# java的项目流程

1.需求分析

2.概要设计

3.详细设计(用例图，流程图，类图)

4.数据库设计(powerdesigner)

5.代码开发（编写）

6.单元测试（junit 白盒测试）(开发人员)

svn版本管理工具(提交，更新代码，文档)

7.集成测试 （黑盒测试，loadrunner（编写测试脚本）(高级测试)）

8.上线试运行 （用户自己体验）

9.压力测试（loadrunner）

10.正式上线

11.维护

# 一、 java基础

## 1、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

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

## 2、修饰符

(1)、权限修饰符（权限修饰符不可以同时使用，可以和普通修饰符同时使用。）

public 公共的 在本类本包不同包中的父子类中(任意位置)

proteced 受保护的 在本类本包不同包中的父子类中

default 默认的 在本类本包中

private 私有的 只限于本类中

(2)、普通修饰符

abstract 抽象的

final 最终的

static 静态的

## 3、数据类型

(1).基本数据类型(四大类八大种)

整数型

字节型 byte (1个字节) 取值范围：-128---127(是java中最小得储存空间)

短整型 short (2个字节) 取值范围：-32768---32767

整型 int (4个字节) 取值范围：-2147483648---2147483647

长整型 long (8个字节)

浮点型

单精度浮点型 float (4个字节) 有效位数6-7位

双精度浮点型 double (8个字节) 有效位数15位

字符型

字符型 char (2个字节) 只能储存一个(unicode)字符

布尔类型

布尔类型 boolean (1个字节) true(真) false(假)

(2).引用数据类型

Array 数组 Class 类 Interface 接口

(3)包装类

基本数据类型 包装类

byte Byte

short Short

char Character

int Integer

long Long

float Float

double Double

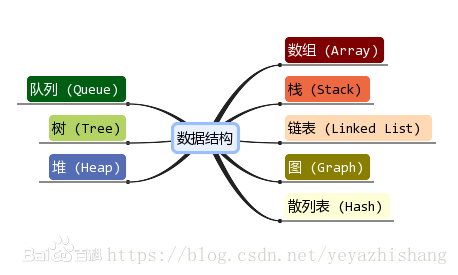
boolean Boolean

将String类型转换成基本数据类型：包装类.parse基本数据类型(); (包装类与基本数据类型对应)

将基本数据类型转换成String类型：String.valueOf();

## 4、数据结构

数据结构是指相互之间存在着一种或多种关系的数据元素的集合和该集合中数据元素之间的关系组成 。  
常用的数据结构有：数组，栈，链表，队列，树，图，堆，散列表等，如图所示：



**1、数组**

数组是可以再内存中连续存储多个元素的结构，在内存中的分配也是连续的，数组中的元素通过数组下标进行访问，数组下标从0开始。例如下面这段代码就是将数组的第一个元素赋值为 1。

int[] data = new int[100]；data[0] = 1;

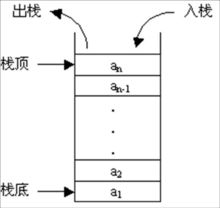
优点：  
1、按照索引查询元素速度快  
2、按照索引遍历数组方便

缺点：  
1、数组的大小固定后就无法扩容了  
2、数组只能存储一种类型的数据  
3、添加，删除的操作慢，因为要移动其他的元素。

适用场景：频繁查询，对存储空间要求不大，很少增加和删除的情况。

**2、栈**

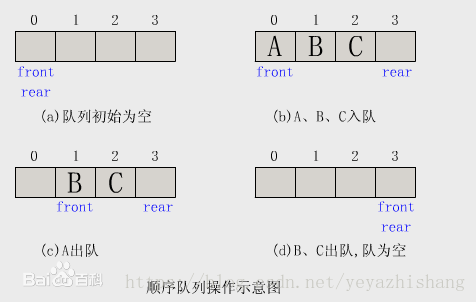
栈是一种特殊的线性表，仅能在线性表的一端操作，栈顶允许操作，栈底不允许操作。 栈的特点是：先进后出，或者说是后进先出，从栈顶放入元素的操作叫入栈，取出元素叫出栈。



栈的结构就像一个集装箱，越先放进去的东西越晚才能拿出来，所以，栈常应用于实现递归功能方面的场景，例如斐波那契数列。

**3、队列**

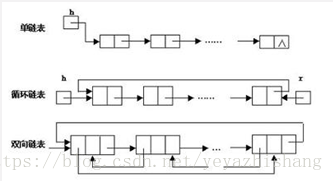
队列与栈一样，也是一种线性表，不同的是，队列可以在一端添加元素，在另一端取出元素，也就是：先进先出。从一端放入元素的操作称为入队，取出元素为出队，示例图如下：



使用场景：因为队列先进先出的特点，在多线程阻塞队列管理中非常适用。

**4、链表**

链表是物理存储单元上非连续的、非顺序的存储结构，数据元素的逻辑顺序是通过链表的指针地址实现，每个元素包含两个结点，一个是存储元素的数据域 (内存空间)，另一个是指向下一个结点地址的指针域。根据指针的指向，链表能形成不同的结构，例如单链表，双向链表，循环链表等。



