# 0、自我介绍

您好，我是内蒙古工业大学软件工程专业的大四应届生席鹏，今年21岁，来自美丽的草原内蒙古赤峰。我在校期间做过四个项目，其中一个属于大学生创新创业项目，拿到了6k的项目基金，已申请论文一篇（审核中），在准备申请软件著作权，在校期间我担任过我校党委学生工作部的第一学生助理，属于一职多能的薪酬岗位，学分评定跟学校的校学生会副主席同级，同时，在美团进行了为期一个月的测试开发工程师的暑期实习工作，独立测试了一个业务模块的接口自动化测试和预上线测试，美团实习组内排名为3/17，因实习时间不足未能转正。

在图像研究领域，图像去雾化大致分为两种方向：图像增强方法和图像复原方法。前者以提高图像对比度为基线，旨在增加图像的细节。后者通过分析图像障质的原理，根据雾图退化模型进行求解。前者由于图像的质量降低与场景点到图像传感器的距离成指数关系，因此这种假设场景景深不变的图像增强技术不能很好地对雾化图像去雾恢复。后者基于大气散射规律建立了图像的退化模型，其先验知识具有内在的优越性，而这种方法一般需要求得场景深度或大气信息，而现实中获取的降质图像并无任何相关信息，因此，此方法成了一个不确定的反问题。为消除这种不确定性，诸多学者用不同天气下统一场景的图像或由用户提供的附加信息进行图像清晰化，因此，这些算法对图像采集有更高的要求。另外，还有学者提出通过扩大复原图像局部对比度来达到去雾目的，然而这样只是在光学原理上达到去雾效果而忽视了透射率。

**学生请销假管理系统** 2019年12月 - 2020年01月

组长

- 使用html，css，js技术完成前端UI设计。
- 使用servlet，jsp技术进行前后端交互。
- 使用mysql数据库对人员信息进行管理。
- 使用DbUtils进行后端与数据库交互。

**基于图像去雾算法的路况鉴别系统（大创项目）** 2019年04月

负责人

- 使用opencv对行车路况进行实时采集。
- 使用opencv对采集的路况进行去雾处理。
- 将处理后的结果展示在显示设备上。
- 使用神经网络对抗将处理结果优化(开发中)。

**E家政APP实现** 2020年05月 - 2020年06月

开发者

使用mui进行前后台前端ui设计

使用mvvm架构进行软件设计

使用json进行数据传输

使用rbac进行权限管理

**新闻管理系统** 2020年08月 - 2020年09月

开发者

使用Ajax进行前端数据处理

使用mvc架构进行后端业务处理

使用jdbc对数据库进行操作

使用mysql存储数据

使用无限表控制子页面

# 1、计算机网络

## 1、http常用状态码：

- 1**：继续等待操作
- 2**：成功并处理
- 3**：重定向
- 4**：客户端错误
- 5**：服务器错误

详细举例：

- **100：继续等待请求。**
- 101：切换协议到高版本
- **200：请求成功**
- 201：已接收请求并创建了新的资源
- 202：已接受请求单位处理完成
- 204：服务器处理成功但无返回值。
- 205：清除表单域
- 206：处理的部分请求
- **300：请求资源包含多个地址，返回一个**
- 301：资源被永久移动
- 302：资源被临时移动
- 304：所有资源未修改，服务器不会返回资源，客户端加载缓存资源
- 305：代理模式
- 307：临时重定向（使用GET方法请求）
- **400：客户端请求语法错误**
- 401：请求要求用户的认证
- 403：服务器理解请求，但拒绝执行
- 404：服务器找不到请求的资源。
- 405：客户端请求的方法被禁止（常见post和get）
- **500：服务器内部错误，无法完成请求**

## 2、http和https（4个）

- https是http+ssl的安全协议，再信息传输上更为保密和安全，http会发生泄露
- http用80端口，https用443
- http无法对应用层的结果进行加密，https再对传输中的数据进行加密，再由应用层解密实现
- https连接前需要下载证书来验证服务器安全性。

## 3、TCP和UDP（6个）

- tcp面向连接，udp面向无连接
- tcp占用系统资源较多，udp较少
- udp程序结构简单
- tcp为流模式，udp为数据报模式
- tcp能保证数据顺序性和不丢包，udp不保证
- tcp无长度限制，udp为8个字节，放一个ip数据报。

## 4、get和post（5个）

- 安全性：get不安全，数据会放在请求的url中，post是安全的。（url会被保存在浏览器中）
- 数据量：get数据量较小，受url长度限制（谷歌8082，ie2083）。
- 字符集支持：get限制为ascll，post为ISOS10646
- 执行效率：get强于post
- 回退：get可直接回退，post回退需要再次提交请求。

