# 基础篇

## Java基础

### 面向对象的特征

继承、封装和多态

### final, finally, finalize 的区别

#### final修饰符（关键字）

##### Final修饰类

如果一个类被final修饰，意味着该类不能派生出新的子类，不能作为父类被继承。因此一个类不能被声明为abstract，又被声明为final。

##### Final修饰方法

被声明为final的方法也只能使用，不能重写。

##### Final修饰变量

final修饰一个成员变量（属性），必须要显示初始化。这里有两种初始化方式，一种是在变量声明的时候初始化；第二种方法是在声明变量的时候不赋初值，但是要在这个变量所在的类的所有的构造函数中对这个变量赋初值。

#### Finally

finally作为异常处理的一部分，它只能用在try/catch语句中，并且附带一个语句块，表示这段语句最终一定会被执行（不管有没有抛出异常），经常被用在需要释放资源的情况下。

#### Finilize

finalize()是在java.lang.Object里定义的，也就是说每一个对象都有这么个方法。这个方法在gc启动，该对象被回收的时候被调用。其实gc可以回收大部分的对象，所以一般是不需要程序员去实现finalize的。

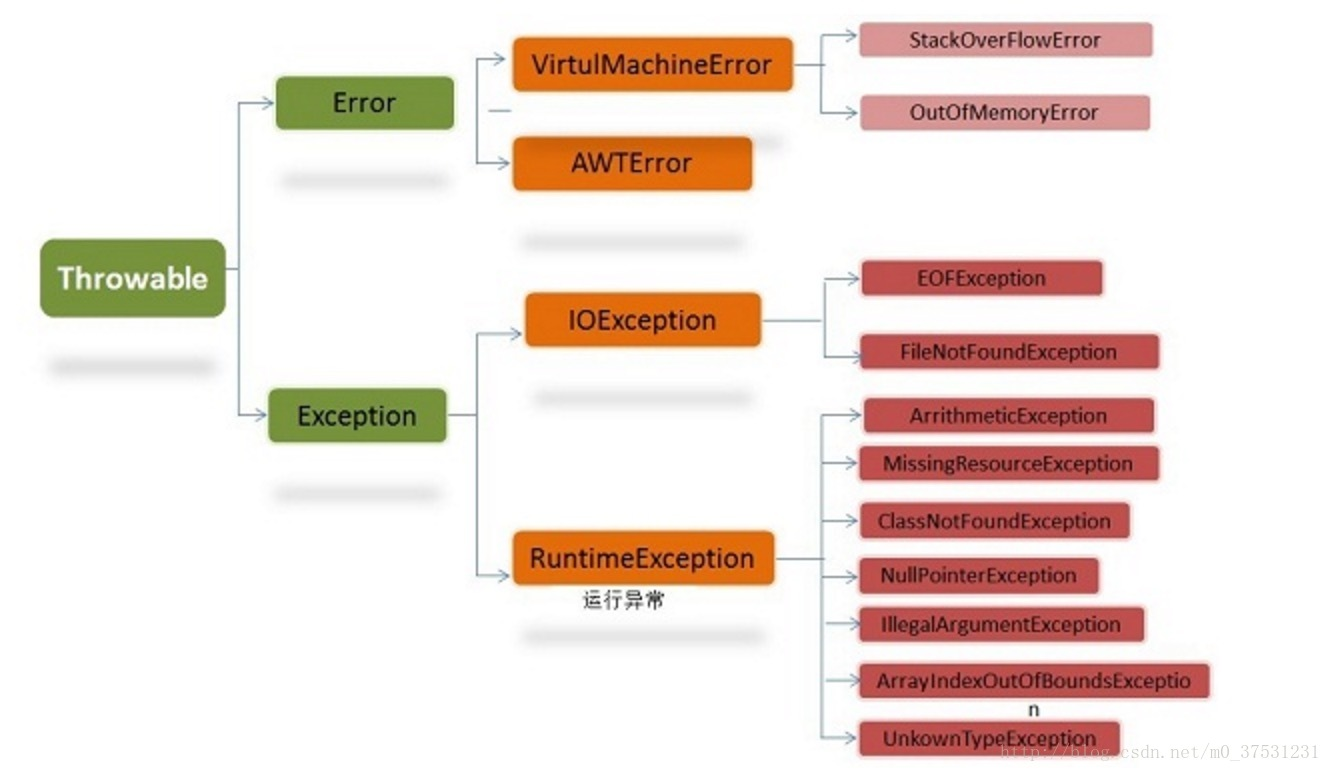
特殊情况下，需要程序员实现finalize，当对象被回收的时候释放一些资源，比如：一个socket链接，在对象初始化时创建，整个生命周期内有效，那么就需要实现finalize，关闭这个链接。

使用finalize还需要注意一个事，调用super.finalize();

