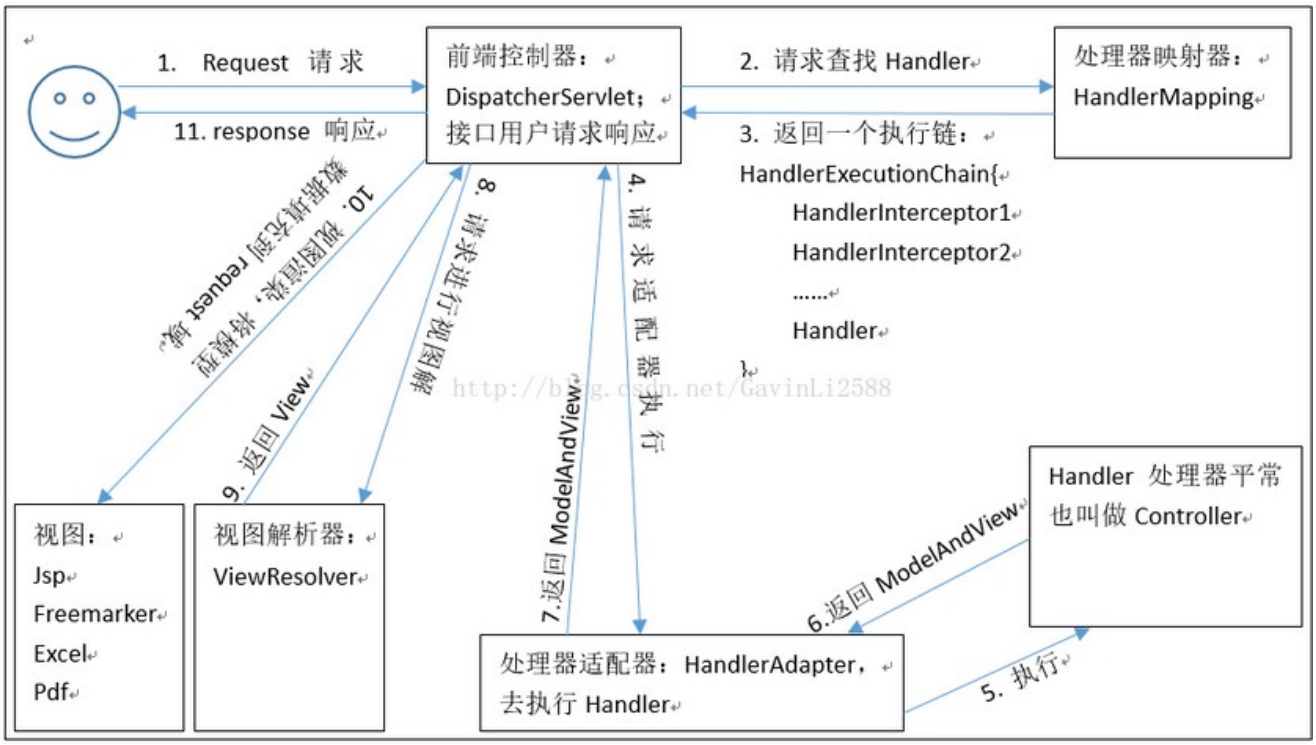
七,SSM,SSH

1.SpringMVC

1.1什么是Spring MVC ？简单介绍下你对springMVC的理解?

Spring MVC是一个基于MVC架构的用来简化web应用程序开发的应用开发框架，它是Spring的一个模块,无需中间整合层来整合，它和Struts2一样都属于表现层的框架。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，简化开发，减少出错，方便组内开发人员之间的配合

1.2SpringMVC的流程?



（1）用户发送请求至前端控制器DispatcherServlet；
（2） DispatcherServlet收到请求后，调用HandlerMapping处理器映射器，请求获取Handle；
（3）处理器映射器根据请求url找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet；
（4）DispatcherServlet通过HandlerAdapter处理器适配器调用处理器；
（5）执行处理器(Handler，也叫后端控制器)；
（6）Handler执行完成返回ModelAndView；
（7）HandlerAdapter将Handler执行结果ModelAndView返回给DispatcherServlet；
（8）DispatcherServlet将ModelAndView传给ViewResolver视图解析器进行解析；
（9）ViewResolver解析后返回具体View；
（10）DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）
（11）DispatcherServlet响应用户。

1.3Spring MVC的主要组键?

（1）前端控制器 DispatcherServlet（不需要程序员开发）

作用：接收请求、响应结果 相当于转发器，有了DispatcherServlet 就减少了其它组件之间的耦合度。

（2）处理器映射器HandlerMapping（不需要程序员开发）

作用：根据请求的URL来查找Handler

（3）处理器适配器HandlerAdapter

注意：在编写Handler的时候要按照HandlerAdapter要求的规则去编写，这样适配器HandlerAdapter才可以正确的去执行Handler。

（4）处理器Handler（需要程序员开发）

（5）视图解析器 ViewResolver（不需要程序员开发）

作用：进行视图的解析 根据视图逻辑名解析成真正的视图（view）

（6）视图View（需要程序员开发jsp）

View是一个接口， 它的实现类支持不同的视图类型（jsp，freemarker，pdf等等）

对比项目	SrpingMVC	Struts2	优势

1.4SpringMVC怎么样设定重定向和转发的？

（1）在返回值前面加"forward:"就可以让结果转发,比如"forward:user.do?name=method4"

（2）在返回值前面加"redirect:"就可以让返回值重定向,必如"redirect:http://www.baidu.com"

1.5SpringMVC优点

（1）它是基于组件技术的。全部的应用对象,无论控制器和视图,还是业务对象之类的都是 java组件.并且和Spring提供的其他基础结构紧密集成.

