# 面试题

**Java基础**  
1.HashMap的源码，实现原理，JDK8中对HashMap做了怎样的优化。  
2.HaspMap扩容是怎样扩容的，为什么都是2的N次幂的大小。  
3.HashMap，HashTable，ConcurrentHashMap的区别。  
4.极高并发下HashTable和ConcurrentHashMap哪个性能更好，为什么，如何实现的。  
5.HashMap在高并发下如果没有处理线程安全会有怎样的安全隐患，具体表现是什么。  
6.java中四种修饰符的限制范围。  
7.Object类中的方法。  
8.接口和抽象类的区别，注意JDK8的接口可以有实现。  
9.动态代理的两种方式，以及区别。  
10.Java序列化的方式。  
11.传值和传引用的区别，Java是怎么样的，有没有传值引用。  
12.一个ArrayList在循环过程中删除，会不会出问题，为什么。  
13.@transactional注解在什么情况下会失效，为什么。  
**数据结构和算法**  
1.B+树。  
2.快速排序，堆排序，插入排序（其实八大排序算法都应该了解）  
3.一致性Hash算法，一致性Hash算法的应用。  
**JVM**  
1.JVM的内存结构。  
2.JVM方法栈的工作过程，方法栈和本地方法栈有什么区别。  
3.JVM的栈中引用如何和堆中的对象产生关联。  
4.可以了解一下逃逸分析技术。  
5.GC的常见算法，CMS以及G1的垃圾回收过程，CMS的各个阶段哪两个是Stop the world的，CMS会不会产生碎片，G1的优势。  
6.标记清除和标记整理算法的理解以及优缺点。  
7.eden survivor区的比例，为什么是这个比例，eden survivor的工作过程。  
8.JVM如何判断一个对象是否该被GC，可以视为root的都有哪几种类型。  
9.强软弱虚引用的区别以及GC对他们执行怎样的操作。  
10.Java是否可以GC直接内存。  
11.Java类加载的过程。  
12.双亲委派模型的过程以及优势。  
13.常用的JVM调优参数。  
14.dump文件的分析。  
15.Java有没有主动触发GC的方式（没有）。  
**多线程**  
1.Java实现多线程有哪几种方式。  
2.Callable和Future的了解。  
3.线程池的参数有哪些，在线程池创建一个线程的过程。  
4.volitile关键字的作用，原理。  
5.synchronized关键字的用法，优缺点。  
6.Lock接口有哪些实现类，使用场景是什么。  
7.可重入锁的用处及实现原理，写时复制的过程，读写锁，分段锁（ConcurrentHashMap中的segment）。  
8.悲观锁，乐观锁，优缺点，CAS有什么缺陷，该如何解决。  
9.ABC三个线程如何保证顺序执行。  
10.线程的状态都有哪些。  
11.sleep和wait的区别。  
12.notify和notifyall的区别。  
13.ThreadLocal的了解，实现原理。  
  
**数据库相关**  
1.常见的数据库优化手段  
2.索引的优缺点，什么字段上建立索引  
3.数据库连接池。  
4.durid的常用配置。  
**计算机网络**  
1.TCP，UDP区别。  
2.三次握手，四次挥手，为什么要四次挥手。  
3.长连接和短连接。  
4.连接池适合长连接还是短连接。  
**设计模式**  
1.观察者模式  
2.代理模式  
3.单例模式，有五种写法，可以参考文章单例模式的五种实现方式  
4.可以考Spring中使用了哪些设计模式  
**分布式相关**  
1.分布式事务的控制。  
2.分布式锁如何设计。  
3.分布式session如何设计。  
4.dubbo的组件有哪些，各有什么作用。  
5.zookeeper的负载均衡算法有哪些。  
6.dubbo是如何利用接口就可以通信的。  
**缓存相关**  
1.redis和memcached的区别。  
2.redis支持哪些数据结构。  
3.redis是单线程的么，所有的工作都是单线程么。  
4.redis如何存储一个String的。  
5.redis的部署方式，主从，集群。  
6.redis的哨兵模式，一个key值如何在redis集群中找到存储在哪里。  
7.redis持久化策略。  
**框架相关**  
1.SpringMVC的Controller是如何将参数和前端传来的数据一一对应的。  
2.Mybatis如何找到指定的Mapper的，如何完成查询的。  
3.Quartz是如何完成定时任务的。  
4.自定义注解的实现。  
5.Spring使用了哪些设计模式。  
6.Spring的IOC有什么优势。  
7.Spring如何维护它拥有的bean。  
**一些较新的东西**  
1.JDK8的新特性，流的概念及优势，为什么有这种优势。  
2.区块链了解。  
3.如何设计双11交易总额面板，要做到高并发高可用。  
**一些小建议**  
1.可以去leetcode上刷题换换思路。  
2.八大排序算法一定要手敲一遍（快排，堆排尤其重要）。  
3.了解一些新兴的技术。  
4.面试之后面试官都会问你有没有什么问题，千万不要没问题，也别傻乎乎的问一些敏感问题。  
5.了解你要面试的公司的产品及竞争产品。

