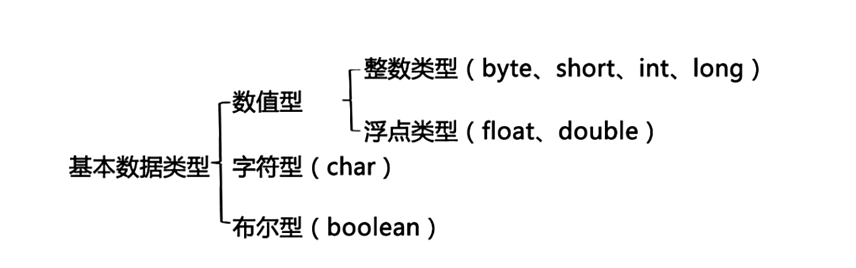
# 1 java基础

## 数据类型（八大数据类型）

Java中的数据类型，分为引用类型和基本数据类型。基本类型又分为8种

### 基本类型（四类八种）



* 整型：byte，short，int,long

long类型的字面量需要使用“L”,"l"后缀，否则编译不通过

* 浮点型：float，double

用于表示小数的数据类型，float类型因为精度过低，所以一般我们更常用double类型。

* 字符型：char

字符类型是一个16位无符号整数，是一个2进制数，这个数值是一个字符的Unicode 编码值

* Boolean型：boolean

表达：真（true）/假（false），一般用于判断语句块中

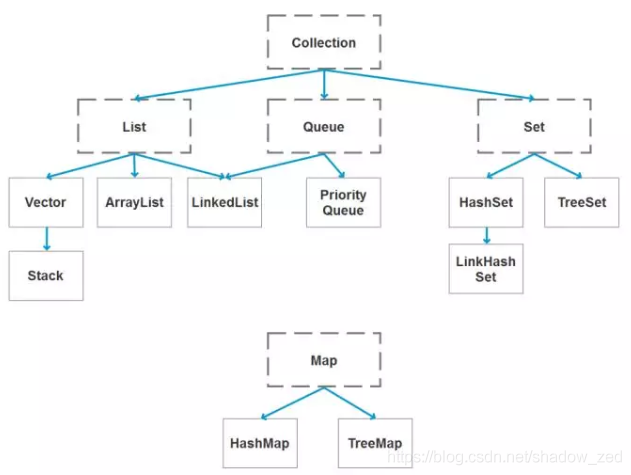
### 引用类型

引用数据类型是建立在基本数据类型的基础上，包括数组、类和接口。引用数据类型是由用户自定义，用来限制其他数据的类型。

## 容器

（list set map ）

### 图





### 文

### Collection

**Collection**（克拉克什）是单列集合类的根接口，用来存储一系列符合某种规则的元素， 它有两个重要的子接口，分别是java.util.List和java.util.Set。

其中，List的特点是元素有序、元素可重复。Set的特点是元素不可重复。

* List接口的主要实现类有java.util.ArrayList和java.util.LinkedList，
* Set接口的主要实现类有java.util.HashSet和java.util.LinkedHashSet。

#### list

1. **ArrayList**：线程不安全，继承自 AbstractList，实现了 List 接口，底层基于数组实现容量大小动态变化, 随机访问速度快， 删除或插入元素速度慢。
2. **LinkedList** ：线程不安全，内部实现是用链表, 随机访问速度慢，删除和插入元素相对较快
3. 平常一般是将ArrayList作为默认首选，只有当需要使用额外的功能，或者因为经常从表中插入或删除元素导致程序性能变差的时候，才去选择LinkedList。
4. **Vector（**线程安全**）**和**Stack**是过去遗留下来的类，目的只是为了支持老程序，应该在编写程序时尽量避免使用它们。

##### Array 和 ArrayList 有何区别？

* Array可以容纳基本类型和对象，而ArrayList只能容纳对象。
* Array是指定大小后不可变的，而ArrayList大小是可变的。
* Array没有提供ArrayList那么多功能，比如addAll、removeAll和iterator等

##### ArrayList 和 Vector 的区别是什么

* Vector是同步的，而ArrayList不是。然而，如果你寻求在迭代的时候对列表进行改变，你应该使用CopyOnWriteArrayList。
* ArrayList比Vector快，它因为有同步，不会过载。
* ArrayList更加通用，因为我们可以使用Collections工具类轻易地获取同步列表和只读列表

##### ArrayList 和 LinkedList 的区别是什么

* 最明显的区别是 ArrayList底层的数据结构是数组，支持随机访问，而 LinkedList 的底层数据结构是双向循环链表，不支持随机访问。
* 使用下标访问一个元素，ArrayList 的时间复杂度是 O(1)，而 LinkedList 是 O(n)。

#### Set

1. **HashSet**: 无序，线程不安全，查询速度快，HashSet底层由HashMap实现，HashSet的值存放于HashMap的key上，而HashMap的value统一为PRESENT（帕Z的），存入HashSet的元素必须定义hashCode()值
2. **LinkedHashSet**: 有序，线程不安全，具有HashSet的查询速度且内部使用链表维护元素的顺序（按插入的顺序或最近最少使用顺序）
3. **TreeSet**：有序，线程不安全，底层为树结构（红黑树）。使用它可以从Set中提取有序的序列。它的元素必须实现Comparable接口或者在构造TreeSet时传入Comparator参数。

#### Map

1. HashMap线程不安全，它是基于散列表的实现（它取代了HashTable,），
2. LinkedHashMap 类似与HashMap， 但是迭代遍历时，取得“键值对”的顺序是其插入的顺序或最近最少使用的顺序(如果在构造LiskedHashMap传入参数accessOrder=true)。比HashMap慢一点，但是在迭代访问时更快，因为它使用链表维护内部的次序。
3. TreeMap 基于红黑树的实现。查看key 或 key-value是，它们会被排序（次序由Comparator或Comparable决定）

##### HashMap 和 Hashtable 有什么区别

1. HashMap是线程非安全的，HashTable是线程安全的所以HashMap的效率高于 HashTable。
2. HashTable之所以是线程安全的，是因为在它的方法上加上了synchrnized锁。
3. hashMap去掉了HashTable 的contains方法，但是加上了containsValue（）和containsKey（）方法。
4. hashTable同步的，而HashMap是非同步的，效率上比hashTable要高。
5. hashMap允许空键值，而hashTable不允许。

##### HashMap的底层原理

1. HashMap底层就是一个数组结构，数组中的每一项又是一个链表
2. 当新建一个HashMap的时候，就会初始化一个数组。
3. Entyr就是数组中的元素，每个Entry其实就是一个Key‐Value对
4. 它持有一个指向下一个元素的引用，这就构成了链表
5. 默认是构建一个初始容量为16，负载因子为0.75的Hashmap。
6. 也就是说，默认情况下，数组大小为16，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，既扩大一倍，然后重新计算每个元素在数组中的位置，而这是一个非常消耗性能的操作，所以如果我们已知HashMap中元素的个数，那么预设元素的个数能够有效的提高HashMap的性能。Jdk1.8引入了红黑树，目的是避免单挑链表过长而影响查询效率。

1. **HashMap在底层将Key‐Value当成一个整体进行处理，这个整体就是一个Entry对象，HashMap底层采用一个Entry[]数组来保存有Key-Value对，当需要存储一个Entry对象时，会根据hash算法来决定其在数组中的存储位置，再根据equals方法决定其在该数组位置上的链表中的存储位置；当需要取出一个Entry时，也会根据hash算法找到其在数组中的存储位置，再根据equuals方法从该位置上的链表中取出该Entry。**

## 包装类

（file，date，string，封装类，exception，stream）

### 基本数据类型包装类

因为Java中的基本数据类型没有方法和属性，而包装类就是为了让这些拥有方法和属性，实现对象化交互。包装类，就能够直接将简单类型的变量表示为一个类。包装类对象的初始值为null（是一个对象）

byte Byte

short Short

int Integer

long Long

float Float

double Double

char Character

boolean Boolean



#### 基本数据和包装类之间的转换

* 装箱：是指将基本数据类型转换为包装类；
* 拆箱：是指将包装类转换为基本数据类型

#### 基本数据类型和包装类的转换

* 通过包装类Integer.toString()将整型转换为字符串；
* 通过Integer.parseInt()将字符串转换为int类型；
* 通过valueOf()方法把字符串转换为包装类然后通过自动拆箱

#### 包装类的用途

1. 集合不允许存放基本数据类型，故常用包装类
2. 包含了每种基本类型的相关属性，如最大值，最小值，所占位数等
3. 作为基本数据类型对应的类类型，提供了一系列实用的对象操作，如类型转换，进制转换等

#### 基本类型与包装类型的区别

1、在Java中，一切皆对象，但八大基本类型却不是对象。

2、声明方式的不同，基本类型无需通过new关键字来创建，而封装类型需new关键字。  
3、存储方式及位置的不同，基本类型是直接存储变量的值保存在堆栈中能高效的存取，封装类型需要通过引用指向实例，具体的实例保存在堆中。  
4、初始值的不同，封装类型的初始值为null，基本类型的的初始值视具体的类型而定，比如int类型的初始值为0，boolean类型为false；  
5、使用方式的不同，比如与集合类合作使用时只能使用包装类型。  
6、什么时候该用包装类，什么时候用基本类型，看基本的业务来定：这个字段允允许null值，就需要使用包装类型，如果不允许null值，，使用基本类型就可以了，用到比如泛型和反射调用函数，就需要用包装类！

### File类

Java File类主要用于文件和目录的创建、文件的查找和文件的删除等。

File对象代表磁盘中实际存在的文件和目录

#### File类的构造方法：

通过给定的父抽象路径名和子路径名字符串创建一个新的File实例。

通过指定的路径名 字符串转换成抽象路径名来创建一个新 File 实例。

通过指定的file:URi转换成一个抽象路径名来创建一个新的 File 实例。

#### File类的操作方法

public String getName()；返回由此抽象路径名表示的文件或目录的名称。

public String getParent()；返回此抽象路径名的父路径名的路径名字符串，如果此路径名没有指定父目录，则返回 null

public String getPath()；将此抽象路径名转换为一个路径名字符串。

public String getAbsolutePath()；返回抽象路径名的绝对路径名字符串

public boolean delete()； 删除此抽象路径名表示的文件或目录。

### Date类

#### Date类对象的创建以及获取所包含的信息

**1、创建一个当前时间的Date对象**

  Date date= new Date();

**2、创建一个我们指定的时间的Date对象：**

使用带参数的构造方法Date(int year, int month, int day) ，可以构造指定日期的Date类对象，Date类中年份的参数应该是实际需要代表的年份减去1900，实际需要代表的月份减去1以后的值。

Date date= new Date(2014-1900, 6-1, 12); （注意参数的设置）

**3、获得一个date对象所包含的信息**

Date date= new Date(2014-1900, 6-1, 12); （注意参数的设置）

获得年份（注意年份要加上1900）

int year = date.getYear() + 1900;

获得月份（注意月份要加1）

int month = date.getMonth() + 1;

//获得日期

int date = date.getDate();

//获得小时

int hour = date.getHours();//不设置默认为0

//获得分钟

int minute = date.getMinutes();

//获得秒

int second = date.getSeconds();

//获得星期 （注意：0代表星期日、1代表星期1、2代表星期2,,,,）

int day = date.getDay();

#### Date类对象与long型时间的互转

1.将Date类的对象转换为long型时间

使用对象的getTime（）方法完成

Date d= new Date();

long dLong = d.getTime();

2.将long型时间转换为Date类的对象

使用Date的构造方法完成

long time = 1290876532190L;

Date d2 = new Date(time);

#### Date类对象和String对象之间的转换

1.Date 转化String

SimpleDateFormat sdf= new SimpleDateFormat("yyyy-MM-dd");

String dateStr=sdf.format(new Date());

2.String 转化Date

String str="2010-5-27";

SimpleDateFormat sdf= new SimpleDateFormat("yyyy-MM-dd");

Date birthday = sdf.parse(str);

### String

String 创建的字符串存储在公共池中，而 new 创建的字符串对象在堆上

String 类是不可改变的，所以你一旦创建了 String 对象，那它的值就无法改变了

如果需要对字符串做很多修改，那么应该选择使用 [StringBuffer & StringBuilder 类](https://www.runoob.com/java/java-stringbuffer.html" \t "https://www.runoob.com/java/_blank" \o "StringBuffer & StringBuilder 类)

#### String类连接字符串

String 类提供了连接两个字符串的方法

1. string1.concat(string2);返回 string2 连接 string1 的新字符串。也可以对字符串常量使用 concat() 方法
2. 更常用的是使用'+'操作符来连接字符串，“string1”+“string2”

#### String类创建格式化字符串

String 类使用静态方法 format() 返回一个String 对象而不是 PrintStream 对象。

String 类的静态方法 format() 能用来创建可复用的格式化字符串，而不仅仅是用于一次打印输出。

#### String类的常用方法

将此字符串与指定的对象比较

[boolean equals(Object anObject)](https://www.runoob.com/java/java-string-equals.html)  
将此String与另一个String比较，不考虑大小写

[boolean equalsIgnoreCase(String anotherString)](https://www.runoob.com/java/java-string-equalsignorecase.html)  
返回此字符串的哈希码

i[nt hashCode()](https://www.runoob.com/java/java-string-hashcode.html)  
返回指定字符在此字符串中第一次出现处的索引

[int indexOf(int ch)](https://www.runoob.com/java/java-string-indexof.html)  
返回字符串对象的规范化表示形式

[String intern()](https://www.runoob.com/java/java-string-intern.html)

返回指定字符在此字符串中最后一次出现处的索引

int lastIndexOf(int ch)  
返回此字符串的长度。

Int Ienfth

### Exception

#### 什么是异常

异常就是在运行时产生的问题。通常用Exception描述。

在java中，把异常封装成了一个类，当出现问题时，就会创建异常类对象并抛出异常相关的信息（如详细信息，名称以及异常所处的位置）。

#### 异常的继承关系

* Throwable类是所有错误跟异常类的超类（祖宗类）。
* Exception异常类及其子类都是继承自Throwable类，用来表示java中可能出现的异常，并且合理的处理这些异常。
* RuntimeException类是运行异常类，继承自Exception类，它以及它的子类只能在运行过程中存在，当出现时，只能修改源代码，此异常无法处理。
* Error类是与Exception的平级的类，用来表示Java中存在的严重错误，只能通过修改代码来解决问题。

#### 异常与错误的区别

* 异常是指程序在编译或者运行时出现的某种异常问题，我们可以对异常进行某种处理，如果不处理异常的话，程序将会停止运行。
* 错误是指程序在运行时出现的严重问题，无法处理，程序将会停止运行，Error通常都是系统级别的问题，都是虚拟机jvm所在系统发生的，只能通过修改源代码解决问题。

#### 异常产生的过程

 1.运行或编译时产生异常

2.创建异常类的对象

3.声明异常类

4.将异常类对象传给调用者（main()方法）处理

5.调用者无法处理，再将异常类对象传给jvm虚拟机

6.jvm虚拟机将异常类的信息（名称、详细信息、异常所处的位置）打印在屏幕上，并且停止程序的运行

#### 声明异常throws

 声明异常格式：

修饰符 返回值类型 方法名称 （参数）throws 异常1名称，异常2名称{

}

#### 抛出异常throw

 在java中，提供了一个throw关键字，用来抛出一个指定的异常。

 使用方法：

1.创建一个异常对象。封装一些提示信息

2.将这个异常告知调用者

 使用格式：

throw new 异常类名（参数）；

#### 捕获异常try...catch...finally

捕获异常：Java中对异常有针对性的语句进行捕获，可以对出现的异常进行指定方式的处理

try：该代码块中编写可能产生异常的代码。

catch：用来进行某种异常的捕获，实现对捕获到的异常进行处理。

finally：有一些特定的代码无论异常是否发生，都需要执行。另外，因为异常会引发程序跳转，导致有些语句执行不到。而finally就是解决这个问题的，在finally代码块中存放的代码都是一定会被执行的。