## 5、三握四挥

- 序号：seq：tcp源向目的端发送的字节流，发起方发送数据时对此进行标记。
- 确认号：ACK：ack=seq+1
- 标志位：SYN：发起一个连接，FIN，释放一个连接。

### 握手：

- 客户端标识SYN=1，发起一个连接，seq=x。
- 服务器返回SYN=1，ACK=1（确认连接），ack=x+1，seq=y；
- 客户端发送ACK=1，seq=x+1，ack=y+1

| | SYN | SEQ | ACK | ACK(小写) |
|---|---|---|---|---|
| 客户端 | 1 | x | | |
| 服务器 | 1 | y | 1 | x+1 |
| 客户端 | | x+1 | 1 | y+1 |

### 挥手：服务器第一次

- 客户端：FIN关闭标志为1，seq字节流为u
- 服务器：ACK确认号为1，seq字节流为v，ack确认为u+1
- 服务器：FIN关闭信号为1，确认号为1，seq字节流为w，ack确认为v+1
- 客户端：确认号ACK=1，seq字节流为u+1，ack确认为w+1

| | FIN | SEQ | ACK | ACK（小写） |
|---|---|---|---|---|
| 客户端 | 1 | u | | |
| 服务器 | | v | 1 | u+1 |
| 服务器 | 1 | w | 1 | u+1 |
| 客户端 | | u+1 | 1 | w+1 |

## 6、；OSI七层网络模型

- 物理层：传输高低电压。（以太网，无线lan）
- 数据链路层：划分数据报，定义分组方式。
- 网络层：引入ip划分广播域及子网。（ARP，IPV4，IPV6，ICMP）
- 传输层：建立连接，处理数据包错误。（TCP，UDP，SCTP，DCCP）
- 会话层：建立端连接，恢复通信，断点续传。（SSH，HTTP，TELNET，POP，SSL）
- 表示层：数据格式转换，加密解密。

- 应用层：规定应用程序的数据格式。

## 7、跨域

请求的URL的协议、域名和端口三者任意一个与当前页面url不同。一般为向服务器之外的请求相应，会因浏览器的同源策略被拦截。

解决方案：

- JSONP：一种json的使用模式，将其设置为非同源策略，将数据放入支持跨域请求资源的script中的src进行处理，这种方法只支持get请求。
- 后端cors：浏览器发送预请求，获取服务器的origin并放入浏览器中，当origin不为空时，浏览器默认请求合法。

## 8、浏览器输入url到页面显示发生了什么

1. url解析对应ip：查询hosts文件-》本地DNS服务器-》根服务器-》域服务器-》域名解析服务器-》获得ip（同时存入缓存）
2. 使用随机端口向服务器的80端口发起tcp连接请求。
3. 发起http请求，包含请求方法、请求头和请求正文。
4. 服务器解析请求，并返回。
5. 关闭tcp连接（四次挥手）
6. 浏览器解析资源
7. 布局渲染

## 9、浏览器常见http请求

- get：请求页面。服务器将资源放在响应数据的报文中进行发送。参数长度受限、不安全。
- post：以键值对的形式将数据封装在请求数据中，可传输大量数据，且不会7受限于参数长度。
- head：参数响应头，一般会将token等登录鉴权的信息放到这里面。
- PUT：添加新内容
- delete：删除某个内容

## 10、session和cookie的区别

- session在服务器中，没有大小限制，安全。
- cookie在客户端，一次性可以发送多个。浏览器关闭后被销毁。最长4kb，服务器对客户端登录状态的识别。

## 11、socket编程

- socket应用层与TCP/IP中间的软件抽象层，TCP/IP协议隐藏在socket后面，对外提供一组接口。
- **服务器步骤：**
- 创建套接字
- 将套接字绑定到端口上
- 设置为监听模式
- 等待客户到来，接收连接请求，返回对应的一个新的套接字
- 用返回的套接字与客户端建立通信，同时等待另一个连接

- 关闭连接。
- **客户端步骤：**
- 创建套接字
- 向服务器发送请求
- 和服务器进行通信
- 关闭请求

## 12、http报文组成

- http报文分请求报文和响应报文
- http由开始行、首部行和实体主体三部分组成
- **请求报文的开始行：**方法、【空格】、URL、【空格】、Http版本。
- 方法：向请求资源指定的资源发送请求报文的方法、其作用是可以指定请求的资源按期望产生某种行为。
- URL：链接
- HTTP：目前由HTTP/1.0、HTTP/1.1、HTTP/2.0版本，其中HTTP1.0使用广泛。
- **响应报文的开始行：**HTTP版本、【空格】、状态码组成。
- **首部行：**说明信息
- 字段名、【空格】、字段值
- 结束有回车换行
- 字段分为通用首部字段、请求首部字段、响应首部字段、实体首部字段。
- 请求报文：post/put提交的表单信息、与首部有空行。