链表的优点：

链表是很常用的一种数据结构，不需要初始化容量，可以任意加减元素；

添加或者删除元素时只需要改变前后两个元素结点的指针域指向地址即可，所以添加，删除很快；

缺点：

因为含有大量的指针域，占用空间较大；

查找元素需要遍历链表来查找，非常耗时。

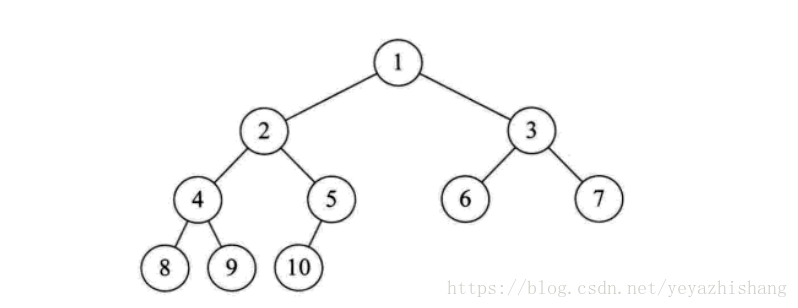
适用场景：数据量较小，需要频繁增加，删除操作的场景

**5、树**

**树**是一种数据结构，它是由n（n>=1）个有限节点组成一个具有层次关系的集合。把它叫做 “树” 是因为它看起来像一棵倒挂的树，也就是说它是根朝上，而叶朝下的。它具有以下的特点：

* 每个节点有零个或多个子节点；
* 没有父节点的节点称为根节点；
* 每一个非根节点有且只有一个父节点；
* 除了根节点外，每个子节点可以分为多个不相交的子树；

在日常的应用中，我们讨论和用的更多的是树的其中一种结构，就是**二叉树**。



二叉树是树的特殊一种，具有如下特点：

1、每个结点最多有两颗子树，结点的度最大为2。

2、左子树和右子树是有顺序的，次序不能颠倒。

3、即使某结点只有一个子树，也要区分左右子树。

二叉树是一种比较有用的折中方案，它添加，删除元素都很快，并且在查找方面也有很多的算法优化，所以，二叉树既有链表的好处，也有数组的好处，是两者的优化方案，在处理大批量的动态数据方面非常有用。

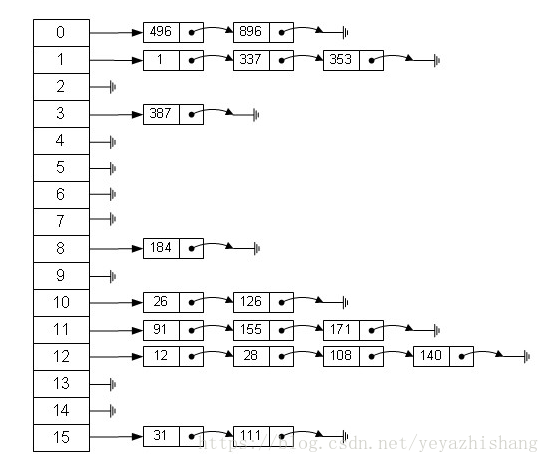
扩展：二叉树有很多扩展的数据结构，包括平衡二叉树、红黑树、B+树等，这些数据结构二叉树的基础上衍生了很多的功能，在实际应用中广泛用到，例如mysql的数据库索引结构用的就是B+树，还有HashMap的底层源码中用到了红黑树。这些二叉树的功能强大，但算法上比较复杂，想学习的话还是需要花时间去深入的。

**6、散列表**

散列表，也叫哈希表，是根据关键码和值 (key和value) 直接进行访问的数据结构，通过key和value来映射到集合中的一个位置，这样就可以很快找到集合中的对应元素。**记录的存储位置=f(key)**

这里的对应关系 f 成为散列函数，又称为哈希 (hash函数)，而散列表就是把Key通过一个固定的算法函数既所谓的哈希函数转换成一个整型数字，然后就将该数字对数组长度进行取余，取余结果就当作数组的下标，将value存储在以该数字为下标的数组空间里，这种存储空间可以充分利用数组的查找优势来查找元素，所以查找的速度很快。

哈希表在应用中也是比较常见的，就如Java中有些集合类就是借鉴了哈希原理构造的，例如HashMap，HashTable等，利用hash表的优势，对于集合的查找元素时非常方便的，然而，因为哈希表是基于数组衍生的数据结构，在添加删除元素方面是比较慢的，所以很多时候需要用到一种数组链表来做，也就是拉链法。拉链法是数组结合链表的一种结构，较早前的hashMap底层的存储就是采用这种结构，直到jdk1.8之后才换成了数组加红黑树的结构，其示例图如下：



从图中可以看出，左边很明显是个数组，数组的每个成员包括一个指针，指向一个链表的头，当然这个链表可能为空，也可能元素很多。我们根据元素的一些特征把元素分配到不同的链表中去，也是根据这些特征，找到正确的链表，再从链表中找出这个元素。