一个对象的finalize()方法只会被调用一次，而且finalize()被调用不意味着gc会立即回收该对象，所以有可能调用finalize()后，该对象又不需要被回收了，然后到了真正要被回收的时候，因为前面调用过一次，所以不会调用finalize()，产生问题。 所以，推荐不要使用finalize()方法，它跟析构函数不一样。

### Exception、Error、运行时异常与一般异常有何异同

#### 图



#### Error

描述了java运行时系统的内部错误和资源耗尽错误。大多数错误与代码编写者执行的操作无关，而表示代码运行时 JVM（Java 虚拟机）出现的问题。应用程序不应该抛出这种类型的对象。

#### Exception

##### RuntimeException

##### 非RuntimeException

### 请写出5种常见到的runtime exception

#### ClassCastException

Object x = new Integer(0);

System.out.println((String)x);

#### ArithmeticException

int a=5/0;

#### NullPointerException

String s=null;

int size=s.size();

#### ArrayIndexOutOfBoundsException

#### StringIndexOutOfBoundsException

"hello".indexOf(-1);

#### NegativeArraySizeException

String[] ss=new String[-1];

#### UnsupportedOperationException

#### ConcurrentModificationException

#### FileSystemNotFoundException

#### DOMException

#### ParseException

#### WebServiceException

### 请列出５种常见的Error

#### OutOfMemoryError

#### StackOverflowError

#### NoSuchMethodError

#### NoSuchFieldError

#### UnsupportedClassVersionError

### 请写出５种常见的Checked异常

#### DataFormatException

#### FileNotFoundException

#### EOFException

#### XMLParseException

#### ClassNotFoundException

#### NoSuchFieldException

属性不存在异常。当访问某个类的不存在的属性时抛出该异常。

#### NoSuchMethodException

#### UnsupportedEncodingException

### int 和 Integer 有什么区别，Integer的值缓存范围

Integer是int的包装类；int是基本数据类型；

Integer变量必须实例化后才能使用；int变量不需要；

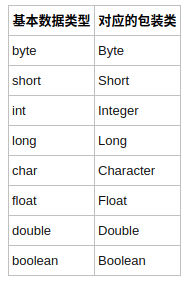
Integer实际是对象的引用，指向此new的Integer对象；int是直接存储数据值 ；

Integer的默认值是null；int的默认值是0。

java在编译Integer i = 100 ;时，会翻译成为Integer i = Integer.valueOf(100)。而java API中对Integer类型的valueOf的定义如下，对于-128到127之间的数，会进行缓存

### 包装类，装箱和拆箱

#### 基本数据类型及对应的包装类



#### 装箱和拆箱

Integer i = 10; //装箱

int n = i; //拆箱

在装箱的时候自动调用的是Integer的valueOf(int)方法。而在拆箱的时候自动调用的是Integer的intValue方法。

### String、StringBuilder、StringBuffer

#### 执行速度

StringBuilder > StringBuffer > String

#### 线程安全

StringBuilder：线程非安全的

StringBuffer：线程安全的

### 重载和重写的区别

#### 重写(Override)

重写是子类对父类的允许访问的方法的实现过程进行重新编写。

访问权限不能比父类中被重写的方法的访问权限更低。例如：如果父类的一个方法被声明为public，那么在子类中重写该方法就不能声明为protected。

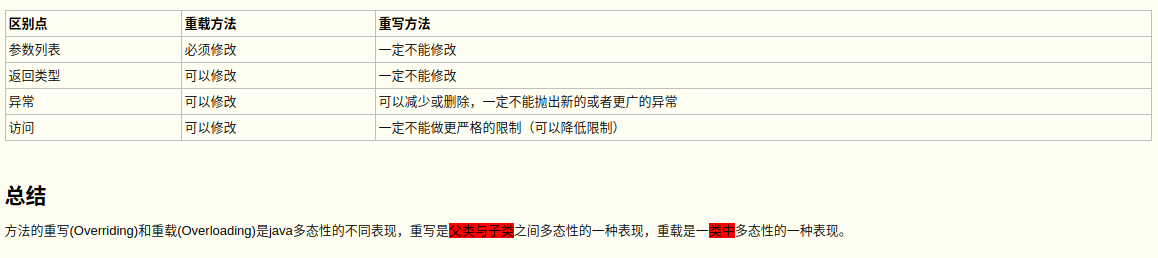
重写方法不能抛出新的检查异常或者比被重写方法申明更加宽泛的异常。例如： 父类的一个方法申明了一个检查异常 IOException，但是在重写这个方法的时候不能抛出 Exception 异常，因为 Exception 是 IOException 的父类，只能抛出 IOException 的子类异常。

