1、Spring IOC的问题

（1）IOC就是控制反转，指将创建对象的控制权转移给Spring框架进行管理，并由Spring根据配置文件去创建实例和管理各个实例之间的依赖关系，对象与对象之间松散耦合。DI是依赖注入，和控制反转是同一个概念的不同角度的描述，即应用程序在运行时依赖IOC容器来动态注入对象需要的外部依赖。

（2）BeanFactory和ApplicationContext的联系和区别

它们是核心容器的两个接口。

1. BeanFactory：对象的创建采用延迟加载的方式，即什么时候根据id获取对象，什么时候才真正创建对象，比较适合多例对象；
2. ApplicationContext：创建核心容器时，对象的创建采用的策略是立即加载，只要一读完配置文件，就马上创建配置文件中配置的所有对象，比较适合单例对象。

```
//获取核心容器对象
ApplicationContext ac = new ClassPathXmlApplicationContext("application.xml");
//根据id获取对象
Person p=(Person)ac.getBean("person");
//或  Person p1=ac.getBean("person",Person.class);
```

（3）xml创建bean的方式

1. 默认构造函数

```
<bean id="id名" class="全限定类名"></bean>
```

2. 实例工厂

```
public class InstanceFactory {
    public IAccount getAccount(){
        return new AccountImpl();
    }
}
```

要调用getAccount方法，创建IAccount的实现类对象

```
<bean id="instanceFactory" class="InstanceFactory的全限定类名"></bean>
<bean id="account" factory-bean="instanceFactory" factory-
method="getAccount"></bean>
```

相当于先创建InstanceFactory类的对象，再调用里面的方法。

3. 静态工厂

```
public class InstanceFactory {
    public static IAccount getAccount(){
        return new AccountImpl();
    }
}
```

静态方法可以直接用类名调用

```
<bean id="account" class="InstanceFactory的全限定类名" factory-
method="getAccount"></bean>
```

需要一个factory-method属性，告诉类需要调用哪个静态方法。

(4) 注解创建bean的方式

1. @Component系列

```
@Component：把当前类的对象存入spring容器，属性value用于指定bean的id，不写默认当前类名，且首字母小写
@Controller：用在表现层
@Service：用在业务层
@Repository：用在持久层
@Configuration：指定当前类是一个配置类
```

2. 依附于@Configuration的注解

@Bean：把当前方法的返回值作为bean对象存入spring的ioc容器。

```
@Configuration
public class Demo {
    @Bean
    public Demo demo(){
        return new Demo();
    }
}
```

@Import：用于导入其他配置类，加载此主配置时，也加载import导入的配置类

```
@Configuration
@Import(Config.class)
public class Demo {
}
```

@ComponentScan：一般和@Configuration注解一起使用，指定spring在创建容器时要扫描的包

```
@ComponentScan
public class BeanConfig {
}
```

(5) 依赖注入的方式

1. 构造函数注入

```
<bean id="user" class="全限定类名">
    <constructor-arg name="username" value="Tom"></constructor-arg>
</bean>
```

2. set方法注入

```
<bean id="user" class="全限定类名">
    <property name="username" value="Tom"></property>
</bean>
```

3. 注解方式注入

（6）依赖注入的相关注解

1. @Autowired：自动按照类型注入，只要容器中有唯一的一个bean对象的类型和要注入的变量类型匹配，就可以注入成功，如果没有，就报错，多个匹配也会报错。
2. @Qualifier：在按照类型注入的基础上，再按照名称注入，给类成员注入时不能单独使用，要和Autowired一起使用，但给方法参数注入时可以，例：(@Qualifier(id名) 参数)。
3. @Resource：直接按照bean的id注入，可以独立使用。
4. @Value：用于注入基本类型和String类型的数据。

2、mybatis概述，开发步骤、一对多映射、Mybatis的sql语句中#和$的区别

（1）mybatis是基于java的持久层框架，内部封装了jdbc，使开发者只需关注sql语句本身，而不需要花费精力去处理加载驱动、创建连接、创建statement等繁杂的过程。