哈希表的应用场景很多，当然也有很多问题要考虑，比如哈希冲突的问题，如果处理的不好会浪费大量的时间，导致应用崩溃。

**7、堆**

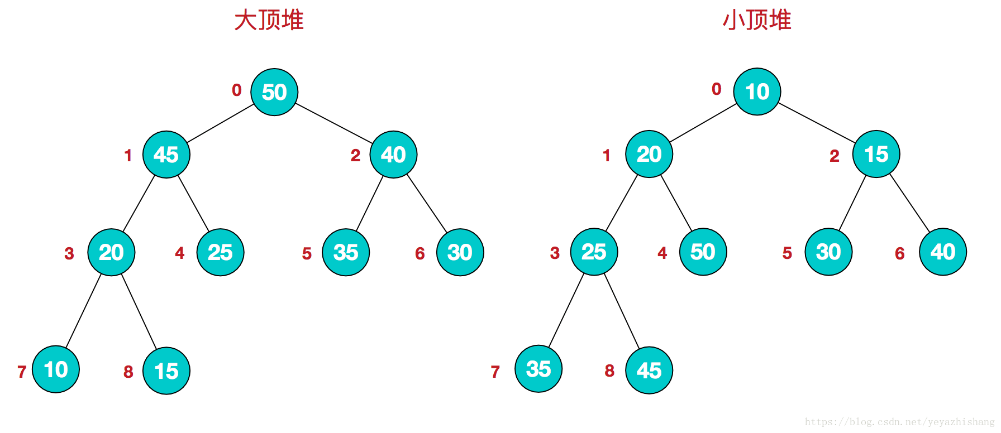
堆是一种比较特殊的数据结构，可以被看做一棵树的数组对象，具有以下的性质：

* 堆中某个节点的值总是不大于或不小于其父节点的值；
* 堆总是一棵完全二叉树。

将根节点最大的堆叫做最大堆或大根堆，根节点最小的堆叫做最小堆或小根堆。常见的堆有二叉堆、斐波那契堆等。

堆的定义如下：n个元素的序列{k1,k2,ki,…,kn}当且仅当满足下关系时，称之为堆。

(ki <= k2i,ki <= k2i+1)或者(ki >= k2i,ki >= k2i+1), (i = 1,2,3,4…n/2)，满足前者的表达式的成为小顶堆，满足后者表达式的为大顶堆，这两者的结构图可以用完全二叉树排列出来，示例图如下：

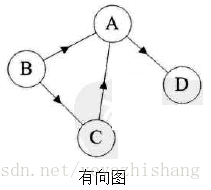
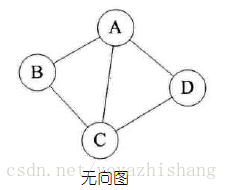


因为堆有序的特点，一般用来做数组中的排序，称为堆排序。

**8、图**

图是由结点的有穷集合V和边的集合E组成。其中，为了与树形结构加以区别，在图结构中常常将结点称为顶点，边是顶点的有序偶对，若两个顶点之间存在一条边，就表示这两个顶点具有相邻关系。

按照顶点指向的方向可分为无向图和有向图：



图是一种比较复杂的数据结构，在存储数据上有着比较复杂和高效的算法，分别有邻接矩阵 、邻接表、十字链表、邻接多重表、边集数组等存储结构，这里不做展开，读者有兴趣可以自己学习深入

详情参考: <https://blog.csdn.net/yeyazhishang/article/details/82353846>

## 5、String的常用方法

char At(下标) 返回指定下标对应的字符,如果找不到返回空的字符串” ”;

index Of() 返回的是字符串对应的下标, 当字符串不存在的时候返回-1

split() 从指定字符串的位置分割，得到相应的字符串数组

substring() 从指定下标截取到字符串末尾(一个参数)

从开始的下标，截取到结束的下标的前一个字符 包前不包后(两个参数)