#### 重载(Overload)

重载(overloading) 是在一个类里面，方法名字相同，而参数不同。返回类型可以相同也可以不同。

法以返回值类型作为重载函数的区分标准。

#### 区别



### 抽象类和接口有什么区别

#### 抽象类

抽象方法是一种特殊的方法：它只有声明，而没有具体的实现。

如果一个类含有抽象方法，则称这个类为抽象类，抽象类必须在类前用abstract关键字修饰。

抽象方法必须为public或者protected（因为如果为private，则不能被子类继承，子类便无法实现该方法），缺省情况下默认为public。

如果一个类继承于一个抽象类，则子类必须实现父类的抽象方法。如果子类没有实现父类的抽象方法，则必须将子类也定义为为abstract类。

#### 接口

接口中的变量会被隐式地指定为public static final变量（并且只能是public static final变量，用private修饰会报编译错误）。

方法会被隐式地指定为public abstract方法且只能是public abstract方法（用其他关键字，比如private、protected、static、 final等修饰会报编译错误）。

#### 抽象类和接口的对比

语法层面上的区别

　　1）抽象类可以提供成员方法的实现细节，而接口中只能存在public abstract 方法；

　　2）抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是public static final类型的；

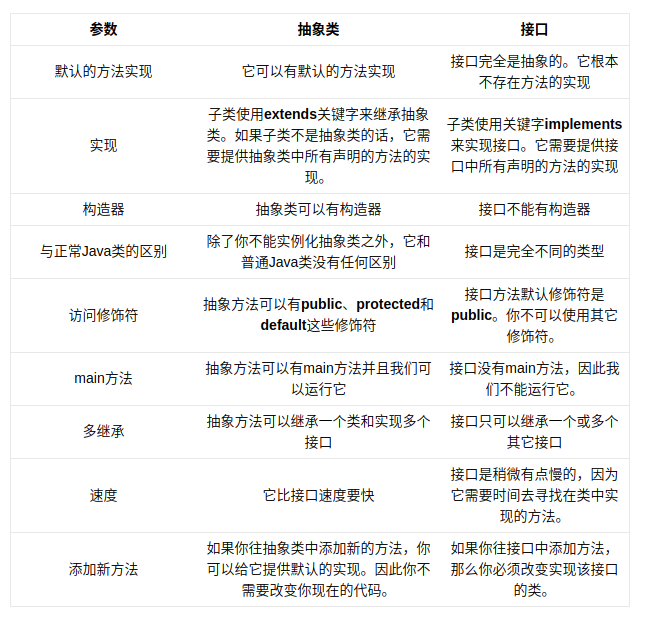
　　3）接口中不能含有静态代码块以及静态方法，而抽象类可以有静态代码块和静态方法；

　　4）一个类只能继承一个抽象类，而一个类却可以实现多个接口。

设计层面上的区别

1）抽象类是对一种事物的抽象，即对类抽象，而接口是对行为的抽象。

2）设计层面不同，抽象类作为很多子类的父类，它是一种模板式设计。而接口是一种行为规范，它是一种辐射式设计。



### 说说反射的用途及实现

反射机制提供的功能主要有：获取类字段、获取类方法、获取对应的实例构造器，并生成类实例、通过newInstance()方法生成类实例、设置私有变量、获取私有变量。

#### 获取一个对象对应的反射类，在Java中有三种方法可以获取一个对象的反射类

通过getClass()方法

通过Class.forName()方法；

使用类.class

通过类加载器实现，getClassLoader()

### 说说自定义注解的场景及实现

#### 定义注解格式

public @interface 注解名 {定义体}

#### 元注解（meta-annotation）

@Target(ElementType.TYPE) :接口、类、枚举、注解

@Target(ElementType.FIELD) :字段、枚举的常量

@Target(ElementType.METHOD) :方法

@Target(ElementType.PARAMETER) :方法参数

@Target(ElementType.CONSTRUCTOR) :构造函数

@Target(ElementType.LOCAL\_VARIABLE) :局部变量

@Target(ElementType.ANNOTATION\_TYPE):注解

@Target(ElementType.PACKAGE) :包

@Retention(RetentionPolicy.SOURCE) :在源文件中有效（即源文件保留）

@Retention(RetentionPolicy.CLASS) :在class文件中有效（即class保留）

@Retention(RetentionPolicy.RUNTIME) :在运行时有效（即运行时保留）

@Inherited：说明子类可以继承父类中的该注解

@Document：说明该注解将被包含在javadoc

#### 示例

package com.sinitek.common.db;

import java.lang.annotation.\*;

/\*\*

\* Created by wfwt on 2018/3/14 0014.