（2）开发步骤：

1. 导包，包括mybatis核心包、依赖包、mysql驱动包；
2. 写核心配置文件SqlMapConfig.xml，里面包含了连接数据库的信息，映射配置文件的位置信息；
3. 写映射配置文件（每个映射配置文件与一个持久层的mapper接口对应），

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!-- namespace：命名空间，用于隔离sql，使用动态代理时找到相应的接口 -->
<mapper namespace="it.pra.dao.IStudent">
	<!-- 查询所有 -->
	<select id="findAll" resultType="it.pra.javabean.Student">
		select * from student;
	</select>
	<!-- 插入一条数据 -->
	<insert id="insertStu" parameterType="it.pra.javabean.Student">
		insert into student(SId,Sname,Sage,Ssex) values(#{SId},#{Sname},#{Sage},#{Ssex});
	</insert>
	<!-- 更新数据 -->
	<update id="updateStu" parameterType="it.pra.javabean.Student">
		update student set Ssex=#{Ssex} WHERE SId=#{SId};
	</update>
	<!-- 删除数据 -->
	<delete id="deleteById" parameterType="Integer">
		delete from student where SId=#{SId};
	</delete>
</mapper>
```

映射配置文件Mapper的约束

使用实现类实现时可以自己定义；使用动态代理实现时，要写接口相对于src的地址

方法名

返回值类型，属性名与数据表的列名一致时，可使用自动映射；不一致时，使用resultMap属性，手动映射

参数类型

与属性名对应

4. 通过接口中的方法，找到对应的sql语句，执行并返回结果。

```
//加载核心配置文件
InputStream is=Resources.getResourceAsStream("sqlMapConfig.xml");
//创建SqlSessionFactory
SqlSessionFactory ssf=new SqlSessionFactoryBuilder().build(is);
//创建SqlSession
SqlSession sqlSession=ssf.openSession();
//创建代理对象（帮助接口生成一个实现类）
IStudent iStudent=sqlSession.getMapper(IStudent.class);
//执行查询所有操作
List<Student> students=iStudent.findAll();
for(Student s:students) {
	System.out.println(s);
}
sqlSession.close();
is.close();
```

接口

在控制台中打印，要写log4j.properties配置文件

若对数据库中的数据表进行了修改（插入，更新，删除），在sqlSession关闭前，要先提交：sqlSession.commit();

（3）实体类属性名与数据表列名不一致时，可采用手动映射，也可用在sql语句中写别名的方式。

