## JAVA浅析

1. JAVA类型占用内存
2. 一个内存地址：1）8字节（8=64/8）2）4字节（4=32/8）
3. boolean:1, char:2, int,float:4, double,long:8
4. Integer：24 = 16 + 4（一个int） + 4（填充字节）
5. Date：32 = 16 + 3\*4（年月日三个int） + 4（填充字节）
6. 一个引用类型所占用的内存大小是8字节的倍数：

对象开销：16，头，hashcode，类信息

引用开销：8（固定）

基本数据类型：加上用到的基本数据类型，如Date中的3\*4

填充字节：为了补充成8字节的倍数

1. StringBuffer、StringBuilder

这两个类型内，都用了System.arraycopy(value,srcBegin,des,desBegin,length)，解决字符串不可变长的问题。

对字符串的其他操作是重用了value数组，仅仅是偏移量、长度、hash值变了，构造一个子字符串时间为常数。

这两个类型中的，reverse方法，也是对value数组做的变化，很高效。

1. 日期格式

用Calendar类，可以获取各种时间类型。Calendar calendar = Calendar.getInstance();

用SimpleDateFormat，可以定制时间格式。

SimpleDateFormat myFmt2=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

y 年，M 月，d 日，h 时(1~12)，H 时(0~23)，m 分，s 秒，S 毫秒。

1. 参数生命周期
2. page：页面运行完成后结束
3. Request：请求响应后结束

表单参数或URL里的参数用getParameter()来获取

在请求过程中设置/取值用get/setAtrribute()

1. session：HTTP的无状态特性，要想获取会话功能，必须在请求和响应上加上一个ID来标识一次会话。有2中技术：cookie，URL重写。Cookie技术可以在用户关闭浏览器后再次打开浏览器访问时保持会话。
2. Application：全局，随着服务器的关闭而关闭

Context包含：servlet、filter、listener

1. 类型转换

高类型向低类型转换，需要强制转换。反之，不需要。

因为低类型有高类型没有的细节，所以高类型要强转。而低类型到高类型是一一对应的关系。不需要额外的动作。

例：

float f = (float) 3.4;因为默认小数是双精度的。或float f = 3.4f;

Short s=1;s = (short)(s+1);或 s += 1;后一个隐含强转。

1. 字符串与基本类型间的相互转换
2. 字符串转换成基本数据类型

基本包装类型.valueOf(String);//Double.valueOf(“1.2”);内部调用parseDouble();

基本包装类型.parse基本类(String);//Integer.parseInteger(“12”);

1. 基本数据类型转换成字符串

String.valueOf(基本数据类型);//String.valueOf(1.2);内部调用toString();

基本包装类型.toString(基本类型);//Integer.toString(2);

1. final修饰符
2. 修饰类：类不能被继承
3. 修饰方法：方法不能被重写
4. 修饰变量

基本类型变量，只能赋值一次。

引用类型变量，地址不能变，当可改变地址的值。

如果用static修饰，则为常量，只能在类里定义并且必须初始化，不能在方法。

1. 反射技术

基础是类类型，class类，是在类加载阶段生成的，在方法区中。

通过类类型可以得到一个类的属性和方法：

1. reflect.constructor：获得类的构造器
2. reflect.Filed：获得类的成员变量
3. reflect.Method：获得类的方法
4. reflect.Modifier：获得成员、方法、类的修饰符
5. reflect.Array：直接对类中集合设置
6. 注解

@Target：对象可用的范围（METHOD、PACKAGE、PARAMETER、CONSTRUCTOR、FIELD、LOCAL、VARIABLE）

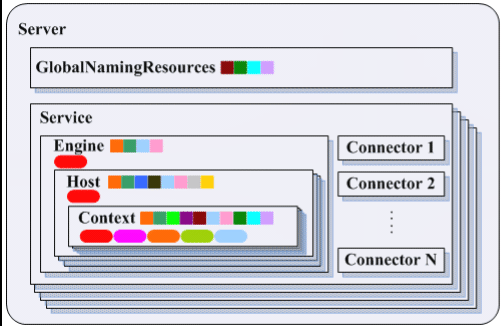
@Retention：生命周期

@Document：生成文档

@Inherited：允许子类继承父类中的注释，只有当前类不存在指定注释时，才能获取父类注释，通过superClass.getAnnotation获取父类注释

@Repeatable：允许同一位置使用相同的注解

1. Tomcat结构



1. Shiro

Apache开源的安全管理框架（security manager）

功能：

1. 用户身份验证
2. 用户访问权限控制
3. 在任何环境下使用Session API
4. 使用多个数据源
5. 单点登录
6. rememberMe

技术：

1. Authentication
2. Authorization
3. Session Management
4. Cryptography
5. CacheManager
6. Spring 控制反转（IOC）、依赖注入（DI）

1）IOC：把创建对象的权利交给框架。需要依赖注入的配合，将原来顶层的东西依赖于底层，反转成底层依赖于顶层。将底层类，作为参数注入到上层类，实现上层类对下层的控制。这样底层的对象怎么实现，上层可以不用关心了，专注做自己的事情。

