目录

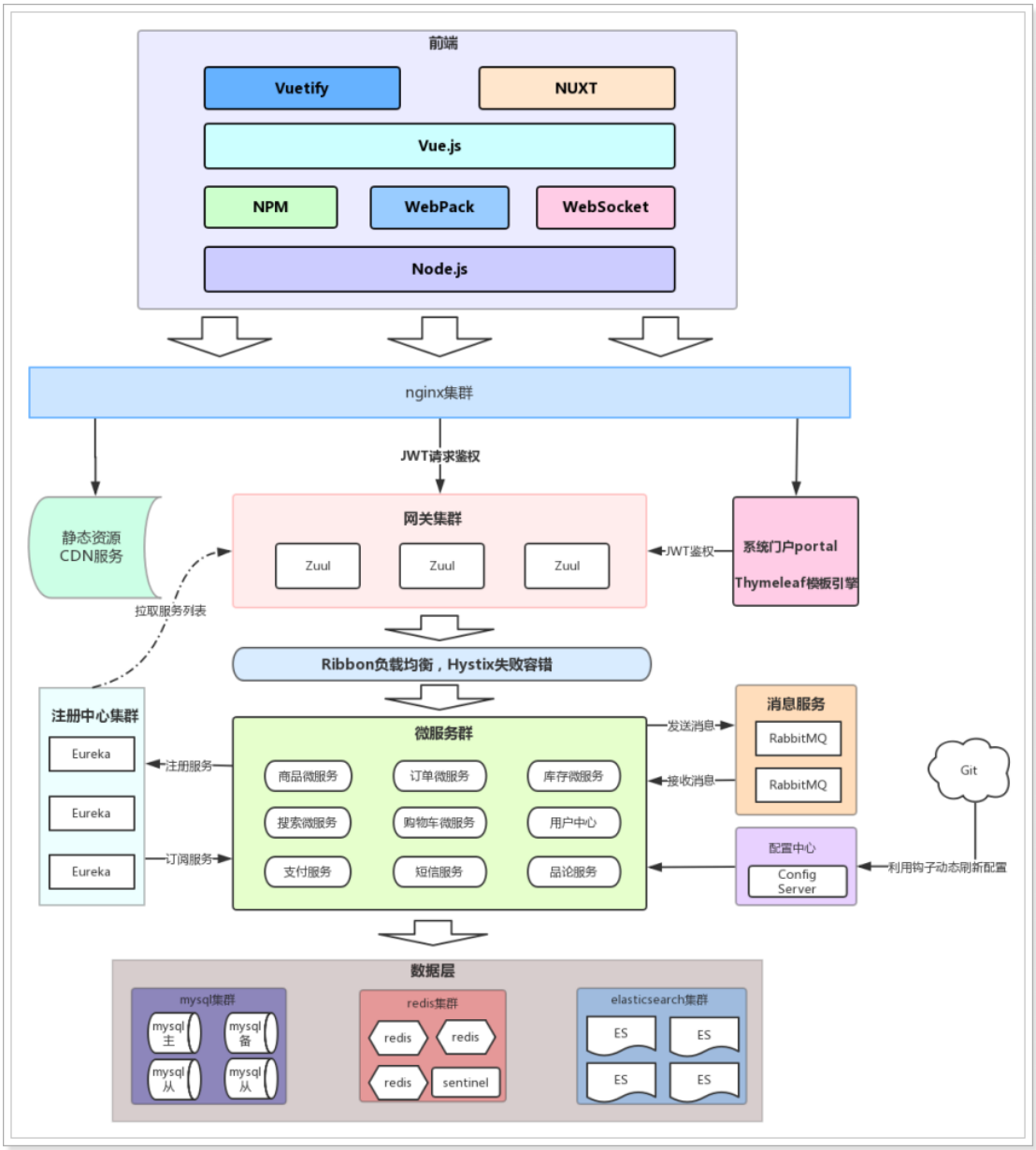
前言

此面试宝典（以下简称文档）旨在按照常规 Java 开发岗位面试流程，结合黑马 Java 课程，作一次综合的要点梳理。适用于黑马 Java 就业班学员在课程将毕，开始正式面试前作为复习指南，以及在后续的 Java 开发工作中，跳槽前的面试要点回顾。

对于 Java 开发面试而言，文档着重强调对于 Java 技术面试整体的把控，而非抓住细节面面俱到，因此建议阅读者根据个人就业和跳槽的需求，对于文档中没有涉及到的 Java 开发技术，自行查阅和整理相应的技术文档。我们认为，面试是一个学习和巩固的过程，而非重述一段概念。达到岗位基本要求以后，每个人的能力展现不尽相同。如果不能从面试中学习，将面试和复习相结合，那么两者往往会背道而驰，事倍功半。

电商项目

项目架构图

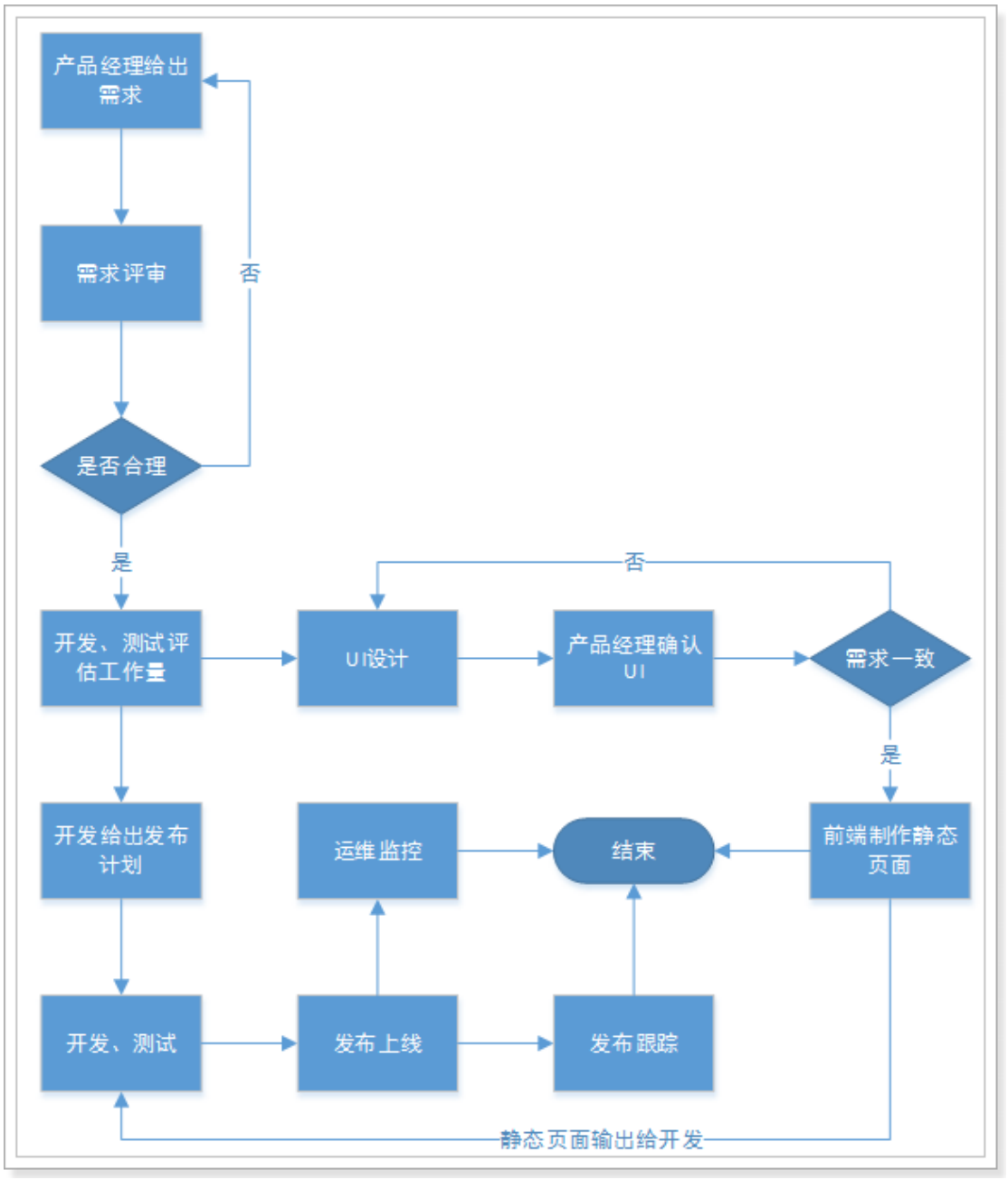


项目人员配置

产品经理：1 人，确定需求以及给出产品原型图。
项目经理：1 人，项目管理，项目质量、进度管理，人力、资源整合调度。
设计团队：2 人，根据产品经理给出的原型制作静态页面。

