JavaSE

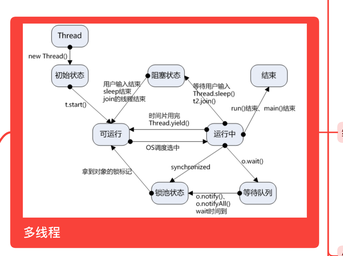
Java的优势

- 跨平台
  - 由于有JVM 的存在，将Java程序编译为字节码文件，Java程序可以在不同操作系统上运行。
- 面向对象
  - Java是一门面向对象的语言
    - 封装
      - 将类的属性和方法进行封装，不被外界随意调用，保证程序的安全性
    - 继承
      - 子类可以继承父类非private的属性和方法，提高程序的复用性，可以重写父类方法
        - 重载: 发生在同一个类中，方法名必须相同，参数类型不同.个数不同.顺序不同，方法返回 值和访问修饰符可以不同，发生在编译时。
        - 重写: 发生在父子类中，方法名.参数列表必须相同，返回值范围小于等于父类，抛出的异常范围小于等于父类， 访问修饰符范围大于等于父类;如果父类方法访问修饰符为 private 则子类就不能重写
          该方法。
      - super与this
        - super关键字可以在子类的构造方法中显示地调用父类的构造方法，super()必须为子类构造函数中的第一行。
        - super可以用来访问父类的成员方法或变量，当子类成员变量或方法与父类有相同的名字时也会覆盖父类的成员变量或方法，这个时候要想访问父类的成员变量或方法只能通过super关键字来访问，子类方法中的super.方法名()不需要位于第一行。
        - this关键字指向的是当前对象的引用，用来区分成员变量和局部变量（重名问题）
        - this() 不能使用在普通方法中 只能写在构造方法中。
        - 必须是构造方法中的第一条语句。
    - 多态
      - 在程序运行时才确定调用方法的对象，同一方法被不同对象调用时，会运行出不同的结果
      - Java实现多态有三个必要条件：继承、重写、向上转型(父类引用指向子类对象)。
- 解释执行
  - 程序在运行Java代码时，会将Java代码编译为Java字节码文件，然后在有Java环境的操作系统上运行
- 支持多线程
  - 并支持多线程之间同步机制

流程控制

- if else
- while
- do-while
- for
- switch case
  - 注意加break，防止穿透

多线程



- 五种状态
  - 新建New
  - 就绪Runnable
  - 运行Running
  - 阻塞Blocked
  - 终止Dead
- 线程创建和使用
  - implements Runnable
  - extends Thread
  - implements Callable 接口并重写 call 方法有返回值，创建线程。
  - ThreadLocal
    - ThreadLocal叫做线程变量，意思是ThreadLocal中填充的变量属于当前线程，该变量对其他线程而言是隔离的，也就是说该变量是当前线程独有的变量。
      - get()
      - set()
      - remove()
    - 适用场景：让某个需要用到的对象在线程间隔离（每个线程都有自己的独立的对象）。
  - 启动线程
    - 线程对象调用 run 方法不开启线程。仅是对象调用方法。
    - 线程对象调用 start 开启线程，并让 jvm 调用 run 方法在开启的线程中执行
    - 调用 start 方法可以启动线程，并且使得线程进入就绪状态，而 run 方法只是 thread 的一个普通方法，还是在主线程中执行
- 线程池
  - ThreadPoolExecutor
    - 属性
      - corePoolSize:核心线程池的大小 maximumPoolSize:线程池能创建线程的最大个数 keepAliveTime:空闲线程存活时间 unit:时间单位，为 keepAliveTime 指定时间单位
    - 方法
      - execute()
        - 向线程池提交一个任务，让线程池中的线程去执行这个任务。
    - 优点
      - 1、使用线程池可以重复利用已有的线程继续执行任务，避免线程在创建销毁时造成的消耗; 2、由于没有线程创建和销毁时的消耗，可以提高系统响应速度; 3、通过线程可以对线程进行合理的管理，根据系统的承受能力调整可运行线程数量的大小等;
- 线程同步
  - synchronized
- 多线程通信
  - 线程调度
    - setPriority()和getPriority()
    - 线程让步：Thread.yield()
    - 线程加入：join()方法，等待其他线程终止。
      - 在当前线程中调用另一个线程的join()方法，则当前线程转入阻塞状态，直到另一个进程运行结束，当前线程再由阻塞转为就绪状态。
  - 线程等待与唤醒
    - sleep(ms)线程休眠
    - wait()线程阻塞来自Object类
    - notify()线程唤醒
    - notifyAll()
- 守护线程
  - thread.setDaemon(true)
  - 常见的做法，就是将守护线程用于后台支持任务，比如垃圾回收、释放未使用对象的内存、从缓存中删除不需要的条目。大多数 JVM 线程都是守护线程.
- 线程:是进程的一个实体，是 cpu 调度和分派的基本单位，是比进程更小的可以独立运行的基本单位。
  进程:具有一定独立功能的程序关于某个数据集合上的一次运行活动，是操作系统进行资源分配和调度的一个独立单位。