（2）不依赖于Servlet API(目标虽是如此,但是在实现的时候确实是依赖于Servlet的)

（3）可以任意使用各种视图技术,而不仅仅局限于JSP

（4） 支持各种请求资源的映射策略

（5）它应是易于扩展的

1.6SpringMVC和struts2区别

对比项目	SrpingMVC	Struts2	优势
国内市场情况	有大量用户，一般新项目启动都会选用springmvc	有部分老用户，老项目组，由于习惯了，一直在使用。	国内情况，springmvc的使用率已经超过Struts2
框架入口	基于servlet	基于filter	本质上没太大优势之分，只是配置方式不一样
框架设计思想	控制器基于方法级别的拦截，处理器设计为单实例	控制器基于类级别的拦截， 处理器设计为多实例	由于设计本身原因，造成了Struts2，通常来讲只能设计为多实例模式，相比于springmvc设计为单实例模式，Struts2会消耗更多的服务器内存。
参数传递	参数通过方法入参传递	参数通过类的成员变量传递	Struts2通过成员变量传递参数，导致了参数线程不安全，有可能引发并发的问题。
与spring整合	与spring同一家公司，可以与spring无缝整合	需要整合包	Springmvc可以更轻松与spring整合

1.7如何解决POST请求中文乱码问题，GET的又如何处理呢？

- POST, 在web.xml配置编码过滤器

```
<filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>utf-8</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

- GET, Tomcat8.0以后不需要处理. 8.0之前可以修改tomcat配置文件添加编码与工程编码一致