开发团队：6 人，包括前端和后端业务实现，实现产品功能。
测试团队：2 人，测试项目功能、性能、安全和稳定性等。
运维团队：2 人，服务器环境支持和维护。

工作流程图



业务模块

后台管理系统

定位：在电商项目中，项目运营者和商家需要对整个项目或自家店铺进行运营管理，这便是通过电商中的后台管理系统来进行。该系统功能包括但不限于：商品管理（商品分类、商品品牌、商品规格等）、订单管理、用户评价管理、营销活动管理、物流管理、售后管理和各项数据报表导出等。在乐优中，实现的主要功能为：商品管理中的分类管理、品牌管理、商品规格参数管理。
商品分类管理中，通过自定义 vue 组件以树状展示，通过 CORS 解决 ajax 跨域问题。
商品品牌管理中，通过 data-tables 组件展示品牌列表，通过 FastDFS 实现图片上传，并实现了品牌分页和过滤。
商品规格参数管理中，通过对商品规格属性分类（通用属性和特殊属性）来进行管理，同样以树状展示。
通过以上功能的实现，进而实现了对商品的管理（新增、查询、修改、删除）。

搜索系统

定位：在电商项目中，系统需要给用户提供快捷高效的购物服务，用户也需要从庞大的种类繁多的商品信息中准确定位自身需要的商品，因此搜索系统便承担了这一重任。在乐优中，通过 Spring Data Elaticsearch 搭建针对于商品的搜索系统。用户在首页键入关键字然后点击搜索时，跳转到搜索页，同时发起异步请求到搜索接口，得到结果后集成分页返回到页面进行渲染。主要包括以下功能：商品搜索、分页展示、商品排序、商品规格参数过滤和多 sku 展示等。

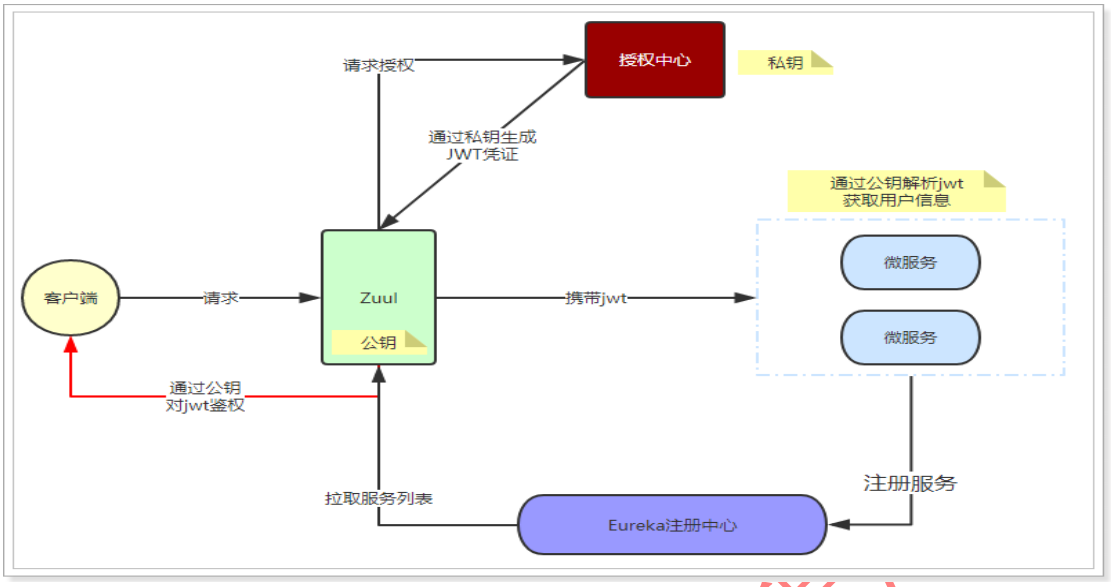
商品详情服务

定位：用户通过搜索系统找到适合自己的商品后，需要点击单个商品了解其详细情况，这里就会访问到商品详情服务。该服务主要功能为展示商品 sku、商品详情、规格与包装和商品评价等。同时，由于商品数量庞大，且服务访问量很高，通过 Nginx 作为反向代理服务器，使用 Thymeleaf 对页面进行静态化，保存在 Nginx 服务器，减轻服务压力。

注册登录系统

定位：注册和登录功能在 web 项目开发中的作用无须赘述。在项目中，注册核心问题包括用户身份验证（短信验证码）和用户密码保存（密码加密），登录核心问题为用户身份校验。同时，基于项目自身架构设计，保证用户登录状态和用户信息在各个服务中实现共享。

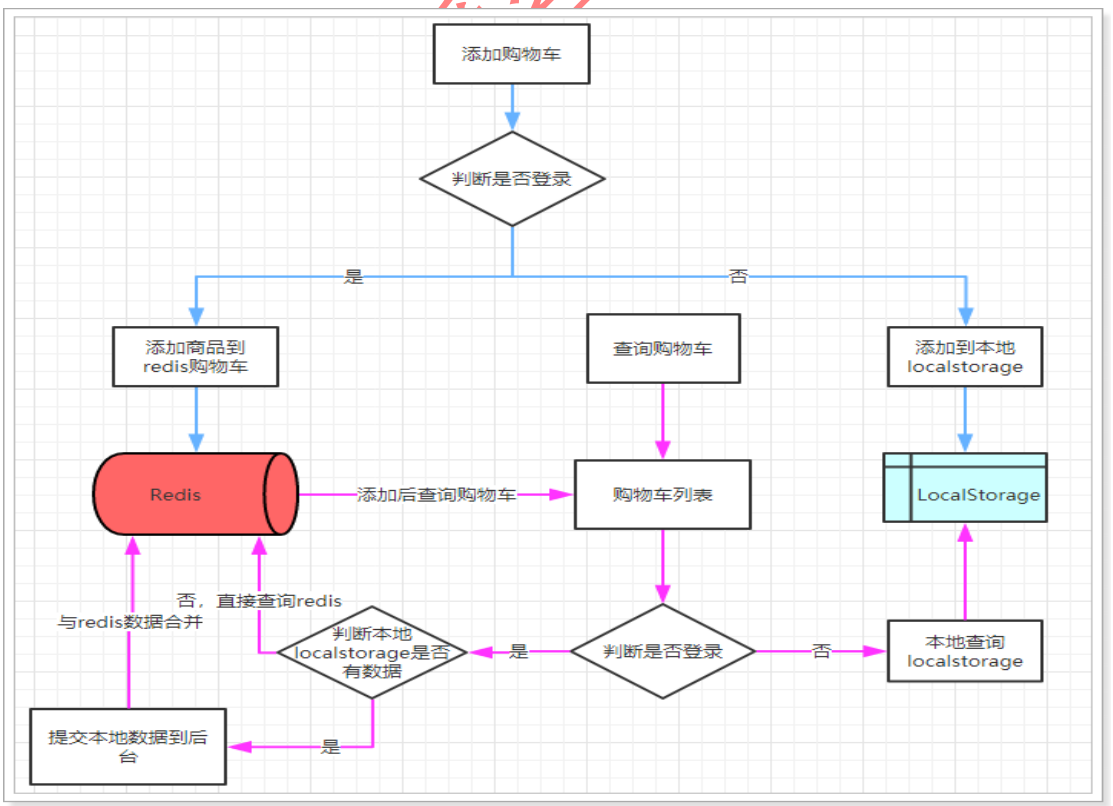
登录系统流程图:



购物车服务

定位：电商项目中，为了避免用户在一次购物中多次结算带来的不友好的体验，通过购物车功能实现一次结算的功能。

流程图:



订单服务

定位：在项目中提供下单功能，包括收件人信息设置、支付方式选择、选购商品信息列表、优惠卷使用和发票服务等。由于电商中订单系统并发压力较大，大型电商的订单系统往往对数据库和订单表进行切分，同时采用更加高效的订单 ID 生成算法（例如 UUID、snowflake）。

项目技术

Redis

Redis 简介

Remote Dictionary Server (Redis) 是一个基于 key-value 键值对的持久化数据库存储系统。支持多种数据结构，这些数据类型都支持 push/pop、add/remove 及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。

Redis 支持的数据类型

字符串（strings）
散列（hashes）
列表（lists）
集合（sets）
有序集合（sorted sets）

Redis 应用场景

缓存
计数器
发布订阅构建消息系统
排行榜

Redis 持久化

RDB 持久化可以在指定的时间间隔内生成数据集的时间点快照（point-in-time snapshot）。AOF 持久化记录服务器执行的所有写操作命令，并在服务器启动时，通过重新执行这些命令来还原数据集。 AOF 文件中的命令全部以 Redis 协议的格式来保存，新命令会被追加到文件的末尾。 Redis 还可以在后台对 AOF 文件进行重写（rewrite），使得 AOF 文件的体积不会超出保存数据集状态所需的实际大小。

Redis 的优势

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 支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。

原子 – Redis 的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。单个操作是原子性的。多个操作也支持事务，即原子性，通过 MULTI 和 EXEC 指令包起来。
丰富的特性 – Redis 还支持 publish/subscribe，通知, key 过期等等特性。

Solr

Solr 简介

Solr 是一个基于 Lucene 的 Java 搜索引擎服务器。Solr 提供了层面搜索、命中醒目显示并且支持多种输出格式（包括 XML/XSLT 和 JSON 格式）。它易于安装和配置，而且附带了一个基于 HTTP 的管理界面。Solr 已经在众多大型的网站中使用，较为成熟和稳定。Solr 包装并扩展了 Lucene，所以 Solr 的基本上沿用了 Lucene 的相关术语。更重要的是，Solr 创建的索引与 Lucene 搜索引擎库完全兼容。通过对 Solr 进行适当的配置，某些情况下可能需要进行编码，Solr 可以阅读和使用构建到其他 Lucene 应用程序中的索引。此外，很多 Lucene 工具（如 Nutch、 Luke）也可以使用 Solr 创建的索引。