```
<resultMap id="orderMap" type="实体类的全限定类名">
    <id property="sid" column="tid"></id>
    <result property="sname" column="tname"></result>
    <association property="user" column="外键列名" javaType="User实体类全限定类名">
一对一的关系映射
        <id property="uid" column="uid"></id>
        <result property="uname" column="uname"></result>
    </association>
    <collection property="goods" ofType="goods实体类的全限定类名"> 一对多的关系映射
        <id property="gid" column="gid"></id>
        <result property="gname" column="gname"></result>
    </collection>
</resultMap>
```

（4）Mybatis的sql语句中#和$的区别：

#将传入的数据都当成一个字符串，会对自动传入的数据加一个引号，能够很大程度防止sql注入；#{} 这种取值是编译好SQL语句再取值，

$将传入的数据直接显示在sql中，没有引号，${} 这种是取值以后再去编译SQL语句，不能防止sql注入，而${}一般用于order by的后面，Mybatis不会对这个参数进行任何的处理，直接生成了sql语句。例：传入一个年龄age的参数，select * from 表名 order by ${age}。

3、inner join和outer join的区别、where和having的区别、用group by时，对select后查询的字段有什么要求

（1）inner join和outer join的区别：

1. A inner join B 得到的结果是两张表的交集；
2. A left outer join B 得到的结果是A表的全部数据和AB两张表的交集部分；
3. A right outer join B 得到的结果是B表的全部数据和AB两张表的交集部分。

（2）where和having的区别：

1. where是一个约束声明；having是一个过滤声明；
2. where是在结果返回之前起作用的；having是在查询返回结果集以后，对查询结果进行过滤的操作；
3. where中不能使用聚合函数；having中可用使用聚合函数。聚合函数需要配合group by使用。
4. where和having的执行顺序：where 早于 group by 早于 having.

（3）用group by时，对select后查询的字段有什么要求：如果在有group by的操作中,select后面接的结果集字段只有两种: 要么就只有group by后出现的字段,要么就是group by后出现的字段 + 聚合函数的组合。

4、Exception和Error的区别、在什么情况下需要自定义异常，如何在发生自定义异常时进行回滚操作、Exception和RuntimeException的区别和联系。

（1）Exception和Error的区别：

1. Exception和Error都是继承于Throwable 类，在 Java 中只有 Throwable 类型的实例才可以被抛出（throw）或者捕获（catch），它是异常处理机制的基本组成类型。
2. Exception是java程序运行中**可预料**的异常情况，可以获取到这种异常，并且对这种异常进行业务外的处理。
3. Error是java程序运行中**不可预料**的异常情况，这种异常发生以后，会直接导致JVM不可处理或者不可恢复的情况。所以这种异常不可能抓取到，比如OutOfMemoryError、NoClassDefFoundError等。

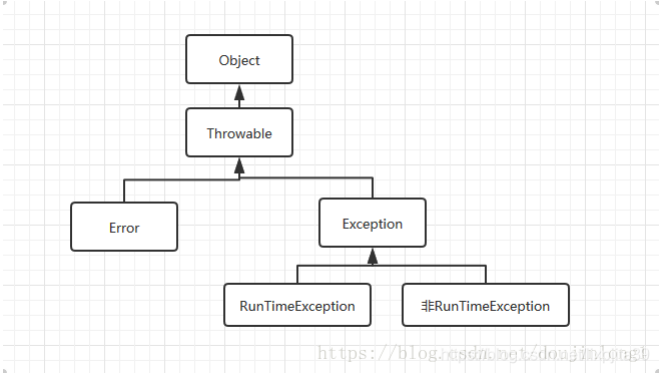
4. Exception又分为检查性异常和非检查性异常，**检查性异常** 必须在编写代码时，使用try catch捕获；**非检查性异常** 在代码编写时，可以忽略捕获操作（比如：ArrayIndexOutOfBoundsException），这种异常是在代码编写或者使用过程中通过规范可以避免发生的。

（2）在什么情况下需要自定义异常：系统定义的异常有限，有时会发生满足不了需求的情况；而且自定义异常可以自己定义异常的名称和特有内容，增强阅读性，方面查找。

（3）如何在发生自定义异常时进行回滚操作：

1. spring默认事务管理：默认当一个方法出现RunTimeException（运行期异常）时会自动回滚事务。
2. 用注解方法进行自定义异常回滚：@Transactional(rollbackFor = MyException.class)
3. 在catch语句中手动回滚：TransactionAspectSupport.currentTransactionStatus().setRollbackOnly();

（4）Exception和RuntimeException的区别和联系：



非RuntimeException：受检查异常，必须要开发者解决以后才能编译通过，解决的方法有两种：1> throw到上层；2> try...catch处理;

RuntimeException：运行时异常，又称不受检查异常。代码中可能会有RuntimeException时，java编译检查时不会告诉你有这个异常，但是在实际运行代码时会暴露出来，比如1/0，空指针等。如果不处理，也会被java自己处理。

5、Linux 中kill和kill -9的区别、Linux常用命令，打包、修改文件权限、在文本里找出关键词等。

（1）Linux 中kill和kill -9的区别：

1. 默认情况下，kill命令的参数是-15，代表的信号是sigterm，告诉进程需要被关闭，请自行停止运行并退出。进程接收到信号后，可能会发送以下事情：1> 进程立刻停止；2> 进程释放相应资源后再停止；3> 进程可能仍然继续运行。
2. kill -9代表的信号是sigkill，表示强制杀死该进程，需要立即退出，这个信号不能被捕获也不能被忽略。

（2）打包和解包：

1. tar -cvf 目标路径 需打包文件目录 （只打包不压缩）

   tar -xvf 解压目录 -C 解压地址
2. tar -zcvf 目标路径 需打包文件目录 （打包并压缩）

   tar -zxvf 目标路径 需打包文件目录
3. zip 目标路径 需打包文件目录

   unzip 需解压文件名

（3）修改文件权限：