##### 捕获异常的组合方式：

 1.  try catch finally组合：检测异常，并传递给catch处理，并在finally中进行资源释放。

 2.  try catch组合 : 对代码进行异常检测，并对检测的异常传递给catch处理。对异常进行捕获处理。

 3.  一个try 多个catch组合 : 对代码进行异常检测，并对检测的异常传递给catch处理。对每种异常信息进行不同的捕获处理。

Ps：这种异常处理方式，要求多个catch中的异常不能相同，并且若catch中的多个异常之间有子父类异常的关系，那么子类异常要求在上面的catch处理，父类异常在下面的catch处理。

4. try finally 组合: 对代码进行异常检测，检测到异常后因为没有catch，所以一样会被默认jvm抛出。异常是没有捕获处理的。但是功能所开启资源需要进行关闭，所有finally。只为关闭资源。

#### 运行时期异常 RuntimeExecption

特点：

　　　方法中抛出运行时期异常,方法定义中无需throws声明,调用者也无需处理此异常

　　　 运行时期异常一旦发生,需要程序人员修改源代码

### Stream

#### 为什么需要 Stream

Java 8 中的 Stream 是对集合（Collection）对象功能的增强，它专注于对集合对象进行各种非常便利、高效的聚合操作（aggregate operation），或者大批量数据操作 (bulk data operation)。Stream API 借助于同样新出现的 Lambda 表达式，极大的提高编程效率和程序可读性。同时它提供串行和并行两种模式进行汇聚操作，并发模式能够充分利用多核处理器的优势，使用 fork/join 并行方式来拆分任务和加速处理过程。通常编写并行代码很难而且容易出错, 但使用 Stream API 无需编写一行多线程的代码，就可以很方便地写出高性能的并发程序。所以说，Java 8 中首次出现的 java.util.stream 是一个函数式语言+多核时代综合影响的产物。

#### 什么是流

Stream 不是集合元素，它不是数据结构并不保存数据，它是有关算法和计算的，它更像一个高级版本的 Iterator。原始版本的 Iterator，用户只能显式地一个一个遍历元素并对其执行某些操作；高级版本的 Stream，用户只要给出需要对其包含的元素执行什么操作，比如 “过滤掉长度大于 10 的字符串”、“获取每个字符串的首字母”等，Stream 会隐式地在内部进行遍历，做出相应的数据转换。

Stream 就如同一个迭代器（Iterator），单向，不可往复，数据只能遍历一次，遍历过一次后即用尽了，就好比流水从面前流过，一去不复返。

而和迭代器又不同的是，Stream 可以并行化操作，迭代器只能命令式地、串行化操作。顾名思义，当使用串行方式去遍历时，每个 item 读完后再读下一个 item。而使用并行去遍历时，数据会被分成多个段，其中每一个都在不同的线程中处理，然后将结果一起输出。

#### 流的构成

当我们使用一个流的时候，通常包括三个基本步骤：

获取一个数据源（source）→ 数据转换→执行操作获取想要的结果，每次转换原有 Stream 对象不改变，返回一个新的 Stream 对象（可以有多次转换），这就允许对其操作可以像链条一样排列，变成一个管道

#### 流的操作类型分为两种

* **Intermediate**：一个流可以后面跟随零个或多个 intermediate 操作。其目的主要是打开流，做出某种程度的数据映射/过滤，然后返回一个新的流，交给下一个操作使用。这类操作都是惰性化的（lazy），就是说，仅仅调用到这类方法，并没有真正开始流的遍历。
* **Terminal**：一个流只能有一个 terminal 操作，当这个操作执行后，流就被使用“光”了，无法再被操作。所以这必定是流的最后一个操作。Terminal 操作的执行，才会真正开始流的遍历，并且会生成一个结果，或者一个 side effect。

#### 流的使用详解

简单说，对 Stream 的使用就是实现一个 filter-map-reduce 过程，产生一个最终结果，或者导致一个副作用（side effect）

#### 流的操作

当把一个数据结构包装成 Stream 后，就要开始对里面的元素进行各类操作了。常见的操作可以归类如下。

**Intermediate：中间**

map (mapToInt, flatMap 等)、 filter、 distinct、 sorted、 peek、 limit、 skip、 parallel、 sequential、 unordered

**Terminal：结束**

forEach、 forEachOrdered、 toArray、 reduce、 collect、 min、 max、 count、 anyMatch、 allMatch、 noneMatch、 findFirst、 findAny、 iterator

**Short-circuiting：环绕**

anyMatch、 allMatch、 noneMatch、 findFirst、 findAny、 limit

#### Stream 的比较典型用法

* Map：它的作用就是把 input Stream 的每一个元素，映射成 output Stream 的另外一个元素。
* flatMap： 把 input Stream 中的层级结构扁平化，就是将最底层元素抽出来放到一起，最终 output 的新 Stream 里面已经没有 List 了，都是直接的数字。
* forEach： 方法接收一个 Lambda 表达式，然后在 Stream 的每一个元素上执行该表达式。

## 控制语句

（if for while foreach 迭代器）

### If

#### if 语句

if 语句会判断括号中的条件是否成立，如果成立则执行 if 语句中的代码块，否则跳过代码块继续执行。

if(布尔表达式) {

//如果布尔表达式为true将执行的语句

}

#### if...else 语句

if 语句后面可以跟 else 语句，当 if 语句的布尔表达式值为 false 时，else 语句块会被执行。

if(布尔表达式) {

//如果布尔表达式的值为true

} else {

//如果布尔表达式的值为false

}

#### if...else if...else 语句

* if 语句至多有 1 个 else 语句，else 语句在所有的 else if 语句之后。
* If 语句可以有若干个 else if 语句，它们必须在 else 语句之前。
* 一旦其中一个 else if 语句检测为 true，其他的 else if 以及 else 语句都将跳过执行。

if (布尔表达式 1) {

//如果布尔表达式 1的值为true执行代码

} else if (布尔表达式 2) {

//如果布尔表达式 2的值为true执行代码

} else if (布尔表达式 3) {

//如果布尔表达式 3的值为true执行代码

} else {

//如果以上布尔表达式都不为true执行代码

}

#### 嵌套的 if…else 语句

使用嵌套的 if else 语句是合法的。也就是说你可以在另一个 if 或者 else if 语句中使用 if 或者 else if 语句。

if (布尔表达式 1) {

////如果布尔表达式 1的值为true执行代码

if (布尔表达式 2) {

////如果布尔表达式 2的值为true执行代码

}

}

### for

#### for 循环

虽然所有循环结构都可以用 while 或者 do while 表示，但 Java 提供了另一种语句 —— for 循环，使一些循环结构变得更加简单。  
for 循环执行的次数是在执行前就确定的。

for (初始化; 布尔表达式; 更新) {

//代码语句

}

1. 最先执行初始化步骤。可以声明一种类型，但可初始化一个或多个循环控制变量，也可以是空语句。
2. 然后，检测布尔表达式的值。如果为 true，循环体被执行。如果为 false，循环终止，开始执行循环体后面的语句。
3. 执行一次循环后，更新循环控制变量。
4. 再次检测布尔表达式。循环执行上面的过程。

#### foreach 循环

Java5 引入了一种主要用于数组的增强型 for 循环。

for (声明语句 : 表达式) {

//代码句子

}

声明语句：声明新的局部变量，该变量的类型必须和数组元素的类型匹配。其作用域限定在循环语句块，其值与此时数组元素的值相等。

表达式：表达式是要访问的数组名，或者是返回值为数组的方法。

### While

#### while 循环

只要布尔表达式为 true，while 循环体会一直执行下去。

语法

while( 布尔表达式 ) {

//循环内容

}

#### do while 循环

对于 while 语句而言，如果不满足条件，则不能进入循环。但有时候我们需要即使不满足条件，也至少执行一次。

do while 循环和 while 循环相似，不同的是，do while 循环至少会执行一次。

语法

do {

//代码语句

} while (布尔表达式);

### 关键字

#### break 关键字

break 主要用在循环语句或者 switch 语句中，用来跳出整个语句块。

break 跳出最里层的循环，并且继续执行该循环下面的语句。

#### continue 关键字

continue 适用于任何循环控制结构中。作用是让程序立刻跳转到下一次循环的迭代。在 for 循环中，continue 语句使程序立即跳转到更新语句。在 while 或者 do while 循环中，程序立即跳转到布尔表达式的判断语句。

#### return 关键字

跳出整个函数体，函数体后面的部分不再执行。

### 迭代器

Java Iterator（迭代器）不是一个集合，它是一种用于访问集合的方法，可用于迭代 [ArrayList](https://www.runoob.com/java/java-arraylist.html" \t "https://www.runoob.com/java/_blank) 和 [HashSet](https://www.runoob.com/java/java-hashset.html) 等集合。

Iterator 是 Java 迭代器最简单的实现，ListIterator 是 Collection API 中的接口， 它扩展了 Iterator 接口。

#### 迭代器 it 的两个基本操作

迭代器 it 的两个基本操作是 next 、hasNext 和 remove。

* 调用 it.next() 会返回迭代器的下一个元素，并且更新迭代器的状态。
* 调用 it.hasNext() 用于检测集合中是否还有元素。
* 调用 it.remove() 将迭代器返回的元素删除。

#### 获取一个迭代器

ArrayList<String> sites = new ArrayList<String>();  
// 获取迭代器  
Iterator<String> it = sites.iterator();  
// 输出集合中的第一个元素  
System.out.println(it.next());

#### 循环集合元素

让迭代器 it 逐个返回集合中所有元素最简单的方法是使用 while 循环：

while(it.hasNext()) {

System.out.println(it.next());

}

#### 删除元素

要删除集合中的元素可以使用 remove() 方法。

 while(it.hasNext()) {  
            Integer i = it.next();  
            if(i < 10) {    
                it.remove();  // 删除小于 10 的元素  
            }  
        }

## 反射

java反射机制就是在运行过程中，对于类来说，咱们可以获取它的属性和方法，对于对象来说，可以动态调用它的任意方法和属性。动态获取信息 以及 动态调用对象的方法 就是反射机制！！！

想要用反射首先得获取类，获取类的方式我知道的有这几种，

* 可以通过对象的getClass()方法获取类，
* 可以通过类名.class 获取类。
* 可以通过Class.forName("类的全称")来获取类，

之后可以通过 getDeclaredFields()获取类的所有属性，getDeclaredField(属性名)获取类中指定 名字的 属性,getDeclaredMethods()获取类的所有方法，获取属性后还可以通过getType()获取属性的类型，通过getName()获取属性名。

动态调用方法这块如果是get方法可以使用 属性.get(对象) 来调用get方法，如果是set方法可以通过 属性.set(对象，值)来调用，但因为get,set方法一般都是私有的，所以得先通过 属性.setAccessible(true) 来开启访问权限。如果是普通的方法可以通过 方法.invoke(对象，参数) 来调用。

### Java反射的定义

Java反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意方法和属性；这种动态获取信息以及动态调用对象方法的功能称为java语言的反射机制。

### 用途

在日常的第三方应用开发过程中，经常会遇到某个类的某个成员变量、方法或是属性是私有的或是只对系统应用开放，这时候就可以利用Java的反射机制通过反射来获取所需的私有成员或是方法。当然，也不是所有的都适合反射，之前就遇到一个案例，通过反射得到的结果与预期不符。阅读源码发现，经过层层调用后在最终返回结果的地方对应用的权限进行了校验，对于没有权限的应用返回值是没有意义的缺省值，否则返回实际值起到保护用户的隐私目的。

### 获得类相关的方法

1. asSubclass(Class<U> clazz) 把传递的类的对象转换成代表其子类的对象 Cast 把对象转换成代表类或是接口的对象
2. getClassLoader() 获得类的加载器
3. getClasses() 返回一个数组，数组中包含该类中所有公共类和接口类的对象 getDeclaredClasses() 返回一个数组，数组中包含该类中所有类和接口类的对象
4. forName(String className) 根据类名返回类的对象
5. getName() 获得类的完整路径名字
6. newInstance() 创建类的实例
7. getPackage() 获得类的包
8. getSimpleName() 获得类的名字
9. getSuperclass() 获得当前类继承的父类的名字
10. getInterfaces() 获得当前类实现的类或是接口

### 获得类中属性相关的方法

1. getField(String name) 获得某个公有的属性对象
2. getFields() 获得所有公有的属性对象
3. getDeclaredField(String name) 获得某个属性对象
4. getDeclaredFields() 获得所有属性对象

### 获得类中注解相关的方法

1. getAnnotation(Class<A> annotationClass) 返回该类中与参数类型匹配的公有注解对象
2. getAnnotations() 返回该类所有的公有注解对象
3. getDeclaredAnnotation(Class<A> annotationClass) 返回该类中与参数类型匹配的所有注解对象
4. getDeclaredAnnotations() 返回该类所有的注解对象

### 获得类中方法相关的方法

1. getMethod(String name, Class...<?> parameterTypes) 获得该类某个公有的方法
2. getMethods() 获得该类所有公有的方法
3. getDeclaredMethod(String name, Class...<?> parameterTypes) 获得该类某个方法
4. getDeclaredMethods() 获得该类所有方法

### 如果源码中明确进行了权限验证，而你的应用又无法获得这个权限的话，建议就不要浪费时间反射了。

## 注解

Annotation

### 注解的理解

可以看作是对 一个 类／方法的一个扩展的模版，它可以加在类上，方法上，成员变量上，参数上。

注解本质是一个继承了Annotation的特殊接口，其具体实现类是Java运行时生成的动态代理类，我们通过反射获取注解，他有四种元注解： Java 8 又加入了 @Repeatable

1. @Documented (地思考门特)是一个简单的标记注解，表示是否将注解信息添加在 Java 文档
2. @Retention (瑞特深)定义了该注解的生命周期。当 @Retention 应用到一个注解上的时候，作用就是说明这个注解的存活时间
3. @Target (他给特)表示该注解用于什么地方，可以理解为：当一个注解被 @Target 注解时，这个注解就被限定了运用的场景
4. @Inherited 是指继承，@Inherited 定义了一个注释与子类的关系。如果一个超类带有@Inherited 注解，那么对于该超类，它的子类如果没有被任何注解应用的话，那么这个子类就继承了超类的注解
5. @Repeatable是 Java 8 中加入的，是指可重复的意思。通常使用 @Repeatable 的时候指注解的值可以同时取多个

其中@Retention定义了注解的生命周期他有三种配置，

1.RetentionPolicy.SOURCE : 在编译阶段丢弃。这些注解在编译结束之后就不再有任何意义，例如@Override

2. RetentionPolicy.CLASS : 在类加载的时候丢弃。在字节码文件的处理中有用。注解默认使用这种方式

3.RetentionPolicy.RUNTIME : 始终不会丢弃，运行期也保留该注解，因此可以使用反射机制读取该注解的信息。我们自定义的注解通常使用这种方式。

#### 常见的参数有

●ElementType.FIELD:成员变量、对象、属性

● ElementType.LOCAL\_VARIABLE:用于描述局部变量

● ElementType.METHOD:用于描述方法

● ElementType.PARAMETER:用于描述参数

● ElementType.TYPE:用于描述类、接口(包括注解类型) 或enum声明

### 定义自定注解呢：

1. 新建一个java接口类在interface的关键字上加上@符号，目的是自动继承java.lang.Annotation这一接口,并且不能再去继承别的类或是接口.
2. 在类上加上三个元注解，
3. 然后编写一个参数成员只能用public或默认(default)这两个访问权修饰
4. 要获取类方法和字段的注解信息，必须通过Java的反射技术来获取 Annotation对象,因 为你除此之外没有别的获取注解对象的方法

### 自定义注解类编写的规则：

1. 注解类型定义为 @interface，所有的注解会自动继承 java.lang.Annotation 这一接口，而且不能再去继承其他的类或接口；
2. 参数成员只能用 public 或 default 两个关键字修饰；
3. 参数成员只能用基本类型：byte, short, char, int, long, float, double, boolean，以及 String, Enum, Class, Annotations 等数据类型，以及这些类型的数组；
4. 要获取类方法和字段的注解信息，必须通过 Java 的反射技术；
5. 注解也可以不定义成员变量，但这样的注解没有什么卵用；
6. 自定义注解需要使用元注解进行编写；