接口和抽象类

- 抽象类的子类使用 extends 来继承
  - 抽象类可以有main方法也可以运行，方法可以是任意修饰符
    - 有抽象方法的类一定是抽象类，抽象类不一定有抽象方法，继承了抽象类的子类必须实现父类中的抽象方法，除非子类也是抽象类
- 接口必须使用 implements 来实现接口
  - 接口不能有main方法，方法默认是public修饰

迭代器

- Iterator对象称为迭代器，主要用于遍历Collection集合中的元素。

```java
public static void main(String[] args) {
    List<Integer> list = new ArrayList<>();
    list.add(111);
    list.add(222);
    list.add(333);

    System.out.println("删除前:" + list);
    Iterator<Integer> iterator = list.iterator();
    while (iterator.hasNext()) {
        Integer value =  iterator.next();
        if(value == 111) iterator.remove();
    }
    System.out.println("删除后" + list);
}
```

  - Java增强for循环可以代替Iterator迭代器，可以把它看做简化版的Iterator，和迭代器本质一样，其实它的底层实现就是Iterator迭代器，只能用于遍历集合或数组。
- 所有实现了Collection接口的集合类都有一个iterator()方法，用以返回一个实现了Iterator接口的对象，即可以返回一个迭代器。
- Iterator仅用于遍历集合，Iterator本身并不存放对象。

Java8新特性

- Lambda表达式
  - `new Thread(()-> System.out.println("abc")).start();`
    Lambda 允许把函数作为一个方法的参数。
    - 方法体只有一句代码时大括号return和唯一一句代码的分号可以省略
    - 参数类型可以省略
    - 方法只有一个参数时小括号可以省略
- Optional
  - Optional的静态方法ofNullable来把数据封装成一个Optional对象。无论传入的参数是否为null都不会出现问题

```java
public static void main(String[] args) {
        Author author = getAuthor();
        Optional<Author> authorOptional = Optional.ofNullable(author);
        authorOptional.ifPresent(author1 -> System.out.println(author1.getName()));
    }

    public static Author getAuthor(){
        Author author = new Author(1L, "蒙多", 33, "一个从菜刀中明悟哲理的坦安人", null);
        return null;
    }
```

- Stream流
  - 创建流
    - 单列集合： 集合对象.stream()
    - 数组：Arrays.stream(数组)或者使用Stream.of （）来创建
  - 中间流
    - filter
    - distinct
    - sort
    - limit
      - 限制流的长度，超出的部分将被抛弃。
    - skip
      - 跳过前几个元素
    - map
      - 将流元素取出进行操作

```java
authors.stream() Stream<Author>
        .map(author -> author.getAge()) Stream<Integer>
        .map(age->age+10)
        .forEach(age-> System.out.println(age));
```

    - faltMap
      - map只能把一个对象转换成另一个对象来作为流中的元素。而flatMap可以把一个对象转换成多个对象作为流中的元素。

```java
//    //不同唯有数据的形式分类，要求对分类进行去重，不能出现这种格式：哲学,爱情      爱情
        authors.stream() Stream<Author>
                .flatMap(author -> author.getBooks()).stream() Stream<Book>
                .distinct()
                .flatMap(book -> Arrays.stream(book.getCategory().split(",",-1))) Stream<String>
                .distinct()
                .forEach(category-> System.out.println(category));
```

  - 终结流
    - count
    - collect

```java
////        获取一个存放所有作者名字的List集合。
        List<String> nameList = authors.stream() Stream<Author>
                .map(author -> author.getName()) Stream<String>
                .collect(Collectors.toList());
        System.out.println(nameList);
```

    - reduce 归并

```java
Optional<Integer> minOptional = authors.stream() Stream<Author>
        .map(author -> author.getAge()) Stream<Integer>
        .reduce((result, element) -> result = element ? element : result);
minOptional.ifPresent(age-> System.out.println(age));
```