```
chmod u=rwx  文件名（文件夹名）  u表示user（拥有者）
chmod u=r,g=r,o=r  文件名（文件夹名） g表示group，o表示other其他人
数字表示法：r是4 读权限，w是2 写权限；x是1 执行权限
chmod 137  第一个数字表示user权限，第二个数字表示group权限，第三个数字表示other权限
```

（4）在文本里找出关键词：grep [-选项] "搜索内容" 文件名

grep "nf" xxx.txt 搜索xxx.txt中含有nf的内容 搜索内容可以用正则表达式

（5）常用命令

```
pwd：显示当前路径
cp： 拷贝
find：在特定的目录下搜索符合条件的文件 find ./ -name test.sh
ps：查看进程信息
top：动态显示正在运行的程序的情况
ifconfig：查看或配置网卡信息
```

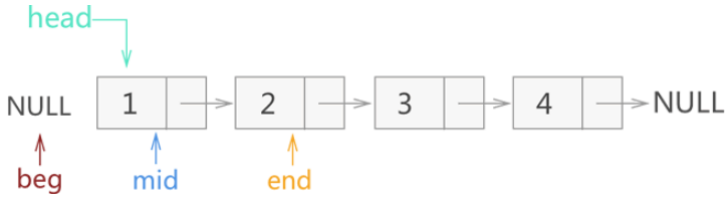
6、如何判断链表中是否有环、链表反转

（1）如何判断链表中是否有环：

快慢指针的方法，定义p、q两个指针，p指针每次向前走一步，q每次向前走两步，若p或q遇到了bull，则证明该链表没有环；若在某个时刻出现p==q时，则存在环。

（2）链表反转：

1. 迭代反转链表：



借助3个指针，3个指针每次各向后移动一个节点，直到mid指向链表中的最后一个节点。这 3 个指针每移动之前，都需要做一步操作，即改变 mid 所指节点的指针域，令其指向 beg 。

2. 递归反转链表：

和迭代反转法的思想恰好相反，递归反转法的实现思想是从链表的尾节点开始，依次向前遍历，遍历过程依次改变各节点的指向，即另其指向前一个节点

3. 头插入法反转链表：

是指在原有链表的基础上，依次将位于链表头部的节点摘下，然后采用从头部插入的方式生成一个新链表，则此链表即为原链表的反转版。

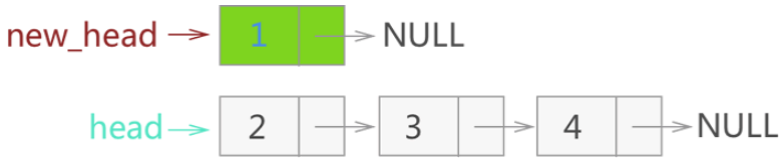
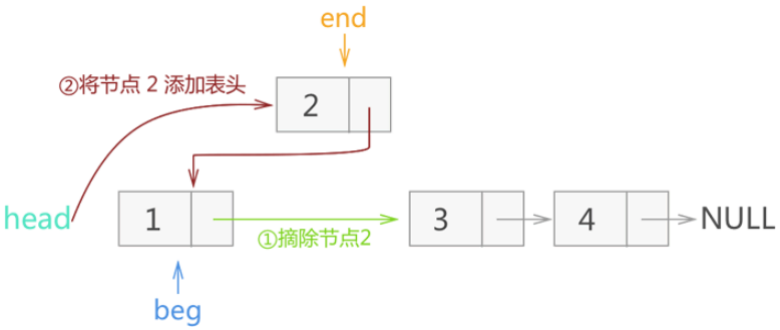


图 13 从原链表摘除节点 1，再添加到新链表中

4. 就地逆置法反转链表：

就地逆置法和头插法的实现思想类似，唯一的区别在于，头插法是通过建立一个新链表实现的，而就地逆置法则是直接对原链表做修改，从而实现将原链表反转。