## jdk新特性

lamda表达式（接口的新实现方式）

### Lambda表达式

lambda表达式本质上是一段匿名内部类，也可以是一段可以传递的代码

Comparator<Integer> cpt2 = (x,y) -> Integer.compare(x,y);

TreeSet<Integer> set2 = new TreeSet<>(cpt2);

Lmabda表达式的语法总结： () -> ();

前置 语法

无参数无返回值 () -> System.out.println(“Hello WOrld”)

有一个参数无返回值 (x) -> System.out.println(x)

有且只有一个参数无返回值 x -> System.out.println(x)

有多个参数，有返回值，有多条lambda体语句 (x，y) -> {System.out.println(“xxx”);return xxxx;}；

有多个参数，有返回值，只有一条lambda体语句 (x，y) -> xxxx

当一个接口中存在多个抽象方法时，如果使用lambda表达式，并不能智能匹配对应的抽象方法，因此引入了函数式接口的概念

### 函数式接口

函数式接口的提出是为了让我们更加方便的使用lambda表达式，不需要自己再手动创 建一个函数式接口，直接拿来用就好了

#### 什么是函数式接口？

简单来说就是只定义了一个抽象方法的接口（Object类的public方法除外），就是函数 式接口，并且还提供了注解：@FunctionalInterface

#### 常见的四大函数式接口

* Consumer 《T》：消费型接口，有参无返回值
* Supplier 《T》：供给型接口，无参有返回值
* Function 《T,R》：:函数式接口，有参有返回值
* Predicate《T》： 断言型接口，有参有返回值，返回值是boolean类型

在四大核心函数式接口基础上，还提供了诸如BiFunction、BinaryOperation、toIntFunction 等扩展的函数式接口，都是在这四种函数式接口上扩展而来的，

### \*方法引用和构造器调用

若lambda体中的内容有方法已经实现了，那么可以使用“方法引用”  
也可以理解为方法引用是lambda表达式的另外一种表现形式并且其语法比lambda表达式更加简单

#### 方法引用三种表现形式：

1. 对象：：实例方法名  
   2. 类：：静态方法名  
   3. 类：：实例方法名 （lambda参数列表中第一个参数是实例方法的调用 者，第二个参数是实例方法的参数时可用）

PS：

1. lambda体中调用方法的参数列表与返回值类型，要与函数式接口中抽象方法的函数列表和返回值类型保持一致！
2. 2.若lambda参数列表中的第一个参数是实例方法的调用者，而第二个参数是实例方法的参数时，可以使用ClassName::method

#### 构造器引用

格式：ClassName::new

#### 数组引用

格式：Type[]::new

### Stream API

#### Stream操作的三个步骤

* 创建stream
* 中间操作（过滤、map）
* 终止操作

还有功能比较强大的两个终止操作 reduce和collect  
 **reduce**操作： reduce:(T identity,BinaryOperator)/reduce(BinaryOperator)-可以将流中元素反复结合起来，得到一个值

**collect**操作：Collect-将流转换为其他形式，接收一个Collection接口的实现，用于给Stream中元素做汇总的方法

### 接口中的默认方法和静态方法

在接口中可以使用default和static关键字来修饰接口中定义的普通方法

### 新时间日期API

#### LocalDate | LocalTime | LocalDateTime

新的日期API都是不可变的，更使用于多线程的使用环境中

#### 新的日期API的几个优点：

* 之前使用的java.util.Date月份从0开始，我们一般会+1使用，很不方便，java.time.LocalDate月份和星期都改成了enum
* java.util.Date和SimpleDateFormat都不是线程安全的，而LocalDate和LocalTime和最基本的String一样，是不变类型，不但线程安全，而且不能修改。
* java.util.Date是一个“万能接口”，它包含日期、时间，还有毫秒数，更加明确需求取舍
* 新接口更好用的原因是考虑到了日期时间的操作，经常发生往前推或往后推几天的情况。用java.util.Date配合Calendar要写好多代码，而且还不一定能写对

## 多线程

* 进程：进程指正在运行的程序。确切的来说，当一个程序进入内存运行，即变成一个进程，进程是处于运行过程中的程序，并且具有一定独立功能。
* 线程：线程是进程中的一个执行单元，负责当前进程中程序的执行，一个进程中至少有一个线程。一个进程中是可以有多个线程的，这个应用程序也可以称之为多线程程序。
* 简而言之：一个程序运行后至少有一个进程，一个进程中可以包含多个线程

1. 单线程程序：即，若有多个任务只能依次执行。当上一个任务执行结束后，下一个任务开始执行
2. 多线程程序：即，若有多个任务可以同时执行。

### 程序运行原理

* 分时调度 所有线程轮流使用 CPU 的使用权，平均分配每个线程占用 CPU 的时间。
* 抢占式调度 优先让优先级高的线程使用 CPU，如果线程的优先级相同，那么会随机选择一个(线程随机性)，Java使用的为抢占式调度。

### Thread多线程

#### 进程与多进程

* 一个进程对应一个应用程序。在java的开发环境下启动JVM，就表示启动了一个进程。在同一个操作系统中，可以启动多个进程
* 对于单核处理器来讲，在同一个时间点上运行了两个，但实际二者并不是同时运行
* 而计算机的CPU处理器在某个时间点上只能做一件事，所以计算机在两个进程之间频繁的切换执行，由于切换速度极高，所以我们感觉两者在同时进行
* 多进程的作用不是提高执行速度，而是提高CPU的使用率
* 进程和进程之间的内存是独立的

#### 线程与多线程

* 线程是一个进程中的执行场景。一个进程可以启动多个线程
* 多线程作用不是为了提高执行速度，而是提高应用程序的使用率
* 线程和线程共享“堆内存和方法区内存”，栈内存是独立的，一个线程一个栈
* 由于多线程在来回切换，所以给我们一种错觉：感觉多个线程在同时并发执行。

#### Java语言中实现多线程的两种方式

* 继承 java.lang.Thread 重写 run 方法
* 实现 java.lang.Runnable 实现 run 方法

#### 线程的调度与控制

CPU在某个时刻只能执行一条指令，线程只有得到CPU时间片，才能执行命令  
JVM负责线程的调度，取得CPU的时间片。优先级高的线程获取的CPU时间片相对多一些

#### 线程阻塞(sleep)与终止(interrupt)

Thread.sleep(ms)，是一个静态方法，阻塞当前线程，腾出CPU，让给其他线程

### 安全问题

**当多个线程同时执行，多个线程之间是相互抢占资源执行，并且抢占是发生在线程的执行的每一步过程中，导致出现非法数据。这种现象就称之为多线程的并发安全问题。**

#### 线程的同步（加锁）important！

异步编程模型：t1线程执行t1的，t2线程执行t2的，两线程之间谁也不等谁

同步编程模型：t1线程和t2线程执行，t1必须等t2线程执行结束后才能执行

#### 为什么要引入线程同步呢

为了数据的安全。尽管应用程序的使用率降低，但是为了保证数据是安全的，必须加入线程同步机制。(线程同步机制让程序等同于单线程)

#### 什么条件下要使用线程同步

* 必须是多线程环境
* 多线程环境共享同一个数据
* 共享的数据涉及到修改操作

#### 原理：

T1线程执行到此处遇到synchronized关键字，就会去找this的对象锁，找到则进入同步语句块执行程序，执行完再归还对象锁  
在T1线程执行同步语句块的过程中，若T2线程执行到此处也遇到synchronized关键字，但未找到this的对象锁，只能等待T1归还  
(synchronized添加到静态方法上，线程执行此方法时会找类锁)

#### 死锁示例

　　死锁：

　　　　在多线程执行任务的时候，可能出现需要获取到多把锁才能去完成某个任务。

　　　　Thread-0  、Thread-1

　　　　它们要完成线程的任务，而它们都需要获取不同的锁才能完成。

　　　　Thread-0  需要先获取A锁  再获取B才能去完成任务

　　　　而Thread-1线程需要先获取B锁，在获取A锁才能完成任务。

　　　　这时Thread-0 获取到了A的时候，CPU切换到Thread-1上，这时Thread-1就获取到了B锁。

　　　　这时就出现了2个线程要执行任务都需要获取对方线程上的那个锁。

1. js基础

**1、js概念**

Js全称javascript，是一种基于对象和事件驱动的具有安全性能的脚本语言

**2、js数据类型**

基本数据类型（3种）

字符串(string) "hello"、"学习" 'javascript'

数值(number) 18, 365.25, -12.7

布尔(boolean) true, false

复合数据类型（2种）

数组Array

对象 object

特殊数据类型（2种）

空 null

未定义 undefined

**3、js检测数据类型**

typeof() 检测类型。

**4、请写出js的5个内置函数并说明其作用**

isNaN() 判断一个值是否为数值,如果是数值返回false,否则返回true

Number() 把指定值转换为数字,如果无法转换返回 NaN

parseInt() 将指定值转为整数,如果无法转换返回 NaN

parseFloat() 将指定值转为小数,如果无法转换返回 NaN

eval() 将传入的字符串解析为JavaScript 代码并执行

**5、js中的内置对象**

--math对象(掌握5个)：

ceil（）向上取整 floor（）向下取整 abs（）绝对值

round（）四舍五入 random（）随机数 min() 最小值 max() 最大值

--string对象(掌握5个)：

indexOf（）查找子字符串在字符串中首次出现的位置

lastIndexOf() 查找子字符串在字符串中最后一次出现的位置

split（）将字符串分割为数组

replace（）替换字符串

substr（）按下标和个数截取字符串

substring（）按下标截取字符串

toLowerCase() 将字符串转为小写

toUpperCase() 将字符串转为大写

Replace()替换

--date对象(掌握5个)：

getFullyear（）返回指定日期的年份

getMonth（）返回指定日期的月份

getDay（）返回指定日期星期中的第几天（0-6）

getDate（）返回指定日期月份中的第几天（1-31）

getHours() 返回指定日期对象的小时数

getMinutes() 返回指定日期对象的分钟数

getSeconds() 返回指定日期对象的秒数

getMilliseconds() 返回指定日期对象的毫秒数

toLocalestring（）根据当前区域设置并转化为字符串返回

toString（）将日期转化为字符串返回

--array对象(掌握5个)：

concat（）将两个数组或多个数组拼接成一个新的数组

shift() 移除数组中第一个元素并返回该元素

unshifit() 向数组头部添加一个或多个元素，并返回数组的新长度

pop（）移除数组中最后一个元素并返回该元素

push（）向数组中添加一个元素并返回数组的长度

splice() 在任意位置添加或删除任意个元素

toString（）将数组以字符串的形式返回

join（）返回一个字符串，字符串是以指定的符号分隔

**6、DOM和BOM 对象**

--document:

getElementById() 返回对拥有指定 id 的第一个对象的引用。

getElementsByName() 返回带有指定名称的对象集合。

getElementsByTagName() 返回带有指定标签名的对象集合。

--window:

alert() 弹出警告框

confirm() 弹出确认框

prompt()弹出输入框

open() 打开新窗口

close() 关闭窗口

--history

forward() 加载历史列表中的下一个url，相当于浏览器前进按钮功能

back() 加载历史列表中的前一个url，相当于浏览器后退按钮的功能

go() 加载历史列表中某个具体页面

--location

href 设置或返回浏览器地址栏

reload() 重新加载当前页面，相当于浏览器刷新按钮的功能

**7、dom常用操作(掌握5个)**

getElementById() 通过id获取html元素

getElementsByName() 通过name获取html元素数组

getElementsByTagName() 通过标签名获取html元素数组

body 获取页面中body元素

innerHTML 设置或获取html元素html内容

childNodes 返回html元素的子节点数组

 firstChild 返回html元素的首个子节点

 lastChild 返回html元素的最后一个子节点。

 nextSibling 返回节点的下一个兄弟节点

 parentNode 返回节点的父节点

 previousSibling 返回节点的上一个兄弟节点

 appendChild() 向元素添加新的子节点，并作为最后一个子节点。

 cloneNode(是否深度克隆) 克隆元素

 insertBefore(新节点,已有子节点) 在指定的已有的子节点之前插入新节点。

 removeChild(已有子节点) 从元素中移除子节点

 replaceChild(新节点,旧节点) 替换元素中的子节点

**8、js全局变量和局部变量**

全局变量：声明在函数外(var可用可不用)一般为了规范加上var

局部变量：声明在函数内部(加上var是局部变量，不加var是全局变量)

**9、js事件**

**事件就是用户对网页或浏览器做出的动作.通常来说,鼠标和键盘的动作称之为事件。**

页面事件：

载入页面 onload

离开页面 onunload

鼠标事件:

单击事件 onclick

双击事件 ondblclick

鼠标按下事件 onmousedown

鼠标抬起事件 onmouseup

鼠标移上事件 onmouseover【实战】

鼠标移开事件 onmouseout 【实战】

键盘事件：

键盘按下事件 onkeydown

键盘抬起事件 onkeyup

键盘按下并抬起事件 onkeypress

表单事件：

表单提交事件 onsubmit

表单重置事件 onreset

获得焦点事件 onfocus

失去焦点事件 onblur

内容改变事件 onchange

**10、JS正则表达式**

元字符:元字符是具有特殊含义的字符

\d 匹配一个数字字符，等价于[0-9]

\w 匹配一个大小写字母、数字、下划线，等价于[0-9a-zA-Z\_]。

\s 匹配一个空白字符(空格、制表符(tab)、换行符(\n))

.匹配除\n之外的任意一个字符

\D 匹配一个非数字字符,等价于[^0-9]

\S 匹配一个非空白字符(非空格、非制表符(tab)、非换行符(\n))

\W 匹配一个非大小写字母、非数字、非下划线，等价于[^0-9a-zA-Z\_]

边界符：

^ 匹配字符串的起始位置

$ 匹配字符串的结束位置

量词

? 匹配前面的子表达式0次或1次

+ 匹配前面的子表达式1次或多次

\* 匹配前面的子表达式0次或多次

{n} 匹配确定的n(非负整数)次

{n,} 至少匹配n(非负整数)次，最多匹配无数次

{n,m} 至少匹配n次，最多匹配m次(n必须小于等于m)

修饰符

g 全局匹配

m 多行匹配

i 忽略大小写匹配

**11、oracle数据类型**

字符类型：

varchar2:最多4000字节,可变长度,最为常用

nvarchar2:最多2000个字符，可变长度，内部实现存在问题不常用

char:最多2000字节，固定长度，长度不够以空格填充 大文本数据类型

数值类型：

number(总长度,小数点后的位数) 最大精度38位

(小数点后的位数:小数位置四舍五入)

日期类型：

date (年月日 时分秒)

通常情况下如果是GBK编码则一个汉字对应2个字节，如果是UTF-8编码则一个汉字对应3个字节。

**12、Oracle的5个聚合函数 ，并说明用途**

-oracle的聚合函数

--count（）统计数量

--max（）最大值

--min（）最小值

--avg（）平均值

--sum（）总和

**13、主键外键**

主键--唯一标识一条记录，不能有重复的，不允许为空

primary key

constraint pk\_id primary key(字段名)

外键--表的外键是另一表的主键, 外键可以有重复的, 可以是空值

constraint 外键名 foreign key(字段名) references 主表(主键字段名)

**14、oracle常用函数函数**

 --数学函数

abs(字段名|值) 绝对值

ceil(字段名|值) 向上取整

floor(字段名|值 ) 向下取整

trunc(字段名|值) 截断

round(字段名|值) 四舍五入

 --字符串函数

concat(字段名|值, 字段名|值) 字符串拼接

initcap(字段名|值) 将首字母转为大写，剩余的全转为小写

instr(字段名|值,子字符串[,起始位置[,第几次出现]]) 按照起始位置和指定次数查找子字符串在字符串中出现的位置

length(字段名|值) 字符串长度

lower(字段名|值) 转为小写

upper(字段名|值) 转为大写

rpad|lpad(字段名|值,字符串总长度,填充的字符串) 左填充和右填充

rtrim|ltrim (字段名|值,要删除的字符串) 左去除|右去除

substr(字段名|值,起始位置,个数) 按照起始位置和个数截取字符串

replace(字段名|值,s1,s2) 替换字符串

reverse(字段名|值) 字符串反转，这里反转的是字节不是字符

--转换函数

to\_char(值|字段名[,格式] ) 将日期或数字转为字符串

to\_date(值|字段[,格式] ) 将字符串转为日期

 --日期和时间函数:

add\_months 增加或减去月份

last\_day 返回日期所在月的最后一天

next\_day(date,day) 给出日期date和星期day之后计算下一个星期的日期

 --通用函数:

nvl(字段名|值,默认值) 如果指定值为空则显示默认值，否则显示指定值

nvl2(字段名|值,默认值1,默认值2) 如果指定值为空则显示默认值2，否则显示默认值1

 decode(字段名|值,条件1,值1,条件2,值2,…,默认值)