```
<ConnectorURIEncoding="utf-8" connectionTimeout="20000" port="8080" protocol="HTTP/1.1"
redirectPort="8443"/>
```

1.8 SpringMvc和Struts2的核心入口类分别是什么?

SpringMvc的是DispatchServlet,

Struts2的是StrutsPrepareAndExecuteFilter。

1.9SpringMvc怎么和AJAX相互调用的

通过Jackson框架就可以把Java里面的对象直接转化成Js可以识别的Json对象。具体步骤如下：

1. 导入jackson坐标

2. 把什么对象转成json，方法的返回值就是什么类型

3. 把方法返回值前面/方法上面 添加@ResponseBody

2.Spring

2.1Spring 框架中都用到了哪些设计模式

（1）代理模式—在AOP和remoting中被用的比较多。

（2）单例模式—在spring配置文件中定义的bean默认为单例模式。

（3）工厂模式—BeanFactory用来创建对象的实例。

（4）模板方法—用来解决代码重复的问题。比如. RestTemplate, JmsTemplate, JpaTemplate。

2.2谈谈你对Spring的IoC和依赖注入(di)的理解

1. IOC就是控制反转。就是对象的创建权反转交给Spring，由容器控制程序之间的依赖关系，作用是实现了程序的解耦合，而非传统实现中，由程序代码直接操控。(依赖)控制权由应用代码本身转到了外部容器，由容器根据配置文件去创建实例并管理各个实例之间的依赖关系，控制权的转移，是所谓反转，并且由容器动态的将某种依赖关系注入到组件之中。BeanFactory 是Spring IoC容器的具体实现与核心接口，提供了一个先进的配置机制，使得任何类型的对象的配置成为可能，用来包装和管理各种bean。

最直观的就是，IOC让对象的创建不用去new了，可以由spring自动生产，这里用的就是java的反射机制，通过反射在运行时动态的去创建、调用对象。spring就是根据配置文件在运行时动态的去创建对象，并调用对象的方法的。

2. DI机制（Dependency Injection，依赖注入）：可以说是IoC的其中一个内容，在容器实例化对象的时候主动的将被调用者（或者说它的依赖对象）注入给调用对象。比如对象A需要操作数据库，以前我们总是要在A中自己编写代码来获得一个Connection对象，有了 spring我们就只需要告诉spring，A中需要一个Connection，至于这个Connection怎么构造，何时构造，A不需要知道。在系统运行时，spring会在适当的时候制造一个Connection，然后像打针一样，注射到A当中，这样就完成了对各个对象之间关系的控制。

   我们可以通过构造方法方式注入, Set方法方法注入,P和C名称空间方式注入, SPEL方式注入,还可以通过注解注入

2.3 BeanFactory和ApplicationContext有什么区别？

BeanFactory和ApplicationContext是Spring的两大核心接口，而其中ApplicationContext是BeanFactory的子接口。它们都可以当做Spring的容器，生成Bean实例的，并管理容器中的Bean。

（1）BeanFactory：是Spring里面最底层的接口，提供了最简单的容器的功能，负责读取bean配置文档，管理bean的加载与实例化，维护bean之间的依赖关系，负责bean的生命周期，但是无法支持spring的aop功能和web应用。

（2）ApplicationContext接口作为BeanFactory的派生，因而具有BeanFactory所有的功能。而且ApplicationContext还在功能上做了扩展，以一种更面向框架的方式工作以及对上下文进行分层和实现继承，相较于BeanFactorty，ApplicationContext还提供了以下的功能：

- 默认初始化所有的Singleton，也可以通过配置取消预初始化。
- 继承MessageSource，因此支持国际化。
- 资源访问，比如访问URL和文件。
- 事件机制。
- 同时加载多个配置文件。
- 以声明式方式启动并创建Spring容器。
- 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。

（3）BeanFactroy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.

而ApplicationContext则相反，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。

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。

（4）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。

（5）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

2.4Spring支持的几种bean的作用域

Spring容器中的bean可以分为5个范围：

（1）singleton：这种bean范围是默认的，这种范围确保不管接受到多少个请求，每个容器中只有一个bean的实例，单例的模式由bean factory自身来维护。