      - result 初始值为0
      - 对流中的数据按照你指定的计算方式计算出一个结果。（缩减操作）
  - 注意事项
    - 惰性求值（如果没有终结操作，只有中间操作是不会得到执行的）
    - 流是一次性的（一旦一个流对象经过一个终结操作后。这个流就不能再被使用）
    - 不会影响原数据（我们在流中可以多数据做很多处理。但是正常情况下是不会影响原来集合中的元素的。这往往也是我们期望的）
- 函数式接口
  - 有且仅有一个抽象方法的接口（不包括默认方法、静态方法以及对Object方法的重写）通常接口上会添加@FunctionInterface
    - 引用类的实例方法- 类名::方法名
    - 引用对象的实例方法- 对象名::方法名
    - 构造器引用- 类名::new

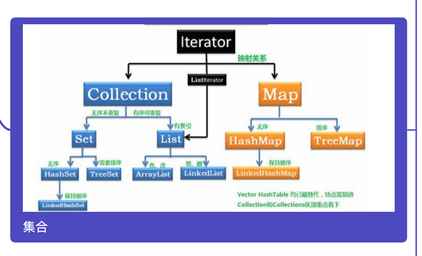
泛型

- Java 泛型（generics）是 JDK 5 中引入的一个新特性, 泛型提供了编译时类型安全检测机制，该机制允许程序员在编译时检测到非法的类型。
  - 可以很大程度上提升代码的重用性
    - 泛型类
      - public class 类名 <泛型类型,...> {}
    - 泛型接口
      - public <泛型类型> 返回类型 方法名（泛型类型变量名）{}
    - 泛型方法
      - public <泛型类型> 返回类型 方法名（泛型类型变量名）{}
  - 项目实例

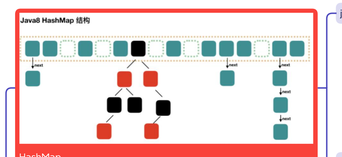
```java
/**
 * 数据源接口（新接入的数据源必须实现）
 *
 * @param <T>
 */
public interface DataSource<T> {

    /**
     * 搜索
     *
     * @param searchText
     * @param pageNum
     * @param pageSize
     * @return
     */
    Page<T> doSearch(String searchText, long pageNum, long pageSize);
}
```

```java
SearchVO searchVO = new SearchVO();
DataSource<?> dataSource = dataSourceRegistry.getDataSourceByType(type);
Page<?> page = dataSource.doSearch(searchText, current, pageSize);
searchVO.setDataList(page.getRecords());
return searchVO;
```

集合



- Colletcion
  - List
    - ArrayList
      - 底层维护了一个动态数组，长度可变。查询元素时比较快，因为使用的是下标查询，但是增删元素较慢，由于用到数组元素的移动。每次扩容50%，线程不安全
        - add()
        - size()
        - get()
      - 如果需要在多线程情况下使用可以用Collection.synchronizedList(list);
    - Vector
      - 线程安全
        - add方法上用synchronized修饰，每次扩容一倍
    - LinkedList
      - 底层维护了一个双向链表，有头节点和尾节点，查询元素较慢，需要挨个遍历。增删元素快，将头节点和尾节点改变即可。线程不安全。
  - Set
    - HashSet
      - 基于HashMap实现，插入元素是无序的并且唯一，可以插入一个null。线程不安全
        - isEmpty()
        - contains()
        - size()
        - remove()
        - clear()
      - 插入元素时不仅需要比较hash值，还要结合equals方法比较
    - TreeSet
      - 底层是通过红黑树实现，基于key有序的key value散列表。线程不安全。
- Map
  - HashMap