equals() 比较字符串是否相等，相等返回true，不相等返回false

contains() 包含 如果包含字符串，返回true，否则返回false

replace() 替换

concat(值1,值2,..) 拼接字符串,相当于+号

toLowerCase() 将字符串转为小写

toUpperCase() 将字符串转为大写

## 6、Date的常用方法

toString() 将日期转为字符串

toLocaleString() 将日期转为本地格式字符串

getFullYear()【否页r】 获取指定日期的年份

getMonth()【忙此】 获取指定日期的月份,返回0-11,0代表1月份，11代表12月份

getDate() 获取指定日期的天数,返回1-31

getDay() 获取指定日期的星期数,返回0-6,0代表星期日,6代表星期六

getHours()【阿窝斯】 获取指定日期的小时数,返回0-23

getMinutes()【米内斯】 获取指定日期的分钟数,返回0-59

getSeconds()【赛啃斯】 获取指定日期的秒数,返回0-59

getMilliseconds()【米林赛啃此】获取指定日期的毫秒数,返回0-999

setFullYear(年份) 设置年份

setMonth(月份) 设置月份

setDate(天数) 设置天数

setHours(小时数) 设置小时数

setMinutes(分钟数) 设置分钟数

setSeconds(秒数) 设置秒数

setMilliseconds(毫秒数) 设置毫秒数

getTime() 获取从1970年1月1号到指定日期经过的毫秒数

## 7、Session和Cookie的区别

1、cookie数据存放在客户的浏览器上，session数据放在服务器上

2、cookie存储在浏览器里不安全，session相对安全

3、cookie保存的数据不能超过4K，session理论上来说没有限cookie

4、cookie是预先设置的生存周期或永久的保存于本地的文件、session是浏览器页面一关 ,session就会消失

## 8、单例原型

单例模式的三要素：

私有的无参构造；

私有的静态的本类对象；

公共的静态的返回本类对象的方法

单例模式特点

单例类只能有一个实例。

单例类必须自己创建自己的唯一实例。

单例类必须给所有其他对象提供这一实例

单例原型的区别：

单例的内存地址是一样的，单例是共享的，单例节约内存

单例是在频繁进行创建和销原型毁的时候创建的

原型的内存地址是不一样的，值是不共享的，原型不节约内存

## 9、String StringBuilder StringBuffer三者区别

String 是一个字符串常量，长度固定

StringBuffer 是一个长度可以改变，线程安全，效率高于String

StringBuilder 是一个长度可以改变，线程不安全，效率高 于StringBuffer、String

## 10、equals和==

equals相等类似于==

==在比较基本数据类型时，比较的是值是否相等；而当==比较对象的时候，比较的是内存地址是否相等

在比较对象的值是否相等的时候我们应该使用equals，由于equals内部也使用的是==比较，所以我们需要重写equals方法

## 11、继承（关键字：extends）

1、写在子类中

2、方式：public class 子类 extends 父类{}

3、子类继承父类，拥有父类的所有方法和属性

4、子类不能继承父类的私有特性(私有属性和私有方法)，和构造方法

5、创建子类对象的时候，默认先调用父类的构造方法

6、一个子类只能有一个父类，一个父类看可以有多个子类，也就是java中的单继承

7、我们将类中相同的属性和方法向上提取封装成父类，增加代码的复用性，减少代码量（继承的作用）

## 12、重写与重载

**方法的重载**

1、在同一类中，方法名称相同，参数列表的个数、顺序、类型至少一个不同，那么就构成了方法的重载

2、方法的重载与返回值的类型和修饰符无关

**方法的重写**

1. 重写发生在继承关系中、父子类关系中

2. 当父类的方法 无法满足子类的需求，我们需要重写父类的方法（为什么）

3. 重写时（方法名 返回值类型 参数列表）必须和父类完全一致（怎么重写）

4. 子类不能重写父类的 私有方法 和 static方法（静态的，只限于本类）

5. 子类重写父类 访问权限不能小于父类

6. 子类重写父类的方法 不能抛出比父类更大的异常

**重写（overWrite）和重载（overLoad）的区别：**

1.重载要求在同一个类中，重写发生在存在继承关系的两个类或多个类中，要求是非静态非私有的方法（类）

2.重载要求参数的顺序、个数、类型至少有一个不同，与返回值类型无关，重写是参数和返回值类型必须相同（方法定义方式）

3.方法的重载与访问修饰符无关，重写要求子类的访问权限必须大于或者等于（不小于）父类（访问修饰符）

4.在重载中与异常无关，但是重写要求子类不能抛出比父类更多的异常（异常）

## 13、抽象类与接口

**抽象类**

1.abstract 修饰类 叫做抽象类

2.抽象类不能被实例化，如果要用抽象类里面的属性和方法，就创建一个类，去继承抽象类

3.abstract 修饰的方法称为抽象方法，抽象方法没有方法体

4.抽象方法必须写在抽象类中

5.抽象类中既可以有抽象方法又可以有普通方法

6.一个类继承了抽象类，就必须重写抽象类中的所有抽象方法

**接口（interface）**

1、接口中所有的属性都是常量，final修饰的变量称为常量，必须赋初始值，并且不能二次赋值 (属性默认的是public static final修饰的)

2、接口中的所有的方法都是抽象方法 (方法默认的是public abstract修饰的)

3、接口不能被实例化，如果要用接口中的属性和方法，就创建一个类，去实现接口

4、实现接口 必须重写接口中的所有抽象方法

5、一个类可以实现多个接口(多实现)

**接口和抽象类之间的关系**

共同点：

都可以有抽象方法，都不能被实例化

一个类要继承一个抽象类那么必须重写它里面的所有抽象方法，一个类要实现接口，也必须重写接口中的所有的（抽象）方法

不同点：

接口不是类，用interface定义，抽象类是一个用abstract修饰符修饰的类

接口里面只能有抽象方法，抽象类可以有普通方法和抽象方法

一个类可以实现多个接口，但是只能继承一个类

接口里面定义的变量都是 final修饰的常量，而抽象类则没有这个限制

## 14、Final（骡子）

Final 修饰的类，不能被继承

Final 修饰的属性是常量，必须赋初始值，不能二次赋值

Final 修饰的方法，不能被重写

## 15、什么是反射?(算命先生)

对类来说 我能知道所有属性和方法

对于对象来说 我能得到所有的属性值和调用它的方法

三种创建类的方法

1、类名.Class

2、对像 对象名=new 对象()

对象名.getClass();

3、class.forName(类的全路径);

里面的方法

类对象名.getDeclaredFields() 得到所有属性

类对象名.getFields() 得到所有公共的属性

类对象名.getDeclaredMethods() 获取类对象中所有的方法

类对象名.getMethods() 获取类对象中所有公共的方法

getName() 获取属性(方法)名

getType() 获取属性(方法)类型

getModifiers() 获取属性(方法)修饰符

getParameterTypes() 得到方法中所有参数集合

setAccessible(布尔类型) 强制访问

getAnnotation() 判断是否拥有注解

五种对象：类 属性 方法 构造函数 注解

## 16、注解

什么是注解？

Java 注解(Annotation)又称 Java 标注

Java 语言中的类、方法、变量、参数和包等都可以被标注。和 Javadoc 不同，Java 标注可以通过反射获取标注内容。在编译器生成类文件时，标注可以被嵌入到字节码中。Java 虚拟机可以保留标注内容，在运行时可以获取到标注内容 。 当然它也支持自定义 Java 标注

自定义注解

示例如下：

@Documented//元注解 可有可无

@Target(ElementType.*METHOD*) //元注解 声明注解的 用的地方(方法上)

@Retention(RetentionPolicy.*RUNTIME*) //元注解 声明注解的生命周期  
public @interface LoggingAnn {  
 String value() default "";  
}

**(1) @interface**

使用 @interface 定义注解时，意味着它实现了 java.lang.annotation.Annotation 接口，即该注解就是一个Annotation。定义 Annotation 时，@interface 是必须的。

注意：它和我们通常的 implemented 实现接口的方法不同。Annotation 接口的实现细节都由编译器完成。通过 @interface 定义注解后，该注解不能继承其他的注解或接口。