## 13、数据库视图使用

- 视图是一个或多个表中导出的虚表，其内容由查询定义，具有普通表的结构，但不实现数据存储。
- 视图的修改：单表视图一般用于查询和修改，会改变表的数据。多表用于查询，不改变表数据。
- 新建视图：

```
create or replace view v_student as select * from student;
```

- 数据检索：

```
select * from v_student
```

- 删除视图

```
drop view v_student
```

## 14、数据库视图作用

- 简化操作：把常用数据定为视图
- 安全性：用户只能看到查询和修改后的数据
- 逻辑独立：屏蔽了真实表的结构带来的影响。
- 缺点：性能差；修改限制。

# 2、数据库

## 1、查询成绩。

两门及以上不及格同学的学号、姓名和平均成绩

```
SELECT
    score.s_id,
    student.s_name,
    AVG(score)
FROM
    score,
    student
WHERE
    score.s_id = ( SELECT s_id FROM score WHERE score < 60 GROUP BY
s_id HAVING COUNT( s_id ) > 1 )
    AND score.s_id = student.s_id;
```

## 2、数据库模式

模式：对数据库的特征和逻辑结构进行描述。全体用户公共视图。

内模式：数据物理结构和存储方式的描述。数据在数据库内部的表示方式。

外模式：（子模式）用户能看见和使用的局部数据的逻辑结构和特征。

## 3、四种常用数据模型

- 层次模型：二叉树
- 关系模式：而未被
- 网状模型：图
- 面向对象：继承封装多态

## 4、数据库系统特点

- 可共享
- 永久使用
- 有组织

## 5、数据库基本特点

- 数据结构化
- 冗余度低
- DBMS系统管理
- 独立性高

## 7、三范式

- 1NF：原子性。数据库中的数据不可分割。
- 2NF：部份依赖：一个表不能出现多个主键。
- 3NF：传递依赖：每个数据只能依赖主键，不能依赖其他非主键。

## 8、主键和索引的区别：

联系：

- 每个表中只能用一个主键和聚合索引
- 主键和聚合索引都可以通过多个字段定义。
- 数据库创建表指定主键是如果未指定聚合索引，则会在主键中自动生成一个索引。

区别：

- 主键强调表的完整性，索引是对行的排序，方便查询。
- 主键不能为空，索引可以为空。
- 主键数据不能唯一，索引数据可以唯一

## 9、MySQL索引数据结构

- innodb：b+树，叶子节点存储数据
- myisam：b+树，叶子节点存放数据地址

## 10、一级缓存、二级缓存

- 一级缓存：当我们执行查询时，查询的语句和返回的结果会以kv的形式存入mybatis的sqlsession中，当再次执行查询时，mybatis会在sqlsession中寻找对应数据返回。
- mapper级，当一个mapper数据被修改后，会被更新到二级缓存中，再执行查询语句的时候会返回二级缓存中。

## 11、redis缓存穿透、击穿和雪崩

- 穿透：redis空但数据库中有
- 击穿：两者都没有
- 雪崩：大批量数据同时过期

## 12、mysql分页：limit

## 13、数据库优化

### sql优化

- 避免全表查询 select *
- 小表驱动大表
- 使用连接代替子查询

### 优化索引使用

- 尽量使用主键查询
- 计算放入

## 14、jdbc连接步骤

1. 获取驱动类和连接信息
2. 获取DriverMamage.connection连接对象
3. 创建Statment执行静态sql的对象，创建PerparedStatment执行动态sql对象。
4. 执行sql语句获取结果集。
5. 关闭流。

## 15、redis的过期策略

- 定时过期：每个设置过期时间的key都需要创建一个定时器，到期会自动删除，对内存友好，但会占用cpu资源去处理过期数据，影响缓存的响应时间和吞吐量。
- 惰性过期：访问key时判断其是否过期，过期则删除，容易造成资源浪费。
- 定时过期：每隔一段时间扫描一次key，并清除过期的key。

# 3、算法

---

## 1、爬楼梯

```
public class main {
    public static int f1(int n) {
        if (n == 1)
            return 1;
        if (n == 2)
            return 2;
        return f1(n - 1) + f1(n - 2);
    }

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int i = scan.nextInt();
        System.out.println(f1(i));
    }
}
```

