### 1 传音集团

#### **springboot spa ,start。**

配置bean

##### **1.2分布式调用时间设置**

# Feign<一般情况下 都是 ribbon 的超时时间（<）hystrix的超时时间

##### **1.3 docker部署服务**

##### **1.4 SpringCloudAlibaba组件与springcloud**



##### **1.5 nocas与Eureka是拉取还是推送**

<https://www.jianshu.com/p/38b5452c9fec>

客户端拉取服务端的数据

1.6 AOP与拦截器谁先执行

### 2 平安人寿

#### 2.1 算法题:二叉树路径求和

#### 判断链表是否有环

##### 2.1.1.二叉树路径求和

二叉树根节点到叶子节点的节点序列称为路径，如果路径上所有节点和为指定的某个数，就打印该路径

10

/ \

5 12

/\

4 7

有两条路径上的结点和为22,分别是10+5+7和10+12

思路比较简单：先序遍历二叉树，并同步更新直到当前节点为止的sum和path,如果是叶子节点，与指定数比较，若相等，输出序列

2.代码

#include<stdio.h>

#include<vector>

struct BinaryTreeNode

{

int value;

BinaryTreeNode\* left;

BinaryTreeNode\* right;

};

void doFindPath(BinaryTreeNode\* root, int expect, std::vector& path, int& current)

{//注意path和current是引用

current += root->value;//进入本节点，更新本节点对current和path的影响

path.push\_back(root->value);//插入队尾

bool isLeaf = root->left==NULL && root->right==NULL;

if(isLeaf && current == expect)//打印结果

{

std::vector::iterator iter = path.begin();

while(iter != path.end())

{

printf("%d ", \*iter);

iter ++;

}

printf("\n");

}

if(root->left)

doFindPath(root->left, expect, path, current);

if(root->right)

doFindPath(root->right, expect, path, current);

current -= root->value;//返回父节点时，删除本节点造成的负面影响

path.pop\_back();

}

void findPath(BinaryTreeNode\* root, int expect)

{

if(root == NULL)

return;

std::vector path;

int current = 0;

doFindPath(root, expect, path, current);

}

class Solution {

public boolean hasPathSum(TreeNode root, int sum) {

TreeNode left = root.left;

TreeNode right = root.right;

// 左右节点为空表示为叶子节点，做最后的判断

if (left == null && right == null) return sum == root.val;

boolean b1 = false, b2 = false;

// 从左节点看是否有剩下路径和相等

if (left != null) b1 = hasPathSum(left, sum - root.val);

// 从有节点看是否有剩下路径和相等

if (right != null) b2 = hasPathSum(right, sum - root.val);

// 只要有一条路径满足就成功

return b1 || b2;

}

}

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

class Solution {

public boolean hasPathSum(TreeNode root, int sum) {

if (root == null) {

return false;

}

if (root.left == null && root.right == null) {

return sum == root.val;

}

return hasPathSum(root.left,sum-root.val) || hasPathSum(root.right,sum-root.val);

