牛人计划-高级项目课（5）

牛客网
NOWCODER

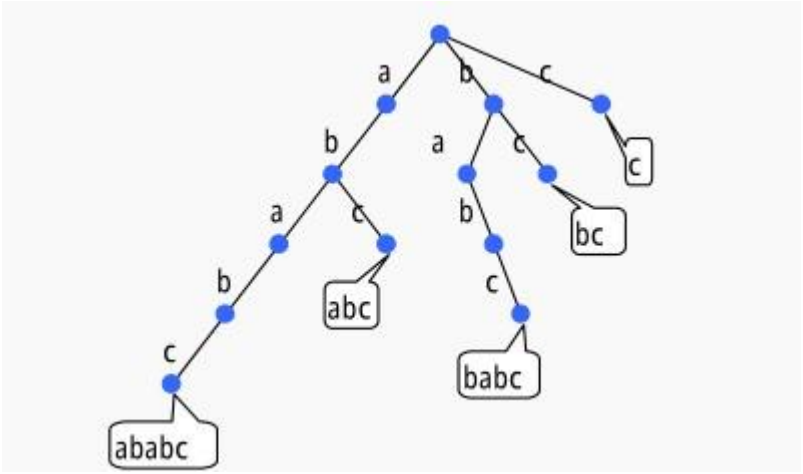
- 问题发布
- HTML/敏感词过滤
- 多线程

问题发布（代码演示）

敏感词/HTML过滤（代码演示）

前缀树

- 根节点不包含字符，除根节点外每一个节点都只包含一个字符
- 从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串
- 每个节点的所有子节点包含的字符都不相同



多线程简介

优势

- 充分利用多处理器
- 可以异步处理任务

挑战

- 数据会被多个线程访问，有安全性问题
- 不活跃的线程也会占用内存资源
- 死锁

Thread

1. extends Thread，重载run()方法
2. implements Runnable()，实现run()方法

```
new Thread(new Runnable() {
    @Override
    public void run() {
        Random random = new Random();
        for (int i = 0; i < 10; ++i) {
            sleep(random.nextInt(1000));
            System.out.println(String.format("T%d : %d", tid, i));


        }
    }
}, String.valueOf(i)).start();
```

Synchronized - 内置锁

1. 放在方法上会锁住所有synchronized方法
2. synchronized(obj) 锁住相关的代码段

```
public static void testSynchronized1() {
    synchronized (obj) {
        Random random = new Random();
        for (int i = 0; i < 10; ++i) {
            sleep(random.nextInt(1000));
        }
    }
}
```

BlockingQueue 同步队列

Summary of BlockingQueue methods

	Throws exception	*Special value*	*Blocks*	*Times out*
Insert	add(e)	offer(e)	put(e)	offer(e, time, unit)
Remove	remove()	poll()	take()	poll(time, unit)
Examine	element()	peek()	*not applicable*	*not applicable*

ThreadLocal

1. 线程局部变量。即使是一个static成员，每个线程访问的变量是不同的。
2. 常见于web中存储当前用户到一个静态工具类中，在线程的任何地方都可以访问到当前线程的用户。
3. 参考HostHolder.java里的users

Executor

Executor

1. 提供一个运行任务的框架。
2. 将任务和如何运行任务解耦。
3. 常用于提供线程池或定时任务服务

```
ExecutorService service = Executors.newFixedThreadPool(2);
service.submit(new Runnable() {
    @Override
    public void run() {
        for (int i = 0; i < 10; ++i) {
            sleep(1000);
            System.out.println("Execute %d" + i);
        }
    }
});
```

Future

```
public static void testFuture() {
    ExecutorService service = Executors.newSingleThreadExecutor();
    Future<Integer> future = service.submit(new Callable<Integer>() {
        @Override
        public Integer call() throws Exception {
            sleep(1000);
            //throw new IllegalArgumentException("一个异常");
            return 1;
        }
    });
    service.shutdown();

    try {
        System.out.println(future.get());
        //System.out.println(future.get(100, TimeUnit.MILLISECONDS));
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

1. 返回异步结果
2. 阻塞等待返回结果
3. timeout
4. 获取线程中的Exception

课后作业

a) 问题字段检测加强
b) 多线程练习

Thanks