\*/

@Retention(RetentionPolicy.RUNTIME)

@Inherited

@Target({ElementType.TYPE, ElementType.METHOD})

public @interface UseDataSource {

String value() default DataSourceConstants.MASTER;

}

### HTTP请求的GET与POST方式的区别

#### GET方法

使用GET方法时，查询字符串（键值对）被附加在URL地址后面一起发送到服务器：/test/demo\_form.jsp?name1=value1&name2=value2

特点：

GET请求能够被缓存

GET请求会保存在浏览器的浏览记录中

以GET请求的URL能够保存为浏览器书签

GET请求有长度限制

·GET请求主要用以获取数据

#### POST方法

使用POST方法时，查询字符串在POST信息中单独存在，和HTTP请求一起发送到服务器：

POST/test/demo\_form.jsp HTTP/1.1

Host:w3schools.com

name1=value1&name2=value2

特点：

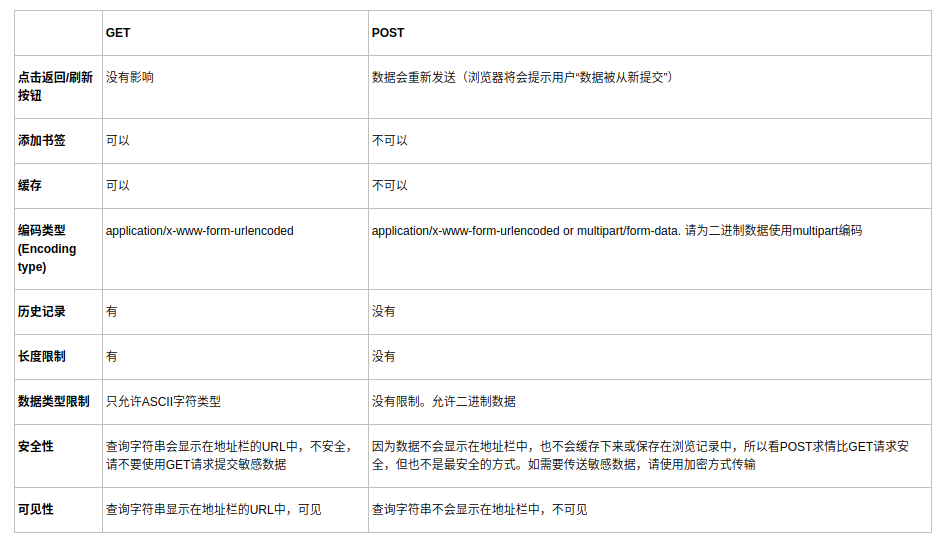
POST请求不能被缓存下来

POST请求不会保存在浏览器浏览记录中

以POST请求的URL无法保存为浏览器书签

POST请求没有长度限制

#### GET和POST的区别



#### Java HTTP get/post

<https://www.cnblogs.com/xjh713/p/6075844.html>

### Session与Cookie区别

#### session的原理

http是无状态的协议，客户每次读取web页面时，服务器都打开新的会话，而且服务器也不会自动维护客户的上下文信息。

session就是一种保存上下文信息的机制，她是针对每一个用户的，session的内容在服务器端，通过sessionId来区分不同的客户，session是以cookie或url重写为基础的，默认用cookie来实现，系统会创造一个JSESSIONID的输出cookie，我们成为session cookie，以区分persistent coookies，注意session cookie是存储于浏览器内存中的，并不是写到硬盘上的；我们通常是看不见JSESSIONID的，但是当我们禁用浏览器的cookie后，web服务器会采用url重写的方式传递sessionid，我们就可以在浏览器看到sessionid=HJHADKSFHKAJSHFJ之类的字符串；session cookie针对某一次会话而言，会话结束session cookie也就消失了。

#### session与cookie的区别

session保存在服务器，客户端不知道其中的信息；cookie保存在客户端，服务端可以知道其中的信息

session中保存的是对象，cookie中保存的是字符串

session不能区分路径，同一个用户在访问一个网站期间，所有的session在任何一个地方都可以访问到；而cookie中如果设置了路径参数，那么同一个网站中不同路径下的cookie互相是访问不到的。

### 列出自己常用的JDK包

### MVC设计思想

model view controller

1)模型: 业务逻辑包含了业务数据的加工与处理以及相应的基础服务(为了保证业务逻辑能够正常进行的事务、安全、权限、日志等等的功能模块)

2)视图:展现模型处理的结果；另外，还要提供相应的操作界面，方便用户使用。

3)控制器:视图发请求给控制器，由控制器来选择相应的模型来处理；模型返回的结果给控制器，由控制器选择合适的视图。

### equals与==的区别

"=="除了比较基本数据之外都是比较的内存地址

"equals"除了没有没有重写equals方法的类之外都是比较的内容（没有重写equals方法的类都是调用的Object的equals的方法，类似＝＝）

### hashCode和equals方法的区别与联系

#### 为什么覆盖(不是重载)equals方法一定要覆盖hashCode方法

因为如果只覆盖了equals而没有覆盖hashCode, 则两个不同的instance a和b虽然equals结果(业务逻辑上)相等,但却会有不同的hashcode,这样hashmap里面会同时存在a和b,而实际上我们需要hashmap里面只能保存其中一个,因为从业务逻辑方向看它们是相等的.