如果指定值等于条件1则显示值1，

如果指定值等于条件2则显示值2，

…

如果指定值不等于任何条件则显示默认值

**15、Oracle分页：**

--假设现在数据库里有839条记录，页面上每页显示37条，现在查出第13页要显示的数据

select \* from (

select rownum rn,t.\* from (

select \* from t\_student

) t where rownum <= (13\*37)--页数\*每页显示条数

) where rn > (13-1)\*37--(页数-1)\*每页显示条数;

**16、DML数据操作语言**

--增加

insert into 表名 (字段1,字段2,字段3,...) values (值1,值2,值3,...);

insert into 表名 values (值1,值2,值3....);

--删除

delete from 表名 [where子句];

--修改

update 表名 set 字段1=值1,字段2=值2,… [where子句];

**17、序列**

--创建序列语法

create sequence 序列名

minvalue 最小值

maxvalue 最大值

start with 起始值

increment by 增量(每次递增的值)

cache 缓存的序列值的个数

order;

--序列名.nextval 获取下一个值

--序列名.currval 获取当前值

**18、两表联查**

内连接

内连接 inner join 只显示两张表有关联的数据

select 字段名,字段名 from 表1 inner join 表2 on 表1和表2的关联条件

select \* from t\_student s inner join t\_class c on s.classid=c.id;

左连接

左连接 left join 会将左边表的全部数据进行展示，另外一张表有和它关联的就展示，没有就展示为null

select 字段名,字段名 from 表1 left join 表2 on 表1和表2的关联条件

select \* from t\_student left join t\_class on t\_class.id=t\_student.classid;

右连接

右连接 right join 会将右边表的全部数据进行展示，另外一张表有和它关联的就展示，没有就展示为null

select 字段名,字段名 from 表1 right join 表2 on 表1和表2的关联条件

**19、JDBC操作数据库的实现步骤及其主要代码**

--导jar包

--加载驱动为数据库连接做准备 关键字：Class.forName

--连接数据库 关键字：DriverManager.getConnection()

--写sql语句

--创建数据库操作对象 关键字：preparestatement

--执行sql语句 关键字：executeupdate()/executequery()

--关闭连接释放资源 关键字：.close()

**20、端口**

tomcat默认端口8080

oracle默认端口 1521

**21、Servlet的生命周期**

1.实例化：由web服务器进行servlet的实例化

2.初始化：使用init()方法进行初始化

3.提供服务：使用service()方法提供服务。（具体使用doGet()和doPost()方法提供服务）

4.销毁：使用destroy()方法进行销毁

5.不可用：标记为垃圾回收

**22、jsp运行原理**

--jsp的运行原理:

1、用户发送一个请求---->服务器(tomcat)

2、服务器根据请求的url(地址)----->找到相应的jsp文件

3、将找到的jsp文件转译成java源文件(servlet)

4、将转译出来的java源文件进行编译，得到一个可执行文件(.class)

5、执行.class文件并且告诉服务器(tomcat)

6、服务器返回一个响应信息------>用户

**23、Jsp 9大隐式对象。**

**输入输出对象**：

request 对应的实体类是(HttpServletRequest)

获取请求的信息(这是用户提交上来的信息)

同时还是作用域通讯对象

在当前请求中，一直有效

response 对应的实体类是(HttpServletResponse)

响应用户的信息(这是要返回给用户的信息)

out 对应的实体类是(JspWriter)

输出对象，一般输出一些调试信息

out.println(); 向页面输出一些信息

System.out.println(); 向控制台输出一些信息

**作用域通讯对象**：

application 对应的实体类是(ServletContext)

只要服务器不关闭，一直有效

session 对应的实体类是(HttpSession)

只要当前会话不关，一直有效

pageContext 对应的实体类是(PageContext)

只在本页面上有用(离开里本页面就没用了)

request

在当前请求中，一直有效

**servlet对象：**

page 对应的实体类是(Page)

相当于new了一个实例，类似于java中的this关键字

config 对应的实体类是(ServletConfig)

可以获取servlet初始化的一些信息，不太常用

**错误对象：**

exception 对应的实体类是(Throwable)

专门处理错误

**24、Jsp的四个作用域、作用域范围、以及对应的作用域对象**

page 当前页面 pageContext

request 当前请求 可以跨越多个页面 request

session 当前会话 多次请求 session

application 整个项目 整个服务应用 application

**25、Redirect和forward的区别**

转发和重定向的区别；

redirect重定向 特点：

地址栏会发生变化

数据不共享(用户请求信息不会带到新页面)

执行效率低，好几次请求

写法：

response.sendRedirect(目标文件地址);

forward转发 特点：

地址栏不会发生变化

数据共享(用户请求信息会被带到新页面)

执行效率高，只有一次请求

写法：

request.getRequestDispatcher(目标文件地址).forward(request, response);

26、**get和post的区别：**

1.get是一般用于从服务器上获取数据， post一般用于向服务器传送数据。

2.get提交的数据会存放在URL之后,以？分割url和传输数据，参数之间以&相连。get对数据不安全，post相对安全

3.get提交的数据大小有限制，最多只能有1024字节(因为浏览器对url长度有限制)而post提交没有限制。

4.通常用post做文件上传而get是不支持的。

## 数据库

**oracle数据类型**

字符类型：

varchar2:最多4000字节,可变长度,最为常用

nvarchar2:最多2000个字符，可变长度，内部实现存在问题不常用

char:最多2000字节，固定长度，长度不够以空格填充 大文本数据类型

数值类型：

number(总长度,小数点后的位数) 最大精度38位

(小数点后的位数:小数位置四舍五入)

日期类型：

date (年月日 时分秒)

通常情况下如果是GBK编码则一个汉字对应2个字节，如果是UTF-8编码则一个汉字对应3个字节。

--oracle的聚合函数

--count（）统计数量

--max（）最大值

--min（）最小值

--avg（）平均值

--sum（）总和

--数学函数

abs(字段名|值) 绝对值

ceil(字段名|值) 向上取整

floor(字段名|值 ) 向下取整

trunc(字段名|值) 截断

round(字段名|值) 四舍五入

--转换函数

to\_char(值|字段名[,格式] ) 将日期或数字转为字符串

to\_date(值|字段[,格式] ) 将字符串转为日期

 --日期和时间函数:

add\_months 增加或减去月份

last\_day 返回日期所在月的最后一天

next\_day(date,day) 给出日期date和星期day之后计算下一个星期的日期

sql语句

--假设现在数据库里有839条记录，页面上每页显示37条，现在查出第13页要显示的数据

select \* from (

select rownum rn,t.\* from (

select \* from t\_student

) t where rownum <= (13\*37)--页数\*每页显示条数

) where rn > (13-1)\*37--(页数-1)\*每页显示条数;

**DML数据操作语言**

--增加

insert into 表名 (字段1,字段2,字段3,...) values (值1,值2,值3,...);

insert into 表名 values (值1,值2,值3....);

--删除

delete from 表名 [where子句];

--修改

update 表名 set 字段1=值1,字段2=值2,… [where子句];

--创建序列语法

create sequence 序列名

minvalue 最小值

maxvalue 最大值

start with 起始值

increment by 增量(每次递增的值)

cache 缓存的序列值的个数

order;

--序列名.nextval 获取下一个值

--序列名.currval 获取当前值

两表联查

内连接

内连接 inner join 只显示两张表有关联的数据

select 字段名,字段名 from 表1 inner join 表2 on 表1和表2的关联条件

select \* from t\_student s inner join t\_class c on s.classid=c.id;

左连接

左连接 left join 会将左边表的全部数据进行展示，另外一张表有和它关联的就展示，没有就展示为null

select 字段名,字段名 from 表1 left join 表2 on 表1和表2的关联条件

select \* from t\_student left join t\_class on t\_class.id=t\_student.classid;

右连接

右连接 right join 会将右边表的全部数据进行展示，另外一张表有和它关联的就展示，没有就展示为null

select 字段名,字段名 from 表1 right join 表2 on 表1和表2的关联条件

# 2 框架

## Mvc思想

Model 持久层 进行数据库操作

Controller 控制层 接参传参 页面跳转 编写业务逻辑

View 视图层 项目的页面展示’

优点 解耦 ’

让我们开发 各司其职 前端转注与页面设计 程序员专业与业务逻辑 减少开发难度

更易于后续维护修改

## Spring

Spring是一个轻量级的开源框架 为解决企业开发难度的一站式分层框架 ;

他在我们的项目中扮演一个类似粘合剂的角色 帮我们将MVC层连接起来

Spring的两大核心 IOC /di aop

Di依赖注入 是IOC的一种体现方式

就是将我们的IOC 容器中的bean 拿出来使用

分为手动注入

属性注入 property 依赖set

构造函数注入 construction 依赖有参构造

接口注入 不常用

自动注入

byname 根据属性名拿bean

Byteype 根据bean类型

Ioc 是一种思想不是一个技术

他就是将我们在程序中手动创建对象的权利 交给IOC控制

从正空 变为反恐(我们不需要知道怎么创建的; 只需要知道生产出来的对象即可 需要直接去IOC容器去拿)

IOC控制反转 是将我们在程序中手动创建对象的控制权交给IOC实现; 做到了解耦 其设计思想是工厂模式 我们不需要知道怎么创建的对象;只需要知道生产出来的对象是谁即可;工作流程就是 在容器启动后; 初始化好我们配置的bean ;在我们需要调用的时候;就会把已经初始化好的bean 分配我们要调用的类,而不用区再创一个实例;

Aop

是编程思想发展到一定阶段的产物;是oop的有益补充 ;一般适用于具有横切逻辑的场所,如日志记录,权限控制;安全认证;性能统计;事务管理等等;说白了就是在不改变源码的基础上;进行增强处理;增加新的功能; aop 的代理通过JDK动态代理;也可以通过cglib来代理;默认是JDK动态代理;但需要接口的支持;如果弄没有就是只能超给力吧来代理;

Spring 有点

Di依赖注入 可以让我们的依赖关系一目了然;

IOC控制反转 更轻量级的使用了单利;节约内从;

实现了解耦;通过Spring容器管理;不需要在程序中创建

Aop面向切面编程 可以在不修改源码的基础上进行代码增强处理;

## SpringMVC

是一个控制层框架; 主要用来接参传参;页面跳转;

SpringMVC 的运行原理

1. 首先用户发送一个请求到核心请求分发器;dispatcherservlet
2. Dispatcherservlet 查找一个或多个handlerMapping,找到可以处理请求的handler;
3. 通过handleradapter 执行handler 调用业务逻辑处理后;返回modelandview;
4. 核心请求分发器会超找一个或多个view resolver 视图解析器;将model渲染到view;
5. 将view返回给用户

springMVC搭建

1. 在web.xml 配置核心请求分发器;
2. 创建springMVC.xml’放到src目录下
3. .扫描controller层;开启springMVC注解 配置视图解析器;
4. 创建一个controller类,并加上@controller注解 标识这是一个控制类

请求的类。并加@RequestMapping(“映射地址”)来配置本类的映射地址

(5)新建方法，返回值为一个ModelAndView对象，并进行操作，在方法上加上@RequestMapping(“映射地址”)来配置本方法的映射地址

(6)配置springMVC核心配置文件中的扫描器,扫描的是controller层

SpringMVC跳转页面的方式

1. 通过modelandview setViewName方法跳转

2. 返回类型是String的时候直接return页面 可以把要返回的数据放入到model中

3. 通过request.getRequestDispartcherServlet返回

Springmvc是如何跳转页面的

1. 默认是forward

2. 重定向 返回的时候在return或者setViewName中 加上redirect:请求

带值返回

第一种 通过request .setAtt...

第二种 通过ModelAndView addObject("key","value")

第三种 Model model.addAttribute("","")

第四种 ModelMap addAttribute("","")

接收参数

第一种 request.getParameter()获取

第二种 通过属性

第三种 通过对象

第四种 restFul风格 接口开发

redirect:请求路径 重定向

## hibernate

hibernate 是一个orm 框架 对jdbc进行了轻量级的封装;

可以在对象对与数据建立映射; hibernate可以自动生成SQL语句，自动执行，

简化我们的开发，提高我们的开发效

**hibernate中session中的常用方法**

save():保存

update()修改

saveOrUpdate()保存或者修改

delete()删除

get()根据唯一标示获取对象 立即加载

load()根据唯一标示获得对象 ，延迟加载

hibernate 的运行原理

**1.    通过configuration加载了hibernate.cfg.xml文件。**

**2.    加载后创建sessionFactory.(sessionFactory是线程安全的)**

**3.    然后同sessionFactory获得session。（创建会话，相当于jdbc中的connection 是线程不安全的）**

**4.    通过session操作数据库，最后通过transaction来进行事物的控制.**

**Hibernate五大核心接口**

1. **Configuration(类):**

加载配置文件hibernate.cfg.xml文件中的配置信息，从而得到：

**1.hibernate的底层信息：**

数据库连接，jdbc驱动，方言，用户名，密码。

**2.hibernate的映射文件(\*.hbm.xml)**

1. **SessionFactory（接口）**

通过configuration创建SessionFactory。

通过SessionFactory可以用来获得session。

SessionFactory是线程安全的，

里边加载的数据信息有数据库的配置信息和映射关系.

1. **Session(接口)**

线程不安全。

相当于jbdc的connection.

用来操作数据库.