**并发编程：**

1. 什么是多线程并发和并行？
2. 什么是线程安全问题？
3. 什么是共享变量的内存可见性问题？
4. 什么是Java中原子性操作？
5. 什么是Java中的CAS操作,AtomicLong实现原理？
6. 什么是Java指令重排序？
7. Java中Synchronized关键字的内存语义是什么？
8. Java中Volatile关键字的内存语义是什么？
9. 什么是伪共享,为何会出现，以及如何避免？
10. 什么是可重入锁、乐观锁、悲观锁、公平锁、非公平锁、独占锁、共享锁？
11. 讲讲ThreadLocal 的实现原理？
12. ThreadLocal 作为变量的线程隔离方式，其内部是如何做的？
13. 说说InheritableThreadLocal 的实现原理？
14. InheritableThreadLocal 是如何弥补 ThreadLocal 不支持继承的特性？
15. CyclicBarrier内部的实现与 CountDownLatch 有何不同？
16. 随机数生成器 Random 类如何使用 CAS 算法保证多线程下新种子的唯一性？
17. ThreadLocalRandom 是如何利用 ThreadLocal 的原理来解决 Random 的局限性？
18. Spring 框架中如何使用 ThreadLocal 实现 request scope 作用域 Bean？
19. 并发包中锁的实现底层（对AQS的理解）？
20. 讲讲独占锁 ReentrantLock 原理？
21. 谈谈读写锁 ReentrantReadWriteLock 原理？
22. StampedLock 锁原理的理解？
23. 谈下对基于链表的非阻塞无界队列 ConcurrentLinkedQueue 原理的理解？
24. ConcurrentLinkedQueue 内部是如何使用 CAS 非阻塞算法来保证多线程下入队出队操作的线程安全？
25. 基于链表的阻塞队列 LinkedBlockingQueue 原理。
26. 阻塞队列LinkedBlockingQueue 内部是如何使用两个独占锁 ReentrantLock 以及对应的条件变量保证多线程先入队出队操作的线程安全？
27. 分析下JUC 中倒数计数器 CountDownLatch 的使用与原理？
28. CountDownLatch 与线程的 Join 方法区别是什么？
29. 讲讲对JUC 中回环屏障 CyclicBarrier 的使用？
30. CyclicBarrier内部的实现与 CountDownLatch 有何不同？
31. Semaphore 的内部实现是怎样的？
32. 并发组件CopyOnWriteArrayList 是如何通过写时拷贝实现并发安全的 List？

**JVM**

1. Java 内存分配？
2. Java 堆的结构是什么样子的？
3. 什么是堆中的永久代（Perm Gen space）?
4. 说说各个区域的作用？
5. Java 中会存在内存泄漏吗，简述一下？
6. Java 类加载过程？
7. 描述一下 JVM 加载 Class 文件的原理机制?
8. 什么是类加载器？
9. 类加载器有哪些？
10. 什么是tomcat类加载机制？
11. 类加载器双亲委派模型机制？
12. 什么是GC? 为什么要有 GC？
13. 简述一下Java 垃圾回收机制？
14. 如何判断一个对象是否存活？
15. 垃圾回收的优点和原理，并考虑 2 种回收机制？
16. 垃圾回收器的基本原理是什么？
17. 垃圾回收器可以马上回收内存吗？有什么办法主动通知虚拟机进行垃圾回收？
18. 深拷贝和浅拷贝？
19. System.gc() 和 Runtime.gc() 会做些什么？
20. 什么是分布式垃圾回收（DGC）？它是如何工作的？
21. 串行（serial）收集器和吞吐量（throughput）收集器的区别是什么？
22. 在 Java 中，对象什么时候可以被垃圾回收？
23. 简述Minor GC 和 Major GC？
24. Java 中垃圾收集的方法有哪些？
25. 讲讲你理解的性能评价及测试指标？
26. 常用的性能优化方式有哪些？
27. 说说分布式缓存和一致性哈希？
28. 同步与异步？阻塞与非阻塞？
29. 什么是GC调优？
30. 常见异步的手段有哪些？

**Spring**

1. 为什么需要代理模式？
2. 讲讲静态代理模式的优点及其瓶颈？
3. 对Java 接口代理模式的实现原理的理解？
4. 如何使用 Java 反射实现动态代理？
5. Java 接口代理模式的指定增强？
6. 谈谈对Cglib 类增强动态代理的实现？
7. 怎么理解面向切面编程的切面？
8. 讲解OOP与AOP的简单对比？
9. 讲解JDK 动态代理和 CGLIB 代理原理以及区别？
10. 讲解Spring 框架中基于 Schema 的 AOP 实现原理？
11. 讲解Spring 框架中如何基于 AOP 实现的事务管理？
12. 谈谈对控制反转的设计思想的理解？
13. 怎么理解 Spring IOC 容器？
14. Spring IOC 怎么管理 Bean 之间的依赖关系，怎么避免循环依赖？
15. 对Spring IOC 容器的依赖注入的理解？
16. 说说对Spring IOC 的单例模式和高级特性？
17. BeanFactory 和 FactoryBean 有什么区别？
18. BeanFactory 和 ApplicationContext 又有什么不同？
19. Spring 在 Bean 创建过程中是如何解决循环依赖的？
20. 谈谈Spring Bean 创建过程中的设计模式？

**数据库**

1. MySQL 有哪些存储引擎啊？都有什么区别？
2. Float、Decimal 存储金额的区别？
3. Datetime、Timestamp 存储时间的区别？
4. Char、Varchar、Varbinary 存储字符的区别？
5. 对比一下B+树索引和 Hash索引？
6. MySQL索引类型有？
7. 如何管理 MySQL索引？
8. 对Explain参数及重要参数的理解？
9. 索引利弊是什么及索引分类？
10. 聚簇索引和非聚簇索引的区别？
11. B+tree 如何进行优化？索引遵循哪些原则？
12. 索引与锁有什么关系？
13. 还有什么其他的索引类型，各自索引有哪些优缺点？
14. 谈谈对Innodb事务的理解？
15. 说说数据库事务特点及潜在问题？
16. 什么是MySQL隔离级别？
17. 有多少种事务失效的场景，如何解决？
18. 一致性非锁定读和一致性锁定读是什么？
19. Innodb如何解决幻读？
20. 讲讲Innodb行锁？
21. 死锁及监控是什么？
22. 自增长与锁 ，锁的算法，锁问题，锁升级是什么？
23. 乐观锁的线程如何做失败补偿？
24. 高并发场景（领红包）如何防止死锁，保证数据一致性？
25. 谈谈MySQL的锁并发？
26. 查询优化的基本思路是什么？
27. 说说MySQL读写分离、分库分表？
28. 表结构对性能有什么影响?
29. 浅谈索引优化？
30. 说说Sql优化的几点原则？
31. MySQL表设计及规范？
32. 说说MySQL几种存储引擎应用场景？
33. MySQL常用优化方式有哪些？
34. MySQL常用监控？
35. MySQL瓶颈分析？