}

}

##### **2.1.2判断链表是否有环**

**ADT:**

**class ListNode {**

**int val;**

**ListNode next;**

**ListNode(int x) {**

**val = x;**

**next = null;**

**}**

**}**

**方法一：循环遍历节点，遍历一个便标记一个，遍历过程判断是否被标记，若已被标记则表示有环**

**方法说明：头指针移动，若到达之前到达过的位置则表示有环，若无环则会走到链表末端。**

**public class Solution {**

**public boolean hasCycle(ListNode head) {**

**//声明一个set存放已遍历的节点，即为标记节点（Set中不允许重复元素）**

**Set<ListNode> set = new HashSet<>();**

**while(head!=null) {**

**if(set.contains(head)) {**

**return true;**

**}else {**

**set.add(head);**

**head = head.next;**

**}**

**}**

**return false;**

**}**

**}**

**方法二：声明两个指针，一个指针走一次经过两个节点(快指针quick)，另一个走一次经过一个节点(慢指针slow)**

**方法说明：快指针走的比较快，若链表有环，则一定会追上慢指针，若无环，则会走到链表末端。**

**public class Solution {**

**public boolean hasCycle(ListNode head) {**

**==//声明两个节点从头开始遍历节点==**

**ListNode quick = head;**

**ListNode slow = head;**

**//当快指针能够走到头表示无环**

**while(quick!=null&&quick.next!=null){**

**quick = quick.next.next;**

**slow = slow.next;**

**if(quick==slow){**

**return true;**

**}**

**}**

**return false;**

**}**

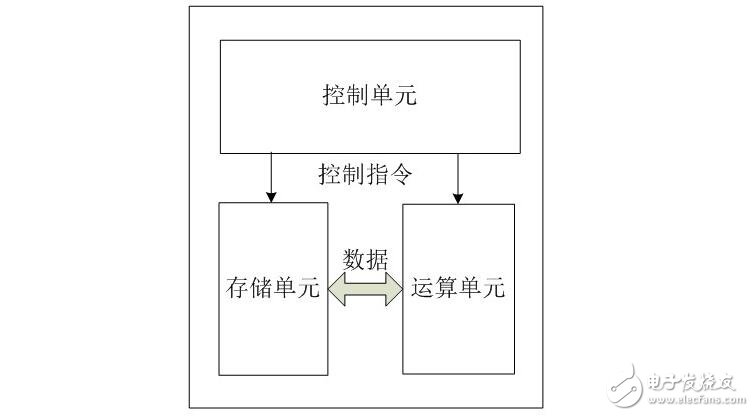
**}**

#### **2.2计算机专业知识:CPU的原理，时间片，网络知识，数据库，索引，spring源码**

##### 2.2.1 cpu原理

**<https://blog.csdn.net/yedushu/article/details/85068432>**

　我们都知道CPU的根本任务就是执行指令，对计算机来说最终都是一串由“0”和“1”组成的序列。CPU从逻辑上可以划分成3个模块，分别是控制单元、运算单元和存储单元，这三部分由CPU内部总线连接起来。如下所示：



### IMG_256

**总结一下，CPU的运行原理就是：**

　　1、取指令：CPU的控制器从内存读取一条指令并放入指令寄存器。指令的格式一般是这个样子滴：

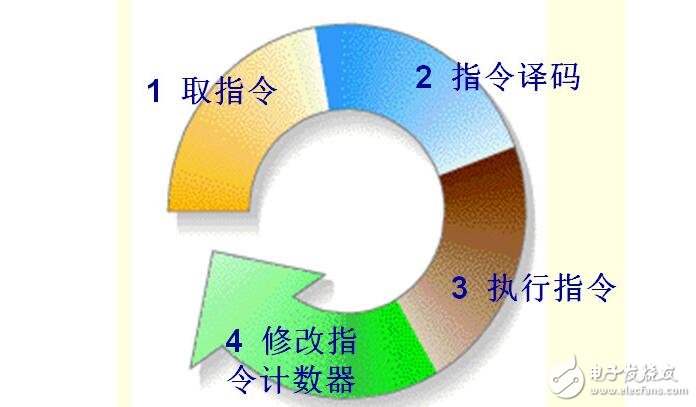


　　操作码就是汇编语言里的mov，add，jmp等符号码；操作数地址说明该指令需要的操作数所在的地方，是在内存里还是在CPU的内部寄存器里。

　　2、指令译码：指令寄存器中的指令经过译码，决定该指令应进行何种操作（就是指令里的操作码）、操作数在哪里（操作数的地址）。

　　3、 执行指令，分两个阶段“取操作数”和“进行运算”。

　　4、 修改指令计数器，决定下一条指令的地址。



##### **2.2.2 时间片**

https://www.jianshu.com/p/44585830f009

时间片即CPU分配给各个程序的时间，每个线程被分配一个时间段，称作它的时间片，即该进程允许运行的时间，使各个程序从表面上看是同时进行的。

##### **2.2.3 网络知识面试题**

<https://www.jianshu.com/p/1cf5e20e11a9>

##### **2.2.4 数据库锁**

##### **2.2.5 数据库索引**

##### **2.2.6 spring源码**

https://www.cnblogs.com/yanggb/p/11004887.html

#### **2.3 项目:并发处理，知识线程，线程池的类型、特点、选择**

##### **2.3.1 并发处理**

https://www.cnblogs.com/javazhiyin/p/10118078.html

**java中有几种方法可以实现一个线程？**

继承Thread类；  
实现Runnable接口；  
实现Callable接口通过FutureTask包装器来创建Thread线程；  
使用ExecutorService、Callable、Future实现有返回结果的多线程（也就是使用了ExecutorService来管理前面的三种方式）。