## 2、快速排序

> *双指针遍历*

```
package day0930;

/**
 * @Author: XiPeng
 * @Description:
 * @Date: Create in 14:22 2020/9/30
 * @Modified by:
 */
/**
 * 快速排序演示
 * @author Lvan
 */
public class QuickSort {
    public static void main(String[] args) {
        int[] arr = {5, 1, 7, 3, 1, 6, 9, 4};

        quickSort(arr, 0, arr.length - 1);

        for (int i : arr) {
            System.out.print(i + "\t");
        }
    }

    /**
     * @param arr        待排序列
     * @param leftIndex  待排序列起始位置
     * @param rightIndex 待排序列结束位置
     */
    private static void quickSort(int[] arr, int leftIndex, int
rightIndex) {
        if (leftIndex >= rightIndex) {
            return;
        }

        int left = leftIndex;
        int right = rightIndex;
        //待排序的第一个元素作为基准值
        int key = arr[left];

        //从左右两边交替扫描，直到left = right
        while (left < right) {
            while (right > left && arr[right] >= key) {
                //从右往左扫描，找到第一个比基准值小的元素
                right--;
            }

            //找到这种元素将arr[right]放入arr[left]中
            arr[left] = arr[right];
```

```
            while (left < right && arr[left] <= key) {
                //从左往右扫描，找到第一个比基准值大的元素
                left++;
            }

            //找到这种元素将arr[left]放入arr[right]中
            arr[right] = arr[left];
        }
        //基准值归位
        arr[left] = key;
        //对基准值左边的元素进行递归排序
        quickSort(arr, leftIndex, left - 1);
        //对基准值右边的元素进行递归排序。
        quickSort(arr, right + 1, rightIndex);
    }
}
```

## 3、大顶堆

```
package day0930;

/**
 * @Author: XiPeng
 * @Description: 大顶堆
 * @Date: Create in 14:10 2020/9/30
 * @Modified by:
 */
public class BigHeap {

        public static void heapSort(int[] arr) {
            if (arr == null || arr.length == 0) {
                return;
            }
            int len = arr.length;
            // 构建大顶堆，这里其实就是把待排序序列，变成一个大顶堆结构的数组
            buildMaxHeap(arr, len);

            // 交换堆顶和当前末尾的节点，重置大顶堆
            for (int i = len - 1; i > 0; i--) {
                swap(arr, 0, i);
                len--;
                heapify(arr, 0, len);
            }
        }

        private static void buildMaxHeap(int[] arr, int len) {
            // 从最后一个非叶节点开始向前遍历，调整节点性质，使之成为大顶堆
            for (int i = (int)Math.floor(len / 2) - 1; i >= 0; i--)
{
```

```
                heapify(arr, i, len);
            }
        }

        private static void heapify(int[] arr, int i, int len) {
            // 先根据堆性质，找出它左右节点的索引
            int left = 2 * i + 1;
            int right = 2 * i + 2;
            // 默认当前节点（父节点）是最大值。
            int largestIndex = i;
            if (left < len && arr[left] > arr[largestIndex]) {
                // 如果有左节点，并且左节点的值更大，更新最大值的索引
                largestIndex = left;
            }
            if (right < len && arr[right] > arr[largestIndex]) {
                // 如果有右节点，并且右节点的值更大，更新最大值的索引
                largestIndex = right;
            }

            if (largestIndex != i) {
                // 如果最大值不是当前非叶子节点的值，那么就把当前节点和最大值的子节点值互换
                swap(arr, i, largestIndex);
                // 因为互换之后，子节点的值变了，如果该子节点也有自己的子节点，仍需要再次调整。
                heapify(arr, largestIndex, len);
            }
        }

        private static void swap (int[] arr, int i, int j) {
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
}
```

## 4、小顶堆

```
package day0930;

/**
 * @Author: XiPeng
 * @Description: 小顶堆
 * @Date: Create in 14:20 2020/9/30
 * @Modified by:
 */
public class SamllHeap {
    // 堆得存储结构：数组
    private int[] data;
```

```
    /**
     * 构造方法：传入一个数组，并转换为一个最小堆
     *
     * @param data
     */
    public void MinHeap(int[] data) {
        this.data = data;
        buildHeap();
    }

    /**
     * 将数组转化为最小堆
     */
    private void buildHeap() {
        //完全二叉树只有数组下标小于或等于 (data.length) / 2 - 1 的元素有孩子结点，遍历这些结点。
        //比如上面的图中，数组有10个元素， (data.length) / 2 - 1的值为4，a[4]有孩子结点，但a[5]没有
        //即，从下自上开始堆化（从最下层非叶子节点开始）
        for (int i = (data.length) / 2 - 1; i >= 0; i--) {
            heapify(i);
        }
    }

    /**
     * 从当前节点开始堆化
     *
     * @param i
     */
    private void heapify(int i) {
        // 获取左右节点数组下标
        int l = left(i);
        int r = right(i);

        // 假定的当前节点、左子节点、右子节点中 最小值的下标
        int smallest = i;

        // 存在左子节点，且左子节点的值小于当前节点的值
        if (l < data.length && data[l] < data[i])
            smallest = l;

        // 存在右子节点，且右子节点的值小于当前节点的值
        if (r < data.length && data[r] < data[i])
            smallest = r;

        // 左右结点的值都大于根节点，直接return
        if (i == smallest)
            return;

        // 将最小值与当前节点互换位置
        swap(i, smallest);
```

```
        // 从之前最小值节点位置重新堆化
        heapify(smallest);
    }

    /**
     * 获取右节点的数组下标
     *
     * @param i
     * @return
     */
    private int right(int i) {
        return (i + 1) << 1;
    }

    /**
     * 获取左节点的数组下标
     *
     * @param i
     * @return
     */
    private int left(int i) {
        return ((i + 1) << 1) - 1;
    }

    /**
     * 交换元素位置
     *
     * @param i
     * @param j
     */
    private void swap(int i, int j) {
        int tmp = data[i];
        data[i] = data[j];
        data[j] = tmp;
    }

    /**
     * 获取堆中最小元素，即根元素
     *
     * @return
     */
    public int getRoot() {
        return data[0];
    }

    /**
     * 替换根元素，并重新heapify
     *
     * @param root
     */
    public void setRoot(int root) {
        data[0] = root;
```

```
            heapify(0);
        }

    }
```