**缓存**

1. redis数据结构有哪些？
2. Redis缓存穿透，缓存雪崩？
3. 如何使用Redis来实现分布式锁？
4. Redis的并发竞争问题如何解决？
5. Redis持久化的几种方式，优缺点是什么，怎么实现的？
6. Redis的缓存失效策略？
7. Redis集群，高可用，原理？
8. Redis缓存分片？
9. Redis的数据淘汰策略？
10. redis队列应用场景？
11. 分布式使用场景（储存session）？

**网络编程**

1. TCP建立连接和断开连接的过程？
2. HTTP协议的交互流程，HTTP和HTTPS的差异，SSL的交互流程？
3. TCP的滑动窗口协议有什么用？
4. HTTP协议都有哪些方法？
5. Socket交互的基本流程？
6. 讲讲tcp协议（建连过程，慢启动，滑动窗口，七层模型）？
7. webservice协议（wsdl/soap格式，与restt办议的区别）？
8. 说说Netty线程模型，什么是零拷贝？
9. TCP三次握手、四次挥手？
10. DNS解析过程？
11. TCP如何保证数据的可靠传输的？

**分布式**

1. 什么是CAP定理？
2. 说说CAP理论和BASE理论？
3. 什么是最终一致性？最终一致性实现方式？
4. 什么是一致性Hash？
5. 讲讲分布式事务？
6. 如何实现分布式锁？
7. 如何实现分布式 Session?
8. 如何保证消息的一致性?
9. 负载均衡的理解？
10. 正向代理和反向代理？
11. CDN实现原理？
12. 怎么提升系统的QPS和吞吐？
13. Dubbo的底层实现原理和机制？
14. 描述一个服务从发布到被消费的详细过程？
15. 分布式系统怎么做服务治理？
16. 消息中间件如何解决消息丢失问题？
17. Dubbo的服务请求失败怎么处理？
18. 对分布式事务的理解？
19. 如何实现负载均衡,有哪些算法可以实现?
20. Zookeeper的用途,选举的原理是什么?
21. 讲讲数据的垂直拆分水平拆分？
22. zookeeper原理和适用场景？
23. zookeeper watch机制？
24. redis/zk节点宕机如何处理？
25. 分布式集群下如何做到唯一序列号？
26. 用过哪些MQ,怎么用的,和其他mq比较有什么优缺点,MQ的连接是线程安全的吗？
27. MQ系统的数据如何保证不丢失？
28. 列举出能想到的数据库分库分表策略？

# 动态编译

动态编译可以做一个浏览器端写Java代码，上传服务器编译和运行的在线评测系统，服务器动态加载某些类文件进行编译。

动态编译有两种做法：

1、 通过Runtime调用javac，启动新的进程去操作

Runtime run = Runtime.getRuntime();

Process process = run.exec(“javac –cp d:/myjava/ HelloWorld.java”);

2、 通过JavaCompiler动态编译

String sourceFile = ?;

JavaCompiler compuler = ToolProvider.getSystemJavaCompiler();

int result = compiler.run(null,null,null,sourceFile)；

System.out.println(result==0?“success”:”fail”);

第一个参数：为Java编译器提供参数

第二个参数：得到Java编译器的输出信息

第三个参数：接收编译器的错误信息

第四个参数：可变参数（String数组）能传入一个或多个Java源文件

result：result的值为0表示编译成功，非0表示编译失败

# 反射机制性能问题

setAccessible

– 启用和禁用访问安全检查的开关,值为 true 则指示反射的对象在使用时应该取消 Java 语

言访问检查。值为 false 则指示反射的对象应该实施 Java 语言访问检查。并不是为true

就能访问为false就不能访问。

– 禁止安全检查，可以提高反射的运行速度。

# 使用Filter和ThreadLocal组合来控制事务

两个问题？

1、为什么要使用Filter来控制事务。

2、ThreadLocal有什么用？

我们先来看第一个问题。我们以去结账生成订单为例。在结账生成订单的时候，我们要插入订单的信息，还要插入订单项的信息。还有修改图书的库存信息。这一系列的操作要么应该都成功 。要么应该都失败。比如说，当我们插入订单信息，成功之后，插入订单项之前抛了一个异常，而使得插入订单项，修改图书库存信息这一系列的操作都无法执行。那么数据库里就只有订单的信息，而找不到这个订单购买了什么商品。这种情况下，我们就需要使用数据库的事务来管理这一系列的数据库操作。当所有数据库的操作都成功之后 。我们再手动的提交事务。来确保这些操作的原子性。

现在我们来看看第二个问题。ThreadLocal类可以让我们在每个线程中存取当前线程使用的Connection对象，这样在下次获取的时候，就是同一个Connection连接

ThreadLocal的使用

ThreadLocal使用，我们只需要记住三点：

一：ThreadLocal类可以让我们以当前线程为key保存一个线程内部变量。而不必担心线程安全问题。

二：ThreadLocal类一般是和static静态关键字组合使用。

三：ThreadLocal类存储的变量在线程销毁后。会被虚拟机自动gc，释放。

我们先来看一下。在线程里保存变量，然后在线程中取自己保存的变量的情况

1）map来实现线程保存变量：

package com.atguigu.threadlocal;

import java.util.HashMap;

import java.util.Map;

import java.util.Random;

public class TestThreadLocal1 {

// 定义一个整型

private static Map<String, Integer> map = new HashMap<String, Integer>();