初始状态下，令 beg 指向第一个节点，end 指向 beg->next， 将 end 所指节点 2 从链表上摘除，然后再添加至当前链表的头部。

7、接口，后期如何向接口中添加方法。

（1）接口中可以定义的内容：

1. Java7中：常量（定义格式：[public] [static] [final] 数据类型 常量名=数据值；常量必须赋值，而且一旦赋值不能改变；常量名称完全大写，用下划线进行分隔）；

   抽象方法（定义格式：[public] [abstract] 返回值类型 方法名(参数列表)； ）
2. Java8中：默认方法；静态方法。
3. Java9中：私有方法。

（2）默认方法：（后期如何向接口中添加方法 的解决方案）

1. 定义格式：public default 返回值类型 方法名(参数列表) {方法体}；
2. 作用：可以解决接口升级问题，即 接口在后续添加了方法，但是该接口以前已经有实现类，若添加抽象方法，以前的实现类就会报错；添加默认方法，以前的实现类不会报错，且能调用默认方法。
3. 默认方法的使用：1> 可以通过接口实现类对象，直接调用； 2> 可以被接口实现类覆盖重写。

（3）静态方法：

1. 定义格式：public static 返回值类型 方法名(参数列表){方法体};
2. 调用：通过接口名称，直接调用。

（4）私有方法：抽取一个共有方法，用来解决两个默认方法之间的重复代码的问题，但是这个共有方法不应该让实现类使用，应该是私有化的。

1. 普通私有方法：解决多个默认方法之间重复代码问题，格式：private 返回值类型 方法名(参数列表){方法体};
2. 静态私有方法：解决多个静态方法之间重复代码问题，格式：private static 返回值类型 方法名(参数列表){方法体};

（4）注意事项：

1. 接口中不能有静态代码块或构造方法；
2. 实现类所实现的多个接口中，若存在重复的抽象方法，只需要覆盖重写一次即可；
3. 如果实现类实现的多个接口中，存在重复的默认方法，实现类一定要对冲突的默认方法进行覆盖重写；
4. 一个类 如果直接父类中的方法，和接口中的默认方法产生了冲突，优先用父类的方法。

8、单例模式（还有双重检验锁、静态内部类等的方式）