1. **Transaction（接口）**

进行事务的控制实现事务的commit（提交)和rollback（回滚）

1. **Query(接口)**

用来执行数据库的查询操作。

**hibernate主键生成策略**

**increment(银科瑞门特) :先选出当前表的最大ID，在此基础上加1，hibernate会自行处理，适用于int short,long类型的主键**

**sequence:oracle数据库会自行处理，会增加一个序列配置**

**native（通用多个数据库，数据库的本地生成策略）**

**uuid：生成一个32的，不重复的字符串，可以达到夸数据库**

identity :由底层数据库生成标识符(自动增长),返回的标识符是 long, short 或者int类型的。

sequence hibernate根据底层数据库序列生成标识符,返回的标识符 是long, short或者 int类型的。

uuid.hex :用一个128-bit的UUID算法生成32位字符串类型的标识符。

 native :根据底层数据库的能力选择identity, sequence 。

assigned :让应用程序在save()之前为对象分配一个标示符。

foreign :使用另外一个相关联的对象的标识符。和<one-to-one>联合一起使用。

**jdbc和hibernate的区别**

hibernate是JDBC的轻量级封装，hibernate基于jdbc

**jdbc**:

jdbc是纯手工的原生sql语句

执行效率高于hibernate

jdbc可以直接操作数据库的表如:select \* from 表名

**hibernate：**

hibernate是全自动

跨数据库

开发效率高于jdbc

使用的是hql语句最终会转化为sql语句.语法是 from 对象名

**get和load的区别**

1.    load延迟加载 访问对象时发送sql语句，访问实体不存在时返回（对象不存在异常）,如果设置了lazy=“false”,load就不再进行延迟加载

2.  get立即加载 查询时立即发送sql语句，当访问实体不存在是返回null

**hibernate三种状态**

***临时状态、持久化状态、托管状态/游离状态***

**transient['trænzɪənt](临时状态)：**

        在创建对象即在new之后，在save之前的状态：

        session中和数据库中都不存在该数据

**persistent[pə'sɪst(ə)nt](持久化状态):**

        在save之后，session关闭之前，

        session中和数据库中都存在对应的数据

**detached[dɪ'tætʃt](托管状态/游离状态):**

        session关闭之后，session不存在，数据中有对应的数据

## Jpa

jpa是个规范 不是一个框架

**1.Jpa是什么**

JPA即Java持久化API，JPA的出现主要是为了简化持久层（Dao层）开发以及整合ORM技术，结束Hibernate、TopLink(套铺赁克)、JDO等ORM框架各自为营的局面。

JPA包括以下3方面的技术：

* **ORM映射：支持XML和注解描述对象和表之间的映射关系**
* **API：操作实体对象来执行CRUD(create read update delete)(增删改查)操作**
* **查询语言：通过面向对象而非面向数据库的查询语言（JPQL）查询数据**

**Spring Data Jpa的使用**

1. **在项目的pom.xml中添加spring-data-jpa的依赖配置**
2. **在spring核心配置文件中配置spring-data-jpa相关项**
3. **编写实体类并使用JPA注解配置实体类和数据库表的映射关系**
4. **编写符合 Spring Data JPA 规范的 Dao 层接口**

**2.**   **Spring Data JPA是什么**

Spring Data JPA 是 Spring 基于 **ORM 框架**、**JPA 规范**的基础上封装的一套**JPA应用框架**，可使开发者用**极简的代码**即可实现对**数据的访问和操作**。Spring Data JPA不需要过多的关心Dao层的实现，只需关注我们继承的接口，按照一定的规则去编写我们的接口即可。

**3.**   **Spring Data JPA的核心接口**

* **Repository(瑞胖丝特瑞): 所有接口的父接口，而且是一个空接口，目的是为了统一所有Repository的类型，让组件扫描的时候能进行识别。**
* **CrudRepository：是Repository的子接口，提供CRUD(增删改查)的功能。**
* **Paging(培经)AndSorting(消停)Repository：是CrudRepository的子接口，添加分页和排序的功能。**
* **JpaRepository：是PagingAndSortingRepository的子接口，增加了一些实用的功能，例如批量操作。**
* **JpaSpecification(丝拍谁非K鑫)Executor(A克丝Q特儿)：用来做负责查询的接口。**
* **Specification(丝拍谁非K鑫)：是Spring Data JPA提供的一个查询规范，要做复杂的查询，只需围绕这个规范来设置查询条件即可。**

**4.**   **JPA中常用的注解**

@Entity【嗯忒忒】 标识当前类为实体类

@Table 配置与实体类相对应的数据库表信息(如表名)

@Id 标识当前字段为主键

@Generated(杰呢瑞忒德)Value 配置主键生成策略

@Column(靠乐木) 配置属性对应的数据库字段信息(如字段名)

@ManyToOne【美图玩】 配置一对多关系

@ManyToMany【没图没】 配置多对多关系

@Transient这个注解表示忽略这个属性，那么建表的时候就不会在数据库表中生成一个对应的字段了！

@JoinColumn外键

**5.**   **JPA主键生成策略**

**Identity(爱den忒忒**):主键由数据库自动生成 （主要是自动增长型）

**Sequence(谁困死):**根据底层数据库的序列来生成主键，条件是数据库支持序列

**Table**:使用特定的数据表格来保存主键

**Auto**:由持久化引擎来决定从以上三种策略中选择最合适的

**6.**   **Spring Data Jpa查询方法命名规则**

如果spring data jpa现有的查询方法无法满足你的需求，你只需要在dao层接口上按照spring data jpa的查询方法命名规则去编写方法，spring data jpa会自动解析你的方法名，从而去执行对应的查询，查询方法命名规则(掌握5个)如下:

* **And:等价于 SQL 中的 and 关键字，比如 findByUsernameAndPassword(String user, Striang pwd)；**
* **Or :等价于 SQL 中的 or 关键字，比如 findByUsernameOrAddress(String user, String addr)；**
* **Between :等价于 SQL 中的 between 关键字，比如 findBySalaryBetween(int max, int min)；**
* **LessThan:等价于 SQL 中的 "<"，比如 findBySalaryLessThan(int max)；**
* **GreaterThan:等价于 SQL 中的">"，比如 findBySalaryGreaterThan(int min)；**
* **IsNull :等价于 SQL 中的 "is null"，比如 findByUsernameIsNull()；**
* **IsNotNull:等价于 SQL 中的 "is not null"，比如 findByUsernameIsNotNull()；**
* **NotNull:与 IsNotNull 等价；**
* **Like :等价于 SQL 中的 "like"，比如 findByUsernameLike(String user)；**
* **NotLike :等价于 SQL 中的 "not like"，比如 findByUsernameNotLike(String user)；**
* **OrderBy:等价于 SQL 中的 "order by"，比如 findByUsernameOrderBySalaryAsc(String user)；**
* **Not:等价于 SQL 中的 "!="，比如 findByUsernameNot(String user)；**
* **In :等价于 SQL 中的 in"，比如 findByUsernameIn(Collection userList) ，方法的参数可以是 Collection 类型，也可以是数组或者不定长参数；**
* **NotIn:等价于 SQL 中的 "not in"，比如 findByUsernameNotIn(Collection userList) ，方法的参数可以是 Collection 类型，也可以是数组或者不定长参数；**

## 什么是mybatis？（持久层框架  对数据库进行操作 （crud增删改查） sql语句）

MyBatis 是一款优秀的持久层框架，它支持自定义 SQL、存储过程以及高级映射。MyBatis 免除了几乎所有的 JDBC 代码以及设置参数和获取结果集的工作。MyBatis 可以通过简单的 XML 或注解来配置和映射原始类型、接口和 Java POJO（Plain Old Java Objects，普通老式 Java 对象）为数据库中的记录。

使用sql语句操作的是数据库，底层都是JDBC。

通过xml 文件或注解的方式将要执行的各种 statement 配置起来，并通过java对象和 statement中sql的动态参数进行映射生成最终执行的sql语句，最后由mybatis框架执行sql并将结果映射为java对象并返回。（从执行sql到返回result的过程）。

//新增返回主键

@Options(useGeneratedKeys=true, keyProperty="userId", keyColumn="id")

**传入参数**

　　(1) 可以传入一个JavaBean

　　(2) 可以传入一个Map

　　(3) 可以传入多个参数，需使用@Param("ParamName") 修饰参数

**#{}和${}的区别是什么？**

${}是字符串替换，#{}是预处理；

Mybatis在处理${}时，就是把${}直接替换成变量的值。而Mybatis在处理#{}时，会对sql语句进行预处理，将sql中的#{}替换为?号，调用PreparedStatement的set方法来赋值；

使用#{}可以有效的防止SQL注入，提高系统安全性。

3.特点：

hibernate: hibernate.cfg.xml,数据对象和数据库有一个xml映射文件对应起来。

mybatis：持久层只有接口没有实现类，所有的sql语句写在一个mapper.xml文件中

jpa:持久层有没有集成一些核心接口

## 什么是 Spring Boot？

Spring Boot 的设计是为了让你尽可能快的跑起来 Spring 应用程序并且尽可能减少你的配置文件。

**设计目的：** 用来简化新 Spring 应用的初始搭建以及开发过程。

Spring boot是一种微服务 用于接口开发

从最根本上来讲，Spring Boot 就是一些库的集合，它能够被任意项目的构建系统所使用。它使用 “习惯优于配置” （项目中存在大量的配置，此外还内置一个习惯性的配置）的理念让你的项目快速运行起来。用大佬的话来理解，**就是 spring boot 其实不是什么新的框架，它默认配置了很多框架的使用方式，就像 maven 整合了所有的 jar 包，spring boot 整合了所有的框架，**

总结一下及几点：

（1）为所有 Spring 开发提供一个更快更广泛的入门体验。(快速搭建spring相关框架)

（2）零配置。无冗余代码生成和XML 强制配置，遵循“约定大于配置” 。

（3）集成了大量常用的第三方库的配置， Spring Boot 应用为这些第三方库提供了几乎可以零配置的开箱即用的能力。

（4）提供一系列大型项目常用的非功能性特征，如嵌入式服务器、安全性、度量、运行状况检查、外部化配置等。

pom

1.需要 parent 配置springboot 的父级依赖

2.JDK 版本

3.启动器的第三方库的依赖

@SpringBootApplication

标识这是一个springboot启动类 main方法 为启动器

* SpringbootApplication： 一个带有 main() 方法的类，用于启动应用程序
* application.properties：一个空的 properties 文件，可以根据需要添加配置属性
* pom.xml： Maven 构建说明文件

**@SpringBootApplication** 是 Spring Boot 的核心注解，它是一个组合注解，该注解组合了：

**@Configuration**、标记类作为配置类 替换 xml配置文件

**@ComponentScan；** 用于类或接口上主要是指定扫描路径，spring会把指定路径下带有指定注解的类自动装配到bean容器里。

**@EnableAutoConfiguration 让 Spring Boot 根据类路径中的 jar 包依赖为当前项目进行自动配置**，例如，添加了 spring-boot-starter-web 依赖，会自动添加 Tomcat 和 Spring MVC 的依赖，那么 Spring Boot 会对 Tomcat 和 Spring MVC 进行自动配置。

* **Spring Boot 还会自动扫描 @SpringBootApplication 所在类的同级包以及下级包里的 Bean ，所以入口类建议就配置在 grounpID + arctifactID 组合的包名下（这里为 com.xpwi.springboot 包）**

**Spring Boot 的配置文件：**

Spring Boot 使用一个全局的配置文件 application.properties 或 application.yml，放置在【src/main/resources】目录或者类路径的 /config 下。

Spring Boot 不仅支持常规的 properties 配置文件，还支持 yaml 语言的配置文件。yaml 是以数据为中心的语言，在配置数据的时候具有面向对象的特征。

Spring Boot 的全局配置文件的作用是对一些默认配置的配置值进行修改。

//接口文档 提供url 请求方式 () 参数说明 返回值说明

// restful 接口的规范 同一个路径 不同的请求方式 对应的操作是不一样的

// 传参 把参数拼到路径上 @PathVariable

2 spirngboot环境搭建

前提条件 jdk 1.8 maven

声明是springboot项目 声明jdk 版本 引入启动器

创建启动类 @springbootapplication （ 三个注解组成 1 2 3 ）

@**Configuration @ComponentScan @EnableAutoConfiguration**

main方法固定写法

springapplication.run("本类",args)

springboot的应用场景 （接口开发 json 和 xml）

3常用注解

@resetcontroller @getmapping 类似的 PathVariable

## vue

 Vue 构造器中有一个el 参数，它是 DOM 元素中的 id。

**data** 用于定义属性

**methods** 用于定义的函数，可以通过 return 来返回函数值。

当一个 Vue 实例被创建时，它向 Vue 的响应式系统中加入了其 data 对象中能找到的所有的属性。当这些属性的值发生改变时，html 视图将也会产生相应的变化。

**模板语法**

数据绑定最常见的形式就是使用 {{...}}（双大括号）的文本插值 <p>{{ message }}</p>

{{ }} 用于输出对象属性和函数返回值。

指令

v-html 使用 v-html 指令用于输出 html 代码

v-bind HTML 属性中的值应使用 v-bind 指令 (Vue.js 都提供了完全的 JavaScript 表达式支持)

class 与 style 是 HTML 元素的属性，用于设置元素的样式，我们可以用 v-bind 来设置样式属性。

v-for 循环使用 v-for 指令。v-for 指令需要以 **site in sites** 形式的特殊语法，

(提供第二个的参数为键名 第三个参数为索引)

v-if 指令将根据表达式 seen 的值(true 或 false )来决定是否插入 p 元素。

*v-else 、v-else-if 必须跟在 v-if 或者 v-else-if之后。*

**v-show** 立即加载 只是隐藏

 v-on 指令，它用于监听 DOM 事件

**v-model** 1.指令用来在 input、select、textarea、checkbox、radio 等表单控件元素上创建双向数据绑定，根据表单上的值，自动更新绑定的元素的值。

2.按钮的事件我们可以使用 v-on 监听事件，并对用户的输入进行响应。

复选框如果是一个为逻辑值，如果是多个则绑定到同一个数组：

修饰符是以半角句号 **.** 指明的特殊后缀，用于指出一个指令应该以特殊方式绑定。

**表单**

单个复选框：

 true

多个复选框：

 Runoob  Google  taobao

选择的值为: [ "Runoob", "Google", "Taobao" ]

**Vue.js 计算属性**

计算属性关键词: computed。

**computed vs methods**

我们可以使用 methods 来替代 computed，效果上两个都是一样的，但是 computed 是基于它的依赖缓存，只有相关依赖发生改变时才会重新取值。而使用 methods ，在重新渲染的时候，函数总会重新调用执行。可以说使用 computed 性能会更好，但是如果你不希望缓存，你可以使用 methods 属性。

**computed 属性默认只有 getter ，不过在需要时你也可以提供一个 setter**

运行 vm.site = '菜鸟教程 http://www.runoob.com'; 时，setter 会被调用，

computed: { site: { // getter get: function () { return this.name + ' ' + this.url }, // setter set: function (newValue) { var names = newValue.split(' ') this.name = names[0] this.url = names[names.length - 1] } }

watch('counter', function(nval, oval) { alert('计数器值的变化 :' + oval + ' 变为 ' + nval + '!'); });

**Vue.js 组件**

组件（Component）是 Vue.js 最强大的功能之一。

组件可以扩展 HTML 元素，封装可重用的代码。

组件系统让我们可以用独立可复用的小组件来构建大型应用，几乎任意类型的应用的界面都可以抽象为一个组件树

父组件怎么向子组件传值 props 子组件怎么调用父组件的方法 $emit） components:{}

Vue.component(tagName, options)

tagName 为组件名，options 为配置选项。

**钩子函数**

**vue有8种生命周期函数：**

|  |  |  |
| --- | --- | --- |
| 钩子函数 | 触发的行为 | 在此阶段可以做的事情 |
| beforeCreadted | vue实例的挂载元素$el和数据对象data都为undefined，还未初始化。 | 加loading事件 |
| created | vue实例的数据对象data有了，$el还没有 | 结束loading、请求数据为mounted渲染做准备 |
| beforeMount | vue实例的$el和data都初始化了，但还是虚拟的dom节点，具体的data.filter还未替换。 | .. |
| mounted | vue实例挂载完成，data.filter成功渲染 | 配合路由钩子使用 |
| beforeUpdate | data更新时触发 |  |
| updated | data更新时触发 | 数据更新时，做一些处理（此处也可以用watch进行观测） |
| beforeDestroy | 组件销毁时触发 |  |
| destroyed | 组件销毁时触发，vue实例解除了事件监听以及和dom的绑定（无响应了），但DOM节点依旧存在 | 组件销毁时进行提示 |

**Vue.js 路由**

Vue.js 路由允许我们通过不同的 URL 访问不同的内容。

通过 Vue.js 可以实现多视图的单页Web应用（single page web application，SPA）。

项目创建

创建项目 命令：vue init webpack my-project

启动 cnpm run dev

## 脚手架 （node.js ）目录结构

|  |  |
| --- | --- |
| 目录/文件 | 说明 |
| build | 项目构建(webpack)相关代码 |
| config | 配置目录，包括端口号等。我们初学可以使用默认的。 |
| node\_modules | npm 加载的项目依赖模块 |
| src | 这里是我们要开发的目录，基本上要做的事情都在这个目录里。里面包含了几个目录及文件：  assets: 放置一些图片，如logo等。  components: 目录里面放了一个组件文件，可以不用。  App.vue: 项目入口文件，我们也可以直接将组件写这里，而不使用 components 目录。  main.js: 项目的核心文件。 |
| static | 静态资源目录，如图片、字体等。 |
| test | 初始测试目录，可删除 |
| .xxxx文件 | 这些是一些配置文件，包括语法配置，git配置等。 |
| index.html | 首页入口文件，你可以添加一些 meta 信息或统计代码啥的。 |
| package.json | 项目配置文件。 |
| README.md | 项目的说明文档，markdown 格式 |

项目的结构

入口 index.html ---》》》 main.js ->>>>app.vue

脚手架 中的 vue 代码 模块 组件名 要求 首字母大写

<template></template> <script> </script> <style> </style>

脚手架中 怎么引入 组件

/\* 引入 组件 import 变量 from 组件路径 \*/

import jgxi from "./components/jgxi";

如果idea中的terminal 命令不管用

Terminal npm命令不能正常使用 File——>Settings——>Tools——>Terminal,发现Shell Path配置的不是本地的cmd地址,修改为本地cmd路径后C:\Windows\System32\cmd.exe,保存重启...