// 随机数对象

private static Random random = new Random(System.currentTimeMillis());

static class MyTask implements Runnable {

Map<String, Integer> map;

public MyTask(Map<String, Integer> map) {

super();

this.map = map;

}

public void run() {

System.out.println(Thread.currentThread().getName() + " -- begin");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

// 生成随机数，用于相加

int num = random.nextInt(1000);

//

System.out.println(Thread.currentThread().getName() + "生成一个随机数：" + num);

// 把随机数保存到map中

map.put(Thread.currentThread().getName(), num);

// 用线程做key获取自己的变量

System.out.println("获取" + Thread.currentThread().getName() + " -- "

+ map.get(Thread.currentThread().getName()));

System.out.println(Thread.currentThread().getName() + " -- end");

}

}

public static void main(String[] args) throws Exception {

Thread t1 = new Thread(new MyTask(map));

Thread t2 = new Thread(new MyTask(map));

t1.start();

t2.start();

}

}

2）ThreadLocal实现线程保存变量

package com.atguigu.threadlocal;

import java.util.Random;

public class TestThreadLocal2 {

// 定义一个整型

private static ThreadLocal<Integer> threadLocal = new ThreadLocal<Integer>();

// 随机数对象

private static Random random = new Random(System.currentTimeMillis());

static class MyTask implements Runnable {

ThreadLocal<Integer> i;

public MyTask(ThreadLocal<Integer> i) {

this.i = i;

}

public void run() {

System.out.println(Thread.currentThread().getName() + " -- begin");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

// 生成随机数，用于相加

int num = random.nextInt(1000);

//

System.out.println(Thread.currentThread().getName() + "生成一个随机数：" + num);

// 把随机数保存到map中

i.set(num);

// 用线程做key获取自己的变量

System.out.println("获取" + Thread.currentThread().getName() + " -- " + i.get());

System.out.println(Thread.currentThread().getName() + " -- end");

}

}

public static void main(String[] args) throws Exception {

Thread t1 = new Thread(new MyTask(threadLocal));

Thread t2 = new Thread(new MyTask(threadLocal));

t1.start();

t2.start();

}

}

经过上面两个小示例的代码，我们可以知道，在多个线程里我们可以使用ThreadLocal保存线程自己需要的变量，而不需要担心线程安全的问题。所以我们只可以使用ThreadLocal来保存数据库的连接，这样，在一次请求中。是一个线程处理所有的操作。这样可以保正一个请求，使用相同的Connection对象。可以确保多个操作在一个连接的一个事务中完成。来达到。要么都成功 。要么都失败的效果。

使用Filter和ThreadLocal组合管理事务

以生成订单为例。来展示如何使用Filter和ThreadLocal管理事务

先修改JDBCUtils中代码。使用ThreadLocal管理Connection对象

package com.atguigu.util;

import java.sql.Connection;

import java.sql.SQLException;

import com.mchange.v2.c3p0.ComboPooledDataSource;

/\*\*

\* 获取数据库连接的工具类

\*

\* @author wzg

\*

\*/

public class JDBCUtils {

private static ComboPooledDataSource dataSource = new ComboPooledDataSource("book\_devoloper");

/\*\*

\* 使用ThreadLocal保存Connection对象

\*/

private static ThreadLocal<Connection> connectionThreadLocal = new ThreadLocal<Connection>();

private JDBCUtils() {

}

/\*\*

\* 获取数据库连接

\*

\* @return 如果获取连接成功，返回数据的连接对象。<br/>

\* 如果获取数据库连接失败，则返回null

\*/

public static Connection getConnection() {

// 先从ThreadLocal中获取

Connection connection = connectionThreadLocal.get();

try {

if (connection == null) {

// 从c3p0中获取数据库连接

connection = dataSource.getConnection();

// 设置事务为手动提交

connection.setAutoCommit(false);

connectionThreadLocal.set(connection);

}

} catch (SQLException e) {

e.printStackTrace();

}

return connection;

}

/\*\*

\* 释放数据库连接

\*/

public static void closeConnection(Connection conn) {

if (conn != null) {

try {

conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

/\*\*

\* 释放数据库连接

\*/

public static void closeConnection() {

// 从线程ThreadLocal中获取

Connection conn = connectionThreadLocal.get();

if (conn != null) {

try {

// 事务提交

conn.commit();

// 事务关闭

conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

// 移除

connectionThreadLocal.remove();

}

public static void rollback() {

// 从线程ThreadLocal中获取

Connection conn = connectionThreadLocal.get();

if (conn != null) {

try {

// 事务回滚

conn.rollback();

// 关闭连接

conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

// 移除

connectionThreadLocal.remove();

}

public static void main(String[] args) {

for (int i = 0; i < 10; i++) {

System.out.println(getConnection());

}

}

}

然后修改BaseDaoImpl的代码

记住，在BaseDaoImpl里，都不要再关闭Connection，因为要让所有的操作都在同一个Connection中使用，而且所有的异常都要往外抛。直到让Filter可以拦截到，做rollback回滚操作。

package com.atguigu.dao.impl;

import java.lang.reflect.ParameterizedType;

import java.sql.Connection;

import java.sql.SQLException;

import java.util.List;

import org.apache.commons.dbutils.QueryRunner;

import org.apache.commons.dbutils.handlers.BeanHandler;

import org.apache.commons.dbutils.handlers.BeanListHandler;

import org.apache.commons.dbutils.handlers.ScalarHandler;

import com.atguigu.util.JDBCUtils;

/\*\*

\* 我们使用的是DBUtil操作数据库

\*

\* @author wzg

\*

\* @param <T>

\*/

public abstract class BaseDaoImpl<T> {

/\*\*

\* DBUtils操作数据库需要使用的类

\*/

protected QueryRunner queryRunner;

/\*\*

\* 保存BaseDaoImpl类的泛型对象class类型

\*/

protected Class<T> type;