举个例子：我码代码的时候渴了，就去小卖铺买水。老板直接给我一瓶水。这个老板就是框架。如果没有这个框架，我只能自己去找水喝。

好，那我去找水了，第一次我在A处找到了水，我以后都去A处喝水了。如果A处不是太远的话也行（创建对象很方便）。

第一，如果很远的话（创建对象很不方便）就不好了。

第二，如果水源总是变动，也不行。

如果是依赖反转我就不需要再创建底层对象上花时间了，可以把时间留出来干别的有意义的事了。

2）DI：依赖注入方式

如果是bean，则递归调用getBean来创建bean；

如果是集合类型，则调用reflect.Array包方法直接设置；

如果不是，则调用reflect.Method方法，用get/set方法设置。

3）IOC缺点

① 反射效率慢，但随着对其的不断优化，会越来越快

② 生成对象步骤变复杂了

③ 该类名，但改不了XML里的名字

1. JAVA特性集1
2. JAVA虚拟机：一台虚拟的计算机，独立于平台，有着自己的内存模型和指令集，去执行JAVA字节码
3. JRE：java运行时最基本的环境，有JVM和Java SE API构成
4. JDK：java开发环境，包含JRE、JAVA程序设计语言、JAVA API类库
5. 不能覆盖Static方法：因为方法覆盖是基于运行时动态绑定的，而static方法编译时静态绑定，已经唯一确定了
6. Private方法不能覆盖：因为修饰的是私有成员，外界根本看不到
7. Static方法中不能访问非static方法：因为类加载的时候需要对static成员进行初始化，而非static变量要到对象初始化时才初始化，所以会报错。但非static中可以有static
8. 构造函数重载，不同参数的构造函数
9. 创建线程的三种方式
10. 实现Runnable类
11. 继承Thread类
12. Callable类，创建线程池，使用Executor框架
13. Runnable更常用，单继承优势
14. N个线程访问N个资源

破坏循环等待条件，在一个资源中设置phaser进行并发阶段任务

1. hashcode与equals方法
2. hashmap先判断hash值是否一样，再判断key值是否一样
3. Equals一样hashcode肯定一样，反之不对
4. 判断相等时总是先判断hashcode再判断equals如果hashcode不同肯定不相等
5. **两个不同对象的hashcode可能相同**
6. 如果类没有复写这两个方法，则比较的就是地址
7. comparable和comparator区别
8. comparable：对象内部实现的排序接口，当使用collection.sort()排序时，默认实现内部的排序方式，静态绑定
9. Comparator：对象外部实现排序的接口，使用collection.sort(XX,new Mycomparator())，来指定外部排序规则，比comparable更灵活，可以随时修改排序规则，为动态绑定
10. 集合的最佳使用
11. 若已知元素大小固定，且知道大小，则用Array（数组）代替ArrayList
12. 如果能估计出存储元素的大致数目，可以设置初始化数量，避免resize
13. 编程时接口优先于实现
14. 底层集合实际上是空的情况下，返回长度是0的集合或者数组而不是Null
15. finally块是一定会被执行的，主要是用来释放资源，如I/O和数据库连接
16. 当对象引用被值为null是，gc会再下一次gc时回收，而不是这次，这次是标记
17. 但永久代满了或是超过了临界值则触发一次fullGC
18. 面向对象
19. OOP达到的目标：重用性、灵活性、扩展性
20. 重要特性：封装、继承、多态
21. 封装：隐藏代码的内部状态，使代码模块化，各模块之间独立，可以单独修改一个模块，不影响其他模块
22. 继承：在不改变类的情况下给现存的类添加新的特性，提高了代码的重用性
23. 多态：多态的概念比较复杂，有多种意义的多态，一个有趣但不严谨的说法是：继承是子类使用父类的方法，而多态则是父类使用子类的方法。父类的引用可以指向子类对象的方法。  
    一般，我们使用多态是为了避免在父类里大量重载引起代码臃肿且难于维护。

如果使用多态的话，最终所选取的方法是按照以下顺序来的：**this是运行时对象**

This.show(O)->super.show(o);->this.show(super(o));->super.show(super(o));

如果父类指向的子类对象，重写了父类的方法，则调用子类的方法。

1. 五大基本原则
2. 单一职责原则：一个类功能要单一，这样类的复杂性最低，耦合性最小，维护、扩展容易
3. 开闭原则：对扩展开放、对修改闭合
4. 替换原则：父类出现的地方、子类一定能出现
5. 依赖倒置原则：具体依赖于抽象、底层依赖于顶层
6. 接口隔离原则：模块间要通过接口隔离开来，而不是通过具体的类来强耦合
7. 设计模式多了两个原则
8. 迪米特法则：又最少知识原则，一个对象应该对其他对象有尽可能少的了解。
9. 合成复用原则：尽量使用合成/聚合的方式，而不是继承
10. 多态的定义：同一个方法可以根据不同的调用对象和参数列表，选择多种行为方式
11. 实现多态的技术：动态绑定，是指在执行期间判断所引用对象的实际类型，来调用相应方法
12. 实现多态的方式：接口实现、继承父类方法重写、同一个类进行方法重载