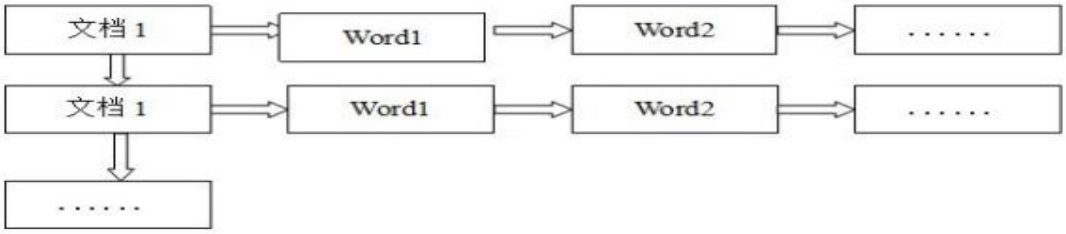
Solr 配置

Schema.xml：
在下载 solr 包的安装解压目录的\solr\example\solr\collection1\conf 中找到，它就是 solr 模式关联的文件。
fieldtype 节点主要用来定义数据类型；
field 节点指定建立索引和查询数据的字段；
solrQueryParser 指定搜索时多个词之间的关系，可以是 or 或 and。

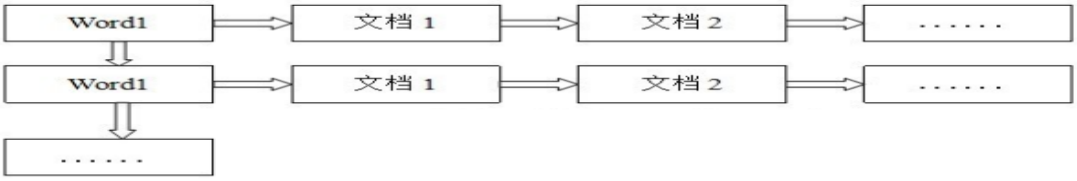
solrconfig.xml：
配置文件主要定义了 SOLR 的一些处理规则，包括索引数据的存放位置，更新，删除，查询的一些规则配置。
datadir 节点定义了索引数据和日志文件的存放位置；
lib 节点表示 solr 引用包的位置。

倒排索引

正排表（索引）是以文档的 ID 为关键字，表中记录文档中每个字的位置信息，查找时扫描表中每个文档中字的信息直到找出所有包含查询关键字的文档。



倒排表（索引）以字或词为关键字进行索引，表中关键字所对应的记录表项记录了出现这个字或词的所有文档，一个表项就是一个字表段，它记录该文档的 ID 和字符在该文档中出现的位置情况。



RabbitMQ/ActiveMQ

RabbitMQ 简介

RabbitMQ 是一个由 Erlang 语言开发的 AMQP 的开源实现。
AMQP：Advanced Message Queue，高级消息队列协议。它是应用层协议的一个开放标准，为面向消息的中间件设计，基于此协议的客户端与消息中间件可传递消息，并不受产品、开发语言等条件的限制。

RabbitMQ 特点

可靠性（Reliability）：使用持久化、传输确认和发布确认机制来保证可靠性。
灵活的路由（Flexible Routing）：在消息进入队列之前，通过 Exchange 来路由消息的。对于典型的路由功能，RabbitMQ 已经提供了一些内置的 Exchange 来实现。针对更复杂的路由功能，可以将多个 Exchange 绑定在一起，也通过插件机制实现自己的 Exchange 。
高可用（Highly Available Queues）：队列可以在集群中的机器上进行镜像，使得在部分节点出问题的情况下队列仍然可用。
多种协议（Multi-protocol）：RabbitMQ 支持多种消息队列协议，比如 STOMP、MQTT 等。
多语言客户端（Many Clients）：RabbitMQ 几乎支持所有常用语言，比如 Java、.NET、Ruby 等。
管理界面（Management UI）：RabbitMQ 提供了一个易用的用户界面，使得用户可以监控和管理消息 Broker 的许多方面。
跟踪机制（Tracing）：如果消息异常，RabbitMQ 提供了消息跟踪机制，使用者可以找出发生了什么。
插件机制（Plugin System）：RabbitMQ 提供了许多插件，来从多方面进行扩展，也可以编写自己的插件。

RabbitMQ 工作模式

简单模式：一个生产者发送消息到队列，一个消费者接收。
工作队列模式：一个生产者，多个消费者，每个消费者获取到的消息唯一，多个消费者只有一个队列。
发布/订阅模式：一个生产者发送的消息会被多个消费者获取，每个消费者只能从自己订阅的队列中获取。
路由模式：生产者发布消息的时候添加路由键，消费者绑定队列到交换机时添加键值，这样就可以接收到需要接收的消息。
通配符模式：基本思想和路由模式是一样的，只不过路由键支持模糊匹配，符号“#”匹配一个或多个词，符号“*”只匹配一个词。

ActiveMQ 简介

ActiveMQ 是 Apache 推出的一款开源的，完全支持 JMS1.1 和 J2EE1.4 规范的 JMS Provider 实现的消息中间件。

ActiveMQ 工作模式

点对点模式：一个消息只有一个消费者消费。
发布/订阅模式：订阅一个主题的消费者只能消费自它订阅之后发布的消息。JMS 规范允许客户创建持久订阅，这在一定程度上放松了时间上的相关性要求。持久订阅允许消费者消费它在未处于激活状态时发送的消息。

MQ 对比

	ActiveMQ	RabbitMQ	Kafka
所属社区/公司	Apache	Mozilla Public License	Apache/LinkedIn
开发语言	Java	Erlang	Java
支持的协议	OpenWire、STOMP、REST、XMPP、AMQP	AMQP	仿AMQP
事物	支持	不支持	不支持
集群	支持	支持	支持
负载均衡	支持	支持	支持
动态扩容	不支持	不支持	支持(zk)

Dubbo

Dubbo 简介

Dubbo 是一个分布式服务框架，致力于提供高性能和透明化的 RPC 远程服务调用方案，以及 SOA 服务治理方案。

其核心部分包括:
远程通讯：提供对多种基于长连接的 NIO 框架抽象封装，包括多种线程模型、序列化、"请求-响应"模式的信息交换方案;
集群容错：提供基于借口方法的透明远程过程调用，包括多协议支持、软负载均衡、失败容错、地址路由、动态配置等集群支持;
自动发现：基于注册中心目录服务，使服务消费方能动态地查找服务提供方，使地址透明，使服务提供方可以平滑增加或减少机器。

Dubbo 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。
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。

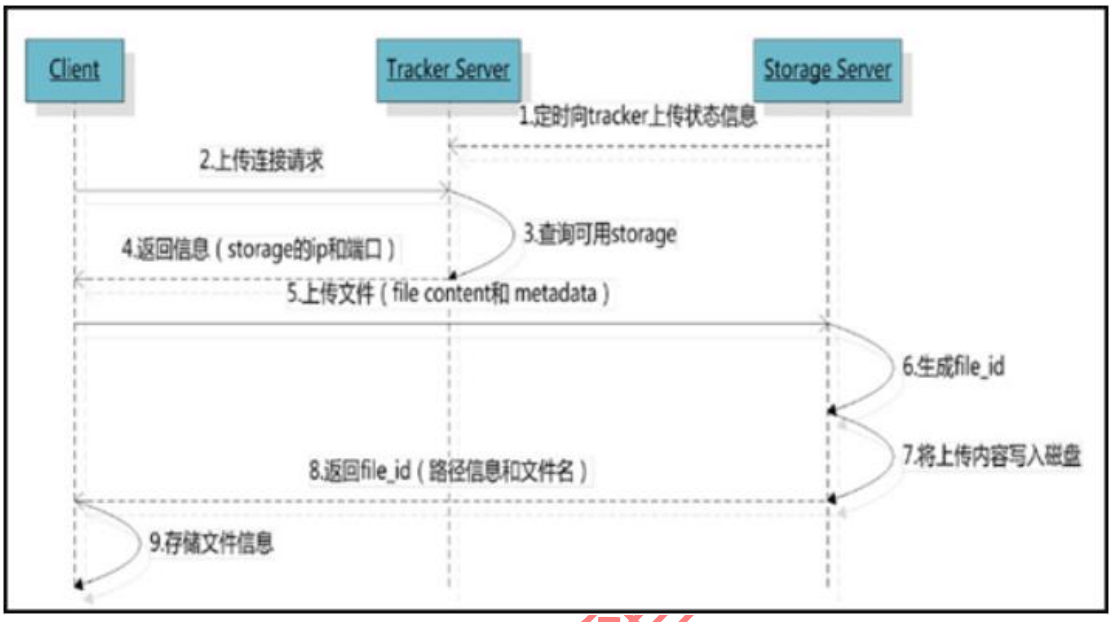
FastDFS

FastDFS 简介

FastDFS 是一个开源的高性能分布式文件系统（DFS）。 它的主要功能包括：文件存储，文件同步和文件访问，以及高容量和负载平衡。主要解决了海量数据存储问题，特别适合以中小文件（建议范围：4KB < file_size <500MB）为载体的在线服务。
FastDFS 系统有三个角色：跟踪服务器(Tracker Server)、存储服务器(Storage Server)和客户端(Client)。
Tracker Server：跟踪服务器，主要做调度工作，起到均衡的作用；负责管理所有的 storage server 和 group，每个 storage 在启动后会连接 Tracker，告知自己所属 group 等信息，并保持周期性心跳。
Storage Server：存储服务器，主要提供容量和备份服务；以 group 为单位，每个 group 内可以有多台 storage server，数据互为备份。