**sleep()和 wait()有什么区别?**

对于sleep()方法，我们首先要知道该方法是属于Thread类中的。而wait()方法，则是属于Object类中的。

sleep()方法导致了程序暂停执行指定的时间，让出cpu该其他线程，但是他的监控状态依然保持者，当指定的时间到了又会自动恢复运行状态。在调用sleep()方法的过程中，线程不会释放对象锁。

当调用wait()方法的时候，线程会放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象调用notify()方法后本线程才进入对象锁定池准备，获取对象锁进入运行状态。

**synchronized和java.util.concurrent.locks.Lock的异同？**

主要相同点：Lock能完成Synchronized所实现的所有功能。

主要不同点：Lock有比Synchronized更精确的线程予以和更好的性能。Synchronized会自动释放锁，但是Lock一定要求程序员手工释放，并且必须在finally从句中释放。

**乐观锁和悲观锁的理解及如何实现，有哪些实现方式？**

乐观锁是假设每次操作都不会冲突，若是遇到冲突失败就重试直到成功；悲观锁是让其他线程都等待，等锁释放完了再竞争锁。

乐观锁实现方式：cas，volatile

悲观锁实现方式：synchronized，Lock

**使用特殊域变量(volatile)实现线程同步**

a.volatile关键字为域变量的访问提供了一种免锁机制，

b.使用volatile修饰域相当于告诉虚拟机该域可能会被其他线程更新，

c.因此每次使用该域就要重新计算，而不是使用寄存器中的值

d.volatile不会提供任何原子操作，它也不能用来修饰final类型的变量

##### **2.3.2 线程池的类型**

public ThreadPoolExecutor(int corePoolSize,

int maximumPoolSize,

long keepAliveTime,

TimeUnit unit,

BlockingQueue<Runnable> workQueue,

ThreadFactory threadFactory,

RejectedExecutionHandler handler)

　① shutdown()：拒收新的任务，立马关闭正在执行的任务，可能会引起报错，需要异常捕获

　　② shutdownNow()：拒收新的任务，等待任务执行完毕，要确保任务里不会有永久等待阻塞的逻辑，否则会导致线程关闭不了

第四部分：线程池的应用场景

1、newSingleThreadExecutor：一个单线程的线程池，可以用于需要保证顺序执行的场景，并且只有一个线程在执行。

2、newFixedThreadPool：一个固定大小的线程池，可以用于已知并发压力的情况下，对线程数做限制。

3、newCachedThreadPool：一个可以无限扩大的线程池，比较适合处理执行时间比较小的任务。

4、newScheduledThreadPool：**可以延时启动，定时启动的线程池**，适用于需要多个后台线程执行周期任务的场景。

5、newWorkStealingPool：一个拥有多个任务队列的线程池，可以减少连接数，创建当前可用cpu数量的线程来并行执行。

 submit()方法有返回值Future，而execute()方法没有返回值

**下面介绍常用的四种：**

**(1)FixedThreadPool**

　　FixedThreadPool的特点：固定池子中线程的个数。使用静态方法newFixedThreadPool()创建线程池的时候指定线程池个数。

**(2)CachedThreadPool（弹性缓存线程池）**

　　CachedThreadPool的特点：用newCachedThreadPool()方法创建该线程池对象， 创建之初里面一个线程都没有，当execute方法或submit方法向线程池提交任务时， 会自动新建　　线程；如果线程池中有空余线程，则不会新建；这种线程池一般最多情况可 以容纳几万个线程，里面的线程空余60s会被回收。

**(3)SingleThreadPool（单线程线程池）**

　　SingleThreadPool的特点：池中只有一个线程，如果扔5个任务进来，那么有4个任务将排队；作用是保证任务的顺序执行。

**(4)ScheduledThreadpool（定时器线程池）**

**算法**

查找（带权）二叉树中的最大值和最小值。

链表A升序排列，链表B降序排列，归并链表A和B，使新链表正序排列。