## 5、冒泡

```
package day0930;

import java.util.Arrays;

/**
 * @Author: XiPeng
 * @Description: 冒泡排序
 * @Date: Create in 14:25 2020/9/30
 * @Modified by:
 */
public class BubbleSort {
    public static void BubbleSort(int[] arr) {
        boolean flag = true;
        while(flag){
            int temp;//定义一个临时变量
            for(int i=0;i<arr.length-1;i++){//冒泡趟数，n-1趟
                for(int j=0;j<arr.length-i-1;j++){
                    if(arr[j+1]<arr[j]){
                        temp = arr[j];
                        arr[j] = arr[j+1];
                        arr[j+1] = temp;
                        flag = true;
                    }
                }
                if(!flag){
                    break;//若果没有发生交换，则退出循环
                }
            }
        }
    }
    public static void main(String[] args) {
        int arr[] = new int[]{1,6,2,2,5};
        BubbleSort.BubbleSort(arr);
        System.out.println(Arrays.toString(arr));
    }
}
```

# 4、linux

查看进程：ps aux

# 5、java

## 1、java体系结构

- jvm虚拟机
- jdk
- 编程语言
- API

## 2、java默认类型及取值范围

- bit:1（-2^7 ~ 2^7）
- short:2
- char:2
- int:4
- float:8
- double:16
- long:8
- boolean:true/false

## 3、java集合框架

### collection

**list:有序，可重复**

- ArrayList:数组。查询快，增删慢，线程不安全
- LinkedList:链表。查询慢，增删快，线程不安全。
- Vector:数组。查询快，增删慢，线程安全。

**set:无序，不可重复**

- HashSet：哈希表，无序，线程不安全，允许为空，hashcode保证唯一性。
- LinkedHashSet：链表。有序。线程不安全
- TreeSet：红黑树保证元素顺序。线程不安全。

**queue:**栈

map：

- TreeMap：有序
- HashMap：无序，同步，可为空，初始容量16，0.75倍增长，两次哈希，底层Entry1.8之前数组+链表，1.8之后数组+红黑树
- HashTable：无序，异步，不为空，初始容量11，翻倍增长，一次哈希
- ConcurrcuentHashMap：1.2：数组，1.8数组+链表+红黑树

### 4、Java求随机数

- new Random()
- Math.random()
- currentTimeMillis()

### 5、为什么重写equals的时候必须重写hashcode

- 不重写会引起equals不同而哈希表相同的情况

### 6、面向对象三大特征

- 继承：继承运行创建分等级的类，子类可以继承父类的属性和方法，提高代码的重用性。
- 封装：把客观的事物抽象成类，把数据和方法封装，对外界提供特定方法操作，把没必要的信息隐藏。以此增强安全性和简化编程。
- 多态：不同对象针对同一消息做出的不同响应。由动态绑定实现，消除类型之间的耦合关系。代码具有可替换性（替换实现的子类）、可扩充性（新增子类不影响已存在类的多态性），接口性（向子类提供一个共同接口），灵活性，简化性。

### 7、接口和抽象类的区别

- 接口是对行为（方法）的抽象，抽象方法集合，不能被实例化，不能包含非常量，所有成员默认被public static final修饰，接口中没有非静态方法的实现，所有方法都是抽象/静态方法。接口实现了定义与实现分离。
- 抽象类也不能被实例化，但可以有零个或多个抽象方法，抽象类是对共同抽象方法的抽取，主要用于代码重用。