@SuppressWarnings("unchecked")

public BaseDaoImpl() {

// 创建一个QueryRunner对象实例

queryRunner = new QueryRunner();

// 获取父类中带有泛型的父类的class类型

ParameterizedType supperClass = (ParameterizedType) getClass().getGenericSuperclass();

// 获取泛型中的具体的类型的class

type = (Class<T>) supperClass.getActualTypeArguments()[0];

}

/\*\*

\* 执行update，delete,insert语句

\*

\* @param sql

\* 要执行的sql语句

\* @param params

\* 执行的sql语句的参数

\* @return 如果语句执行成功返回true<br/>

\* 如果执行失败返回false

\* @throws SQLException

\*/

public int update(String sql, Object... params) {

Connection connection = null;

int updateCount = 0;

try {

connection = JDBCUtils.getConnection();

updateCount = queryRunner.update(connection, sql, params);

} catch (Exception e) {

e.printStackTrace();

throw new RuntimeException(e);

}

return updateCount;

}

/\*\*

\* 执行查询语句，只返回第一个记录

\*

\* @param sql

\* 要执行的sql语句

\* @param params

\* 执行的sql语句的参数

\* @return 返回查询对象的具体实例 <br/>

\* 如果查询的数据不存在，则返回null<br/>

\* 查询失败也返回null

\* @throws SQLException

\*/

public T queryOne(String sql, Object... params) {

Connection connection = null;

try {

// 获取数据库连接

connection = JDBCUtils.getConnection();

// 执行查询语句

return queryRunner.query(connection, sql, new BeanHandler<T>(type), params);

} catch (Exception e) {

e.printStackTrace();

throw new RuntimeException(e);

}

}

/\*\*

\* 执行查询语句，返回查询后的对象实例集合

\*

\* @param sql

\* 要执行的sql语句

\* @param params

\* sql语句的参数

\* @return 返回的是查找到的对象集合<br/>

\* 查询失败返回null

\* @throws Exception

\*/

public List<T> queryList(String sql, Object... params) {

Connection connection = null;

try {

// 获取数据库连接

connection = JDBCUtils.getConnection();

// 执行查询语句

return queryRunner.query(connection, sql, new BeanListHandler<T>(type), params);

} catch (Exception e) {

e.printStackTrace();

throw new RuntimeException(e);

}

}

/\*\*

\* 执行批量操作

\*

\* @param sql

\* 要执行的sql语句

\* @param params

\* 参数

\* @return 返回每个语句修改的数量

\* @throws Exception

\*/

public int[] batch(String sql, Object[][] params) {

Connection connection = null;

try {

// 获取数据库连接

connection = JDBCUtils.getConnection();

// 执行查询语句

return queryRunner.batch(connection, sql, params);

} catch (Exception e) {

e.printStackTrace();

throw new RuntimeException(e);

}

}

/\*\*

\* 查询只有一个返回值的语句

\*

\* @param sql

\* @param params

\* @return

\* @throws Exception

\*/

public Object querySingleValue(String sql, Object... params) {

Connection connection = null;

try {

// 获取数据库连接

connection = JDBCUtils.getConnection();

// 执行查询语句

return queryRunner.query(connection, sql, new ScalarHandler(), params);

} catch (Exception e) {

e.printStackTrace();

throw new RuntimeException(e);

}

}

}

修改BaseServlet的方法调用，代码如下 ：

在下面的代码中，一定要把所有的异常，都使用RuntimeException包装起来往外抛。这样Filter里才能收到异常，否则异常只会到BaseServlet就会终止，而不会继续往外抛

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// 解决post乱码问题，设置请求的参数字符集为UTf-8

request.setCharacterEncoding("UTF-8");

// action参数就可以用来判断是当前用户模块的哪一个操作。

String action = request.getParameter("action");

// 定义一个Method方法对象

Method actionMethod;

try {

// 通过反射获取到与action字符串对应的操作方法

actionMethod = getClass().getDeclaredMethod(action, HttpServletRequest.class,

HttpServletResponse.class);

// System.out.println(actionMethod);

// 调用Method对象的invoke函数执行方法

actionMethod.invoke(this, request, response);

} catch (NoSuchMethodException | SecurityException | IllegalAccessException

| IllegalArgumentException | InvocationTargetException e) {

e.printStackTrace();

throw new RuntimeException(e);

}

}

添加TransactionFilter类管理事务

package com.atguigu.filter;

import java.io.IOException;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import com.atguigu.util.JDBCUtils;

/\*\*

\* TrasactionFilter类用于管理事务

\*/

public class TransactionFilter implements Filter {

public TransactionFilter() {

}

public void destroy() {

}

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)

throws IOException, ServletException {

try {

// 放行操作

chain.doFilter(request, response);

// 关闭连接，并提交事务

JDBCUtils.closeConnection();

} catch (Exception e) {

// 回滚 事务

JDBCUtils.rollback();

throw e;

}

}

public void init(FilterConfig fConfig) throws ServletException {

}

}

在web.xml文件中的配置

<filter>

<filter-name>TransactionFilter</filter-name>

<filter-class>com.atguigu.filter.TransactionFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>TransactionFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 配置500错误页面 -->

<error-page>

<error-code>500</error-code>

<location>/error500.jsp</location>

</error-page>

error500.jsp页面中的错误提示

一般每个项目都有自己错误提示页面。等在实际项目中，只需要把这个内容改成你公司提示的页面即可。

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="pragma" content="no-cache" />