（2）prototype：原形范围与单例范围相反，为每一个bean请求提供一个实例。
（3）request：在请求bean范围内会每一个来自客户端的网络请求创建一个实例，在请求完成以后，bean会失效并被垃圾回收器回收。
（4）Session：与请求范围类似，确保每个session中有一个bean的实例，在session过期后，bean会随之失效。

（5）global-session：global-session和Portlet应用相关。当你的应用部署在Portlet容器中工作时，它包含很多portlet。如果你想要声明让所有的portlet共用全局的存储变量的话，那么这全局变量需要存储在global-session中。全局作用域与Servlet中的session作用域效果相同

2.5谈谈你对Spring的AOP理解

AOP，一般称为面向方面（切面）编程，作为面向对象的一种补充，用于解剖封装好的对象内部，找出其中对多个对象产生影响的公共行为，并将其封装为一个可重用的模块，这个模块被命名为“切面”（Aspect），切面将那些与业务无关，却被业务模块共同调用的逻辑提取并封装起来，减少了系统中的重复代码，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AOP框架自动创建的AOP代理，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动态代理通过反射来接收被代理的类，并且要求被代理的类必须实现一个接口。JDK动态代理的核心是InvocationHandler接口和Proxy类。生成的代理对象的方法调用都会委托到InvocationHandler.invoke()方法，当我们调用代理类对象的方法时，这个“调用”会转送到invoke方法中，代理类对象作为proxy参数传入，参数method标识了我们具体调用的是代理类的哪个方法，args为这个方法的参数。

②如果目标类没有实现接口，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，覆盖方法时可以添加增强代码，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

2.6解释下AOP中的名词

- JoinPoint: 连接点

  类里面哪些方法可以被增强，这些方法称为连接点. 在spring的AOP中，指的是所有现有的方法。

- Pointcut: 切入点

  在类里面可以有很多方法被增强，但是实际开发中，我们只对具体的某几个方法而已，那么这些实际增强的方法就称之为切入点

- Advice: 通知/增强

增强的逻辑、称为增强，比如给某个切入点(方法) 扩展了校验权限的功能，那么这个校验权限即可称之为增强 或者是通知

通知分为:

前置通知： 在原来方法之前执行.

后置通知： 在原来方法之后执行. 特点: 可以得到被增强方法的返回值

环绕通知：在方法之前和方法之后执行. 特点:可以阻止目标方法执行

异常通知： 目标方法出现异常执行. 如果方法没有异常,不会执行. 特点:可以获得异常的信息

最终通知： 指的是无论是否有异常，总是被执行的。

- Aspect: 切面

  切入点和通知的结合。

2.7通知有哪些类型

（1）前置通知（Before advice）：在某连接点（join point）之前执行的通知，但这个通知不能阻止连接点前的执行（除非它抛出一个异常）。

（2）返回后通知（After returning advice）：在某连接点（join point）正常完成后执行的通知：例如，一个方法没有抛出任何异常，正常返回。
（3）抛出异常后通知（After throwing advice）：在方法抛出异常退出时执行的通知。
（4）后通知（After (finally) advice）：当某连接点退出的时候执行的通知（不论是正常返回还是异常退出）。

（5）环绕通知（Around Advice）：包围一个连接点（join point）的通知，如方法调用。这是最强大的一种通知类型。 环绕通知可以在方法调用前后完成自定义的行为。它也会选择是否继续执行连接点或直接返回它们自己的返回值或抛出异常来结束执行。

2.8Spring事务的种类和各自的区别

spring支持编程式事务管理和声明式事务管理两种方式：

（1）编程式事务管理使用TransactionTemplate或者直接使用底层的PlatformTransactionManager。对于编程式事务管理，spring推荐使用TransactionTemplate。

（2）声明式事务管理建立在AOP之上的。其本质是对方法前后进行拦截，然后在目标方法开始之前创建或者加入一个事务，在执行完目标方法之后根据执行情况提交或者回滚事务。声明式事务最大的优点就是不需要通过编程的方式管理事务，这样就不需要在业务逻辑代码中掺杂事务管理的代码，只需在配置文件中做相关的事务规则声明(或通过基于@Transactional注解的方式)，便可以将事务规则应用到业务逻辑中。

（3）显然声明式事务管理要优于编程式事务管理，这正是spring倡导的非侵入式的开发方式。声明式事务管理使业务代码不受污染，一个普通的POJO对象，只要加上注解就可以获得完全的事务支持。和编程式事务相比，声明式事务唯一不足地方是，后者的最细粒度只能作用到方法级别，无法做到像编程式事务那样可以作用到代码块级别。

2.9Spring优点

1.方便解耦，简化开发