**(2) @Documented**

类和方法的 Annotation 在缺省情况下是不出现在 javadoc 中的。如果使用 @Documented 修饰该 Annotation，则表示它可以出现在 javadoc 中。

定义 Annotation 时，@Documented 可有可无；若没有定义，则 Annotation 不会出现在 javadoc 中。

**(3) @Target(ElementType.TYPE)**

前面我们说过，ElementType 是 Annotation 的类型属性。而 @Target 的作用，就是来指定 Annotation 的类型属性。

@Target(ElementType.TYPE) 的意思就是指定该 Annotation 的类型是 ElementType.TYPE。这就意味着，MyAnnotation1 是来修饰"类、接口（包括注释类型）或枚举声明"的注解。

定义 Annotation 时，@Target 可有可无。若有 @Target，则该 Annotation 只能用于它所指定的地方；若没有 @Target，则该 Annotation 可以用于任何地方。

**(4) @Retention(RetentionPolicy.RUNTIME)**

前面我们说过，RetentionPolicy 是 Annotation 的策略属性，而 @Retention 的作用，就是指定 Annotation 的策略属性。

@Retention(RetentionPolicy.RUNTIME) 的意思就是指定该 Annotation 的策略是 RetentionPolicy.RUNTIME。这就意味着，编译器会将该 Annotation 信息保留在 .class 文件中，并且能被虚拟机读取。

RetentionPolicy的3种策略方式

a) 若 Annotation 的类型为 SOURCE，则意味着：Annotation 仅存在于编译器处理期间，编译器处理完之后，该 Annotation 就没用了。 例如，" @Override" 标志就是一个 Annotation。当它修饰一个方法的时候，就意味着该方法覆盖父类的方法；并且在编译期间会进行语法检查！编译器处理完后，"@Override" 就没有任何作用了。

b) 若 Annotation 的类型为 CLASS，则意味着：编译器将 Annotation 存储于类对应的 .class 文件中，它是 Annotation 的默认行为。

c) 若 Annotation 的类型为 RUNTIME，则意味着：编译器将 Annotation 存储于 class 文件中，并且可由JVM读入。

定义 Annotation 时，@Retention 可有可无。若没有 @Retention，则默认是 RetentionPolicy.CLASS。

详情参考：<https://www.runoob.com/w3cnote/java-annotation.html>

详情参考：<https://www.cnblogs.com/hncboy/p/12208132.html>

其他注解及作用：

@Component//把类交给ioc管理

@Aspect//声明是切面类

@Before("execution(\* com.jgxi.controller.\*.\*(..))")//前置通知 切点表达式（具体方法 支持通配符）

@Around("execution ( \* com.jgxi.controller.\*.\*(..))")环绕通知

@ControllerAdvice//声明了一个aop 默认横切所有的controller 通知是异常通知

## 17、什么是接口?

接口的概念：

java中，接口是一个特殊的抽象类，接口里面的所有方法都是抽象方法，所有属性都是常量。只定义了方法，但是没有方法的具体实现

接口的思想

由内部操作分离出外部沟通方法

或

Java接口是一系列方法的声明，是一些方法特征的集合，一个接口只有方法的特征没有方法的实现，

因此这些方法可以在不同的地方被不同的类实现，而这些实现可以具有不同的行为（功能）。

## 18、递归

递归是什么

特性:

1.自己调自己

2.应该有一个退出条件/临界条件

3.递归在自己调自己的时候，传递的参数在不停的发生变化

为了解决什么问题/什么时候用?

当你不知道要循环多少次的时候/当不知道有几层嵌套的时候，就需要用递归！！！

1.求一个文件夹下有多个个文件？

2.求一个文件夹的大小？

3.涉及到树形结构的话，会使用递归，比如ztree？

\*4.在项目中可以生成一个无限级菜单

## 19、事务

什么是事务

事务是一组具有逻辑思维的sql语句，要么都执行成功，要么全部都不执行

事务的四大特性ACID

 原子性（Atomicity）

原子性是指事务是一个不可分割的工作单位，事务的操作要么都发生要么都不发生

 一致性（Consistentcy）

事务必须使数据库从一个一致性状态到另一个一致性状态

 隔离性（Isolation）

一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。

 持久性（Durability）

指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。

接下来的其他操作或故障不应该对其有任何影响。

事务的传播特性

1、PROPAGATION\_REQUIRED 如果当前没有事务，就新建一个事务，如果已经存在一个事务中，加入到这个事务中。这是最常见的选择。

2、PROPAGATION\_SUPPORTS 支持当前事务，如果当前没有事务，就以非事务方式执行

3、PROPAGATION\_MANDATORY 使用当前的事务，如果当前没有事务，就抛出异常。

4、PROPAGATION\_REQUIRES\_NEW 新建事务，如果当前存在事务，把当前事务挂起。

5、PROPAGATION\_NOT\_SUPPORTED 以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

6、PROPAGATION\_NEVER 以非事务方式执行，如果当前存在事务，则抛出异常。

7、PROPAGATION\_NESTED 如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行与PROPAGATION\_REQUIRED类似的操作

参考：<https://blog.csdn.net/weixin_41313504/article/details/78769535>

详情参考：<https://www.jianshu.com/p/e1848e2aa7c3>

详情参考：<https://www.cnblogs.com/myseries/p/10800430.html>

事务的隔离级别

1、ISOLATION\_DEFAULT：这是一个PlatfromTransactionManager默认的隔离级别，使用数据库默认的事务隔离级别。另外四个与JDBC的隔离级别相对应。