<meta http-equiv="cache-control" content="no-cache" />

<meta http-equiv="Expires" content="0" />

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Insert title here</title>

</head>

<body>

系统出现故障，我们马上为您解决。请耐心等待！<a href="${pageContext.request.contextPath }">返回首页</a>

</body>

</html>

# CDN技术

cdn指的是内容分发网络，一般用于网络静态资源的加速，即cdn服务就就近获取服务器上的资源传输给用户。

# JavaScript编码规范

**js编码规范参考**

无分号风格：<https://standardjs.com/readme-zhcn.html>

有分号风格（airbnb）：<https://www.kancloud.cn/kancloud/javascript-style-guide/43119>

**无分号的代码风格注意事项**

 当一行代码是以：

(

 [

 `

开头的时候，要在行首补上一个分号用避免一些语法解析错误。

结论：

无论你的代码是否有分号，都建议如果一行代码是以 (、[、` 开头的，则最好都在其前面补上一个分号

# Node.js图书管理系统开发流程

## 一、项目初始化

### 1、创建mybook文件夹

### 2、创建入口文件index.js

### 3、初始化项目生成package.json

npm init -y

### 4、安装依赖

npm install express art-template express-art-template body-parser --save

### 5、编辑入口文件 index.js

#### 1）定义依赖

/\*\*

图书管理系统入口文件

\*/

const express = require('express');

const router = require('./router.js');

#### 2）创建app对象

const app = express();

#### 3）启动服务器，配置路由

// 启动服务器功能

// 配置路由

app.use(router);

#### 4）监听端口

// 监听端口

app.listen(3000,()=>{

   console.log('running...');

});

### 6、创建路由文件router.js

/\*

  路由模块

\*/

const express = require('express');

const router = express.Router();

​

// 路由处理

// 渲染主页

router.get('/',(req,res)=>{

   console.log('主页');

   res.send('主页');

});

​

// 添加图书(跳转到添加图书的页面)

router.get('/addBook',(req,res)=>{

   console.log('添加图书');

   res.send('添加图书');

});

​

// 编辑图书(跳转到编辑图书信息页面)

router.get('/editBook',(req,res)=>{

   console.log('编辑图书');

   res.send('编辑图书');

});

​

module.exports = router;

### 7、测试

#### 1）启动服务

node .

#### 2）访问定义的路由

<http://localhost:3000>

<http://localhost:3000/addBook>

<http://localhost:3000/editBook>

## 二、添加模板引擎

### 1、引入模板引擎

#### 1）定义依赖

在index.js中添加依赖

const template = require('art-template');

const path = require('path');

#### 2）设置模板引擎

在index.js中添加

// 设置模板的路径

app.set('views',path.join(\_\_dirname,'views'));

// 设置模板引擎

app.set('view engine','art');

// 使express兼容art-template模板引擎

app.engine('art', require('express-art-template'));

### 2、创建模板文件

#### 1）创建views文件夹

#### 2）创建文件views/index.art

<!DOCTYPE html>

<html lang="en">

<head>

   <meta charset="UTF-8">

   <title>图书管理系统</title>

</head>

<body>

   <div>图书管理系统<a href="/addBook">添加图书</a></div>

   <div>

       <table cellpadding="0" cellspacing="0">

           <thead>

               <tr>

                   <th>编号</th>

                   <th>名称</th>

                   <th>作者</th>

                   <th>分类</th>

                   <th>描述</th>

                   <th>操作</th>

               </tr>

           </thead>

           <tbody>

               <tr>

                   <td>1</td>

                   <td>西游记</td>

                   <td>吴承恩</td>

                   <td>文学</td>

                   <td>佛教与道教的斗争</td>

                   <td><a href="#">修改</a>|<a href="#">删除</a></td>

               </tr>

               <tr>

                   <td>1</td>

                   <td>西游记</td>

                   <td>吴承恩</td>

                   <td>文学</td>

                   <td>佛教与道教的斗争</td>

                   <td><a href="#">修改</a>|<a href="#">删除</a></td>

               </tr>

               <tr>

                   <td>1</td>

                   <td>西游记</td>

                   <td>吴承恩</td>

                   <td>文学</td>

                   <td>佛教与道教的斗争</td>

                   <td><a href="#">修改</a>|<a href="#">删除</a></td>

               </tr>

           </tbody>

       </table>

   </div>

</body>

</html>

#### 3）创建文件views/addBook.art

<!DOCTYPE html>

<html lang="en">

<head>

   <meta charset="UTF-8">

   <title>图书管理系统-添加图书</title>

</head>

<body>

   <div>添加图书</div>

   <form action="/saveBook" method="post">

      名称：<input type="text" name="name"><br>

      作者：<input type="text" name="author"><br>

      分类：<input type="text" name="category"><br>

      描述：<input type="text" name="desc"><br>

       <input type="submit" value="提交">

   </form>

</body>

</html>

#### 4）创建文件views/editBook.art

<!DOCTYPE html>

<html lang="en">

<head>

   <meta charset="UTF-8">

   <title>图书管理系统-修改图书</title>

</head>

<body>

   <div>修改图书</div>

   <form action="/updateBook" method="post">

       <input type="hidden" name="id" >

      名称：<input type="text" name="name" ><br>

      作者：<input type="text" name="author" ><br>

      分类：<input type="text" name="category" ><br>

      描述：<input type="text" name="desc" ><br>

       <input type="submit" value="提交">

   </form>

</body>

</html>

### 3、调用模板

修改router.js

// 路由处理

// 渲染主页

router.get('/',(req,res)=>{

   ...

   res.render('index');

});

​

// 添加图书(跳转到添加图书的页面)

router.get('/addBook',(req,res)=>{

   ...

   res.render('addBook');

});

​

// 编辑图书(跳转到编辑图书信息页面)

router.get('/editBook',(req,res)=>{

   ...

   res.render('editBook');

});

## 三、处理数据

### 1、数据列表

#### 1）创建json数据文件data.json

[

  {

       "id": "1",

       "name": "三国演义",

       "author": "罗贯中",

       "category": "文学",

       "desc": "一个杀伐纷争的年代"

  },

  {

       "id": "2",

       "name": "水浒传",

       "author": "施耐庵",

       "category": "文学",

       "desc": "108条好汉的故事"