## 6、操作系统

---

## 1、进程五大特性：

> *动态性、并发性、制约性、独立性、结构特性*

1. 动态性：进程是个程序的执行，是个动态概念。
2. 并发性：多个进程同存于内存中，能在一段时间内同时执行。
3. 制约性：并发进程间存在资源等制约关系，导致程序不可预测，需要对程序并发次序和运行速度进行协调。
4. 独立性：进程是资源调度和分配的基本单位。
5. 结构特性：进程由程序块、数据块和进程控制块三部分组成。

## 2、进程和线程的区别

1. 一个进程中包含多个线程
2. 多个线程共享进程的堆和方法区，但每个线程有自己的程序计数器、虚拟机栈和本地方法栈
3. 程序是一个静态文件，交给cpu进行运行后，变成一个进程

## 3、进程的三种基本状态

1. 就绪态：进程准备好，资源已经取到，等待cpu执行。
2. 运行态：进程得到处理机被执行。
3. 等待态：进程等待某一事件发生，放弃处理机并进入等待态。

## 4、进程控制块（PCB）7部分

> *系统为了管理进程设置的数据结构*

- 程序计数器：下一个指令的地址
- 进程状态：新建，就绪，执行，阻塞，死亡
- cpu暂存器：中断时暂存数据
- 存储器管理：标签页
- 会计信息：cpu和实际使用数量，时限，账户，工作号。
- 输出输入方式。

## 5、jvm六组件

### pc寄存器

程序计数器，线程执行的字节码指令地址。就是线程执行的位置。

### 虚拟机栈

存储栈帧。一般是一些局部变量或者未计算好的结果

### 堆（共享）

存放GC管理的对象

### 本地方法栈

其他语言使用的栈

### 运行时常量池（方法区一部分）

存放字面量和符号引用量，编译时会将字符串放入一个常量池中，实现对象共享，节约时间和内存。方法区分配且类和接口被加载到虚拟机后就能创建对应的运行时常量池。

### 方法区（共享）

存放运行时常量池、字段、方法（构造函数、普通函数）

# 7、事务和锁

---

## 1、共享锁、排他锁和更新锁

- 共享锁：多个事务锁住一个表，都可以读取，但不能修改。
- 排他锁：一个事务对其枷锁，其他食物在等待该锁被释放后才能访问。
- 更新锁：预定施加排他锁，先用共享锁锁定，但不允许其他事务对其加锁，当遇到更新是，会将其升级为排他锁。

## 2、死锁产生四条件

- 互斥：资源排他性使用，一个资源被一个进程独占，其他进程拿不到。
- 请求保持：进程持有至少一个资源，但不能获取新资源（被其他资源占用），进程阻塞却不会主动释放占用资源。
- 不剥夺：进程在使用之前不能被剥夺，只能在使用完主动释放。
- 环路等待：出现进程-资源-进程的环路链。

## 3、死锁预防四方案

- 破坏互斥：将资源共享。资源独占绝对性，可行性低。
- 破坏请求保存：静态分配，将资源一次性分配完。但会引起资源严重浪费
- 破坏不剥夺：当一个持有不剥夺资源的进程请求新资源而得不到是，自动释放已剥夺的资源。但反复申请和释放资源对cpu消耗过高。
- 破坏循环等待：为资源分级，按照顺序递增，同类资源一次性申请完，每次申请资源只能申请编号大的资源。

## 4、避免死锁的算法

### 判断系统安全法：

先判断安全性，如果这次资源分配会导致系统进入不安全状态，则不分配，反之分配资源。

### 银行家算法

- 申贷总额度不能超过现有资产总和。
- 分批提款，按时还款。
- 若不能够满足用户额度时，给与承诺一定时间交付。

## 5、事务四特性

- 原子性
- 一致性
- 隔离性
- 持久性

## 6、可重入锁和异常锁

- 可重入锁：AB两锁都被synchronized关键字修饰，A调用B时，A方法持有的锁能被B拿到
- 异常锁：程序执行出现异常则锁会被释放，引起程序乱入，使读入数据发生异常。

## 7、锁升级

- 无锁：无业务
- 偏向锁：第一个线程的id被记录
- 自旋锁：另一线程自旋十次，不休眠
- 重量级锁：十次自旋后变成重量级锁。

## 8、变量控制：volatile

- 保证对多线程可见
- 禁止指令重排序

## 9、原子性、可见性和有序性

- 最小单位：不可分割
- 可见性：线程的一个操作对另一个操作来讲是可见的。
- 有序性：volatile“禁止指令重排序”，synchronized（森阔奈子）保证线程安全。

### 10、事务的隔离级别

- 读未提交：（一个事务读到另一个事务提交的数据）出现脏读、幻读、不可重复读
- 读已提交：（数据提交后才能被读）解决脏读
- 可重复读：（读数据时不允许其他操作）解决不可重复读
- 串行化：解决幻读