### 什么是Java序列化和反序列化，如何实现Java序列化？或者请解释Serializable 接口的作用

#### Java 序列化和反序列化

把JAVA对象转换为字节序列的过程就称为对象的序列化，将字节序列恢复成Java对象的过程称为对象的反序列化。

ObjectOutputStream类：它继承OutputStream类并且实现ObjectOutput接口。利用该类来实现将对象存储（调用ObjectOutput接口中的writeObject()方法）。ObjectInputStream类：它继承InputStream类并且实现ObjectInput接口。利用该类来实现读取一个对象（调用ObjectInput接口中的readObject()方法）。

#### Serializable和Externalizable

只有实现了 serializable和Externalizable接口的类的对象才能被序列化 后者是前者的子类 实现这个借口的类完全由自身来控制序列化的行为，而仅仅实现前者的类可以采用默认的序列化方式。

如果某个类能够被串行化，其子类也可以被串行化。如果该类有父类，则分两种情况来考虑，如果该父类已经实现了可串行化接口。则其父类的相应字段及属性的处理和该类相同；如果该类的父类没有实现可串行化接口，则该类的父类所有的字段属性将不会串行化。

声明为static和transient类型的成员数据不能被串行化。因为static代表类的状态， transient代表对象的临时数据；

### Object类中常见的方法，为什么wait、notify会放在Object里边?

public final native Class<?> getClass();

public native int hashCode();

public boolean equals(Object obj) { return (this == obj); }

protected native Object clone() throws CloneNotSupportedException;

public String toString() {return getClass().getName() + "@" + Integer.toHexString(hashCode());}

public final native void notify();

public final native void notifyAll();

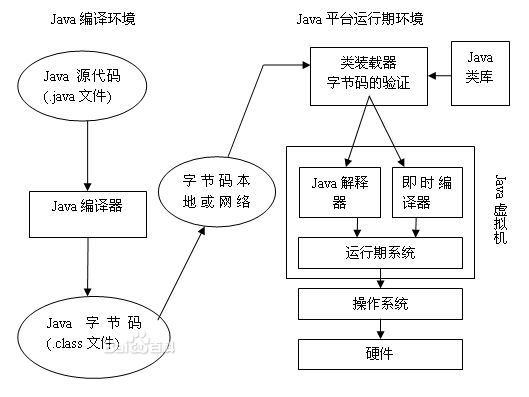
public final native void wait(long timeout) throws InterruptedException;

public final void wait(long timeout, int nanos) throws InterruptedException;

public final void wait() throws InterruptedException { wait(0);}

protected void finalize() throws Throwable { }

### Java的平台无关性如何体现出来的



### JDK和JRE的区别

JDK是Java Development Kit的缩写，是Java的开发工具包。

JRE是Java Runtime Environment的缩写，是Java程序的运行环境。

JDK和JRE是Java开发和运行工具，其中JDK包含了JRE，但是JRE是可以独立安装的。

JDK是Java的开发工具，它不仅提供了Java程序运行所需的JRE，还提供了一系列的编译，运行等工具，如javac，java，javaw等。JRE只是Java程序的运行环境，它最核心的内容就是JVM（Java虚拟机）及核心类库。

### Java 8有哪些新特性

<http://www.importnew.com/11908.html>

#### Lambda表达式与Functional接口

一个lambda可以由用逗号分隔的参数列表、–>符号与函数体三部分表示。

e -> System.out.println( e )

( String e ) -> System.out.println( e )

Lambda可以引用类的成员变量与局部变量（如果这些变量不是final的话，它们会被隐含的转为final，这样效率更高）

函数式接口就是一个具有一个方法的普通接口。像这样的接口，可以被隐式转换为lambda表达式。

默认方法与静态方法并不影响函数式接口的契约，可以任意使用。

#### 接口的默认方法与静态方法

Public interface Defaulable {

default String notRequired() {

return "Default implementation";

}

static Defaulable create( Supplier< Defaulable > supplier ) {

return supplier.get();

}

}

#### 方法引用

##### 构造器引用

它的语法是Class::new，或者更一般的Class< T >::new。请注意构造器没有参数。

##### 静态方法引用

它的语法是Class::static\_method。请注意这个方法接受一个Car类型的参数。

##### 特定类的任意对象的方法引用

它的语法是Class::method。请注意，这个方法没有参数。

##### 特定对象的方法引用

它的语法是instance::method。请注意，这个方法接受一个Car类型的参数

#### 重复注解

@Target( ElementType.TYPE )

    @Retention( RetentionPolicy.RUNTIME )

    public @interface Filters {

        Filter[] value();