Client：客户端，上传下载数据的服务器，也就是我们自己的项目所部署在的服务器。

文件上传流程



Nginx

Nginx 简介

Nginx 是一款高性能的 http 服务器/反向代理服务器及电子邮件(IMAP/POP3)代理服务器。由俄罗斯的程序设计师 Igor Sysoev 所开发，官方测试 nginx 能够支支撑 5 万并发链接，并且 cpu、内存等资源消耗却非常低，运行非常稳定。

Nginx 功能

静态 HTTP 服务器：Nginx 是一个 HTTP 服务器，可以将服务器上的静态文件（如 HTML、图片）通过 HTTP 协议展现给客户端。
反向代理服务器：客户端本来可以直接通过 HTTP 协议访问某网站应用服务器，但如果单台服务器承受不住压力需要使用多台服务器共同处理请求，这时可以在中间加上一个 Nginx，客户端请求 Nginx，Nginx 请求应用服务器，然后将结果返回给客户端，此时 Nginx 就是反向代理服务器。
负载均衡：当客户端访问量很大，通过反向代理的方式，使用轮询、加权轮询和 IP Hash 的策略将请求分配给多台服务器。

Quartz

Quartz 简介

Quartz 是一个任务调度框架。它具有以下特点:
强大的调度功能，例如支持丰富多样的调度方法，可以满足各种常规及特殊需求;
灵活的应用方式，例如支持任务和调度的多种组合方式，支持调度数据的多种存储方式;
分布式和集群能力，Terracotta 收购后在原来功能基础上作了进一步提升;
作为 Spring 默认的调度框架，Quartz 很容易与 Spring 集成实现灵活可配置的调度功能。

Quartz 核心元素

Scheduler:任务调度器,实际执行任务调度的控制器。在 spring 中通过 SchedulerFactoryBean 封装起来;
Trigger: 触发器，用于定义任务调度的时间规则，有 SimpleTrigger、CronTrigger、DateIntervalTrigger 和 NthIncludedDayTrigger，其中 CronTrigger 用的比较多，在 spring 中封装在 CronTriggerFactoryBean 中;
Calendar: 一些日历特定时间点的集合。一个 trigger 可以包含多个 Calendar，以便排除或包含某些时间点;
JobDetail: 用来描述 Job 实现类及其它相关的静态信息。如 Job 名字、关联监听器等信息。在 spring 中有 JobDetailFactoryBean 和 MethodInvokingJobDetailFactoryBean 两种实现，如果任务调度只需要执行某个类的某个方法，可以通过 MethodInvokingJobDetailFactoryBean 来调用;
Job: 是一个接口，只有一个方法 void execute(JobExecutionContext context)，开发者实现该接口定义运行任务，JobExecutionContext 类提供了调度上下文的各种信息。Job 运行时的信息保存在 JobDataMap 实例中。实现 Job 接口的任务，默认是无状态的，若要将 Job 设置成有状态的,在 quartz 中是给实现的 Job 添加@DisallowConcurrentExecution 注解,在与 spring 结合中可以在 spring 配置文件的 job detail 中配置 concurrent 参数。

框架部分

Spring

Spring 的理解

spring 是一个开源框架，Spring 为简化企业级应用开发而生，使用 Spring 可以使简单的 JavaBean 实现以前只有 EJB 才能实现的功能。Spring 是一个 IOC 和 AOP 容器框架。
Spring 主要核心是:

(1)控制反转(IOC)：传统的 java 开发模式中，当需要一个对象时，我们会自己创建一个对象，而在 Spring 开发模式中，Spring 容器使用了工厂模式为我们创建了所需要的对象，我们直接调用 Spring 为我们提供的对象即可，这就是控制反转的思想。实例化一个 java 对象有三种方式：使用类构造器，使用静态工厂方法，使用实例工厂方法。当使用 spring 时我们不需要关心通过何种方式实例化一个对象，spring 通过控制反转机制自动为我们实例化一个对象。
(2)依赖注入(DI)：Spring 使用 Java Bean 对象的 Set 方法或者带参数的构造方法为我们在创建所需对象时将其属性自动设置所需要的值的过程就是依赖注入的基本思想。
(3)面向切面编程(AOP)：在面向对象编程(OOP)思想中，我们将事物纵向抽象成一个个的对象。而在面向切面编程中，我们将一个个对象某些类似的方面横向抽象成一个切面，对这个切面进行一些如权限验证，事物管理，记录日志等公用操作处理的过程就是面向切面编程的思想。
在 Spring 中，所有管理的对象都是 JavaBean 对象，而 BeanFactory 和 ApplicationContext 就是 spring 框架的两个 IOC 容器，现在一般使用 ApplicationContext，其不但包含了 BeanFactory 的作用，同时还进行更多的扩展。

Spring Bean 生命周期

1.Spring 容器 从 XML 文件中读取 Bean 的定义，并实例化 Bean。
2.Spring 根据 Bean 的定义填充所有的属性。
3.如果 Bean 实现了 BeanNameAware 接口，Spring 传递 bean 的 ID 到 setBeanName 方法。
4.如果 Bean 实现了 BeanFactoryAware 接口， Spring 传递 beanfactory 给 setBeanFactory 方法。
5. 如果有任何与 bean 相关联的 BeanPostProcessors ， Spring 会在 postProcesserBeforeInitialization()方法内调用它们。
6.如果 bean 实现 IntializingBean 了，调用它的 afterPropertySet 方法，如果 bean 声明了初始化方法，调用此初始化方法。
7.如果有 BeanPostProcessors 和 bean 关联，这些 bean 的 postProcessAfterInitialization() 方法将被调用。
8.如果 bean 实现了 DisposableBean，它将调用 destroy()方法。
注意：
有两个重要的 bean 生命周期方法，第一个是 setup() ， 它是在容器加载 bean 的时候被调用。第二个方法是 teardown() 它是在容器卸载类的时候被调用。
The bean 标签有两个重要的属性 init-method 和 destroy-method。使用它们你可以自己定制初始化和注销方法。它们也有相应的注解@PostConstruct 和@PreDestroy。

Spring 中的设计模式

代理模式—Spring 中两种代理方式，若目标对象实现了若干接口，spring 使用 JDK 的 java.lang.reflect.Proxy 类代理，若目标对象没有实现任何接口，spring 使用 CGLIB 库生成目标对象的子类。

单例模式—在 spring 配置文件中定义的 bean 默认为单例模式。
模板方法模式—用来解决代码重复的问题。比如： RestTemplate, JmsTemplate, JpaTemplate。
前端控制器模式—Srping 提供了 DispatcherServlet 来对请求进行分发。
视图帮助(View Helper)—Spring 提供了一系列的 JSP 标签，高效宏来辅助将分散的代码整合在视图里。
依赖注入—贯穿于 BeanFactory/ApplicationContext 接口的核心理念。
工厂模式—在工厂模式中，我们在创建对象时不会对客户端暴露创建逻辑，并且是通过使用一个共同的接口来指向新创建的对象。Spring 中使用 BeanFactory 用来创建对象的实例。

Spring 注解

Spring 在 2.5 版本以后开始支持用注解的方式来配置依赖注入。可以用注解的方式来替代 XML 方式的 bean 描述，可以将 bean 描述转移到组件类的内部，只需要在相关类上、方法上或者字段声明上使用注解即可。注解注入将会被容器在 XML 注入之前被处理，所以后者会覆盖掉前者对于同一个属性的处理结果。
注解装配在 Spring 中是默认关闭的。所以需要在 Spring 文件中配置一下才能使用基于注解的装配模式。如果你想要在你的应用程序中使用关于注解的方法的话，请参考如下的配置。

```
<beans>
<context:annotation-config/>
<!-- bean definitions go here -->
</beans>
```

在 <context:annotation-config/>标签配置完成以后，就可以用注解的方式在 Spring 中向属性、方法和构造方法中自动装配变量。
几种比较重要的注解类型：
1. @Required：该注解应用于设值方法。
2. @Autowired：该注解应用于有值设值方法、非设值方法、构造方法和变量。
3. @Qualifier：该注解和@Autowired 注解搭配使用，用于消除特定 bean 自动装配的歧义。
4. JSR-250 Annotations：Spring 支持基于 JSR-250 注解的以下注解，@Resource、@PostConstruct 和 @PreDestroy。

Spring 事务

Spring 支持两种类型的事务管理：
1.编程式事务管理：这意味你通过编程的方式管理事务，给你带来极大的灵活性，但是难维护。
2.声明式事务管理：这意味着你可以将业务代码和事务管理分离，你只需用注解和 XML 配置来管理事务。
Spring 事务配置示例（使用 tx 标签配置的拦截器）

```
<!-- 定义事务管理器（声明式的事务） -->
    <bean id="transactionManager"
        class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory" />
    </bean>
```

```
<!-- 配置 Advice 通知 -->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <tx:method name="*" propagation="REQUIRED" />
        </tx:attributes>
    </tx:advice>
    <!-- 配置切点切面 -->
    <aop:config>
        <aop:pointcut id="interceptorPointCuts"
            expression="execution(* com.bluesky.spring.dao.*.*(..))" />
        <aop:advisor advice-ref="txAdvice"
            pointcut-ref="interceptorPointCuts" />
    </aop:config>
```