  },

  {

       "id": "3",

       "name": "西游记",

       "author": "吴承恩",

       "category": "文学",

       "desc": "佛教与道教的斗争"

  },

  {

       "id": "4",

       "name": "红楼梦",

       "author": "曹雪芹",

       "category": "文学",

       "desc": "一个封建王朝的缩影"

  },

  {

       "name": "天龙八部",

       "author": "金庸",

       "category": "文学",

       "desc": "武侠小说",

       "id": 5

  }

]

#### 2）router中引用json

const data = require('./data.json');

​

// 路由处理

// 渲染主页

router.get('/',(req,res)=>{

   ...

   res.render('index', {list : data});

});

#### 3）模板中显示数据

<tbody>

  {{each list}}

   <tr>

       <td>{{$value.id}}</td>

       <td>{{$value.name}}</td>

       <td>{{$value.author}}</td>

       <td>{{$value.category}}</td>

       <td>{{$value.desc}}</td>

       <td>

           <a href="#">修改</a>|

           <a href="#">删除</a>

       </td>

   </tr>

  {{/each}}

</tbody>

#### 4）重启服务测试

<http://localhost:3000/>

### 2、删除数据

#### 1）修改index.art

<a href="/deleteBook?id={{$value.id}}">删除</a>

#### 2）router中添加删除路由

// 删除图书

router.get('/deleteBook',(req,res)=>{

   console.log('删除图书');

   let id = req.query.id;

   data.forEach((item,index)=>{

       if(id == item.id){

           // 删除数组的一项数据

           data.splice(index,1);

      }

  });

   res.redirect('/');//页面跳转

});

#### 3）测试

### 3、保存数据

#### 1）index.js中定义依赖

const bodyParser = require('body-parser');

#### 2）引入参数处理中间件

在index.js中启动服务的代码之前添加

//挂载参数处理中间件(处理post表单)

// create application/x-www-form-urlencoded parser

app.use(bodyParser.urlencoded({extended: false}));

#### 3）router中添加保存路由和辅助方法

// 自动生成图书编号（自增）

let maxBookCode = ()=>{

   let arr = [];

   data.forEach((item)=>{

       arr.push(item.id);

  });

   return Math.max.apply(null,arr);

};

// 保存图书(提交表单)

router.post('/saveBook',(req,res)=>{

   console.log('添加图书(提交表单)');

   let info = req.body;

   console.log(info);

   let book = {};

   for(let key in info){

       book[key] = info[key];

  }

   book.id = maxBookCode() + 1;

   data.push(book);

​

   res.redirect('/');//页面跳转

});

#### 4）测试

### 4、根据id查询数据

#### 1）修改index.art

<a href="/editBook?id={{$value.id}}">修改</a>

#### 2）router中修改编辑图书的路由

// 编辑图书(跳转到编辑图书信息页面)

router.get('/editBook',(req,res)=>{

   console.log('编辑图书');

   // res.send('编辑图书');

   let id = req.query.id;

   let book = {};

   data.forEach((item)=>{

       if(id == item.id){

           book = item;

      }

  });

   res.render('editBook',book);

});

#### 3）修改editBook.art页面

<form action="/updateBook" method="post">

   <input type="hidden" name="id" value="{{id}}">

    名称：<input type="text" name="name" value="{{name}}"><br>

    作者：<input type="text" name="author" value="{{author}}"><br>

    分类：<input type="text" name="category" value="{{category}}"><br>

    描述：<input type="text" name="desc" value="{{desc}}"><br>

    <input type="submit" value="提交">

</form>

#### 4）测试

### 5、更新数据

#### 1）router中添加更新路由

// 编辑图书(提交表单)

router.post('/updateBook',(req,res)=>{

   console.log('编辑图书提交表单');

​

   let info = req.body;

   data.forEach((item)=>{

       if(info.id == item.id){

           for(let key in info){

               item[key] = info[key];

          }

      }

  });

   res.redirect('/');//页面跳转

});

#### 2）测试

## 四、创建服务层

### 1、数据列表

#### 1）创建service.js文件

将对json数据的引入和辅助方法的定义从router.js中移植到service.js中

/\*

  业务模块

\*/

const data = require('./data.json');

​

​

// 自动生成图书编号（自增）

let maxBookCode = ()=>{

   let arr = [];

   data.forEach((item)=>{

       arr.push(item.id);

  });

   return Math.max.apply(null,arr);

};

#### 2）在service中定义页面显示业务

// 渲染主页面

exports.showIndex = (req,res)=>{

   console.log('主页');

   res.render('index', {list : data});

};

​

// 添加图书

exports.addBook = (req,res)=>{

   console.log('添加图书');

   res.render('addBook');

};

#### 3）router中引入service模块

const service = require('./service.js');

#### 4）调用service模块中的方法

为了避免错误，先屏蔽其他的相关路由（除"/addBook"之外）

// 路由处理

// 渲染主页

router.get('/', service.showIndex);

​

// 添加图书(跳转到添加图书的页面)

router.get('/addBook', service.addBook);

### 2、编辑图书

#### 1）在service中定义根据id查询指定图书的方法

// 编辑图书

exports.editBook = (req,res)=>{

   console.log('编辑图书');

   // res.send('编辑图书');

   let id = req.query.id;

   let book = {};

   data.forEach((item)=>{

       if(id == item.id){

           book = item;

      }

  });

   res.render('editBook',book);

};

#### 2）修改router中的编辑图书的路由

// 编辑图书(跳转到编辑图书信息页面)

router.get('/editBook', service.editBook);

### 3、删除图书

#### 1）在service中定义根据id删除指定图书的方法

// 删除图书

exports.deleteBook = (req,res)=>{

   console.log('删除图书');

   let id = req.query.id;

   data.forEach((item,index)=>{

       if(id == item.id){

           // 删除数组的一项数据

           data.splice(index,1);