    }

    @Target( ElementType.TYPE )

    @Retention( RetentionPolicy.RUNTIME )

    @Repeatable( Filters.class )

    public @interface Filter {

        String value();

    };

    @Filter( "filter1" )

    @Filter( "filter2" )

    public interface Filterable {

}

反射相关的API提供了新的函数getAnnotationsByType()来返回重复注解的类型（请注意Filterable.class.getAnnotation( Filters.class )经编译器处理后将会返回Filters的实例）。

#### 更好的类型推测机制

#### 扩展注解的支持

ElementType.TYPE\_USE和ElementType.TYPE\_PARAMETER是两个新添加的用于描述适当的注解上下文的元素类型。

#### Java编译器的新特性

##### 参数名字

–parameters参数来编译类

#### Java 类库的新特性

##### Optional

##### Stream

##### Date/Time API (JSR 310)

##### JavaScript引擎Nashorn

##### Base64

##### 并行（parallel）数组

#### 新的Java工具

##### Nashorn引擎: jjs

##### 类依赖分析器jdeps

## Java常见集合

### List 和 Set 区别

Iterable<E> <--- Collection<E> <--- List<E>

Iterable<E> <--- Collection<E> <--- Set<E>

### Set和hashCode以及equals方法的联系

当equals()方法被override时，hashCode()也要被override。按照一般hashCode()方法的实现来说，相等的对象，它们的hash code一定相等。

相等（相同）的对象必须具有相等的哈希码（或者散列码）。

如果两个对象的hashCode相同，它们并不一定相同。

### List 和 Map 区别

interface Map<K,V>

### Arraylist 与 LinkedList 区别

Object <--- AbstractCollection<E> <--- AbstractList<E> <--- ArrayList<E>

AbstractList<E> <--- AbstractSequentialList<E> <--- LinkedList<E>

### ArrayList 与 Vector 区别

Object <--- AbstractCollection<E> <--- AbstractList<E> <--- ArrayList<E>

Object <--- AbstractCollection<E> <--- AbstractList<E> <--- Vector<E>

Vector是线程（Thread）同步（Synchronized）的，所以它也是线程安全的，而Arraylist是线程异步（ASynchronized）的，是不安全的。

ArrayList默认大小是10，Vector默认大小是10。

ArrayList调整大小：int newCapacity = oldCapacity + (oldCapacity >> 1);

Vector调整大小：int newCapacity = oldCapacity + ((capacityIncrement > 0) ? capacityIncrement : oldCapacity);

### HashMap 和 Hashtable 的区别

AbstractMap<K,V> <--- HashMap<K,V>

Dictionary<K,V> <--- Hashtable<K,V>

HashMap允许将null作为一个entry的key或者value，而Hashtable不允许。

HashMap是非synchronized，而Hashtable是synchronized，这意味着Hashtable是线程安全的。

HashMap的迭代器(Iterator)是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。（如果在使用迭代器的过程中有其他线程修改了map，那么将抛出ConcurrentModificationException，这就是所谓fail-fast策略。）

HashMap把Hashtable的contains方法去掉了，改成containsvalue和containsKey。因为contains方法容易让人引起误解。

Hashtable中数组默认大小是11，增加的方式是int newCapacity = (oldCapacity << 1) + 1;。HashMap中hash数组的默认大小是16，而且一定是2的指数。

### HashSet 和 HashMap 区别

AbstractMap<K,V> <--- HashMap<K,V>

AbstractCollection<E><--- AbstractSet<E> <--- HashSet<E>

HashSet底层采用的是HashMap进行实现的

### HashMap 和 ConcurrentHashMap 的区别

ConcurrentHashMap将数据分为多个segment，默认16个（concurrency level），然后每次操作对一个segment加锁，避免多线程锁的几率，提高并发效率。

从ConcurrentHashMap代码中可以看出，它引入了一个“分段锁”的概念，具体可以理解为把一个大的Map拆分成N个小的HashTable，根据key.hashCode()来决定把key放到哪个HashTable中。

在ConcurrentHashMap中，就是把Map分成了N个Segment，put和get的时候，都是现根据key.hashCode()算出放到哪个Segment中。