通过Spring提供的IoC容器，我们可以将对象之间的依赖关系交由Spring进行控制，避免硬编码所造成的过度程序耦合。有了Spring，用户不必再为单实例模式类、属性文件解析等这些很底层的需求编写代码，可以更专注于上层的应用。

2.AOP编程的支持

通过Spring提供的AOP功能，方便进行面向切面的编程，许多不容易用传统OOP实现的功能可以通过AOP轻松应付。

3.声明式事务的支持

在Spring中，我们可以从单调烦闷的事务管理代码中解脱出来，通过声明式方式灵活地进行事务的管理，提高开发效率和质量。

4.方便程序的测试

可以用非容器依赖的编程方式进行几乎所有的测试工作，在Spring里，测试不再是昂贵的操作，而是随手可做的事情。例如：Spring对Junit4支持，可以通过注解方便的测试Spring程序。

5.方便集成各种优秀框架

Spring不排斥各种优秀的开源框架，相反，Spring可以降低各种框架的使用难度，Spring提供了对各种优秀框架（如Struts,Hibernate、Hessian、Quartz）等的直接支持。

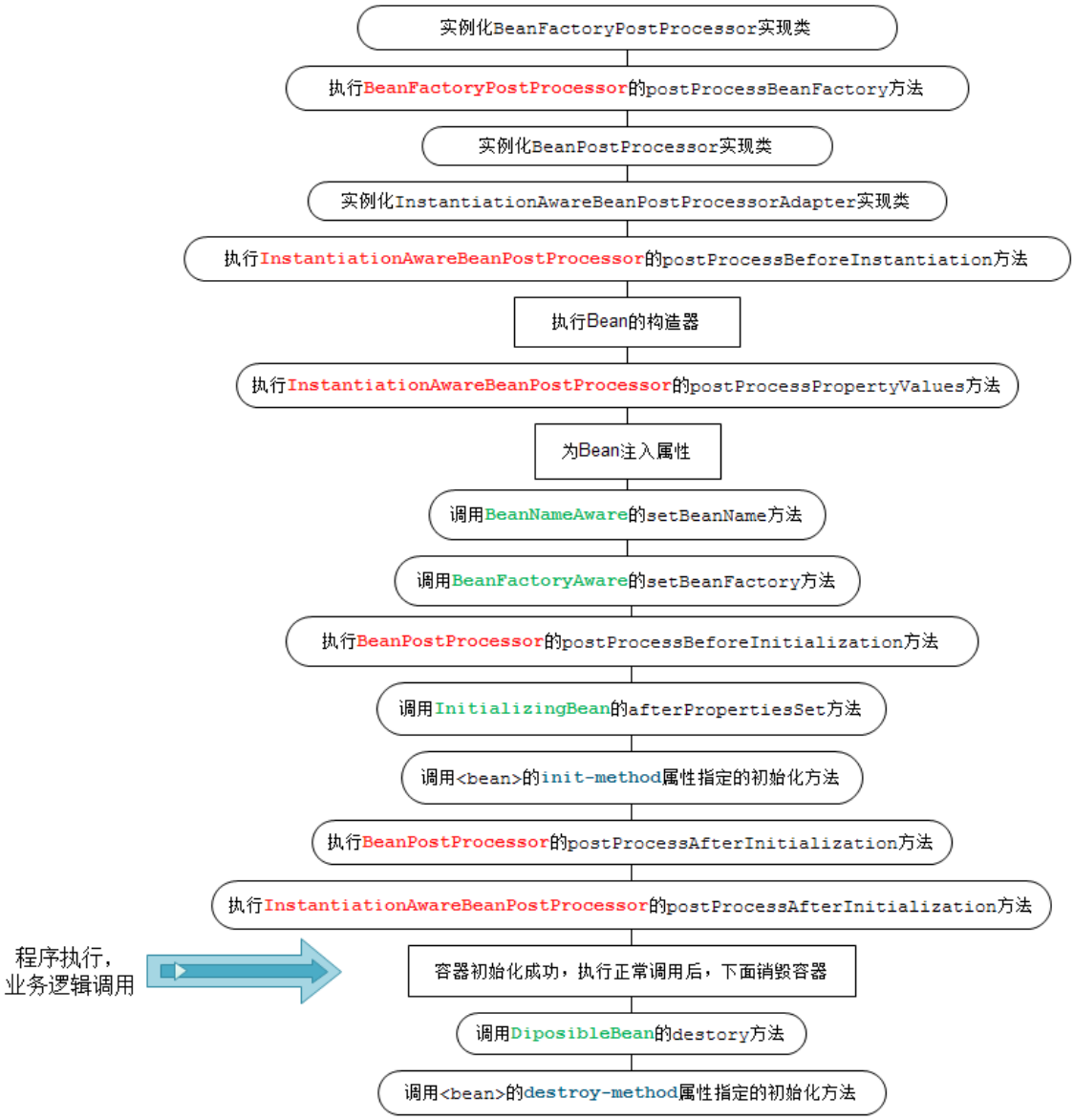
6.降低Java EE API的使用难度

Spring对很多难用的Java EE API（如JDBC，JavaMail，远程调用等）提供了一个薄薄的封装层，通过Spring的简易封装，这些Java EE API的使用难度大为降低。

2.10Spring框架中的单例Beans是线程安全的么？

Spring框架并没有对单例bean进行任何多线程的封装处理。关于单例bean的线程安全和并发问题需要开发者自行去搞定。但实际上，大部分的Spring bean并没有可变的状态(比如Serview类和DAO类)，所以在某种程度上说Spring的单例bean是线程安全的。如果你的bean有多种状态的话（比如 View Model 对象），就需要自行保证线程安全。
最浅显的解决办法就是将多态bean的作用域由“singleton”变更为“prototype”。

2.11Spring Bean的生命周期



3.MyBatis

3.1Mybait的优点和缺点

- 优点
  - 简单易学，容易上手（相比于Hibernate） —- 基于SQL编程；
  - JDBC相比，减少了50%以上的代码量，消除了JDBC大量冗余的代码，不需要手动开关连接；
  - 很好的与各种数据库兼容（因为MyBatis使用JDBC来连接数据库，所以只要JDBC支持的数据库MyBatis都支持，而JDBC提供了可扩展性，所以只要这个数据库有针对Java的jar包就可以就可以与MyBatis兼容），开发人员不需要考虑数据库的差异性。
  - 提供了很多第三方插件（分页插件 / 逆向工程）；
  - 能够与Spring很好的集成；
  - MyBatis相当灵活，不会对应用程序或者数据库的现有设计强加任何影响，SQL写在XML里，从程序代码中彻底分离，解除sql与程序代码的耦合，便于统一管理和优化，并可重用。
  - 提供XML标签，支持编写动态SQL语句。
  - 提供映射标签，支持对象与数据库的ORM字段关系映射。
  - 提供对象关系映射标签，支持对象关系组建维护。
- 缺点

- SQL语句的编写工作量较大，尤其是字段多、关联表多时，更是如此，对开发人员编写SQL语句的功底有一定要求。
- SQL语句依赖于数据库，导致数据库移植性差，不能随意更换数据库。

3.2MyBatis框架适用场合

（1）MyBatis专注于SQL本身，是一个足够灵活的DAO层解决方案。

（2）对性能的要求很高，或者需求变化较多的项目，如互联网项目，MyBatis将是不错的选择。

3.3#{}和${}区别

1.#{}表示一个占位符号

- 通过#{}可以实现 preparedStatement 向占位符中设置值自动进行 java 类型和 数据库 类型转换
- {}可以有效防止 sql 注入。
- {}可以接收简单类型值或 pojo 属性值。 如果 parameterType 传输单个简单类型值， #{}括号中可以是 value 或其它名称。

2.${}表示拼接 sql 串

- 通过${}可以将 parameterType 传入的内容拼接在 sql 中且不进行 jdbc 类型转换.
- ${}不可以防止sql注入
- {}可以接收简单类型值或 pojo 属性值，如果 parameterType 传输单个简单类型值.${}括号中只能是 value

3.4MyBatis与Hibernate有哪些不同？

（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.

3.5Xml映射文件中有哪些常见标签？

除了select|insert|updae|delete标签之外

还有很多其他的标签，`<resultMap>`、`<parameterMap>`、`<sql>`、`<include>`、`<selectKey>`，加上动态sql的9个标签，`trim|where|set|foreach|if|choose|when|otherwise|bind`等，

其中为sql片段标签，通过标签引入sql片段，为不支持自增的主键生成策略标签。

3.6 实体类属性名和表中字段名不一样怎么办

- 通过在查询的sql语句中定义字段名的别名，让字段名的别名和实体类的属性名一致。
- 通过`<resultMap>`来映射字段名和实体类属性名的一一对应的关系

3.7模糊查询like语句该怎么写?

- 方式一: 在Java代码中添加sql通配符。