### 11、脏读、幻读和不可重复读

- 脏读：一个事务读到另一个事务未提交的数据。
- 不可重复读：一个事务两个不同查询返回不同数据。
- 幻读：一个查询结束后有了新的数据插入

## 8、spring

---

### 1、spring常用模块

- spring context：应用上下文。
- spring code：管理bean的组件。
- spring aop：面向切面编程
- spring web：web组件
- spring dao：数据库异常处理。

### 2、SpringBootApplication

- ComponentScan：开启注解扫描
- EnableAutoConfiguration：bean加载到IOC
- SpringBootConfiguration：声明配置类
- Inherited：被子类继承。

### 3、springmvc处理流程

- 用户请求发送到DispatcherServlet。
- DispatcherServlet调用HandlerMapping处理器。
- 处理器根据xml或者注解生成处理器对象和处理器拦截器并返回给DispatcherServlet。
- DispatcherServlet调用HandlerAdapter
- HandlerAdapter调用controller
- controller返回modelandview
- DispatcherServlet将modelandview传给视图解析器。
- 视图解析器返回view
- DispatcherServlet根据view进行视图渲染
- DispatcherServlet响应用户。

### 4、IOc和Dl

- ioc：spring控制对象的声明周期：所有对象都交由spring管理。所有类的创建和销毁均由spring来控制。控制对象的生命周期不再引用其对象，而是spring，所有对象都被spring控制。
- DI:ioc在运行过程中，动态的向某个对象提供他所需要的对象。

## 9、设计模式

---

### 1、观察者：

- 一对多：行为型模式，一个对象被修改会自动通知其他对象。
- 一个对象状态改变要通知其他对象。
- 代码核心：将对象存到list里面。
- 优点：两者抽象耦合，存在触发机制。
- 缺点：多个观察者会导致进度变慢，同时可能会存在多个循环调用导致系统闭环，且观察者只知道通知结果而不知道过程。
- 业务实现
- ```
  package observerPattern;

  import java.util.ArrayList;
  import java.util.List;

  /**
   * @Author: XiPeng
   * @Description: 观察者模式
   * @Date: Create in 9:08 2020/9/30
   * @Modified by:观察者模式使用三个类 Subject、Observer 和
  Client。
   * Subject 对象带有绑定观察者到 Client 对象和从 Client 对象解绑观
  察者的方法。
   * 我们创建 Subject 类、Observer 抽象类和扩展了抽象类 Observer
  的实体类。
   * ObserverPatternDemo，我们的演示类使用 Subject 和实体类对象来
  演示观察者模式。
   */
  public class Subject {
      private List<Observer> observers = new
  ArrayList<Observer>();
      private int status;

      public int getStatus() {
          return status;
      }

      //触发机制
      public void setStatus(int status) {
  ```
  ```

```
        this.status = status;
        notifyAllObservers();
    }
    //通知遍历
    private void notifyAllObservers() {
        for (Observer observer : observers) {
            observer.update();
        }
    }

    public void attach(Observer observer){
        observers.add(observer);
    }
}
```

```
package observerPattern;

/**
 * @Author: XiPeng
 * @Description:
 * @Date: Create in 9:11 2020/9/30
 * @Modified by:
 */
public abstract class Observer {
    protected Subject subject;
    public abstract void update();
}
```

```
package observerPattern;

/**
 * @Author: XiPeng
 * @Description: 观察者3
 * @Date: Create in 10:16 2020/9/30
 * @Modified by:
 */
public class HexaObserver extends Observer {
    public HexaObserver(Subject subject) {
        this.subject = subject;
        this.subject.attach(this);
    }

    public void update() {
        System.out.println("Hex String:"+

 Integer.toHexString(subject.getStatus()).toUpperCase());
    }
}
```

```
import observerPattern.BinaryObserver;
import observerPattern.HexaObserver;
import observerPattern.OctalObserver;
import observerPattern.Subject;
import org.junit.Test;

/**
 * @Author: XiPeng
 * @Description: 观察者模式调用
 * @Date: Create in 10:20 2020/9/30
 * @Modified by:
 */
public class ObserverPatternTest {
    @Test
    public void test(){
        Subject subject = new Subject();
        new HexaObserver(subject);
        new OctalObserver(subject);
        new BinaryObserver(subject);

        System.out.println("first 15:");
        subject.setStatus(15);

        System.out.println("second 10");
        subject.setStatus(10);
    }
}
```

## 2、mvc和mvvm的区别

- v层指挥接收数据，不会对数据进行处理。
- mv层对数据进行间交互，v负责数据的处理。

## 3、单例模式

懒汉式：第一次访问才会被创建。

```
public class Singleton {

    private static Singleton instance = null;

    private Singleton() {
    }

    public static synchronized Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
```