vue中 请求远程接口的数据

axios（第三方插件库）

# 3 架构

## mongodb

mongodb是一个非关系型数据库

专业的说法 ：Mongodb是一个基于分布式文件存储的数据库，由C++语言编写，皆在为WEB应用可扩展的高性能的数据存储解决方案。

Mongodb是一个介于关系型数据库和非关系型数据库之间的产品，是最像关系型数据库的

Mongodb是一个面向文档存储的数据库，操作起来比较简单容易。

你可以在mongodb中设置任何属性的索引来实现更快的排序，

你可以通过本地或者网络创建数据镜像，这使得mongodb有更强的扩展性，

如果负载的增加 可以在其他网络记算计分布节点 这就是所谓的分片。

MongoDB支持各种编程语言:RUBY，PYTHON，JAVA，C++，PHP，C#等多种语言。

MongoDB安装简单。

Mongodb命令

show dbs (查看所有存在数据的数据库)

use (创建或者切换数据库)

db (查看当前数据库)

集合的相关操作

db.createCollection(); (创建集合)

db.集合名称.drop() (删除集合)

show (collections/tables) 查看所有的集合

db.集合名insert（） 集合添加数据 save insertone insertmany

文档的操作

db.集合名.insert(文档) 集合添加数据 save insertone insermany

db.集合名.remove

db.集合名.update

db.集合名.find()

## redis

非关系型数据库 nosql 主要是开源免费的

redis 是一个高性能的key-value数据库 由c++语言编写

数据类型丰富 ：string(字符串) ,list(链表),set(无序集合),zset(有序集合),hash(哈希类型)

这些数据类型都支持push/pop，add/remove及取交集并集和差集及更丰富的操作， 而且这些操作都是原子性的, Redis 是单线程模型，指的是执行 Redis 命令的核心模块是单线程的，而不是整个 Redis 实例就一个线程,redis4.0就开始由多线程的概念了,比如redis需要通过多线程方式在后台删除对象

Redis 6.0 其实也是多线程的 但是默认是关闭的 需要在redis.conf 中开启

数据类型：

String :存进去的数据 直接可以取出来用

List : 链表 可以用来做消息队列 先进先出

Set ：无序集合

Zset ：有序集合

Hash : key filed value

持久化方案 RDB AOF

Rdb : 基于快照的方式将某个时间段的数据进行存储 适用于大数据的场合使用 数据量过小的话可能会丢失某一时间段的数据

AOF : 将对数据库数据造成改变的操作已文本的方式记录到aof文件中

Rdb和Aof的区别

RDB持久化是指在指定的时间间隔内将内存中的数据集快照写入磁盘，实际操作过程是fork一个子进程，先将数据集写入临时文件，写入成功后，再替换之前的文件，用二进制压缩存储。  
  
 RDB持久化保存键空间的所有键值对(包括过期字典中的数据),并以二进制形式保存，符合rdb文件规范，根据不同数据类型会有不同处理。  
  
 AOF持久化以日志的形式记录服务器所处理的每一个写、删除操作，查询操作不会记录，以文本的方式记录，可以打开文件看到详细的操作记录。  
  
 AOF持久化保存redis服务器所执行的所有写命令来记录数据库状态,在写入之前命令存储在aof\_buf缓冲区

RDB 在恢复大数据集时的速度比 AOF 的恢复速度要快，AOF在运行效率上往往会慢于RDB。

Redis的三大问题

击穿 （恶意访问一个不存在的数据）

指的是**单个key**在缓存中查不到，去数据库查询，这样如果数据量不大或者并发不大的话是没有什么问题的。

   如果数据库数据量大并且是**高并发**的情况下那么就可能会造成数据库压力过大而崩溃

解决方案 ：

1**缓存空对象**  当查询的时候，如果缓存和数据库中都没有，我们就将这个数据以空的形式存放在缓存中，(或者是给一个false的标示)这样就不用去数据库就可以知道不存在，减少对数据库查询的次数，当我们这个值发生改变的时候，我们在重新进行赋值；

2使用布隆过滤器 BloomFilter

业务系统查询数据时，先从布隆过滤器判断该key是否存在，不存在则说明数据库也不存在，直接返回空。若存在则走正常流程。

这种方法适用于数据命中不高，数据相对固定实时性低(通常是数据集较大)的应用场景，代码维护较为复杂，但是缓存空间占用少

对于恶意攻击，往往key只会使用一次，即使缓存了也没有用，浪费缓存空间，这种场景适合使用布隆过滤器

雪崩 ：

1、大量key同一时间过期造成雪崩  
 2、缓存服务器宕机造成雪崩

解决方案 ：

1 对缓存做集群高可用部署，防止单节点宕机造成雪崩  
 2 对key过期时间增加一个随机数，防止同时大面积key过期，或者对热点数据设置为永不过期

穿透：

是指一个key非常热点，在不停的扛着大并发，大并发集中对这一个点进行访问，当这个key在失效的瞬间，持续的大并发就穿破缓存，直接请求数据库，就像在一个屏障上凿开了一个洞。

**解决方案 ：**

对热门访问key早早的做好了准备，让缓存永不过期；

## redis主从复制

概念

    主从复制模型中，有多个redis节点。

    其中，有且仅有一个为主节点Master。从节点Slave可以有多个。

只要网络连接正常，Master会一直将自己的数据更新同步给Slaves，保持主从同步。

特点

（1）主节点Master可读、可写.

（2）从节点Slave只读。（read-only）

因此，主从模型可以提高读的能力，在一定程度上缓解了写的能力。因为能写仍然只有Master节点一个，可以将读的操作全部移交到从节点上，变相提高了写能力。

主从模式的一个缺陷 就是突然主挂掉了 那么就没有一个可往数据库中写数据的节点了 那么这个就是主从的缺陷 但是下面的东西可以解决

**Sentinel哨兵模式**

监控主数据库和从数据库是否正常运行。

主数据库出现故障时自动将从数据库转换为主数据库。

Redis 的 Sentinel 系统用于管理多个 Redis 服务器（instance）， 该系统执行以下三个任务：

**监控（Monitoring）**： Sentinel 会不断地检查你的主服务器和从服务器是否运作正常。

**提醒（Notification）**： 当被监控的某个 Redis 服务器出现问题时， Sentinel 可以通过 API 向管理员或者其他应用程序发送通知。

**自动故障迁移**（Automatic failover）： 当一个主服务器不能正常工作时， Sentinel 会开始一次自动故障迁移操作， 它会进行选举，将其中一个从服务器升级为新的主服务器， 并让失效主服务器的其他从服务器改为复制新的主服务器； 当客户端试图连接失效的主服务器时， 集群也会向客户端返回新主服务器的地址， 使得集群可以使用新主服务器代替失效服务器。

**Cluster** **集群模式**

哨兵 Sentinel模式基本可以满足一般生产的需求，具备高可用性。**但是当遇到单机内存，并发和流量等瓶颈的时候，主从模式或哨兵Sentinel模式就不能满足需求了**，这个时候需要**对存储的数据进行分片**，**将数据存储到多个Redis实例中**。

什么是redis集群

集群由**多个节点(Node)组成**，Redis的数据分布在这些节点中。每个节点都保存着数据和整个集群的状态，每个节点都和其它所有节点连接。

Redis集群的作用

数据分区：数据分区(或称数据分片)是Redis集群最核心的功能。

Redis集群将数据分散到多个节点，一方面突破了Redis单机内存大小的限制，存储容量大大增加；另一方面每个主节点都可以对外提供读服务和写服务，大大提高了集群的响应能力。

高可用：集群支持主从复制和主节点的自动故障转移（与哨兵类似）；当任一节点发生故障时，集群仍然可以对外提供服务。

Redis集群特点

1.        所有的Redis节点彼此互联(PING-PONG机制)，内部使用二进制协议优化传输速度和带宽。

2.        节点的fail是通过集群中超过半数的节点检测失效时才生效。

3.        客户端与Redis节点直连，不需要中间proxy层。客户端不需要连接集群所有节点，连接集群中任何一个可用节点即可。

4.        Redis集群把所有的物理节点映射到[0-16383]slot上（不一定是平均分配）,集群负责维护node<->slot<->value。

5.        Redis集群预分将所有的存储空间划分为好16384个哈希槽，集群中的每个节点负责处理一部分哈希槽。当需要在 Redis 集群中放置一个 key-value 时，根据 CRC16(key) mod 16384的值，决定将一个key放到哪个哈希槽中。然后将key放到哈希槽对应的节点上。

## linux命令

ip addr 查看ip地址

pwd 查看当前所在的路径

systemctl status firewalld 查看防火墙状态

systemctl satrt firewalld 启动防火墙

systemctl stop firewalld 关闭防火墙

mkdir 创建文件夹 加 -p的话是创建多级目录

ps –ef|grep 进程名 查看某个进程的运行状态

rz 上传文件

tar –zxvf 文件 解压压缩包

cp file1 file2 复制一个文件

rm –rf file1 删除文件

## docker

**打包他们的应用以及依赖包到一个轻量级、可移植的容器中**

**重要的是容器性能开销极低**。

**一次封装，到处运行**

docker是一个解决运行环境和配置问题的一个软件引擎

主要有 容器 镜像 仓库 这三个部分

Docker 操作

docker images 列出本机的所有镜像 镜像名 版本号 id 创建时间 大小

docker search 镜像名 搜索镜像 收藏数

docker pull 镜像名称 下载镜像不指定版本（下载最新的）

docker rmi 镜像名(id) 注意正在被容器使用的镜像是无法删除

docker rmi -f 镜像名(id) 可以删除正在被容器使用的镜像

**容器的操作命令（有镜像才能创建容器）**

docker run -it --name 容器名称（唯一的） 镜像（创建容器并进入终端一旦退出容器结束）

docker run -itd --name 容器名称（唯一的） 镜像（创建容器进入后台运行不会进行终端 （exec 进入终端）终端退出 容器继续运行）

容器端口映射：docker run -itd --name (名称) -p (服务端口):(映射端口) /bin/bash

退出容器 ctrl +q+p 退出后台运行

Exec 退出后台运行

Exit 退出关闭

进入容器 docker exec -it 容器名称(容器id) /bin/bash

查看运行中的容器 docker ps

查看所有的容器 docker ps -a

删除容器 docker rm 容器id（名称）（运行中的容器不能删）

删除容器 docker rm -f 容器id（名称）（可以删除运行中的容器）

停止容器 docker stop 容器id（名称）

启动容器 docker start 容器id（名称）

创建数据卷

docker run -itd --name 容器名 -v 宿主目录(/opt/docker/gx):/容器目录(/gx) 镜像 /bin/bash

制作镜像

创建DockerFile文件

内容如下：

FROM centos

MAINTAINER liuwenhao

RUN mkdir -p /docker/doc

ADD jdk1.8.0\_45 /docker/doc/jdk

ADD springcloud\_product\_eureka\_server-1.0-SNAPSHOT.jar /docker/doc/

ENV JAVA\_HOME /docker/doc/jdk

ENV PATH $PATH:$JAVA\_HOME/bin

EXPOSE 3400

ENTRYPOINT

生成镜像：docker build -f DockerFile文件 -t 生成的镜像名称:版本号 .

启动容器：docker run -itd --name (名称) -p (服务端口):(映射端口) 镜像id /bin/bash

私有仓库

1、拉取私有仓库镜像

docker pull registry

2、启动私有仓库容器

docker run -id --name=registry -p 5000:5000 registry

3、打开浏览器 输入地址

http://私有仓库服务器ip:5000/v2/\_catalog，看到{"repositories":[]} 表示私有仓库搭建成功

4、修改daemon.json vim /etc/docker/daemon.json

在上述文件中添加一个key，保存退出。此步用于让 docker 信任私有仓库地址；注意将私有仓库服务器ip修改为自己私有仓库服务器真实ip

{"insecure-registries":["私有仓库服务器ip:5000"]}

5、重启docker 服务

systemctl restart docker

docker start registry

把镜像推到私有仓库

1. 标记镜像为私有仓库的镜像

docker tag 本地镜像 私有仓库服务器IP:5000/放到仓库的镜像名称

1. 上传标记的镜像

docker push 私有仓库服务器IP:5000/放到仓库的镜像名称

从私有仓库拉镜像

docker pull 私有仓库服务器IP:5000/放到仓库的镜像名称

Docker rabbitmq

拉取 管理版本 进行

docker pull rabbitmq:management

运行容器

docker run -dit --name Myrabbitmq -e RABBITMQ\_DEFAULT\_USER=admin -e RABBITMQ\_DEFAULT\_PASS=admin -p 15672:15672 -p 5672:5672 rabbitmq:management

## nginx（负载均衡）

如何配置nginx负载均衡配置（轮询,权重,ip绑定）

配置文件解析

worker\_processes 1;#工作进程的个数，一般与计算机的cpu核数一致

events {

worker\_connections 1024;#单个进程最大连接数（最大连接数=连接数\*进程数）

}

http {

include mime.types; #文件扩展名与文件类型映射表

default\_type application/octet-stream;#默认文件类型

sendfile on;#开启高效文件传输模式，sendfile指令指定nginx是否调用sendfile函数来输出文件，对于普通应用设为 on，如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络I/O处理速度，降低系统的负载。注意：如果图片显示不正常把这个改成off。

keepalive\_timeout 65; #长连接超时时间，单位是秒

gzip on;#启用Gizp压缩

#服务器的集群

upstream netitcast.com { #服务器集群名字

server 127.0.0.1:18080 weight=1;#服务器配置 weight是权重的意思，权重越大，分配的概率越大。

server 127.0.0.1:28080 weight=2;

}

#当前的Nginx的配置

server {

listen 80;#监听80端口，可以改成其他端口

server\_name localhost;############## 当前服务的域名

location / {

proxy\_pass http://netitcast.com;

proxy\_redirect default;

}

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root html;

}

}

}

集群是为了解决单节点无法服务高并发的情况，在集群中nginx是如何分配将来自客户端的请求 转发给服务器的

负载均衡可以提高网站的吞吐量（接受和响应），减轻单台服务器的压力

负载均衡提供了三种策略：轮询，权重，ip绑定

打开nginx的nginx.conf配置文件：

在回顾一下执行流程：当一个浏览器的请求 例如：发送www.jiahou.com 请求 以后，会先在本地查找hosts文件，对域名进行解析，然后 找到解析出来的ip地址，nginx http服务器拦截所有的请求，监听80端口 server\_name 和域名比较，相同 则 跳转proxy\_pass的路径

如果是在集群的情况下有两台服务器：

轮询：客户端发送一条请求，第一条请求随机给A 或者B ，例如 第一条请求转发给了A 服务器处理，下一条请求 则会转发给B ，如果在有一条请求来 则又发给A 最后的结果则是ABABAB交替处理请求

打开nginx.conf配置文件 新增配置文件 默认情况下 nginx将会在 该两个服务器来回切换执行

权重配置：

upstream backserver{} 用在nginx 里面是配置 集群的服务器 如上图 weight 配置的该服务器的权重，上面 配置的是2:1 并不是 请求8080的服务器有2次 请求8081的有1次 而是比例：2:1 如果配置 是5：3并不是 一个请求5次一个请求3次

ip绑定：

直接在配置文件里面添加ip\_hash 则可以实现ip绑定，ip绑定 和什么浏览器没有关系，是和本机电脑ip有关系，A B 两个客户端，如果 nginx配置ip绑定以后 A 客户端如果请求的 是8080 服务器，则以后来自客户端的A 的请求都将交给8080服务器处理，如果是IP 绑定，在高并发的情况下，也没有什么作用，但是 ip绑定可以实现 会话共享

nginx负载均衡的集群的时候，提供的三种策略配置

## springboot

SpringBoot

使用场景：1.有spring的地方 2. J2ee应用 3.微服务

@Controller：修饰class，用来创建处理http请求的对象

@RestController：Spring4之后加入的注解，原来在@Controller中返回json需要@ResponseBody来配合，如果直接用@RestController替代@Controller就不需要再配置