```
<select id="selectlike">
     select * from foo where bar like #{value}
</select>
```

- 方式二: 在sql语句中拼接通配符，会引起sql注入

```
<select id="selectlike">
     select * from foo where bar like "%"#{value}"%"
 </select>
```

3.8 Dao方法能重载吗？

Dao接口里的方法，是不能重载的，因为是全限名+方法名的保存和寻找策略。

Dao接口的工作原理是JDK动态代理，Mybatis运行时会使用JDK动态代理为Dao接口生成代理proxy对象，代理对象proxy会拦截接口方法，转而执行MappedStatement所代表的sql，然后将sql执行结果返回。

3.9 Dao接口的工作原理是什么

Dao接口，就是人们常说的Mapper接口，接口的全限名，就是映射文件中的namespace的值，接口的方法名，就是映射文件中MappedStatement的id值，接口方法内的参数，就是传递给sql的参数。Mapper接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为key值，可唯一定位一个MappedStatement，

举例：com.mybatis3.mappers.StudentDao.findStudentById，可以唯一找到namespace为com.mybatis3.mappers.StudentDao下面id = findStudentById的MappedStatement。在Mybatis中，每一个`<select>、<insert>、<update>、<delete>`标签，都会被解析为一个MappedStatement对象。

3.10Mybatis分页插件的原理

Mybatis使用RowBounds对象进行分页，它是针对ResultSet结果集执行的内存分页，而非物理分页，可以在sql内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

分页插件的基本原理是使用Mybatis提供的插件接口，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，根据dialect方言，添加对应的物理分页语句和物理分页参数。

3.11 Mybatis动态sql原理

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。

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

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。

3.12Mybatis的一级、二级缓存

1）一级缓存: 基于 PerpetualCache 的 HashMap 本地缓存，其存储作用域为 Session，当 Session flush 或 close 之后，该 Session 中的所有 Cache 就将清空，默认打开一级缓存。

2）二级缓存与一级缓存其机制相同，默认也是采用 PerpetualCache，HashMap 存储，不同在于其存储作用域为 Mapper(Namespace)，并且可自定义存储源，如 Ehcache。默认不打开二级缓存，要开启二级缓存，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；

3）对于缓存数据更新机制，当某一个作用域(一级缓存 Session/二级缓存Namespaces)的进行了C/U/D 操作后，默认该作用域下所有 select 中的缓存将被 clear。

3.13使用的mapper接口调用时有哪些要求

- Mapper接口方法名和mapper.xml中定义的每个sql的id相同
- Mapper接口方法的输入参数类型和mapper.xml中定义的每个sql 的parameterType的类型相同
- Mapper接口方法的输出参数类型和mapper.xml中定义的每个sql的resultType的类型相同
- Mapper.xml文件中的namespace即是mapper接口的类路径。

3.14Mybatis持延迟加载和实现原理

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。

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。

当然了，不光是Mybatis，几乎所有的包括Hibernate，支持延迟加载的原理都是一样的。

3.15 MyBatis怎么实现一对一

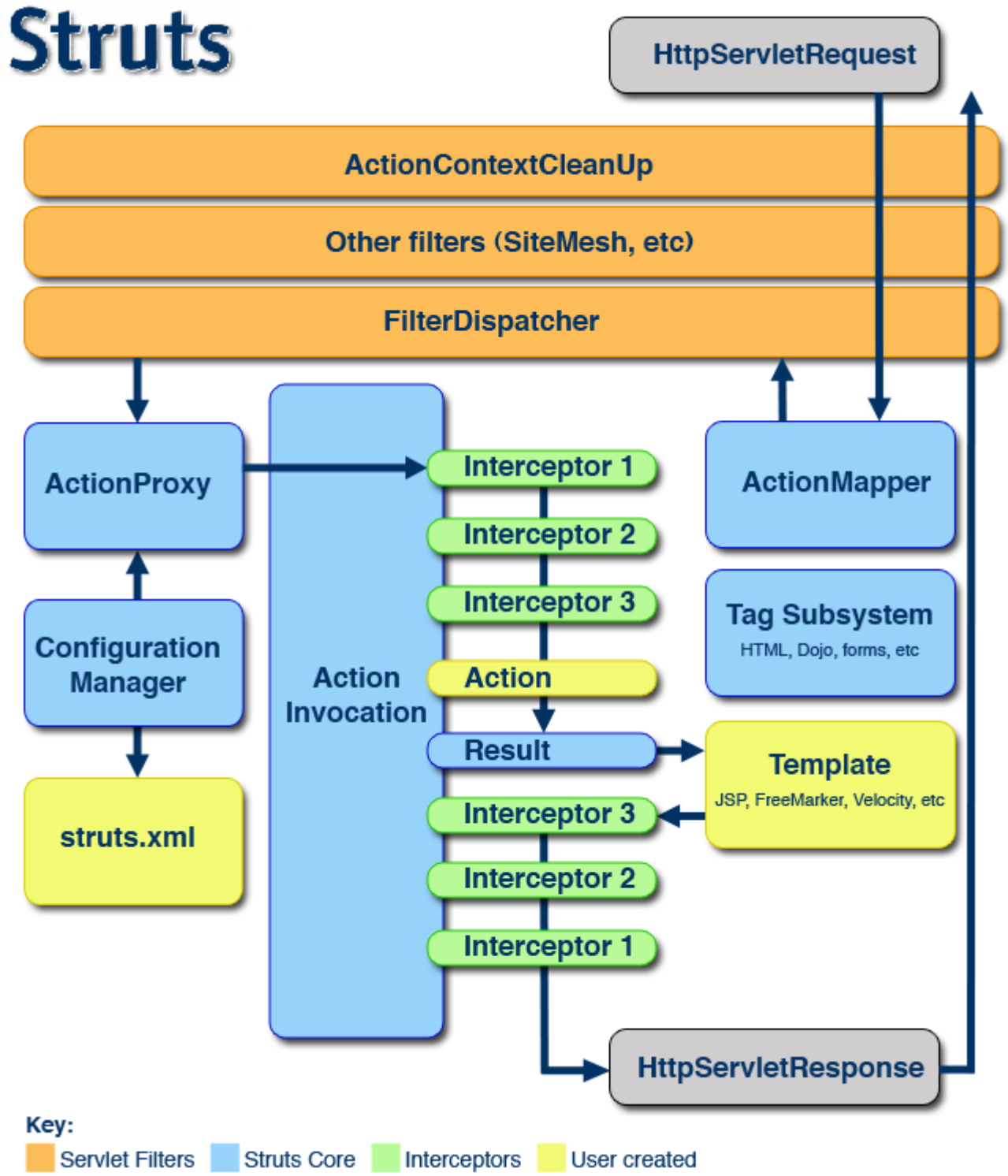
有联合查询和嵌套查询,联合查询是几个表联合查询,只查询一次, 通过在resultMap里面配置association节点配置一对一的类就可以完成; 嵌套查询是先查一个表,根据这个表里面 的结果的外键id,去再另外一个表里面查询数据,也是通过association配置,但另外一个表的查询通过select属性配置。

3.16 MyBatis怎么实现一对多

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collection节点配置一对多的类就可以完成; 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,也是通过配置collection,但另外一个表的查询通过select节点配置。

4.struts2

4.1struts2工作流程



1 、客户端初始化一个指向Servlet容器的请求；

2、 这个请求经过一系列的过滤器（Filter）（这些过滤器中有一个叫做ActionContextCleanUp的可选过滤器，

这个过滤器对于Struts2和其他框架的集成很有帮助，例如：SiteMesh Plugin）

3 、接着FilterDispatcher被调用，

FilterDispatcher询问ActionMapper来决定这个请是否需要调用某个Action

4、如果ActionMapper决定需要调用某个Action，

FilterDispatcher把请求的处理交给ActionProxy

5、ActionProxy通过Configuration Manager询问框架的配置文件，

找到需要调用的Action类

6、ActionProxy创建一个ActionInvocation的实例。

7、ActionInvocation实例使用命名模式来调用，

在调用Action的过程前后，涉及到相关拦截器（Intercepter）的调用。

8、一旦Action执行完毕，ActionInvocation负责根据struts.xml中的配置找到对应的返回结果 。返回结果通常是（但不总是，也可 能是另外的一个Action链）一个需要被表示的JSP或者FreeMarker的模版。 在表示的过程中可以使用Struts2 框架中继承的标签。 在这个过程中需要涉及到ActionMapper

9、响应的返回是通过我们在web.xml中配置的过滤器
10、如果ActionContextCleanUp是当前使用的，则FilterDispatecher将不会清理sreadlocal ActionContext;如果ActionContextCleanUp不使用，则将会去清理sreadlocals。

4.2拦截器和过滤器的区别

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

4.3struts2是如何启动的

struts2框架是通过Filter启动的，即StrutsPrepareAndExecuteFilter，此过滤器为struts2的核心过滤器；

StrutsPrepareAndExecuteFilter的init()方法中将会读取类路径下默认的配置文件struts.xml完成初始化操作。struts2读取到struts.xml的内容后，是将内容封装进javabean对象然后存放在内存中，以后用户的每次请求处理将使用内存中的数据，而不是每次请求都读取struts.xml文件。

4.4struts2框架的核心控制器是什么？它有什么作用？

- Struts2框架的核心控制器是StrutsPrepareAndExecuteFilter。
- 作用：

  负责拦截由 `<url-pattern>/*</url-pattern>` 指定的所有用户请求，当用户请求到达时，该Filter会过滤用户的请求。默认情况下，如果用户请求的路径 不带后缀或者后缀以.action结尾，这时请求将被转入struts2框架处理，否则struts2框架将略过该请求的处理。

  可以通过常量"struts.action.extension"修改action的后缀，如：