    - jdk1.7
      - 底层通过数组+链表实现
        - 采用的是头部插入法：当计算得到的索引值相同时，即hash碰撞，引入链表结构解决Hash冲突，新加入的值会插入到旧数据的头部。
    - jdk1.8
      - 底层通过数组+链表+红黑树
        - 当链表中的元素超过了 8 个以后， 会将链表转换为红黑树(提高查询效率)，当红黑树节点 小于 等于 6 时又会退化为链表。
          - put(k,v) 将K,V 封装到Node节点中，调用K的hashCode方法计算出hash值，通过hash值转换成数组下标，如果当前位置没有值则直接加入，如果当前有链表则与每一个进行equals()比较，有true时覆盖，全为false则加到尾部
          - get(k) 调用K的hashCode()计算出hash值，通过hash算法转为数组下标，如果数组此处无值则返回null，否则挨个进行equals()比较都为false为返回null，如果有ture则是需要返回的值
        - 默认的负载因子大小为 0.75，数组大小为 16。也就是说，默认情况下，那么当 HashMap 中元素个数超过 16*0.75=12 的时候，就把数组的大小扩展为 2*16=32，即扩大一倍。
  - Hash算法：是指将任意长度的二进制，映射为固定长度的较小的二进制值，这个较小的二进制值叫做哈希值
    - Hash 冲突
      - 不同的对象算出来的数组下标是相同的这样就会产生 hash 冲突，当单线链表达到一定长度后效率会非常低。
  - TreeMap
    - 底层是通过红黑树实现，map根据其键的自然顺序排序，Key不可存null.
  - HashTable
    - 实现key和value都不能为null，线程安全,默认大小为11
      - put方法上用synchronized修饰
  - ConcurrentHashMap
    - 线程安全
      - 在jdk1.8，ConcurrentHashMap 取消了 Segment 分段锁，采用 CAS 和 synchronized 来保证线程安全
        - synchronized属于悲观锁，悲观的认为程序中的并发情况严重，所以严防死守。CAS属于乐观锁，乐观地认为程序中的并发情况不那么严重，所以让线程不断去重试更新。（可能会自旋）

Java的常用类及其方法

- String 字符串类型
  - length()
  - contains()
  - equals()
    - String类中重写了equals方法，判断两个字符串只要内容一样，就是相等的。
      - == 的作用:
        基本类型:比较的就是值是否相同
        引用类型:比较的就是地址值是否相同
      - equals 的作用:
        引用类型:默认情况下，比较的是地址值。
        特:String、Integer、Date 这些类库中 equals 被重写，比较的是内容而不是地址!
  - reverse()
  - charAt()
    - 返回指定索引处的字符
  - 与String相关的类继承自 AbstractStringBuilder
    - StringBuilder
      - 线程安全采用append()方法
    - StringBuffer
      - 单线程下操作速度最快，线程不安全
  - 基本概念
    - String是被final修饰的类，不能被继承；
    - String实现了Serializable和Comparable接口，表示String支持序列化和可以比较大小；
    - String底层是通过char类型的数据实现的，并且被final修饰，所以字符串的值创建之后就不可以被修改，具有不可变性。
- Object 所有类的父类
  - equals()
    - 内部是通过==判断
      - 基本数据类型判断值是否相等
      - 引用数据类型判断是否是同一个对象
  - toString()
    - 返回该对象的字符串表示
  - hashCode()
    - 返回该对象的哈希码
      - java对象的内存地址经过哈希算法得到的值
- Math
  - abs()
    - 取绝对值
  - round()
    - 四舍五入
  - pow()
    - 取幂
      - Math.pow(3,2) ==> 9
  - max(),min()
  - random()
  - ceil()/floor()
    - 向上取整，向下取整
      - 向上取整，无论是正数还是负数，都取最大的值
      - 向下取整，无论是正数还是负数，都取最小的值
- File
  - 构造方法
    - File(File parent, String child);
    - File(String pathname);
  - getAbsolutePath()
  - isFile()
  - exists()
  - midir()

包装类

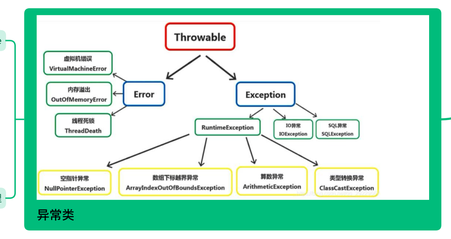
- Byte
- Short
- Integer
- Long
- Float
- Double
- Character
- Boolean
- 自动装箱和拆箱是 JDK 1.5 开始提供的特性。
  - 装箱：将基本类型用它们对应的引用类型包装起来；
  - 拆箱：将包装类型转换为基本数据类型；

Java的八大基本数据类型

- 整数类型
  - byte
    - 1个字节
  - short
    - 2个字节
  - int
    - 4个字节
  - long
    - 8个字节
- 浮点数类型
  - float
    - 4个字节
  - double
    - 8个字节
- 字符型
  - char
    - 2个字节
- 布尔型
  - boolean
    - 1个字节
    - 默认为false