@ResponseBody，默认返回json格式。

@RequestMapping：配置url映射

事务控制在service中，@transaction()  required support

异步任务：@EnableAsync 发送短信，邮件，消息推送，运维任务

定时任务：@EnableScheduling

拦截器：@Configuration   extends WebMvcConfigurerAdapter

重写 addInterceptors

registry.addInterceptor(new OneInterceptor()).addPathPatterns(“/one/\*\*”);

主要是微服务 (全能框架 类似粘合剂)

快速搭建 spring 框架 2 约定大于配置 3 开箱即用 4 嵌入了web容器

（1）为所有 Spring 开发提供一个更快更广泛的入门体验。

（2）零配置。无冗余代码生成和 XML 强制配置，遵循“约定大于配置” 。

（3）集成了大量常用的第三方库的配置,Spring Boot 应用为这些第三方库提供了几乎可以零配置的开箱即用的能力。

（4）提供一系列大型项目常用的非功能性特征，如嵌入式服务器、安全性、度量、运行状况检查、外部化配置等。

（5）Spring Boot 不是Spring 的替代者，Spring 框架是通过 IOC 机制来管理 Bean 的。Spring Boot 依赖 Spring 框架来管理对象的依赖。Spring Boot 并不是Spring 的精简版本，而是为使用 Spring 做好各种产品级准备

启动类注解

@SpringBootApplication

@SpringBootApplication是一个启动类注解，包括@ComponentScan，和@SpringBootConfiguration，@EnableAutoConfiguration。

@SpringBootConfiguration继承自@Configuration，二者功能也一致，标注当前类是配置类，并会将当前类内声明的一个或多个以@Bean注解标记的方法的实例纳入到srping容器中，并且实例名就是方法名。

@EnableAutoConfiguration的作用启动自动的配置，@EnableAutoConfiguration注解的意思就是Springboot根据你添加的jar包来配置你项目的默认配置，比如根据spring-boot-starter-web ，来判断你的项目是否需要添加了webmvc和tomcat，就会自动的帮你配置web项目中所需要的默认配置。在下面博客会具体分析这个注解，快速入门的demo实际没有用到该注解。

@ComponentScan，扫描当前包及其子包下被@Component，@Controller，@Service，@Repository注解标记的类并纳入到spring容器中进行管理。是以前的<context:component-scan>（以前使用在xml中使用的标签，用来扫描包配置的平行支持）。所以本demo中的User为何会被spring容器管理。

## Springcloud

### 什么是Spring Cloud？

SpringCloud是一系列框架的有序集合，它是基于springboot的，通过springboot的开发便利性巧妙的简化了分布式系统基础设施的开发，如:注册中心与发现,配置中心,智能路由,消息总线,负载均衡,断路器,数据监控等,都可以用springboot的开发风格做到一键启动和部署，springcloud并没有重复的制造轮子,它只是将各家公司开发比较成熟、还经得起考验的框架组合起来，再次封装屏蔽掉了一些复杂的配置和实现原理，最后给开发者留出了简单易懂的、容易部署的、容易维护的分布式系统的开发工具包。

1. **为什么要使用springcloud？**

现在不论是商业应用还是用户应用，在开发的初期都是使用单体架构，但是随着逐渐业务的扩展，单体架构的应用也会变得越来越来复杂,这时会带来如下的几个问题:

1. 代码的结构混乱：

业务复杂，导致代码量增大，管理起来也会变得困难，同时，也会给业务的快速迭代来巨大的挑战;

1. 开发效率低：

因为单体架构是一套代码，大家同时开发一套，很难避免代码冲突，还伴随着不断解决冲突的过程，这样的话会严重影响开发效率;

1. 排查解决问题成本高:

线上业务发现bug，修复bug的话简单，但是需要重新编译、打包、上线，成本很高。

由于单体结构的应用随着系统复杂度的增高，会暴露出各种各样的问题。近些年来，微服务架构逐渐取代了单体架构，且这种趋势将会越来越流行。Spring Cloud是目前最常用的微服务开发框架，已经在企业级开发中大量的应用。

### 3.Spring Cloud的优缺点？

优点：

（1）前后端分离开发，便于快速开发

（2）采用去中心化思想，服务之间采用Restful等轻量级通讯，比ESB更轻量

（3）利于扩展模块，乐高积木一般

（4）Springcloud的节点不会影响另一个节点

（5）适于互联网时代，产品迭代周期更短，如果用来做电商网站的话，性能是非常好的

缺点： 配置复杂，节点多，涉及的知识点多，所以治理成本高，不利于维护系统，且对团队挑战大

总的来说优点大过于缺点，目前看来Spring Cloud是一套非常完善的分布式框架，目前很多企业开始用微服务、Spring Cloud的优势是显而易见的。因此对于想研究微服务架构的同学来说，学习Spring Cloud是一个不错的选择。

4.SpringBoot和SpringCloud的区别？

（1）Springboot是单体个体的开发，Springcloud是分布式（多个）的开发

（2）SpringBoot可以离开SpringCloud独立使用开发项目， 但是SpringCloud离不开SpringBoot ，属于依赖的关系.

（3）SpringBoot专注于快速、方便的开发单个微服务个体，SpringCloud关注全局的服务治理框架

### 5.springcloud五大组件

现在介绍一下springcloud的五大组件：

Eureka注册中心与发现、ribbon负载均衡、feign接口调用、hystrix熔断器、zuul服务网关

1.Eureka

Eureka是服务注册中心，分别是eurekaserver服务端和eurekaclient客户端

eurekaserver服务端：里面有个注册中心，注册中心保存的是各个服务所在的机器和端口号

eurekaclient客户端：将服务注册到注册中心中并且发现服务

工作原理：在启动应用后，eureka客户端会向服务器注册自己的服务，同时服务端的信息会缓存到本地,客户端会向服务器周期性的心跳交互更新服务信息。

2.ribbon组建介绍

ribbon在springcloud中主要做负载均衡的作用.它是基于http协议和tcp协议的客户端负载均衡,使得面向REST请求时变成客户端的负载均衡调用.

3.feign组建介绍

feign 组建其实是整合和ribbon的.feign主要是进行远程调用,它通过@FeginClient的注解,进行动态代理(要是某个接口上面标注有这个注解,那么feign会对这个接口进行动态代理)

feign的动态代理,会根据你请求的@RequestMapping 和参数,来动态构造你要请求的服务地址,然后根据地址发送请求.

如果服务部署了多台机器,那么feign会通过负载均衡ribbon的轮询机制确实服务机器(ribbon默认采用轮询机制).

4.Hystrix组建

在微服务架构中,一个系统会有多个服务,服务之间的相互调用如果不做任何保护就会导致服务器雪崩问题.那么Hystrix就是用来解决这些问题的.Hystrix是隔离,熔断以及降级的一个框架,

在某个服务挂掉之后,Hystrix会熔断这个服务,直接返回.(降级:在这个服务熔断之后,记录这期间服务请求的信息,以便后期维护).

5.zuul网关

zuul:微服务网关,主要是负责网络路由的.

在微服务的架构中,可能后台会部署几百个服务,所有的服务名称前端不可能全部记住.那么现在就需要网关,前端通过浏览器发送请求,进入到网关zuul,网关会根据请求的一些特征,将请求转发到后端的服务.

网关还可以做统一的服务降级,限流,认证权限安全等等.

6.springcloud 常见面试问题

1.springcloud 与dubbo的区别

服务调用方式不同,dubbo是采用rpc远程调用,srpingcloud是采用rest api远程调用.

注册中心:dubbo是采用zookeeper,springcloud是采用eureka

服务网关不同:dubbo本身是没有实现的,只是整合了第三方的技术.而springcloud是有一个组建zuul作为路由网关,springcloud还支持断路器,与git完美集成分布式配置文件支持版本控制.

2.微服务之间是如何独立通信的?

feign远程调用.优点:没有中间代理件,简单常见.缺点:可用性低,在请求过程中服务端和客户端都必须是可用的

消息中间件.优点:解耦,提高可用性,可以支持多种相应模式(如:发布/订阅,请求/异步响应等等).缺点:增加系统的复杂度,万一消息中间件挂了,整个服务就挂了.

3.eureka和zookeeper的区别

eureka取用cap的ap,而zookeeper选用cp注重一致性(C:一致性.A:可用性.P:分区容错性.)

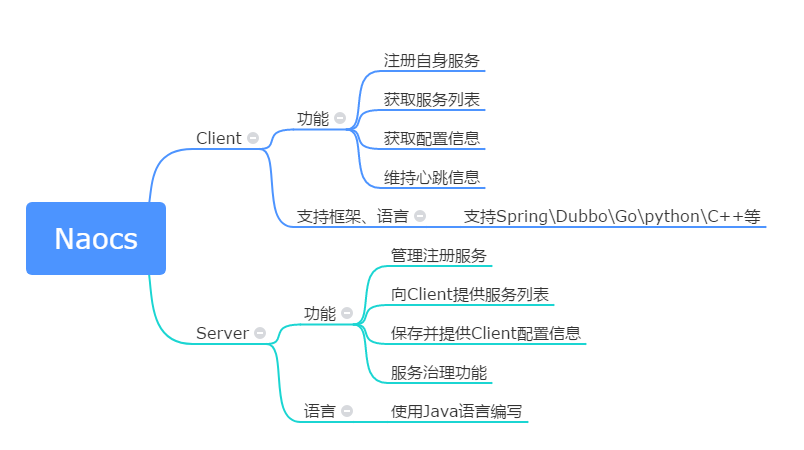
在服务崩溃时,eureka有自我保护机制,各个节点都是平等关系,在一个节点崩溃时,微服务并不会盲目删除这个节点,而是进入保护状态,此时可以正常注册,也可以提供查询服务,但是不与其他节点同步信息,其他节点可以正常使用.zookeeper分为leader和follower两种角色,采用半数存活的原则,超过一般的节点挂机就会使整个服务崩溃 而且在leader崩溃之后需要重新选举,在此期间服务是不可用的.

什么是Nacos

Nacos是阿里巴巴最新开源的项目，核心定位是“一个更易于帮助构建云原生应用的动态服务发现、配置和服务管理平台”

Nacos 致力于帮助您发现、配置和管理微服务。Nacos 提供了一组简单易用的特性集，帮助您快速实现动态服务发现、服务配置、服务元数据及流量管理。 是Spring Cloud A 中的服务注册发现组件，类似于Consul、Eureka，同时它又提供了分布式配置中心的功能，这点和Consul的config类似，支持热加载。

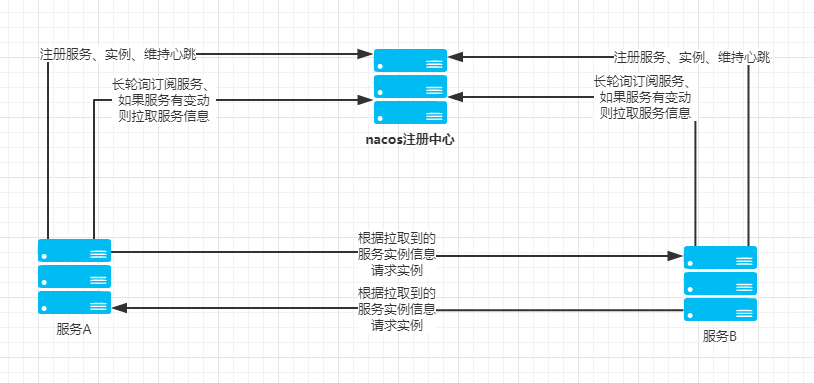
**Nacos原理**



nacos简单介绍

Nacos注册中心分为server与client，server采用Java编写，为client提供注册发现服务与配置服务。而client可以用多语言实现，client与微服务嵌套在一起，nacos提供sdk和openApi，如果没有sdk也可以根据openApi手动写服务注册与发现和配置拉取的逻辑

**注册中心原理**



服务注册原理

注册的策略就是每5秒会向向nacossever发送一次心跳，心跳会带上服务名、IP、服务端口号等，nacosserver会主动的向nacosClent客户端发送健康检查心跳，支持tcp/http协议，15秒内如果没有客户端的心跳认为健康失败

,30秒内还是没有接收到心跳就把改实例给剔除。

Dubbo

什么是dubno？

Dubbo是阿里巴巴开源的基于 Java 的高性能 RPC 分布式服务框架，现已成为 Apache 基金会孵化项目。

面试官问你如果这个都不清楚，那下面的就没必要问了。

官网：http://dubbo.apache.org

2、为什么要用Dubbo？

因为是阿里开源项目，国内很多互联网公司都在用，已经经过很多线上考验。内部使用了 Netty、Zookeeper【nacos】，保证了高性能高可用性。

使用 Dubbo 可以将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，可用于提高业务复用灵活扩展，使前端应用能更快速的响应多变的市场需求。

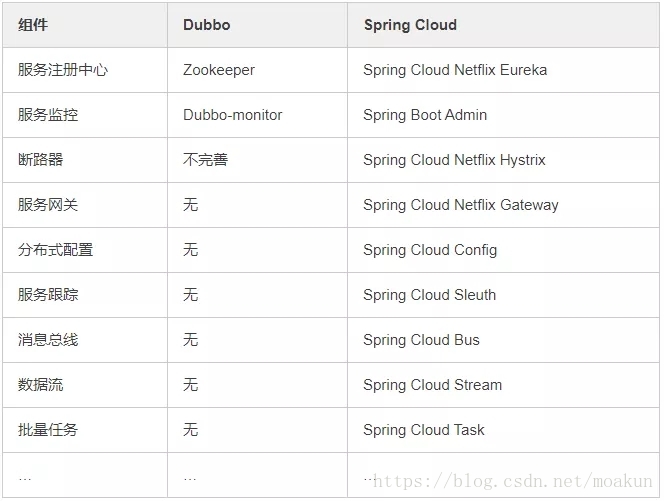
3.Dubbo 和 Spring Cloud 有什么区别？

两个没关联，如果硬要说区别，有以下几点。

1）通信方式不同

Dubbo 使用的是 RPC 通信，而 Spring Cloud 使用的是 HTTP RESTFul 方式。

2）组成部分不同



4、dubbo都支持什么协议，推荐用哪种？

dubbo://（推荐）

rmi://

hessian://

http://

webservice://

thrift://

memcached://

redis://

rest://

5、Dubbo需要 Web 容器吗？

不需要，如果硬要用 Web 容器，只会增加复杂性，也浪费资源。

6、Dubbo内置了哪几种服务容器？

Spring Container

Jetty Container

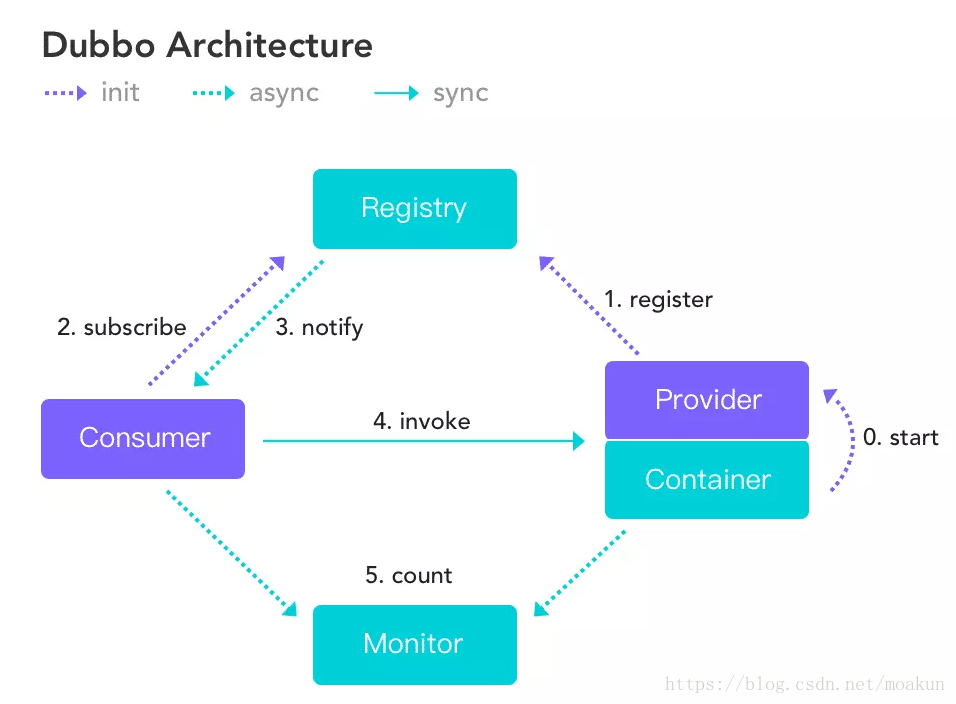
Log4j Container

Dubbo 的服务容器只是一个简单的 Main 方法，并加载一个简单的 Spring 容器，用于暴露服务。

7.Dubbo里面有哪几种节点角色？



8、画一画服务注册与发现的流程图



该图来自 Dubbo 官网，供你参考，如果你说你熟悉 Dubbo, 面试官经常会让你画这个图，记好了。

运行原理

（1）启动容器，相当于在启动Dubbo的Provider

（2）启动后回去注册中心进行注册，注册所有可以提供的服务列表

（3）在Consumer启动后会去Registry中获取服务列表和Provider的地址，进行订阅

（4）当Provider有修改后，注册中心会把消息推送给Consummer，使用了观察者设计模

式

（5）根据获取到的Provider地址，真实调用Provider中的功能，在consummer方使用了代理设计模式，创建了一个Provider方类的一个代理对象。通过代理对象获取Provider中的真实功能，起到保护Provider真实功能的作用。