```
public class Singleton {    单例模式的懒汉式，线程不安全
    //一个静态的实例
    private static Singleton singleton;
    //私有化构造函数
    private Singleton(){}
    //给出一个公共的静态方法返回一个单一实例
    public static (synchronized) Singleton getInstance(){ 加上synchronized，线程安全，但效率低
        if (singleton == null) {  //决定去修改
            singleton = new Singleton(); //执行修改
        }
        return singleton;
    }
}
```

```
public class Singleton1 {  饿汉式，线程安全
    private Singleton1() {}
    private static final Singleton1 single = new Singleton1();
    //静态工厂方法
    public static Singleton1 getInstance() {
        return single;
    }
}
```

9、多线程的作用和应用场景、如何保证指定线程执行结束之后执行新线程

（1）多线程的作用：提高CPU的利用率，更好地利用系统资源；用尽可能少的时间来对用户的要求做出响应...

（2）应用场景：

1. 同时处理多个后台任务：如 同时进行查杀木马、垃圾处理、性能优化等；
2. 分布式计算：当处理一个比较大的耗时任务时，可以将该任务切割成多个小任务，然后开启多个线程同时处理这些小任务，如 使用多线程下载，提高下载速度；
3. 异步处理任务：需要处理一个耗时操作，并且不要立刻知道处理结果，可以开启后台线程异步处理该耗时操作。

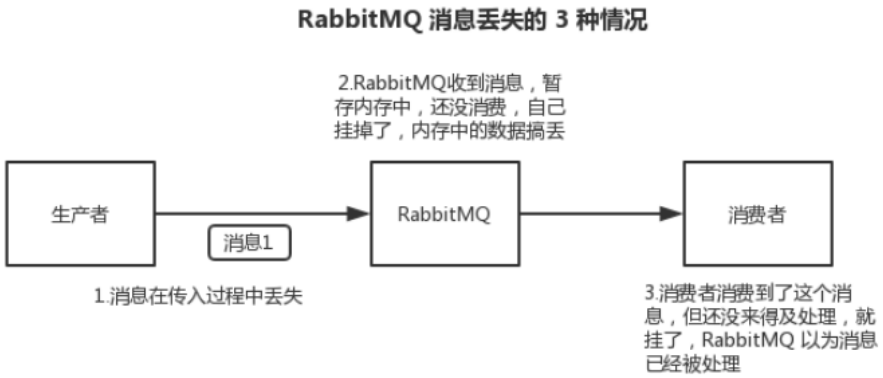
（3）如何保证指定线程执行结束之后执行新线程：thread.join(); 当前线程A等待thread线程终止之后才能继续运行。底层是通过wait/notify实现的。

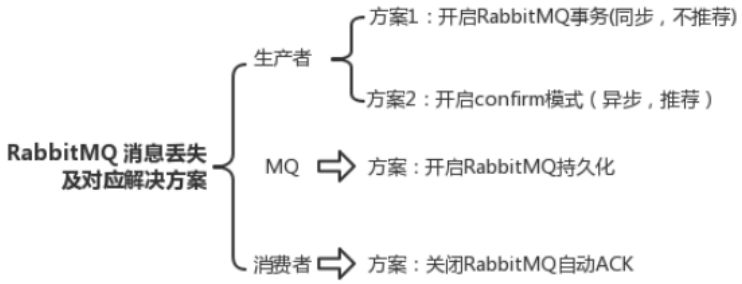
10、消息队列通信的模式、rabbitMQ的消息模型，如何保证消息不丢失、队列的应用场景、优势。

（1）消息队列通信的模式：点对点模式、发布订阅模式...

（2）rabbitMQ的消息模型：基本消息模型（一个生产者，一个队列，一个消费者），work消息模型（一个生产者，一个队列，多个消费者），订阅模型（一个生产者，一个交换机，多个队列，多个消费者）

（3）如何保证消息不丢失：





针对生产者：

1. 开启RabbitMQ事务：用RabbitMQ提供的事务功能，在生产者发送数据之前开启RabbitMQ事务channel.txSelect，然后发送消息；如果消息没有成功被RabbitMQ接收到，生产者会收到异常报错，此时就可以回滚事务channel.txRollback，然后重试发送消息；如果收到了消息，可以提交事务channel.txCommit。缺点：RabbitMQ事务机制是同步的，生产者发送消息会同步阻塞，等待成功或失败的返回信息，太耗性能。
2. 使用confirm机制：事务机制和confirm机制最大的不同在于，事务机制是同步的，提交一个事务之后会阻塞在那儿；confirm机制是异步的，发送消息之后就可以发送下一个消息，RabbitMQ接收了消息之后会异步回调接口，通知你消息收到了。