SpringBoot

SpringBoot 简介

Spring Boot(英文中是“引导”的意思)，是用来简化 Spring 应用的搭建到开发的过程。应用开箱即用，只要通过 “just run”（可能是 java -jar 或 tomcat 或 maven 插件 run 或 shell 脚本)，就可以启动项目。二者，Spring Boot 只要很少的 Spring 配置文件（例如那些 xml，property)。因为“习惯优先于配置”的原则，使得 Spring Boot 在快速开发应用和微服务架构实践中得到广泛应用。

SpringBoot 特性

自动配置：针对很多 Spring 应用程序常见的应用功能，Spring Boot 能自动提供相关配置；
起步依赖：告诉 Spring Boot 需要什么功能，它就能引入需要的库；
命令行界面：这是 Spring Boot 的可选特性，借此你只需写代码就能完成完整的应用程序，无需传统项目构建；
Actuator：让你能够深入运行中的 Spring Boot 应用程序，一探究竟。

SpringBoot 核心

@SpringBootApplication 这个 Spring Boot 核心注解是由其它三个重要的注解组合，分别是：@SpringBootConfiguration 、 @EnableAutoConfiguration 和 @ComponentScan。
@ SpringBootConfiguration
点开查看发现里面还是应用了@Configuration。任何一个标注了@Configuration 的 Java 类定义的都是一个 JavaConfig 配置类。SpringBoot 社区推荐使用基于 JavaConfig 的配置形式，所以，这里的启动类标注了@Configuration 之后，本身其实也是一个 IoC 容器的配置类。

@EnableAutoConfiguration
是一个复合注解。最重要的是@Import(EnableAutoConfigurationImportSelector.class)，借助 EnableAutoConfigurationImportSelector，@EnableAutoConfiguration 可以帮助 SpringBoot 应用将所有符合条件的@Configuration 配置都加载到当前 SpringBoot 使用的 IoC 容器。
@ComponentScan
@ComponentScan 这个注解在 Spring 中很重要，它对应 XML 配置中的元素，@ComponentScan 的功能其实就是自动扫描并加载符合条件的组件（比如@Component 和 @Repository 等）或者 bean 定义，最终将这些 bean 定义加载到 IoC 容器中。

SpringCloud

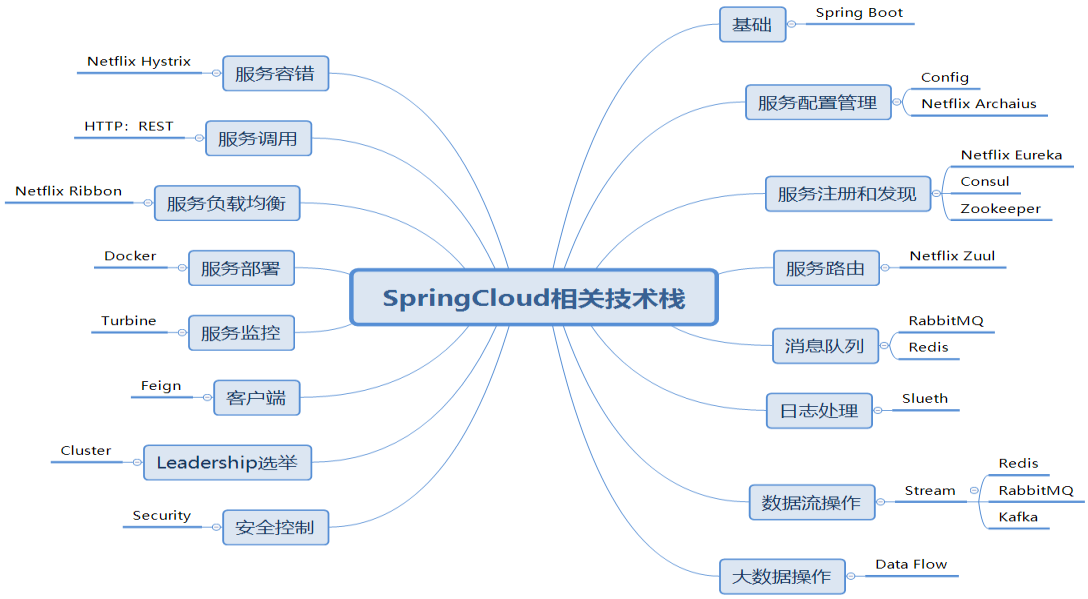
SpringCloud 简介

spring Cloud 是一个基于 Spring Boot 实现的云应用开发工具，它为基于 JVM 的云应用开发中的配置管理、服务发现、断路器、智能路由、微代理、控制总线、全局锁、决策竞选、分布式会话和集群状态管理等操作提供了一种简单的开发方式。

SpringCloud 核心组件

服务注册发现 - Netflix Eureka
配置中心 - spring cloud config
负载均衡-Netflix Ribbon
断路器 - Netflix Hystrix
路由(网关) - Netflix Zuul

SpringCloud 相关技术栈



微服务

微服务是一种可以让软件职责单一、松耦合、自包含、可以独立运行和部署的架构思想。关键思想就是：拆分、单一、独立、组件化。把原本一个庞大、复杂的项目按业务边界拆分一个一个独立运行的小项目，通过接口的方式组装成一个大的项目。

Docker

Docker 简介

Docker 项目的目标是实现轻量级的操作系统虚拟化解决方案。Docker 的基础是 Linux 容器（LXC）等技术。在 LXC 的基础上 Docker 进行了进一步的封装，让用户不需要去关心容器的管理，使得操作更为简便。用户操作 Docker 的容器就像操作一个快速轻量级的虚拟机一样简单。

Docker 理解

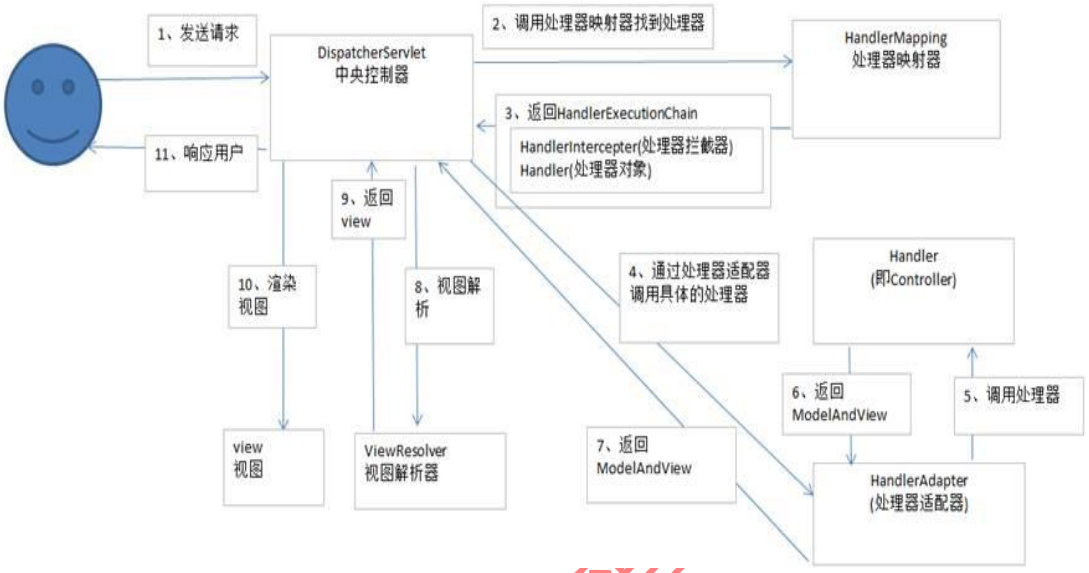
Docker 其实和虚拟机的目的差不多，都是虚拟化技术，但是 docker 比虚拟机更加轻量级，更快，更加易于移植。

镜像：创建虚拟机和 docker 都必不可少的东西。创建一个虚拟机，就先得下载操作系统的 ISO 镜像文件，然后通过镜像文件安装操作系统，和实体机类似，然后能在虚拟机中去安装各种软件。

容器：通俗拿 VM 虚拟机和 Docker 来举例，一个容器就类似于一个虚拟机，只不过在 Docker 技术的术语上称为容器。这个容器里装的就是我们部署的应用在运行，和虚拟机一样可以开机，关机，重启。Docker 称为容器的运行，关闭，重启。而且这个容器可以打包为镜像文件，类似虚拟机快照的文件，放在其它虚拟机上又可以保持原样能运行，Docker 也是如此，把容器打包为镜像文件，然后在新的服务器安装好的 Docker 环境下导入进去，保持原来的状态能够运行。