异常处理

- Throwable
  - Error
    - 一般不编写针对性的代码进行处理
      - 栈溢出
      - 内存泄漏
  - Exception
    - RuntimeException
      - 空指针异常
      - 数组越界异常
      - 非法参数异常
    - 编译时异常
- try
  - 将可能发生的异常代码写入
- catch
  - 捕获异常，一般都是从小到大
- finally
  - 不管发生或不发生异常都会执行，一般用于资源的释放/关闭
- throw
  - throw是语句抛出一个异常。
- throws
  - thorws时方法选择上抛出异常，谁调用抛给谁



异常类

IO流

- 16位以字符为单位传输
  - 字符流
    - 实现类
      - Reader
        - BufferReader
        - InputStreamReader

```java
public static void main(String[] args) throws IOException {
    File file = new File( pathname: "/Users/zhangyujie/Downloads/Acwing算法基础.md");
    FileReader reader = new FileReader(file);
    // 开始读
    char[] chars = new char[4]; // 一次读取4个字符
    int readCount = 0;
    while(readCount = reader.read(chars)) != -1) {
        System.out.print(new String(chars, offset: 0,readCount));
    }
```

FileReader

```java
public static void main(String[] args) throws IOException {
    File file = new File( pathname: "/Users/zhangyujie/Downloads/Acwing算法基础.md");

    BufferedWriter out= new BufferedWriter(new OutputStreamWriter(new FileOutputStream(file, append: true)));
                                                                                    //追加
    // 开始写.
    out.write( str: "hello world!");
    out.write( str: "\n");
    out.write( str: "hello kitty!");
    // 刷新
    out.flush();
    // 关闭操作流
    out.close();
```

BufferWriter

      - Writer
        - FileWrite
        - OutputStreamWrite
- 8位以字节为单位传输
  - 字节流
    - 实现类
      - InputStream

```java
public static void main(String[] args) throws IOException {
    File file = new File( pathname: "/Users/zhangyujie/Downloads/Acwing算法基础.md");
    FileInputStream fis = new FileInputStream(file);
    byte[] bytes = new byte[4];// 准备一个4个长度的byte数组，一次最多读取4个字节.
    int readCount = 0;
    // 这个方法的返回值是: 读取到的字节数量. (不是字节本身) ;1个字节都没有读取到返回-1(文件读到末尾)
    while((readCount = fis.read(bytes)) != -1) {
        // 不应该全部都转换，应该是读取了多少个字节，转换多少个.
        System.out.print(new String(bytes, offset: 0, readCount));
    }
```

FileInputStream

        - BufferInputStream
          - FilterInputStream
        - ObjectInputStream
      - OutputSteam
        - Fileoutpustream
        - BufferOutputStream
          - FilterOutputStream

```java
public static void main(String[] args) throws IOException {
    File file = new File( pathname: "/Users/zhangyujie/Downloads/Acwing算法基础.md");
    // 创建java对象
    Student s = new Student( i: 1111, zhangsan: "zhangsan");
    // 序列化
    ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream( name: "students"));

    // 序列化对象
    oos.writeObject(s);
    // 刷新
    oos.flush();
    // 关闭
    oos.close();
}
```

ObjectOutputStream Student要实现序列化接口 Serializable

JDK，JRE，JVM

- JDK java开发工具包 java的基础类库，如果要编写java程序的话就需要JDK
- JRE java运行时环境，里面包括JVM
- JVM java虚拟机，在不同的操作系统上有不同的java虚拟机，如果只需要运行java程序的话，只需要java虚拟机就够了
- javac的任务就是将Java源代码转化成JVM能够识别的一种语言（Java字节码），这种字节码不是针对某种机器、某种平台的

反射

- 优点
  - 运行时动态的获取类信息
  - 动态的加载类属性和方法
- 缺点
  - 破坏程序的封装
  - 效率比较低
- 常用方法
  - setAccessible()
    - 默认false，设置为true为打破封装
  - Class类方法
    - getFields()
      - 获取public修饰的属性
    - newInstance()
    - getDeclaredConstructors()
      - 所有构造器
    - getDeclaredMethods()
      - 所有方法
  - Field方法
    - set(Object obj, Object value)
      - 设置属性值
    - get(Object obj)
      - 读取属性值
  - Method 反编译一个类的方法
    - invoke(Object obj, Object... args)
      - 调用方法
    - getReturnType()
    - getParameterTypes()
    - 方法.invoke(对象, 实参);
  - Constructor 反编译一个类的构造方法
    - newInstance(Object ... int args)
      - 创建对象
- 使用场景
  - 大多数RPC框架使用反射例如：springcloud Dubbo
  - Junit 单元测试

自由主题

Presented with xmind