```
//开启confirm
channel.confirm();
//发送成功回调
public void ack(String messageId){

}
// 发送失败回调
public void nack(String messageId){
    //重发该消息
}
```

针对RabbitMQ：

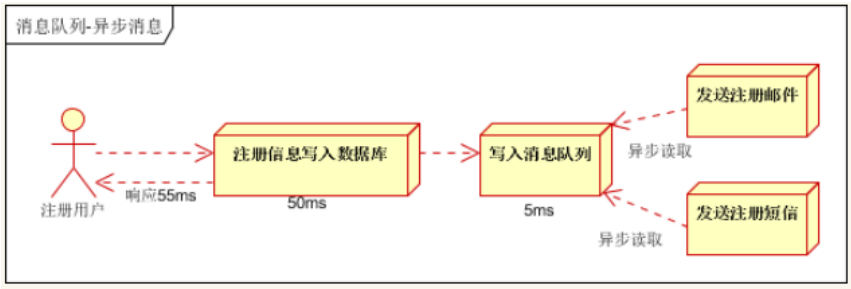
1. 消息持久化： Exchange 设置持久化、Queue 设置持久化、Message持久化发送；
2. 设置集群镜像模式；
3. 消息补偿机制：消息补偿机制需要建立在消息要写入DB日志，发送日志，接受日志，两者的状态必须记录。

针对消费者：

1. ACK确认机制：在消费消息的回调函数中写 channel.basicAck(envelope.getDeliveryTag(),false);手动确认消息；

（4）队列的应用场景、优势：

1. 业务模块解耦：用户下单后，订单系统需要通知库存系统，若使用订单系统调用库存系统的接口，假如库存系统无法访问，则订单减库存将失败，从而导致订单失败。
2. 数据延迟处理：
3. 高并发限流削峰：秒杀活动，一般会因为流量过大，流量暴增，导致应用挂掉。为解决这个问题，一般需要在应用前端加入消息队列。可以控制活动的人数，可以缓解短时间内高流量压垮应用。
4. 异步通信：



用户的响应时间相当于是注册信息写入数据库的时间，也就是50毫秒。写入消息队列后，直接返回，因为写入消息队列的速度很快，基本可以忽略。注册邮件，发送短信异步进行，因此用户的响应时间可能是50毫秒。系统的吞吐量会提高。

11、sql的执行时间，如何判断是否使用了索引，联合索引查询哪些字段会用到索引

（1）sql的执行时间：`show profiles`是mysql提供的，可以用来分析当前会话中语句执行的资源消耗情 况。可以用来SQL的调优测量。

1. 查看是否支持profiling：`select @@have_profiling;` 若结果为Yes，代表支持；
2. 查看profiling是否是开启状态：`select @@profiling;` 结果为0，表示未开启，结果为1，表示开启。若没有开启，使用 `set profiling=1;`开启。
3. 执行一系列查询语句后，输入`show profiles`查询每一条sql的执行时间；

Query_ID	Duration	Query
54	0.002136	SHOW STATUS
55	0.00256675	SELECT QUERY_ID, SUM(DURATION
56	0.00239175	SELECT STATE AS `状态`, ROUND(SU
57	0.0002845	SET PROFILING=1
58	0.0024605	SHOW STATUS

4. 查询每一条sql每个阶段的执行时间：`show  profile for query query_id;`其中query_id是3中查询的query_id；
5. 查看线程在什么资源上耗费过高。`show profile block io,cpu for query 16;`类型 all、cpu、block io 、context、switch、page faults。

（2）如何判断是否使用了索引：使用explain对sql进行分析，例`EXPLAIN SELECT * from yuwu_vue WHERE id=96 and role_id=2`，查询结果为：

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	yuwu_vue	(Null)	const	PRIMARY	PRIMAR	4	const	1	100	(Null)

字段解释：

字段	含义
id	select查询的序列号，表示查询中执行select子句或者操作表的顺序
select_type	select的类型，simple表示简单表，即不使用表连接或子查询
table	输出结果集的表
type	表示表的连接类型
possible_key	表示查询时，可能使用的索引
key	表示实际使用的索引
key_len	索引字段的长度
rows	扫描行的数量
filtered	存储引擎返回的数据在server层过滤后，剩下多少满足查询的记录数量的比例
extra	执行情况的说明和描述

（3）联合索引查询哪些字段会用到索引：如果这个字段在联合索引中所有字段的第一个，那就会用到索引，否则就无法使用到索引。例：联合索引 IDX(字段A,字段B,字段C,字段D)，当使用到字段A查询时，索引 IDX 就会使用到；如果仅使用字段B或字段C或字段D查询，或BCD三个字段的任意组合，则索引 IDX 都不会用到。