      }

  });

   res.redirect('/');//页面跳转

};

#### 2）修改router中的删除图书的路由

// 删除图书

router.get('/deleteBook', service.deleteBook);

### 4、保存图书

#### 1）在service中定义保存图书的方法

// 保存图书

exports.saveBook = (req,res)=>{

   console.log('添加图书(提交表单)');

   let info = req.body;

   console.log(info);

   let book = {};

   for(let key in info){

       book[key] = info[key];

  }

   book.id = maxBookCode() + 1;

   data.push(book);

​

   res.redirect('/');//页面跳转

};

#### 2）修改router中的保存图书的路由

// 保存图书(提交表单)

router.post('/saveBook', service.saveBook);

### 5、更新图书

#### 1）在service中定义根据id更新指定图书的方法

// 编辑图书

exports.updateBook = (req,res)=>{

   console.log('编辑图书提交表单');

​

   let info = req.body;

   data.forEach((item)=>{

       if(info.id == item.id){

           for(let key in info){

               item[key] = info[key];

          }

      }

  });

   res.redirect('/');//页面跳转

};

#### 2）修改router中的更新图书的路由

// 编辑图书(提交表单)

router.post('/updateBook', service.updateBook);

## 五、将数据持久化到json文件中

### 1、修改service

#### 1）添加文件相关依赖

在service.js中添加

const path = require('path');

const fs = require('fs');

#### 2）将数据写入json

在service.js中添加writeDataToFile方法，

// 把内存数据写入文件

let writeDataToFile = (res) => {

   // (data,null,4)：保证数据存入的时候有换行缩进的格式

   fs.writeFile(path.join(\_\_dirname,'data.json'),JSON.stringify(data,null,4),(err)=>{

       if(err){

           return false; // 文件写入失败

      }

       return true; // 文件写入成功

  });

};

#### 3）调用写入文件的方法

在删除、保存、更新的业务方法中分别调用

writeDataToFile();

## 六、将数据持久化到MySQL数据中

### 1、初始化MySQL数据库和表

#### 1）创建MySQL数据库 nodejs\_book2）创建数据库表 book

CREATE TABLE `book` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(255),

`author` varchar(255),

`category` varchar(255),

`description` varchar(255),

PRIMARY KEY (`id`)

);

#### 2）执行MySQL脚本插入数据

insert into book (name,author,category,description) values ('三国演义','罗贯中','文学','一个杀伐纷争的年代');

insert into book (name,author,category,description) values ('水浒传','施耐庵','文学','108条好汉的故事');

insert into book (name,author,category,description) values ('西游记','吴承恩','文学','佛教与道教的斗争');

insert into book (name,author,category,description) values ('红楼梦','曹雪芹','文学','一个封建王朝的缩影');

insert into book (name,author,category,description) values ('天龙八部','金庸','文学','武侠小说');

### 2、引入mysql模块

#### 1）安装模块

npm install mysql --save

#### 2）创建db.config.js文件

//配置链接数据库参数

module.exports = {

   host : 'localhost',

   port : 3306,

   database : 'nodejs\_book',

   user : 'root',

   password : '123456'

};

#### 3）定义通用的db.js

/\*

  定义数据库query函数

\*/

var mysql = require('mysql');

var databaseConfig = require('./database.config');  //引入数据库配置模块中的数据

​

//向外暴露方法

module.exports = {

   base : (sql,params,callback) => {

        // 创建数据库连接

       const connection = mysql.createConnection(databaseConfig);

​

       // 执行连接操作

       connection.connect();

​

       // 操作数据库(数据库操作也是异步的)

       connection.query(sql,params, function(error, results, fields) {

           if (error) throw error;

           callback(results);

      });

​

       // 关闭数据库

       connection.end();

  }

};

### 3、修改service

#### 1）在service中引用db模块

const db = require('./db.js');

#### 2）主页列表

// 渲染主页面

exports.showIndex = (req,res)=>{

   console.log('主页');

   let sql = 'select \* from book';

   db.base(sql,null,(result)=>{

       res.render('index',{list : result});

  });

};

#### 3）编辑图书（根据id选择）

// 编辑图书

exports.editBook = (req,res)=>{

   console.log('编辑图书');

   // res.send('编辑图书');

   let id = req.query.id;

   let sql = 'select \* from book where id=?';

   let data = [id];

   db.base(sql,data,(result)=>{

       res.render('editBook',result[0]);

  });

};

#### 4）删除图书

// 删除图书

exports.deleteBook = (req,res)=>{

   console.log('删除图书');

   let id = req.query.id;

   let sql = 'delete from book where id=?';

   let data = [id];

   db.base(sql,data,(result)=>{

       if(result.affectedRows == 1){

           res.redirect('/');

      }

  });

};

#### 5）保存图书

exports.saveBook = (req,res)=>{

   console.log('添加图书(提交表单)');

   let info = req.body;

   let book = {};

   for(let key in info){

       book[key] = info[key];

  }

   let sql = 'insert into book set ?';

   db.base(sql,book,(result)=>{

       if(result.affectedRows == 1){

           res.redirect('/');

      }

  });

};

#### 6）更新图书

exports.updateBook = (req,res)=>{

   console.log('编辑图书提交表单');

​

   let info = req.body;

   let sql = 'update book set name=?,author=?,category=?,description=? where id=?';

   let data = [info.name,info.author,info.category,info.description,info.id];

   db.base(sql,data,(result)=>{

       if(result.affectedRows == 1){

           res.redirect('/');

      }

  });

};

## 静态资源服务

### 1、添加静态资源目录

### 2、启用静态资源服务

在index.js中添加

// 启动静态资源服务

app.use('/www',express.static('public'));

补充：

express中app.use和app.get的区别及解析

<https://blog.csdn.net/wthfeng/article/details/53366169>