  2、ISOLATION\_READ\_UNCOMMITTED： 这是事务最低的隔离级别，它充许令外一个事务可以看到这个事务未提交的数据。这种隔离级别会产生脏读，不可重复读和幻像读。

  3、ISOLATION\_READ\_COMMITTED： 保证一个事务修改的数据提交后才能被另外一个事务读取。另外一个事务不能读取该事务未提交的数据

  4、ISOLATION\_REPEATABLE\_READ： 这种事务隔离级别可以防止脏读，不可重复读。但是可能出现幻像读。它除了保证一个事务不能读取另一个事务未提交的数据外，还保证了避免下面的情况产生(不可重复读)。

5、ISOLATION\_SERIALIZABLE 这是花费最高代价但是最可靠的事务隔离级别。事务被处理为顺序执行。除了防止脏读，不可重复读外，还避免了幻像读。

详解参考：<https://blog.csdn.net/qq_33290787/article/details/51924963>

详情参考：<https://blog.csdn.net/fuzhongmin05/article/details/91126936>

事务总结：<https://www.jianshu.com/p/d42b8c9aa950>

事务总结：<https://www.jianshu.com/p/5fd7bfb97888>

事务总结：<https://jingyan.baidu.com/article/e8cdb32b43b5e637042bad7d.html>

AOP是如何管理事物

AOP管理的是数据库的事物。

1、首先在spring的配置文件中配置sessionFactory,并加载数据库的连接信息，

声明事物（踹赛身卖嫩着）TransactionManager,并且反射到session工厂。

2、配置事物的传播特性，以save,update,delete等开头的执行增删改操作的方法必须开启事物以及事物的回滚，

其他方法走的是只读事物。

3、将事物交给Aop进行管理，我们用Aop横切的是service层。

## 20、多线程

**创建多线程**

(1)继承Thread(丝瑞的)，重写里面的run方法

使用时创建线程对象

启动 调用start方法

(2)实现Runnable接口

使用时创建线程实例化对象

创建一个线程，通过一个 Thread 类的有参构造，将实例化对象传进去

启动 调用start方法

**线程的生命周期**

1、新建状态（New）：当线程对象对创建后，即进入了新建状态，

如：Thread t = new MyThread();

2、就绪状态（Runnable）：当调用线程对象的start()方法（t.start();），线程即进入就绪状态。处于就绪状态的线程，只是说明此线程已经做好了准备，随时等待CPU调度执行，并不是说执行了t.start()此线程立即就会执行；

3、运行状态（Running）：当CPU开始调度处于就绪状态的线程时，此时线程才得以真正执行，即进入到运行状态。注：就     绪状态是进入到运行状态的唯一入口，也就是说，线程要想进入运行状态执行，首先必须处于就绪状态中；

4、阻塞状态（Blocked）：处于运行状态中的线程由于某种原因，暂时放弃对CPU的使用权，停止执行，此时进入阻塞状态，直到其进入到就绪状态，才 有机会再次被CPU调用以进入到运行状态。根据阻塞产生的原因不同，阻塞状态又可以分为三种：

1.等待阻塞：运行的线程执行wait()方法，JVM会把该线程放入等待池中。(wait会释放持有的锁)  
2.同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中。  
3.其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。（注意,sleep是不会释放持有的锁）

**线程睡眠**：Thread.sleep(long millis)方法，使线程转到阻塞状态。millis参数设定睡眠的时间，以毫秒为单位。当睡眠结束后，就转为就绪（Runnable）状态。sleep()平台移植性好。  
**线程等待**：Object类中的wait()方法，导致当前的线程等待，直到其他线程调用此对象的 notify() 方法或 notifyAll() 唤醒方法。这个两个唤醒方法也是Object类中的方法，行为等价于调用 wait(0) 一样。唤醒线程后，就转为就绪（Runnable）状态。  
**线程让步**：Thread.yield() 方法，暂停当前正在执行的线程对象，把执行机会让给相同或者更高优先级的线程。  
**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I/O**：线程执行某些IO操作，因为等待相关的资源而进入了阻塞状态。比如说监听system.in，但是尚且没有收到键盘的输入，则进入阻塞状态。  
**线程唤醒**：Object类中的notify()方法，唤醒在此对象监视器上等待的单个线程。如果所有线程都在此对象上等待，则会选择唤醒其中一个线程，选择是任意性的，并在对实现做出决定时发生。类似的方法还有一个notifyAll()，唤醒在此对象监视器上等待的所有线程。

5、死亡状态（Dead）：线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

生命周期: <https://www.jianshu.com/p/cb6c3090002f>

**线程安全出现的根本原因**

1、存在两个或者两个以上的线程对象共享同一个资源；

2、多线程操作共享资源代码有多个语句

线程问题:<https://blog.csdn.net/lester_8/article/details/77098127>

避免死锁: <https://blog.csdn.net/sinat_41144773/article/details/89476679>

## 21、JDK 1.8新特性

Lambda表达式(就是函数接口的实现)

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

Stream

Java 8 API添加了一个新的抽象称为流Stream，可以让你以一种声明的方式处理数据。

Stream 使用一种类似用 SQL 语句从数据库查询数据的直观方式来提供一种对 Java 集合运算和表达的高阶抽象。

Stream API可以极大提高Java程序员的生产力，让程序员写出高效率、干净、简洁的代码。

这种风格将要处理的元素集合看作一种流， 流在管道中传输， 并且可以在管道的节点上进行处理， 比如筛选， 排序，聚合等。

元素流在管道中经过中间操作（intermediate operation）的处理，最后由最终操作(terminal operation)得到前面处理的结果。

1、创建stream

2、中间操作：

distinct() 去重 【第四厅可特】集合名.stream.distinct().forEach();

skip() 跳过几条 【斯k扑】

limit() 取出几条 【类米特】

filter() 过滤 【非欧特】集合名.stream.filter(a->a.length()>1).forEach();