```
}
//优化
public class Singleton {

    private static volatile Singleton instance;

    private Singleton() {}

    public static Singleton getInstance() {
        if (instance == null) {
            synchronized (Singleton.class) {
                if (instance == null) {
                    instance = new Singleton();
                }
            }
        }
        return instance;
    }

}
```

饿汉式：初始化就会被new出来

```
public class Singleton {

    private final static Singleton INSTANCE = new Singleton();

    private Singleton(){}

    public static Singleton getInstance(){
        return INSTANCE;
    }
}
```

## 4、mvc和mvp的区别

- MVC：可以通过jsp直接v，完成c和m的交互，代码的主要逻辑放在v层。
- MVP：Presenter层类似controller层，功能相对复杂，负责和view的双向交互，跟v层一对一。

## 5、适配器模式

- 作为两个并不兼容接口的桥梁，结构性模式，结合两个接口的功能。
- 优点：可以让两个没有任何关联的类一起运行，提高了类的复用，增加了类的透明度，灵活性好。
- 缺点：过多使用适配器会让系统非常凌乱。由于java单继承，只能适配一个适配器类。

### 6、命令行模式

- 将请求封装为一个对象，使发出请求的责任和执行请求的责任分开。两者之间通过命令对象进行沟通。
- 优点：降低系统耦合度。增删命令方便。可以与组合模式组成组合命令。方便Undo和Redo
- 缺点：大量命令类会增加系统的复杂性。

# 10、多线程

---

### 1、创建线程的方式：

1. 继承Thread
2. 实现runnable
3. 实现callable，需要抛出返回值。

### 2、sleep、join和yield

- sleep：线程睡眠：暂停一段时间将cpu让出，时间结束后继续执行。
- yield：将现场重新放回队列，重新与其他线程争夺cpu
- join：a中join线程b，当执行到b时，a暂停，等到b执行完毕后a再继续。

### 3、线程状态

> *其状态在Java中都被jvm管理*

- 新建：new但没有start
- 就绪态：start，资源分配完成，等待时间片轮转
- 执行态：running执行。
- 挂起态：某些原因放弃cpu执行。分为**timedwaiting计时阻塞**，**Blocked（锁阻塞）**和**waiting（无限等待）状态**。
- 结束态：teminated

### 4、三种阻塞状态

- Blocked：锁阻塞，由于获取不到锁产生的阻塞。
- Waiting：无限等待，一个线程在等待另一个线程唤醒，唤醒之前一致保持waiting状态，需要其他线程调用notify或者notifyAll来唤醒，进入无限等待之前必须要获取锁对象
- TimeWaiting：计时等待，一个线程进入waiting状态，当计时结束或者被主动唤醒才会被执行。

### 5、创建线程池

- newCachedThreadPool 创建一个可缓存线程池。
- newScheduledThreadPool创建一个定长线程池，支持定时及周期性任务。
- newSingleThreadExecutor 创建一个单线程化的线程池。

### 6、线程局部变量ThreadLocal

- 线程内数据共享

### 7、数据库连接池

```
//创建固定大小的线程池
ExecutorService executorService = Executors.newFixedThreadPool(2);
//创建缓存大小的线程池
ExecutorService pool = Executors.newCachedThreadPool();
//创建单一的线程池
ExecutorService executor = Executors.newSingleThreadExecutor();
```

### 8、线程关闭后如何重新启动

- Executors.newSingleThreadExecutor();

## 11、jvm

---

### 1、jvm的结构

- 堆
- 方法区
- 程序计数器
- 虚拟机栈
- 本地方法栈

### 2、堆内存

- 对象优先分配到新生代
- 大对象直接进入老年代
- 长期存活的对象进入老年代

### 3、新生代GC和老年代GC

- 新生代GC动作频繁，回收速度快
- 老年代GC动作慢

### 4、垃圾判断算法

- 引用计数器：引用生效则+1，引用失效-1，当计数器为0的时候可以被回收。

### 5、类加载过程

- 加载：把.class放入类加载器。
- 验证：保证加载进来的字符流符合标准。
- 准备：为类变量分配内存，赋予初值。
- 解析：将常量池的符号替换为直接引用。
- 初始化：对static语句遍历进行初始化。

## 10.29

---

- 面向对象三大特性
- 接口和抽象类的区别
- 内部类可以引用他包含类的成员吗
- 什么是链表
- hashmap和hashtable的区别
- 适配器模式
- 命令行模式
- 什么是socket，给出基于tcp协议的套接字实现流程
- 什么是页式存储
- 操作系统内的内存碎片怎么处理
- linux操作及应用
- jvm内存模型和类对象加载
- jvm内存泄漏是什么，什么时候发生
- bean生命周期
- Autowired和resource区别
- @Controller和@respController的区别
- 什么是数据库引擎，什么时候用innodb什么时候用mylsam
- 什么是jdbc，jdbc如何进行事务处理
- 抢红包方案