SpringMVC

SpringMVC 执行流程



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

springmvc 常用注解

@RequestMapping：是一个用来处理请求地址映射的注解，可用于类或方法上。用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。
@PathVariable：用于将请求 URL 中的模板变量映射到功能处理方法的参数上，即取出 uri 模板中的变量作为参数。
@requestParam：主要用于在 SpringMVC 后台控制层获取参数，类似一种是 request.getParameter("name"), 它有三个常用参数: defaultValue = "0", required = false, value = "isApp"；defaultValue 表示设置默认值，required 铜过 boolean 设置是否是必须要传入的参数，value 值表示接受的传入的参数类型。
@ResponseBody：该注解用于将 Controller 的方法返回的对象，通过适当的

HttpMessageConverter 转换为指定格式后，写入到 Response 对象的 body 数据区。使用时机：返回的数据不是 html 标签的页面，而是其他某种格式的数据时（如 json、xml 等）使用
@RequestBody：该注解常用来处理 Content-Type：不是 application/x-www-form-urlencoded 编码的内容，例如 application/json, application/xml 等；
@RequestHeader：可以把 Request 请求 header 部分的值绑定到方法的参数上。
@CookieValue：可以把 Request header 中关于 cookie 的值绑定到方法的参数上。

SpringMVC 和 Struts2 对比

机制：spring mvc 的入口是 servlet，而 struts2 是 filter（这里要指出，filter 和 servlet 是不同的。以前认为 filter 是 servlet 的一种特殊），这样就导致了二者的机制不同，这里就牵涉到 servlet 和 filter 的区别了。
性能：spring 会稍微比 struts 快。spring mvc 是基于方法的设计，而 sturts 是基于类，每次发一次请求都会实例一个 action，每个 action 都会被注入属性，而 spring 基于方法，粒度更细，但要小心把握像在 servlet 控制数据一样。spring3 mvc 是方法级别的拦截，拦截到方法后根据参数上的注解，把 request 数据注入进去，在 spring3 mvc 中，一个方法对应一个 request 上下文。而 struts2 框架是类级别的拦截，每次来了请求就创建一个 Action，然后调用 setter getter 方法把 request 中的数据注入；struts2 实际上是通过 setter getter 方法与 request 打交道的；struts2 中，一个 Action 对象对应一个 request 上下文。
参数传递：struts 是在接受参数的时候，可以用属性来接受参数，这就说明参数是让多个方法共享的。
设计思想上：struts 更加符合 oop 的编程思想，spring 就比较谨慎，在 servlet 上扩展。

Mybatis

Mybatis 的理解

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体，将接口和 Java 的 POJOs(Plain Old Java Objects,普通的 Java 对象)映射成数据库中的记录。
Mybatis 的优点：
1、简单易学。mybatis 本身就很小且简单。没有任何第三方依赖，最简单安装只要两个 jar 加配置几个 sql 映射文件，易于学习，易于使用，通过文档和源代码，可以比较完全的掌握它的设计思路和实现；
2、灵活。mybatis 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，便于统一管理和优化。通过 sql 基本上实现不使用数据访问框架可以实现的所有功能；
3、解除 sql 与程序代码的耦合。通过提供 DAO 层，将业务逻辑和数据访问逻辑分离，使系统的设计更清晰，更易维护，更易单元测试。sql 和代码的分离，提高了可维护性；
4、提供映射标签，支持对象与数据库的 orm 字段关系映射；
5、提供对象关系映射标签，支持对象关系组建维护；
6、提供 xml 标签，支持编写动态 sql。

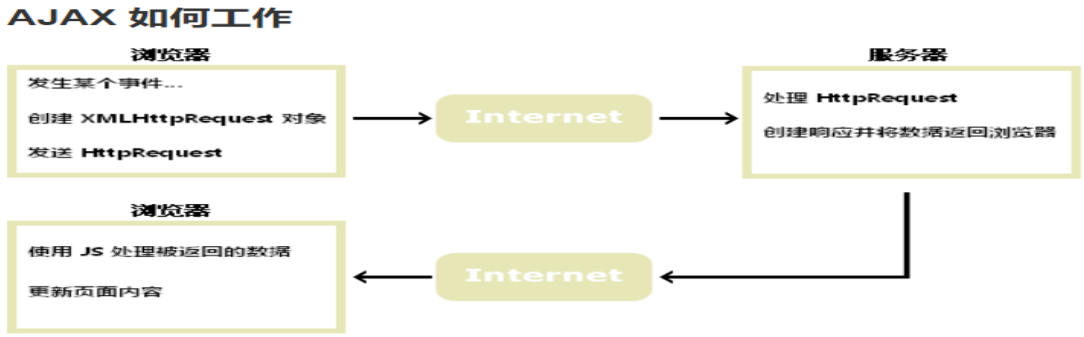
Mybatis 缓存

一级缓存：Mybatis 的一级缓存的作用域是 session，当 openSession()后，如果执行相同的 SQL（相同语句和参数），Mybatis 不进行执行 SQL，而是从缓存中命中返回。
二级缓存：Mybatis 的二级缓存的作用域是一个 mapper 的 namespace，同一个 namespace 中查询 sql 可以从缓存中命中。二级缓存是可以跨 session 的。

Java Web

Ajax

AJAX = Asynchronous JavaScript and XML（异步 JavaScript 和 XML）。
Ajax 的原理简单来说通过 XmlHttpRequest 对象来向服务器发异步请求，从服务器获得数据，然后用 Javascript 来操作 DOM 而更新页面。这其中最关键的一步就是从服务器获得请求数据。
XmlHttpRequest 是 ajax 的核心机制，它是在 IE5 中首先引入的，是一种支持异步请求的技术。简单的说，也就是 Javascript 可以及时向服务器提出请求和处理响应，而不阻塞用户。达到无刷新的效果。



JQuery

JQuery 是一个 JavaScript 库。功能包括 HTML 元素选取和操作、CSS 操作、HTML 事件函数、JavaScript 特效和动画、HTML DOM 遍历和修改、AJAX 和 Utilities。除此之外，JQuery 还提供了大量插件。
基础语法： $(selector).action()。
选择器：主要分四大选择器，分别是基本选择器、层次选择器、过滤选择器、属性过滤选择器。
事件：例如 click()、dblclick()、mouseenter()、mouseleave()、mousedown()等。

Cookie

在 web 程序中是使用 HTTP 协议来传输数据的，因为 http 是无状态协议，一旦数据交换完毕，客户端和服务器端的连接就会关闭，再次交换数据需要建立新的连接，所以无法实现会话跟踪，cookie 技术则弥补了这一缺陷。
cookie 实际上一段的文本信息，客户端请求服务器。如果服务器需要记录该用户的状态，就使用 response 向客户端浏览器颁发一个 cookie。客户端浏览器会把 cookie 保存起来。当浏览器再请求该网站时，浏览器把请求的网址连同该 cookie 一同提交给服务器。服务器检查该 cookie，以此来辨认用户的状态。服务器还可以根据需要修改 cookie 的内容。
cookie 生命周期:
cookie 的 maxAge 决定 cookie 的生命周期，单位为秒（second）。cookie 通过 getMaxAge()方法和 setMaxAge()方法来获得 maxAge 属性，如果 maxAhe 属性为正，则表示 cookie 会在 maxAge 秒之后自动失效。如果 maxAge 属性为负，则说明 cookie 仅在本浏览器窗口和本窗口打开的子窗口下有效，关闭窗口 cookie 则失效。maxAge 的默认值是-1 当 maxAge 的值为 0 时，表示删除 cookie。

Session

session 也是一种记录客户状态的机制，不同的是 cookie 保存在客户端浏览器中，而 session 保存在服务器上。客户端浏览器访问服务器是时候把客户端信息以某种形式记录在服务器上，这就是 session 中查找该客户的状态。

session 生命周期:
session 保存在服务器端，为了获得更高的存取速度，服务器一般把 session 放在内存。每个用户都会有一个独立的 session,如果 session 内容过于复杂，当大量客户访问服务器时可能会导致内存溢出。
session 在用户第一次访问服务器的时候自动创建，需要注意只有访问 JSP，Servlet 等程序时才会创建 session；只要访问 HTML、IMAGE 等静态资源不会创建 session。如果尚未生成 session,可以使用 request.getSession(true)强制生成 session。
session 生成后，只要用户访问，服务器就会更新 session 的最后访问时间，并维护该 session。用户每访问服务器一次，无论是否续写 session 服务器都认为该用户的 session 活跃（active）了一次。
Session 对应的类是 javax.servlet.http.HttpSession，每一个访问者都对应一个 session 对象，并将其状态信息保存在这个 session 对象中，session 对象的创建是在用户第一次访问服务器时产生的。

热门面试问题:

1、原生态 Ajax 执行流程?