（6）Consumer和Provider每隔一分钟会向Monitor发送统计信息，统计信息包括访问次数，频率等。

9、Dubbo默认使用什么注册中心，还有别的选择吗？

推荐使用 Zookeeper 作为注册中心，新版推荐使用nacos做注册中心。

10、Dubbo有哪几种配置方式？

1）Spring 配置方式 xml文件和注解

2）Java API 配置方式

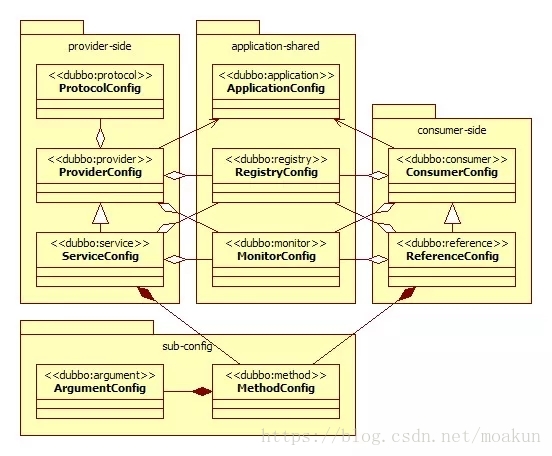
11.Dubbo 核心的配置有哪些？

我曾经面试就遇到过面试官让你写这些配置，我也是蒙逼。。

IMG_256



配置之间的关系见下图。



12、在 Provider 上可以配置的 Consumer 端的属性有哪些？

1）timeout：方法调用超时

2）retries：失败重试次数，默认重试 2 次

3）loadbalance：负载均衡算法，默认随机

4）actives 消费者端，最大并发调用限制

13、Dubbo启动时如果依赖的服务不可用会怎样？

Dubbo 缺省会在启动时检查依赖的服务是否可用，不可用时会抛出异常，阻止 Spring 初始化完成，默认 check="true"，可以通过 check="false" 关闭检查。

14、Dubbo推荐使用什么序列化框架，你知道的还有哪些？

推荐使用Hessian序列化，还有Duddo、FastJson、Java自带序列化。

15、Dubbo默认使用的是什么通信框架，还有别的选择吗？

Dubbo 默认使用 Netty 框架，也是推荐的选择，另外内容还集成有Mina、Grizzly。

16、Dubbo有哪几种集群容错方案，默认是哪种？



17、Dubbo有哪几种负载均衡策略，默认是哪种？

IMG_260



18、注册了多个同一样的服务，如果测试指定的某一个服务呢？

可以配置环境点对点直连，绕过注册中心，将以服务接口为单位，忽略注册中心的提供者列表。

19、Dubbo支持服务多协议吗？

Dubbo 允许配置多协议，在不同服务上支持不同协议或者同一服务上同时支持多种协议。

20、当一个服务接口有多种实现时怎么做？

当一个接口有多种实现时，可以用 group 属性来分组，服务提供方和消费方都指定同一个 group 即可。

21、服务上线怎么兼容旧版本？

可以用版本号（version）过渡，多个不同版本的服务注册到注册中心，版本号不同的服务相互间不引用。这个和服务分组的概念有一点类似。

22、Dubbo可以对结果进行缓存吗？

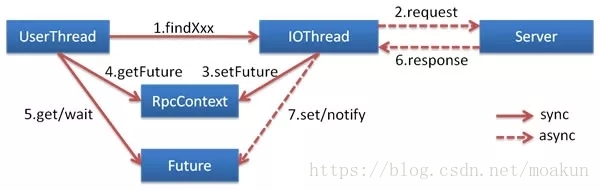
可以，Dubbo 提供了声明式缓存，用于加速热门数据的访问速度，以减少用户加缓存的工作量。

23、Dubbo服务之间的调用是阻塞的吗？

默认是同步等待结果阻塞的，支持异步调用。

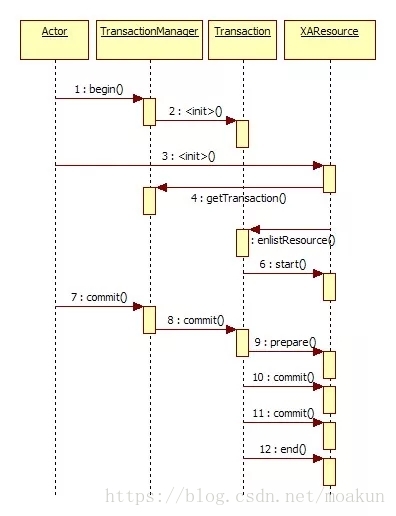
Dubbo 是基于 NIO 的非阻塞实现并行调用，客户端不需要启动多线程即可完成并行调用多个远程服务，相对多线程开销较小，异步调用会返回一个 Future 对象。

异步调用流程图如下。



24、Dubbo支持分布式事务吗？

目前暂时不支持，后续可能采用基于 JTA/XA 规范实现，如以图所示。



25、Dubbo telnet 命令能做什么？

dubbo 通过 telnet 命令来进行服务治理，具体使用看这篇文章《[dubbo服务调试管理实用命令](https://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247483709&idx=1&sn=afe0688c184f00902529583a85d90089&scene=21" \l "wechat_redirect)》。

telnet localhost 8090

26、Dubbo支持服务降级吗？

Dubbo 2.2.0 以上版本支持。

27、Dubbo如何优雅停机？

Dubbo 是通过 JDK 的 ShutdownHook 来完成优雅停机的，所以如果使用 kill -9 PID 等强制关闭指令，是不会执行优雅停机的，只有通过 kill PID 时，才会执行。

28、服务提供者能实现失效踢出是什么原理？

服务失效踢出基于 Zookeeper 的临时节点原理。

29、如何解决服务调用链过长的问题？

Dubbo 可以使用 Pinpoint 和 Apache Skywalking(Incubator) 实现分布式服务追踪，当然还有其他很多方案。

30、服务读写推荐的容错策略是怎样的？

读操作建议使用 Failover 失败自动切换，默认重试两次其他服务器。

写操作建议使用 Failfast 快速失败，发一次调用失败就立即报错。

31、Dubbo必须依赖的包有哪些？

Dubbo 必须依赖 JDK，其他为可选。

32、Dubbo的管理控制台能做什么？

管理控制台主要包含：路由规则，动态配置，服务降级，访问控制，权重调整，负载均衡，等管理功能。

33、说说 Dubbo 服务暴露的过程。

Dubbo 会在 Spring 实例化完 bean 之后，在刷新容器最后一步发布 ContextRefreshEvent 事件的时候，通知实现了 ApplicationListener 的 ServiceBean 类进行回调 onApplicationEvent 事件方法，Dubbo 会在这个方法中调用 ServiceBean 父类 ServiceConfig 的 export 方法，而该方法真正实现了服务的（异步或者非异步）发布。

34、Dubbo 停止维护了吗？

2014 年开始停止维护过几年，17 年开始重新维护，并进入了 Apache 项目。

35、Dubbo 和 Dubbox 有什么区别？

Dubbox 是继 Dubbo 停止维护后，当当网基于 Dubbo 做的一个扩展项目，如加了服务可 Restful 调用，更新了开源组件等。

36、你还了解别的分布式框架吗？

别的还有 Spring cloud、Facebook 的 Thrift、Twitter 的 Finagle 等。

37、Dubbo 能集成 Spring Boot 吗？

可以的，项目地址如下。

https://github.com/apache/incubator-dubbo-spring-boot-project

38、在使用过程中都遇到了些什么问题？

Dubbo 的设计目的是为了满足高并发小数据量的 rpc 调用，在大数据量下的性能表现并不好，建议使用 rmi 或 http 协议。

39、你读过 Dubbo 的源码吗？

要了解 Dubbo 就必须看其源码，了解其原理，花点时间看下吧，网上也有很多教程，后续有时间我也会在公众号上分享 Dubbo 的源码。

40、你觉得用 Dubbo 好还是 Spring Cloud 好？

扩展性的问题，没有好坏，只有适合不适合，不过我好像更倾向于使用 Dubbo, Spring Cloud 版本升级太快，组件更新替换太频繁，配置太繁琐，还有很多我觉得是没有 Dubbo 顺手的地方……

## seata概念

什么是Seata？

Seata是阿里巴巴推出的一款用来解决分布式事务问题的框架，他经过天猫双十一的考验，很有可能成为解决分布式事务问题的主流框架

分布式事务产生的背景？

在传统的单体项目中，多个不同的业务逻辑使用的都是同一个数据源，使用的都是同一个事务管理器，所以不会存在事务问题。

在分布式或者微服务架构中，每个服务都有自己的数据源，使用不同事务管理器，如果A服务去调用B服务，B服务执行失败了，A服务的事务和B服务的事务都会回滚，这时候是不存在事务问题的，但是如果A服务B服务执行成功之后出现异常，A服务的事务会回滚，但是B服务的事务不会回滚，此时就存在分布式事务问题。

（在单体的项目中，有多个不同的数据源，每个数据源中都有自己独立的事务管理器，互不影响，那么这时候也会存在多数据源事务管理：解决方案jta+ atominc）

Seata有三个组成部分：

事务协调器TC：协调

事务管理器TM：发起方

资源管理器RM：参与方

只要写网络通讯框架就要用到netty

Seata实现原理

1.TM（发起方）连接到我们的TC事务协调者，创建一个全局的事务的xid，保存到ThreadLoacl中；

2.TM（发起方）和RM(参与方)都被Seata的数据数据源实现代理，在原生的sql之前和之后保存原来和修改后日志到undo\_log中，方便后期实现回滚。

3.TM（发起方）使feign客户端调用接口时候，在ThreadLoacl中获取xid，设置到请求头中；

4.RM(参与方)从请求中获取到该xid，设置到ThreadLoacl中，同时也会向seataserver注册该分支事务。

5.TM(发起方)将当前本地事务的结果，告诉给协调者TC，协调者TC在通知所有的分支是否回滚。

6. TM(发起方)如果调用接口成功之后抛出异常的情况下，告诉给协调者TC，协调者TC在通知所有的分支根据根据全局的xid和分支事务的id 查询分支数据源的undo\_log日志表逆向生成sql语句实现回滚，同时删除对应的undo\_log日志

7. TM(发起方)如果调用接口成功之后没有抛出任何的异常，告诉给协调者TC，协调者TC在通知所有的分支根据根据全局的xid和分支事务的id 查询分支数据源的 删除对应的undo\_log日志表

Seata原理分析

1.发起方TM会向我们的TC协调者申请一个全局的事务id，保存到threadlocal中；

2.TM和RM都会被Seata代理数据源，写入原来和修改后内容保存到undo\_log表中；

3.TM从请求头中传递该全局的事务id给RM，RM从请求中头中获取到该全局事务id，并且注册该分支。

4.如果TM调用接口成功之后，如果报错的情况下则通知给协调者，协调者在告诉所有的分支都开始回滚，直接根据本地事务id+xid查询undo\_log表 ，逆向生成sql语句回滚，同时删除该undo\_log日志。

5.如果TM调用接口成功之后，如果没有报错的情况下则通知给协调者，协调者在告诉所有的分支都开始提交事务，直接根据本地事务id+xid删除对应的undo\_log表记录即可。

# RabbitMQ

削峰 异步 解耦

面对我们一些高并发场景 ;我们的项目可能会扛不住;

响应慢;

请求量大;

消息队列是什么 就是可以暂时存储生产者的数据;暂存 等待消费者来消费;

削峰 就是我们有3000个请求 生产者将数据交给消息队列 就可以返回了; 我们消费者可以先拿1000个数据消费;

解耦 我们有 ABCD 四个服务 a 生产出一个id ;

放入 消息队列中 其他服务调用与否 我们a是不用关心的

异步 就是我们数据生产出来了 ;我们可以一会消费它;

生产者将数据放入 消息队列中就可以做其他的了;

消费者可以根据策略再去选择消费数据;

我们的消息队列 有两大核心 交换机 和队列组成的

有三种推送模式

**Direct Exchange**

直连型交换机，根据消息携带的路由键将消息投递给对应队列。

大致流程，有一个队列绑定到一个直连交换机上，同时赋予一个路由键 routing key 。

然后当一个消息携带着路由值为X，这个消息通过生产者发送给交换机时，交换机就会根据这个路由值X去寻找绑定值也是X的队列。

**Fanout Exchange**

扇型交换机，这个交换机没有路由键概念，就算你绑了路由键也是无视的。 这个交换机在接收到消息后，会直接转发到绑定到它上面的所有队列。

**Topic Exchange**

主题交换机，这个交换机其实跟直连交换机流程差不多，但是它的特点就是在它的路由键和绑定键之间是有规则的。

**消息确认:Message acknowledgment**

在实际应用中，可能会发生消费者收到Queue中的消息，但没有处理完成就宕机（或出现其他意外）的情况，这种情况下就可能会导致消息丢失。为了避免这种情况发生，我们可以要求消费者在消费完消息后发送一个回执给RabbitMQ，RabbitMQ收到消息回执（Message acknowledgment）后才将该消息从Queue中移除；如果RabbitMQ没有收到回执并检测到消费者的RabbitMQ连接断开，则RabbitMQ会将该消息发送给其他消费者（如果存在多个消费者）进行处理。这里不存在Timeout概念，一个消费者处理消息时间再长也不会导致该消息被发送给其他消费者，除非它的RabbitMQ连接断开。

这里会产生另外一个问题，如果我们的开发人员在处理完业务逻辑后，忘记发送回执给RabbitMQ，这将会导致严重的问题，Queue中堆积的消息会越来越多，消费者重启后会重复消费这些消息并重复执行业务逻辑。

如果我们采用no-ack的方式进行确认，也就是说，每次Consumer接到数据后，而不管是否处理完成，RabbitMQ会立即把这个Message标记为完成，然后从queue中删除了。

**Dead Letter Exchange（死信交换器）**

在队列上指定一个Exchange，则在该队列上发生如下情况，

1.消息被拒绝（basic.reject or basic.nack)，且requeue=false

2.消息过期而被删除（TTL）

3.消息数量超过队列最大限制而被删除

4.消息总大小超过队列最大限制而被删除

就会把该消息转发到指定的这个exchange

需要定义了x-dead-letter-exchange属性，同时也可以指定一个可选的x-dead-letter-routing-key，表示默认的routing-key，如果没有指定，则使用消息原来的routeing-key进行转发

当定义队列时指定了x-dead-letter-exchange（x-dead-letter-routing-key视情况而定），并且消费端执行拒绝策略的时候将消息路由到指定的Exchange中去。

我们知道还有二种情况会造成消息转发到死信队列。

一种是消息过期而被删除，可以使用这个方式使的rabbitmq实现延迟队列的作用。还有一种就是消息数量超过队列最大限制而被删除或者消息总大小超过队列最大限制而被删除