3、结果输出：

forEach()输出

count() 统计有几条【康特】

stream使用: <https://blog.csdn.net/chenhao_c_h/article/details/80691284>

stream参考: <https://blog.csdn.net/qq_28410283/article/details/80633710>

新特性详情参考: <https://blog.csdn.net/qq_29411737/article/details/80835658>

## 22javascript

### 1数据类型

#### 字符串 string

字符串对象常用方法

字符串定义方式：var str="内容";

split(); 将字符串分割为字符串数组

charAt(下标); 返回指定下标对应的字符,如果找不到返回空的字符串" ";

indexOf(); 查找子字符串在字符串中首次出现的位置,如果找不到返回-1

lastindexOf(); 查找子字符串在字符串中最后一次出现的位置,如果找不到返回-1

substring(起始下标,结束下标); 按起始下标和结束下标截取字符串，包前不包后， //如果不传结束下标，则默认截取到字符串结尾

substr(起始下标,[个数]); 按起始下标和个数截取字符串, //如果只给了起始下标，没有给截取的个数，默认截取到字符串末尾

replace(旧字符串,新字符串); 替换字符串（replace(正则表达式); ）

concat(值1,值2,...); 拼接字符串,相当于+号

toLowerCase(); 将字符串转为小写

toUpperCase(); 将字符串转为大写

#### 数值number 【南波儿】

数组的常用方法：

1、var arr=[值1,值2,值3......];

2、var arr=new Array(长度);

定义长度没什么用

3、var arr=new Array();

数组的赋值：数组名[下标]=值;

数组的取值：数组名[下标];

遍历循环数组的方式：for、while、do while、for in

for(var i in 数组名){数组名[i]}

数组的常用方法

push(元素1,...);【抛斯】 向数组末尾添加一个或多个元素,并且返回数组的新长度

unshift(元素1,...); 向数组开头添加一个或多个元素,并且返回数组的新长度

pop();【泡普】 移除数组的最后的一个元素,并返回该元素

shift(); 移除数组的开头第一个元素,并返回该元素

concat(); 将多个数组拼接成一个新的数组,并返回新数组

slice(起始下标,结束下标);【撕莱私】 以数组的形式返回起始下标到结束下标之间的元素,包前不包后

splice(起始下标,个数,[元素,..]);【撕扑莱私】在数组的任意位置添加或删除任意个元素

sort();【扫特】 对数组中的元素进行排序,默认使用unicode编码进行从小到大排序

join(分隔符);【爪恩】 以指定的分隔符将数组中的元素拼接成一个字符串

reverse();【瑞ver斯】 反转数组中的元素

toString(); 将数组转成字符串

toLocaleString();【喽口】 和同string没什么区别

二维数组:二维数组中的每一个元素都是一个一维数组

Math数学对象

abs(); 绝对值

Max(); 最大值

Min(); 最小值

ceil();【赛欧】 向上取整

floor();【负老r】 向下取整

round();【软的】 四舍五入

random();【软的木】随机数，生成0到1之间的随机小数，含0不含1

布尔 boolean

typeof 【泰普奥夫】 用来检测数据类型

复合数据类型

数组 array

三种定义方式：

1、var arr1 = [变量1,变量2,.....];

2、var arr2 = new Array(3); 需要提前定义一个长度

//即便我们赋值时超过了这个长度，也不会报错，下标没什么卵用

3、var arr3 = new Array();

数组的取值方式：

var x = 数组名[下标];

//在取值时，即使下标超过最大的下标也不报错，返回undefined【安迪凡的】未定义

数组的赋值方式：

数组名[下标] = 值;

//注意：即使我们要给数组赋值的下标超出数组长度，也不会报错

日期类型

日期对象及常用方法

日期对象的定义方式：var date = new Date();

toString() 将日期转为字符串

toLocaleString() 将日期转为本地格式字符串

getFullYear()【否页r】 获取指定日期的年份

getMonth()【忙此】 获取指定日期的月份,返回0-11,0代表1月份，11代表12月份

getDate() 获取指定日期的天数,返回1-31

getDay() 获取指定日期的星期数,返回0-6,0代表星期日,6代表星期六

getHours()【阿窝斯】 获取指定日期的小时数,返回0-23

getMinutes()【米内斯】 获取指定日期的分钟数,返回0-59

getSeconds()【赛啃斯】 获取指定日期的秒数,返回0-59

getMilliseconds()【米林赛啃此】获取指定日期的毫秒数,返回0-999

setFullYear(年份) 设置年份

setMonth(月份) 设置月份

setDate(天数) 设置天数

setHours(小时数) 设置小时数

setMinutes(分钟数) 设置分钟数

setSeconds(秒数) 设置秒数

setMilliseconds(毫秒数) 设置毫秒数

getTime() 获取从1970年1月1号到指定日期经过的毫秒数

对象（json）

js的对象的数据格式，也叫json(J森)数据。json数据是一个行业内的数据格式标准！

格式：{"键1":"值1","键2":"值2","键3":"值3".....}

json对象

SON是一种基于JavaScript语言的轻量级的数据交换格式(JavaScript Object Notation),

JSON对象是一个无序的“属性名/值”对集合。

一个对象以“{”（左大括号）开始，“}”（右大括号）结束。

每个“属性名”后跟一个“:”（冒号）；“属性名/值”对之间使用“,”（逗号）分隔。

它是一种严格的JS对象的格式，JSON的属性名必须有双引号；如果值是字符串(数字不用)，也必须是双引号。

json定义方式：var obj={"键1:值1","键2":"值2".......};