创建 XMLHttpRequest 对象；

注册回调函数;
设置连接信息;
发送数据，与服务器开始交互;
接受服务器返回数据。

2、转发（forward）和重定向（redirect）的区别?

forward 是容器中控制权的转向，是服务器请求资源，服务器直接访问目标地址的 URL，把那个 URL 的响应内容读取过来，然后把这些内容再发给浏览器，浏览器根本不知道服务器发送的内容是从哪儿来的，所以它的地址栏中还是原来的地址。
redirect 就是服务器端根据逻辑，发送一个状态码，告诉浏览器重新去请求那个地址，因此从浏览器的地址栏中可以看到跳转后的链接地址，很明显 redirect 无法访问到服务器保护起来资源，但是可以从一个网站 redirect 到其他网站。

3、怎么防止表单重复提交?

i. 禁掉提交按钮。表单提交后使用 Javascript 使提交按钮 disable。
ii .Post/Redirect/Get 模式。在提交后执行页面重定向，这就是所谓的 Post-Redirect-Get (PRG) 模式。简言之，当用户提交了表单后，你去执行一个客户端的重定向，转到提交成功信息页面。
iii.在 session 中存放一个特殊标志。当表单页面被请求时，生成一个特殊的字符标志串，存在 session 中，同时放在表单的隐藏域里。接受处理表单数据时，检查标识字串是否存在，并立即从 session 中删除它，然后正常处理数据。

4、web.xml 文件中可以配置哪些内容?

web.xml 用于配置 Web 应用的相关信息，如：监听器（listener）、过滤器（filter）、 Servlet、相关参数、会话超时时间、安全验证方式、错误页面等。

数据库（MySQL）

连接查询

分类：内连接、外连接、自然连接（略）、交叉连接（略）。

内连接

基本语法：左表 [inner] join 右表 on 左表.字段 = 右表.字段;
从左表中取出每一条记录，去右表中与所有的记录进行匹配：匹配必须是某个条件在左表中

与右表中相同最终才会保留结果，否则不保留。

外连接

基本语法：左表 left/right join 右表 on 左表.字段 = 右表.字段;
left join：左外连接(左连接)，以左表为主表
right join：右外连接(右连接)，以右表为主表
以某张表为主，取出里面的所有记录，然后每条与另外一张表进行连接：不管能不能匹配上条件，最终都会保留。能匹配，正确保留；不能匹配，其他表的字段都置空 NULL。
举例：

student_id	student_name	student_sex
1	隔壁老王	男
2	邻家小玉	女
3	叶良辰	男
4	包青天	男
5	苍井空	女

student_id	student_score	student_grade
1	96	优秀
2	88	良好
3	60	合格
6	54	不合格

内连接

```
SELECT A.*, B.*
FROM student_info A inner join student_score B
ON A.student_id = B.student_id
```

信息 结果1 概况 状态

student_id	student_name	student_sex	student_id1	student_score	student_grade
1	隔壁老王	男	1	96	优秀
2	邻家小玉	女	2	88	良好
3	叶良辰	男	3	60	合格

左外连接

```
SELECT A.*, B.*
FROM student_info A left join student_score B
ON A.student_id = B.student_id
```

信息 结果1 概况 状态

student_id	student_name	student_sex	student_id1	student_score	student_grade
1	隔壁老王	男	1	96	优秀
2	邻家小玉	女	2	88	良好
3	叶良辰	男	3	60	合格
4	包青天	男	(Null)	(Null)	(Null)
5	苍井空	女	(Null)	(Null)	(Null)

右外连接

```
SELECT A.*, B.*
FROM student_info A right join student_score B
ON A.student_id = B.student_id
```

信息 | 结果1 | 概况 | 状态

student_id	student_name	student_sex	student_id1	student_score	student_grade
1	隔壁老王	男	1	96	优秀
2	邻家小玉	女	2	88	良好
3	叶良辰	男	3	60	合格
(Null)	(Null)	(Null)	6	54	不合格

联合查询

基本语法：Select 语句 1
Union [union 选项]
Select 语句 2
.....

将多次查询(多条 select 语句)，在记录上进行拼接(字段不会增加)，每一条 select 语句获取的字段数必须严格一致(但是字段类型无关)。
其中 union 选项有 2 个。ALL：保留所有；Distinct（默认）：去重。
应用：查询同一张表，但是有不同的需求；查询多张表，多张表的结构完全一致，保存的数据也是一样的。
在联合查询中，order by 不能直接使用。需要对查询语句使用括号才行。另外需要配合 limit 使用。

索引

如果说数据库表中的数据是一本书，那么索引就是书的目录。索引能够让我们快速的定位想要查询的数据。
索引的结构：BTree 索引和 Hash 索引。
MyISAM 和 InnoDB 存储引擎：只支持 BTREE 索引， 也就是说默认使用 BTREE，不能够更换。
MEMORY/HEAP 存储引擎：支持 HASH 和 BTREE 索引。
索引的分类：单列索引(普通索引，唯一索引，主键索引)、组合索引、全文索引、空间索引。

数据库引擎

InnoDB：支持事务处理，支持外键，支持崩溃修复能力和并发控制。如果需要对事务的完整性要求比较高（比如银行），要求实现并发控制（比如售票），那选择 InnoDB 有很大的优势。如果需要频繁的更新、删除操作的数据库，也可以选择 InnoDB，因为支持事务的提交（commit）和回滚（rollback）。
MyISAM：插入数据快，空间和内存使用比较低。如果表主要是用于插入新记录和读出记录，

那么选择 MyISAM 能实现处理高效率。如果应用的完整性、并发性要求比较低，也可以使用。
MEMORY：所有的数据都在内存中，数据的处理速度快，但是安全性不高。如果需要很快的读写速度，对数据的安全性要求较低，可以选择 MEMOEY。它对表的大小有要求，不能建立太大的表。所以，这类数据库只使用在相对较小的数据库表。

存储过程

SQL 语句需要先编译然后执行，而存储过程（Stored Procedure）是一组为了完成特定功能的 SQL 语句集，经编译后存储在数据库中，用户通过指定存储过程的名字并给定参数（如果该存储过程带有参数）来调用执行它。
存储过程是可编程的函数，在数据库中创建并保存，可以由 SQL 语句和控制结构组成。当想要在不同的应用程序或平台上执行相同的函数，或者封装特定功能时，存储过程是非常有用的。数据库中的存储过程可以看做是对编程中面向对象方法的模拟，它允许控制数据的访问方式。

存储过程的优点：
增强 SQL 语言的功能和灵活性；
标准组件式编程；
较快的执行速度；
减少网络流量；
作为一种安全机制来充分利用。

热门面试问题：

1、JDBC 编程的步骤?

（1） 注册驱动；
（2） 获取连接对象 Connection；
（3） 创建 Statement 对象；
（4） 运行 SQL 语句；
（5） 处理结果；
（6） 关闭连接释放资源。

2、事务的 ACID 是什么？事务并发会产生哪些问题?

ACID 表示事务的特性：原子性、一致性、隔离性和持久性。
- 原子性(Atomic)：事务中各项操作，要么全做要么全不做，任何一项操作的失败都会导致整个事务的失败；
- 一致性(Consistent)：事务结束后系统状态是一致的；
- 隔离性(Isolated)：并发执行的事务彼此无法看到对方的中间状态；

- 持久性(Durable)：事务完成后所做的改动都会被持久化，即使发生灾难性的失败。通过日志和同步备份可以在故障发生后重建数据。

事务并发产生的问题：脏读、幻读、不可重复读。
脏读（Dirty Read）：A 事务读取 B 事务尚未提交的数据并在此基础上操作，而 B 事务执行回滚，那么 A 读取到的数据就是脏数据。
幻读（Phantom Read）：事务 A 重新执行一个查询，返回一系列符合查询条件的行，发现其中插入了被事务 B 提交的行。
不可重复读（Unrepeatable Read）：事务 A 重新读取前面读取过的数据，发现该数据已经被另一个已提交的事务 B 修改过了。

3、数据库性能优化有哪些方式?

SQL 优化:

尽量避免使用 SELECT *;
只查询一条记录时使用 limit 1;
使用连接查询代替子查询;
尽量使用一些能通过索引查询的关键字。

表结构优化:

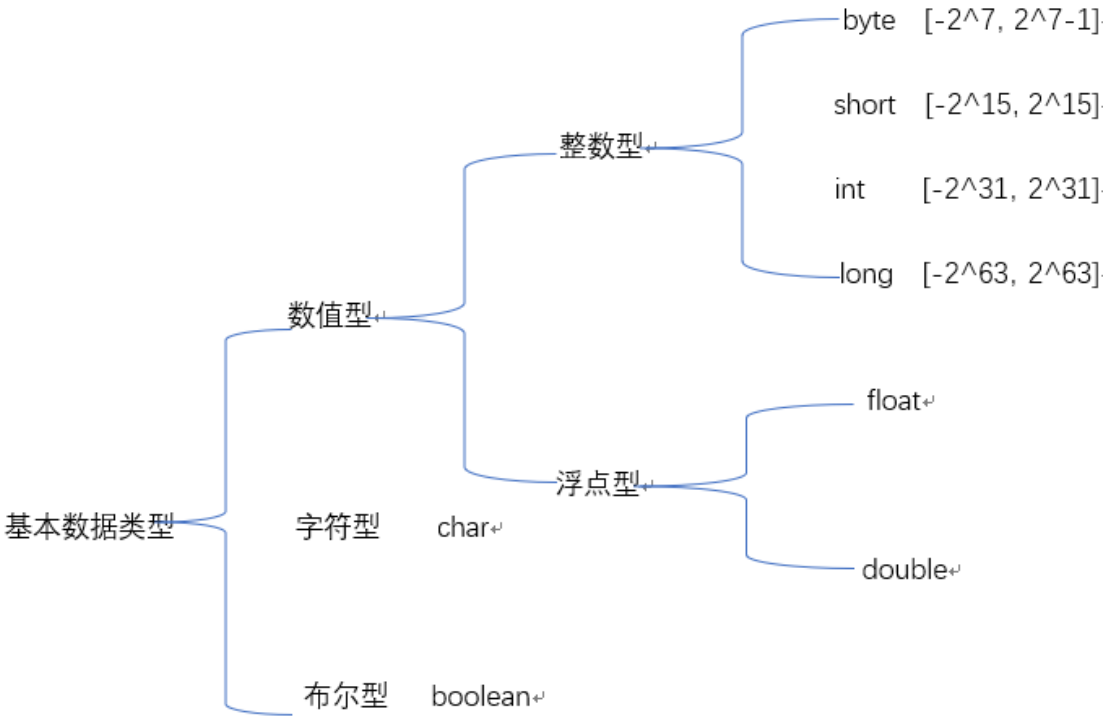
尽量使用数字类型字段，提高比对效率;
长度不变且对查询速度要求高的数据可以考虑使用 char，否则使用 varchar;
表中字段过多时可以适当的进行垂直分割，将部分字段移动到另外一张表;
表中数据量过大可以适当的进行水平分割，将部分数据移动到另外一张表。