  ```
  <constant name="struts.action.extension" value="do"/>
  ```

4.5 struts2配置文件的加载顺序

图示(后面配置的会把前面的覆盖)

顺序	配置文件名	所在位置	说明
1	default.properties	..src\core\src\main\resources\org\apache\struts2	不能修改
2	struts-default.xml	..\struts-2.3.32\src\core\src\main\resources	不能修改
3	strtuts-plugin.xml	在struts2提供的插件jar包中	不能修改
4	struts.xml	我们的应用中	我们修改的：推荐
5	struts.properties	我们的应用中	我们修改的:不建议用

4.6struts2常量的修改方式？

常量可以在struts.xml或struts.properties中配置，两种配置方式如下：

1）在struts.xml文件中配置常量

```
<constant name="struts.action.extension" value="do"/>
```

2）在struts.properties中配置常量（struts.properties文件放置在src下）：

struts.action.extension=do

4.7struts2中至少常见的默认拦截器

fileUpload:提供文件上传功能

i18n :记录用户选择的locale

cookies :使用配置的name,value来是指cookies

checkbox: 添加了checkbox自动处理代码，将没有选中的checkbox的内容设定为false，而html默认情况下不提交没有选中的checkbox。

chain :让前一个Action的属性可以被后一个Action访问，现在和chain类型的result（）结合使用。

alias :在不同请求之间将请求参数在不同名字件转换，请求内容不变

4.8struts2是如何管理action的？

struts2框架中使用包来管理Action，包的作用和java中的类包是非常类似的。

主要用于管理一组业务功能相关的action。在实际应用中，我们应该把一组业务功能相关的Action放在同一个包下。

4.9struts2中的默认包struts-default有什么作用？

1）struts-default包是由struts内置的，它定义了struts2内部的众多拦截器和Result类型，而Struts2很多核心的功能都是通过这些内置的拦截器实现，如：从请求中

把请求参数封装到action、文件上传和数据验证等等都是通过拦截器实现的。当包继承了struts-default包才能使用struts2为我们提供的这些功能。

2）struts-default包是在struts-default.xml中定义，struts-default.xml也是Struts2默认配置文件。 Struts2每次都会自动加载 struts-default.xml文件。

3）通常每个包都应该继承struts-default包。

4.10值栈ValueStack的原理与生命周期

1）ValueStack贯穿整个 Action 的生命周期，保存在request域中，所以ValueStack和request的生命周期一样。当Struts2接受一个请求时，会迅速创建ActionContext，

ValueStack，action。然后把action存放进ValueStack，所以action的实例变量可以被OGNL访问。 请求来的时候，action、ValueStack的生命开始，请求结束，action、 ValueStack的生命结束；

2）action是多例的，和Servlet不一样，Servelt是单例的；

3）每个action的都有一个对应的值栈，值栈存放的数据类型是该action的实例，以及该action中的实例变量，Action对象默认保存在栈顶；

4）ValueStack本质上就是一个ArrayList；

5）关于ContextMap，Struts 会把下面这些映射压入 ContextMap 中：

parameters ： 该 Map 中包含当前请求的请求参数

request ： 该 Map 中包含当前 request 对象中的所有属性 session :该 Map 中包含当前 session 对象中的所有属性

application :该 Map 中包含当前 application 对象中的所有属性

attr:该 Map 按如下顺序来检索某个属性: request, session, application

6）使用OGNL访问值栈的内容时，不需要#号，而访问request、session、application、attr时，需要加#号；

7）注意： Struts2中，OGNL表达式需要配合Struts标签才可以使用。如：`<s:property value="name"/>`

8）在struts2配置文件中引用ognl表达式 ,引用值栈的值 ，此时使用的"$"，而不是#或者%;

4.11拦截器的生命周期与工作过程？

1）每个拦截器都是实现了Interceptor接口的 Java 类；

2）init(): 该方法将在拦截器被创建后立即被调用, 它在拦截器的生命周期内只被调用一次. 可以在该方法中对相关资源进行必要的初始化；

3）intercept(ActionInvocation invocation): 每拦截一个动作请求, 该方法就会被调用一次；

4）destroy: 该方法将在拦截器被销毁之前被调用, 它在拦截器的生命周期内也只被调用一次；

5）struts2中有内置了18个拦截器。

5.Hibernate

5.1hibernate 优缺点

- 优点
  - 对 JDBC 访问数据库的代码做了封装，简化了数据访问层繁琐的重复性代码
  - 映射的灵活性, 它支持各种关系数据库, 从一对一到多对多的各种复杂关系.
  - 非侵入性、移植性会好

- 缓存机制: 提供一级缓存和二级缓存

- 缺点
  - 无法对 SQL 进行优化
  - 框架中使用 ORM原则, 导致配置过于复杂
  - 执行效率和原生的JDBC 相比偏差: 特别是在批量数据处理的时候
  - 不支持批量修改、删除

5.2Hibernate 的检索方式有哪些

① 导航对象图检索