ConcurrentHashMap不允许Null键。

### HashMap 的工作原理及代码实现，什么时候用到红黑树

初始大小为16，装填因子为0.75，则扩容阈值为12。

当链表大小>=8时，则变成红黑树；如果数组长度<64，则直接扩容2倍数组，不变红黑树。

### 多线程情况下HashMap死循环的问题

HashMap出现Hash DOS攻击的问题

ConcurrentHashMap 的工作原理及代码实现，如何统计所有的元素个数

手写简单的HashMap

看过那些Java集合类的源码

1.3、进程和线程

线程和进程的概念、并行和并发的概念

创建线程的方式及实现

进程间通信的方式

说说 CountDownLatch、CyclicBarrier 原理和区别

说说 Semaphore 原理

说说 Exchanger 原理

ThreadLocal 原理分析，ThreadLocal为什么会出现OOM，出现的深层次原理

讲讲线程池的实现原理

线程池的几种实现方式

线程的生命周期，状态是如何转移的

可参考：《Java多线程编程核心技术》

1.4、锁机制

说说线程安全问题，什么是线程安全，如何保证线程安全

重入锁的概念，重入锁为什么可以防止死锁

产生死锁的四个条件（互斥、请求与保持、不剥夺、循环等待）

如何检查死锁（通过jConsole检查死锁）

volatile 实现原理（禁止指令重排、刷新内存）

synchronized 实现原理（对象监视器）

synchronized 与 lock 的区别

AQS同步队列

CAS无锁的概念、乐观锁和悲观锁

常见的原子操作类

什么是ABA问题，出现ABA问题JDK是如何解决的

乐观锁的业务场景及实现方式

Java 8并法包下常见的并发类

偏向锁、轻量级锁、重量级锁、自旋锁的概念

可参考：《Java多线程编程核心技术》

1.5、JVM

JVM运行时内存区域划分

内存溢出OOM和堆栈溢出SOE的示例及原因、如何排查与解决

如何判断对象是否可以回收或存活

常见的GC回收算法及其含义

常见的JVM性能监控和故障处理工具类：jps、jstat、jmap、jinfo、jconsole等

JVM如何设置参数

JVM性能调优

类加载器、双亲委派模型、一个类的生命周期、类是如何加载到JVM中的

类加载的过程：加载、验证、准备、解析、初始化

强引用、软引用、弱引用、虚引用

Java内存模型JMM

1.6、设计模式

常见的设计模式

设计模式的的六大原则及其含义

常见的单例模式以及各种实现方式的优缺点，哪一种最好，手写常见的单利模式

设计模式在实际场景中的应用

Spring中用到了哪些设计模式

MyBatis中用到了哪些设计模式

你项目中有使用哪些设计模式

说说常用开源框架中设计模式使用分析

动态代理很重要！！！

1.7、数据结构

树（二叉查找树、平衡二叉树、红黑树、B树、B+树）

深度有限算法、广度优先算法

克鲁斯卡尔算法、普林母算法、迪克拉斯算法

什么是一致性Hash及其原理、Hash环问题

常见的排序算法和查找算法：快排、折半查找、堆排序等

1.8、网络/IO基础

BIO、NIO、AIO的概念

什么是长连接和短连接

Http1.0和2.0相比有什么区别，可参考《Http 2.0》

Https的基本概念

三次握手和四次挥手、为什么挥手需要四次

从游览器中输入URL到页面加载的发生了什么？可参考《从输入URL到页面加载发生了什么》

# 数据存储和消息队列

2.1、数据库

MySQL 索引使用的注意事项

DDL、DML、DCL分别指什么

explain命令

left join，right join，inner join

数据库事物ACID（原子性、一致性、隔离性、持久性）

事物的隔离级别（读未提交、读以提交、可重复读、可序列化读）

脏读、幻读、不可重复读

数据库的几大范式

数据库常见的命令

说说分库与分表设计

分库与分表带来的分布式困境与应对之策（如何解决分布式下的分库分表，全局表？）

说说 SQL 优化之道

MySQL遇到的死锁问题、如何排查与解决

存储引擎的 InnoDB与MyISAM区别，优缺点，使用场景

索引类别（B+树索引、全文索引、哈希索引）、索引的原理

什么是自适应哈希索引（AHI）

为什么要用 B+tree作为MySQL索引的数据结构

聚集索引与非聚集索引的区别

遇到过索引失效的情况没，什么时候可能会出现，如何解决

limit 20000 加载很慢怎么解决

如何选择合适的分布式主键方案

选择合适的数据存储方案

常见的几种分布式ID的设计方案

常见的数据库优化方案，在你的项目中数据库如何进行优化的

2.2、Redis

Redis 有哪些数据类型，可参考《Redis常见的5种不同的数据类型详解》

Redis 内部结构

Redis 使用场景

Redis 持久化机制，可参考《使用快照和AOF将Redis数据持久化到硬盘中》

Redis 集群方案与实现

Redis 为什么是单线程的？

缓存雪崩、缓存穿透、缓存预热、缓存更新、缓存降级

使用缓存的合理性问题

Redis常见的回收策略

2.3、消息队列

消息队列的使用场景