其它优化:

对查询频率高的字段适当的建立索引，提高效率;
根据表的用途使用合适的数据库引擎;
读写分离。

Java 基础

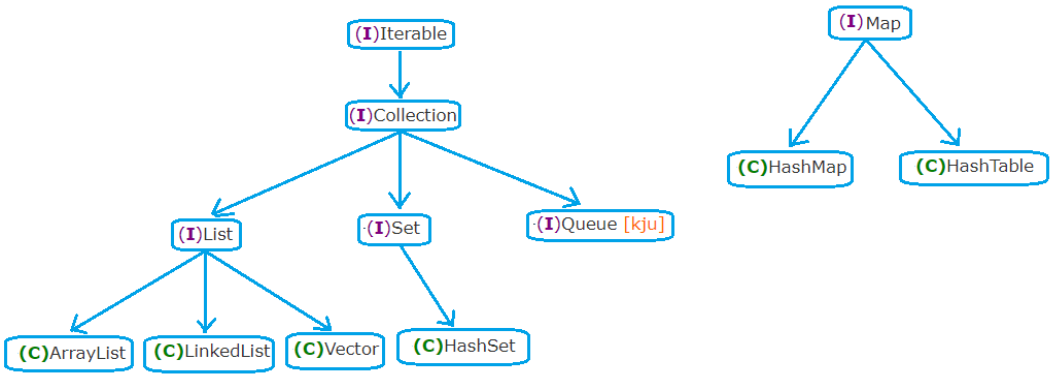
基本数据类型



包装类型

包装类型是对基本数据类型不足之处的补充。
基本数据类型的传递方式是值传递，而包装类型是引用传递，同时提供了很多数据类型间转换的方法。
Java1.5 以后可以自动装箱和拆箱。

集合



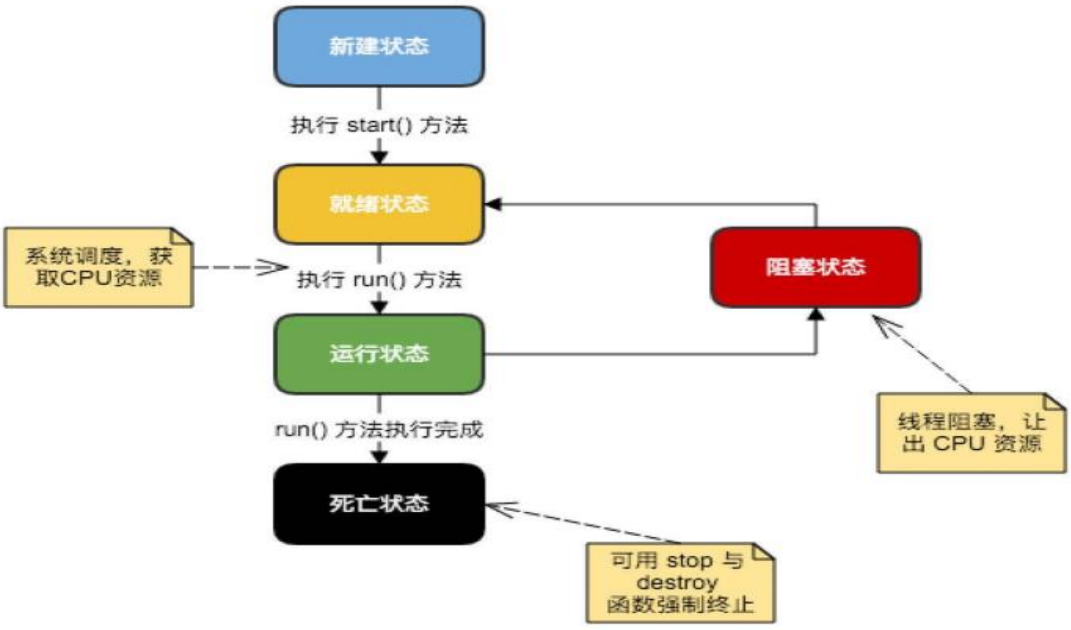
List：有序、可重复。可以通过索引快速查找，但进行增删操作时后续的数据需要移动，所以增删速度慢。

Set：无序、不可重复。

Map：键值对、键唯一、值不唯一。Map 集合中存储的是键值对，键不能重复，值可以重复。根据键得到值，对 map 集合遍历时先得到键的 set 集合，对 set 集合进行遍历，得到相应的值。

多线程

生命周期



新建状态: 一个新产生的线程从新状态开始了它的生命周期。它保持这个状态直到程序 start 这个线程。
运行状态:当一个新状态的线程被 start 以后，线程就变成可运行状态，一个线程在此状态下被认为是开始执行其任务
就绪状态:当一个线程等待另外一个线程执行一个任务的时候，该线程就进入就绪状态。当另一个线程给就绪状态的线程发送信号时，该线程才重新切换到运行状态。
休眠状态: 由于一个线程的时间片用完了，该线程从运行状态进入休眠状态。当时间间隔到期或者等待的时间发生了，该状态的线程切换到运行状态。
终止状态: 一个运行状态的线程完成任务或者其他终止条件发生，该线程就切换到终止状态。

热门面试问题:

1、什么是 GC？为什么要有 GC?

GC（Garbage Collection）是垃圾收集的意思，负责清除对象并释放内存。Java 提供的 GC 功能可以自动检测对象是否超过作用域从而达到自动回收内存的目的，从而防止内存泄漏。

2、final, finally 和 finalize 的区别?

final 用于声明属性，方法和类，表示属性不可变，方法不可被重写，类不可被继承。
finally 是异常处理语句结构的一部分，表示总是执行。
finalize 是 object 类的一个方法，在垃圾收集器执行的时候会调用这个对象回收的方法，工垃圾收集时其他资源的回收，比如关闭文件。

3、什么是单例模式？实现步骤?

单例模式保证了对象唯一。分为懒汉式（在类加载时不初始化）和饿汉式（在类加载时就完成了初始化，所以类加载比较慢，但获取对象的速度快）。
实现步骤：私有化构造函数、创建一个静态的私有对象、提供公共的访问方法。

4、ArrayList 和 LinkedList 有何区别?

ArrayList 是基于动态数组的数据结构，LinkedList 是基于链表的数据结构;
对于随机访问 get 和 set，ArrayList 较优，因为 LinkedList 要移动指针;
对于新增和删除操作 add 和 remove，LinedList 较优，因为 ArrayList 要移动数据。

5、HashMap 和 Hashtable 的区别?

HashMap 允许空键值，Hashtable 不允许;
HashMap 继承自 AbstractMap，Hashtable 继承自 Dictionary 类，两者都实现了 Map 接口;
HashMap 的方法不是同步的，Hashtable 的方法是同步的。

6、Iterater 和 ListIterator 之间有什么区别?

Iterator 用来遍历 Set 和 List 集合，而 ListIterator 只能遍历 List;
Iterator 只可以向前遍历，而 LIstIterator 可以双向遍历;
ListIterator 从 Iterator 接口继承，然后添加了一些额外的功能，比如添加一个元素、替换一个元素、获取前面或后面元素的索引位置。

7、创建线程的方式?

继承 Thread 类
实现 Runnable 接口
使用 Executor 框架

8、什么是死锁?

两个线程或两个以上线程都在等待对方执行完毕才能继续往下执行的时候就发生了死锁。结果就是这些线程都陷入了无限的等待中。

9、wait()与 sleep()的区别?

sleep()来自 Thread 类，wait()来自 Object 类;
调用 sleep()方法，线程不会释放对象锁。而调用 wait 方法线程会释放对象锁;
sleep()睡眠后不出让系统资源，wait 让其他线程可以占用 CPU;
sleep(milliseconds)需要指定一个睡眠时间，时间一到会自动唤醒。而 wait()需要配合 notify()或者 notifyAll()使用。

10、什么是 ThreadLocal? ThreadLocal 和 Synchonized 的区别?

线程局部变量。是局限于线程内部的变量，属于线程自身所有，不在多个线程间共享。Java 提供 ThreadLocal 类来支持线程局部变量，是一种实现线程安全的方式。
synchronized 是利用锁的机制，使变量或代码块在某一时该只能被一个线程访问。而 ThreadLocal 为每一个线程都提供了变量的副本，使得每个线程在某一时间访问到的并不是同一个对象，这样就隔离了多个线程对数据的数据共享。