② OID检索
③ HQL检索
④ QBC检索
⑤ 本地SQL检索

5.3在 Hibernate中Java 对象的状态有哪些

①. 临时状态（transient）：不处于 Session 的缓存中。OID 为null 或等于 id 的 unsaved-value 属性值
②. 持久化状态（persistent）：加入到 Session 的缓存中。
③. 游离状态（detached）：已经被持久化，但不再处于 Session 的缓存中。

5.4Session的清理和清空有什么区别？

清理缓存调用的是 session.flush() 方法.

而清空调用的是 session.clear() 方法.

Session 清理缓存是指按照缓存中对象的状态的变化来同步更新数据库，但不清空缓存；清空是把Session 的缓存置空, 但不同步更新数据库；

5.5load()和get()的区别

①如果数据库中，没有 OID 指定的对象。通过 get方法加载，则返回的是一个null；通过load加载，则返回一个代理对象，如果后面代码如果调用对象的某个属性会抛出异常：org.hibernate.ObjectNotFoundException；
②load 支持延迟加载，get不支持延迟加载。

5.6怎么使用 Hibernate 进行大批量更新

直接通过 JDBC API 执行相关的 SQl 语句或调用相关的存储过程是最佳的方式

5.7getCurrentSession() 和 openSession() 的区别

①. getCurrentSession() 它会先查看当前线程中是否绑定了 Session, 如果有则直接返回, 如果没有再创建. 而 openSession() 则是直接 new 一个新的 Session 并返回。
②. 使用ThreadLocal 来实现线程 Session 的隔离。
③. getCurrentSession() 在事务提交的时候会自动关闭 Session, 而 openSession() 需要手动关闭.

5.8说说 Hibernate 的缓存

Hibernate缓存包括两大类：Hibernate一级缓存和Hibernate二级缓存：
1）. Hibernate一级缓存又称为“Session的缓存”，它是内置的，不能被卸载。由于Session对象的生命周期通常对应一个数据库事务或者一个应用事务，因此它的缓存是事务范围的缓存。在第一级缓存中，持久化类的每个实例都具有唯一的OID。

2）.Hibernate二级缓存又称为“SessionFactory的缓存”，由于SessionFactory对象的生命周期和应用程序的整个过程对应，因此Hibernate二级缓存是进程范围或者集群范围的缓存，有可能出现并发问题，因此需要采用适当的并发访问策略，该策略为被缓存的数据提供了事务隔离级别。第二级缓存是可选的，是一个可配置的插件，在默认情况下，SessionFactory不会启用这个插件。
当Hibernate根据ID访问数据对象的时候，首先从Session一级缓存中查；查不到，如果配置了二级缓存，那么从二级缓存中查；如果都查不到，再查询数据库，把结果按照ID放入到缓存删除、更新、增加数据的时候，同时更新缓存。

5.9update和saveOrUpdate的区别

update()和saveOrUpdate()是用来对跨Session的PO进行状态管理的。
update()方法操作的对象必须是持久化了的对象。也就是说，如果此对象在数据库中不存在的话，就不能使用update()方法。

saveOrUpdate()方法操作的对象既可以使持久化了的，也可以使没有持久化的对象。如果是持久化了的对象调用saveOrUpdate()则会 更新数据库中的对象；如果是未持久化的对象使用此方法,则save到数据库中。

5.10比较hibernate的三种检索策略优缺点

- 立即检索；
  - 优点: 对应用程序完全透明，不管对象处于持久化状态，还是游离状态，应用程序都可以方便的从一个对象导航到与它关联的对象；
  - 缺点: select语句太多;可能会加载应用程序不需要访问的对象白白浪费许多内存空间；
- 延迟检索：
  - 优点： 由应用程序决定需要加载哪些对象，可以避免可执行多余的select语句，以及避免加载应用程序不需要访问的对象。因此能提高检索性能，并且能节省内存空间；
  - 缺点： 应用程序如果希望访问游离状态代理类实例，必须保证他在持久化状态时已经被初始化；
- 迫切左外连接检索
  - 优点：对应用程序完全透明，不管对象处于持久化状态，还是游离状态，应用程序都可以方便地冲一个对象导航到与它关联的对象。2使用了外连接，select语句数目少；
  - 缺点:可能会加载应用程序不需要访问的对象，白白浪费许多内存空间；2复杂的数据库表连接也会影响检索性能；

5.11如何在控制台看到hibernate生成并执行的sql

在定义数据库和数据库属性的文件applicationConfig.xml里面，把hibernate.show_sql 设置为true
这样生成的SQL就会在控制台出现了
注意：这样做会加重系统的负担，不利于性能调优

5.12Hibernate中怎样实现类之间的关系

类与类之间的关系主要体现在表与表之间的关系进行操作，它们都是对对象进行操作，我们程序中把所有的表与类都映射在一起，它们通过配置文件中的many-to-one、one-to-many、many-to-many

5.13谈谈Hibernate中inverse的作用

inverse属性默认是false,就是说关系的两端都来维护关系。
比如Student和Teacher是多对多关系，用一个中间表TeacherStudent维护。

如果Student这边inverse=”true”, 那么关系由另一端Teacher维护，就是说当插入Student时，不会操作TeacherStudent表（中间表）。只有Teacher插入或删除时才会触发对中间表的操作。所以两边都inverse=”true”是不对的，会导致任何操作都不触发对中间表的影响；当两边都inverse=”false”或默认时，会导致在中间表中插入两次关系

5.14如何优化Hibernate

1. 使用双向一对多关联，不使用单向一对多
2. 灵活使用单向一对多关联
3. 不用一对一，用多对一取代
4. 配置对象缓存，不使用集合缓存
5. 一对多集合使用Bag,多对多集合使用Set
6. 继承类使用显式多态
7. 表字段要少，表关联不要怕多，有二级缓存撑腰