消息的重发补偿解决思路

消息的幂等性解决思路

消息的堆积解决思路

自己如何实现消息队列

如何保证消息的有序性

# 开源框架和容器

3.1、SSM/Servlet

Servlet的生命周期

转发与重定向的区别

BeanFactory 和 ApplicationContext 有什么区别

Spring Bean 的生命周期

Spring IOC 如何实现

Spring中Bean的作用域，默认的是哪一个

说说 Spring AOP、Spring AOP 实现原理

动态代理（CGLib 与 JDK）、优缺点、性能对比、如何选择

Spring 事务实现方式、事务的传播机制、默认的事务类别

Spring 事务底层原理

Spring事务失效（事务嵌套），JDK动态代理给Spring事务埋下的坑，可参考《JDK动态代理给Spring事务埋下的坑！》

如何自定义注解实现功能

Spring MVC 运行流程

Spring MVC 启动流程

Spring 的单例实现原理

Spring 框架中用到了哪些设计模式

Spring 其他产品（Srping Boot、Spring Cloud、Spring Secuirity、Spring Data、Spring AMQP 等）

有没有用到Spring Boot，Spring Boot的认识、原理

MyBatis的原理

可参考《为什么会有Spring》

可参考《为什么会有Spring AOP》

3.2、Netty

为什么选择 Netty

说说业务中，Netty 的使用场景

原生的 NIO 在 JDK 1.7 版本存在 epoll bug

什么是TCP 粘包/拆包

TCP粘包/拆包的解决办法

Netty 线程模型

说说 Netty 的零拷贝

Netty 内部执行流程

Netty 重连实现

3.3、Tomcat

Tomcat的基础架构（Server、Service、Connector、Container）

Tomcat如何加载Servlet的

Pipeline-Valve机制

可参考：《四张图带你了解Tomcat系统架构！》

# 分布式

4.1、Nginx

请解释什么是C10K问题或者知道什么是C10K问题吗？

Nginx简介，可参考《Nginx简介》

正向代理和反向代理.

Nginx几种常见的负载均衡策略

Nginx服务器上的Master和Worker进程分别是什么

使用“反向代理服务器”的优点是什么?

4.2、分布式其他

谈谈业务中使用分布式的场景

Session 分布式方案

Session 分布式处理

分布式锁的应用场景、分布式锁的产生原因、基本概念

分布是锁的常见解决方案

分布式事务的常见解决方案

集群与负载均衡的算法与实现

说说分库与分表设计，可参考《数据库分库分表策略的具体实现方案》

分库与分表带来的分布式困境与应对之策

4.3、Dubbo

什么是Dubbo，可参考《Dubbo入门》

什么是RPC、如何实现RPC、RPC 的实现原理，可参考《基于HTTP的RPC实现》

Dubbo中的SPI是什么概念

Dubbo的基本原理、执行流程

# 微服务

5.1、微服务

前后端分离是如何做的？

微服务哪些框架

Spring Could的常见组件有哪些？可参考《Spring Cloud概述》

领域驱动有了解吗？什么是领域驱动模型？充血模型、贫血模型

JWT有了解吗，什么是JWT，可参考《前后端分离利器之JWT》

你怎么理解 RESTful

说说如何设计一个良好的 API

如何理解 RESTful API 的幂等性

如何保证接口的幂等性

说说 CAP 定理、BASE 理论

怎么考虑数据一致性问题

说说最终一致性的实现方案

微服务的优缺点，可参考《微服务批判》

微服务与 SOA 的区别

如何拆分服务、水平分割、垂直分割

如何应对微服务的链式调用异常

如何快速追踪与定位问题

如何保证微服务的安全、认证

5.2、安全问题

如何防范常见的Web攻击、如何方式SQL注入

服务端通信安全攻防

HTTPS原理剖析、降级攻击、HTTP与HTTPS的对比

5.3、性能优化

性能指标有哪些

如何发现性能瓶颈

性能调优的常见手段

说说你在项目中如何进行性能调优

# 其他

6.1、设计能力

说说你在项目中使用过的UML图

你如何考虑组件化、服务化、系统拆分

秒杀场景如何设计

可参考：《秒杀系统的技术挑战、应对策略以及架构设计总结一二！》

6.2、业务工程

说说你的开发流程、如何进行自动化部署的

你和团队是如何沟通的

你如何进行代码评审

说说你对技术与业务的理解

说说你在项目中遇到感觉最难Bug，是如何解决的

介绍一下工作中的一个你认为最有价值的项目，以及在这个过程中的角色、解决的问题、你觉得你们项目还有哪些不足的地方

6.3、软实力

说说你的优缺点、亮点

说说你最近在看什么书、什么博客、在研究什么新技术、再看那些开源项目的源代码

说说你觉得最有意义的技术书籍

工作之余做什么事情、平时是如何学习的，怎样提升自己的能力

说说个人发展方向方面的思考

说说你认为的服务端开发工程师应该具备哪些能力

说说你认为的架构师是什么样的，架构师主要做什么

如何看待加班的问题