查看json对象单个的属性

查看值：对象名.键名 或 对象名["键名"]

新增json对象的属性

对象名.新增属性="新增值"

修改json对象的某个属性

对象名.要修改的属性="要修改的值"

删除json对象的某个属性

delete 对象名.要删除的键 delete【地雷特】

遍历JSON对象中的所有属性

for(var i in 对象名){

i 代表每一个键对象

对象名[i] 代表每一个值对象

}

JSON数组是由JSON对象组成的数组，JSON数组中的每一个元素都是一个JSON对象。

而且json数组有要求，json数组中的每一个json对象的格式必须一样。

遍历JSON数组

for(var i=0;i<json数组名.length;i++){

for(var j in 数组名[i]){

j 代表每一个json对象的每一个键对象

数组名[i][j] 代表每一个json对象的值对象

}

}

特殊数据类型

空 （null）

未定义undefined【按第反的】（undefined指的是未定义数据类型）

### 2 Js选择器 和Documnet对象

##### Id 和name 选择器

在js中id和name都是标识之用（页面元素的调用），id一般来说是唯一的，但name不一定是唯一的。

##### Documnet对象

（js网页特效的核心，//其实就是网页页面的对象） //Elemnet【艾乐门特】\\

document.write(); 在网页上输出

document.getElementById(); 根据id获取html元素对象

value 获取、修改html元素的value值

doucment.getElementsByTagName(); 根据标签名获取HTML元素的数组

document.getElementsByName(); 根据标签的name值来获取html元素数组

innerText和innerHTML 都是可以针对于无法设置value的元素来设置其中的内容

innerText和innerHTML异同点

1、innerText获取内容时，只能获取文本内容，能获取HTML代码，设置值时只能设置文本内容，即便是写了HTML代码，

也会将HTML代码原样输出

2、innerHTML获取内容时，既能获取文本内容，也能获取HTML代码，设置值时文本内容和HTML代码都可以设置

3、他们在设置值时，都会覆盖掉原本的内容！！！

Js常用事件

事件的定义:

**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。**

**当事件发生时，你可以做一些事情.比如在事件触发时可以执行一些JavaScript代码。**

* **为啥要学习js事件**

**如果不使用js事件，我们的一些网页操作也可以实现，但是用户的体验度比较差。**

**js事件可以实现某些特定的功能。可以使网页有动态的效果。**

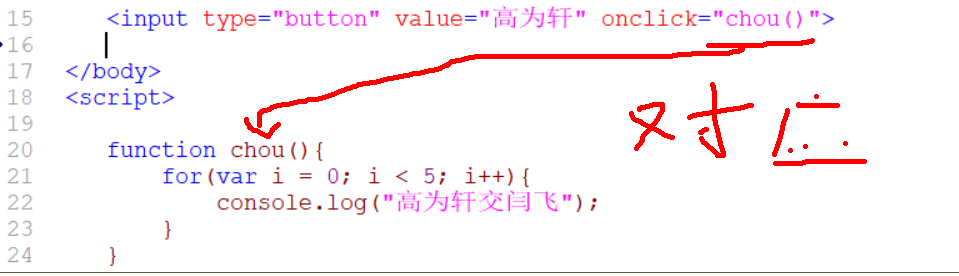
* **事件的使用:**

**为HTML 元素绑定事件**

<HTML元素 事件1=”js代码” 事件2=”js代码” .......></HTML元素>

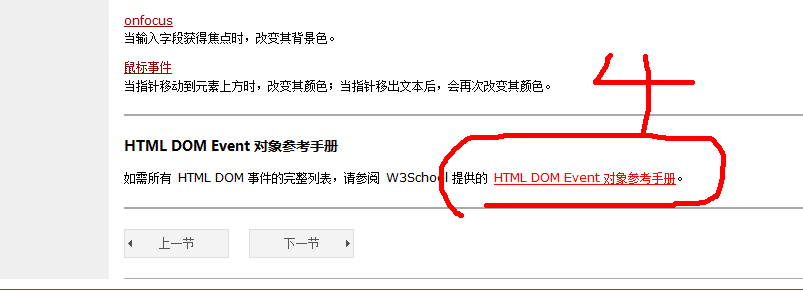


**注意：一个html元素可以绑定多个事件，事件通常与函数配合使用，这样就可以通过发生的事件来驱动函数执行。**



怎么看手册？



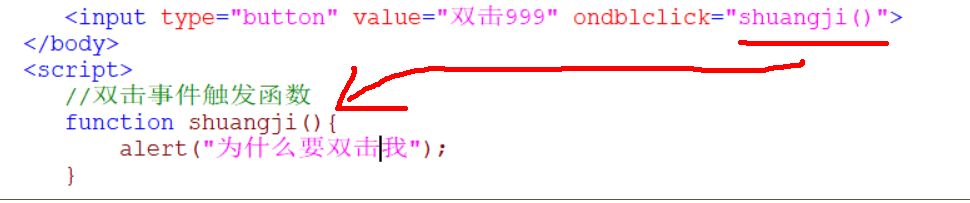


* **事件的分类:**

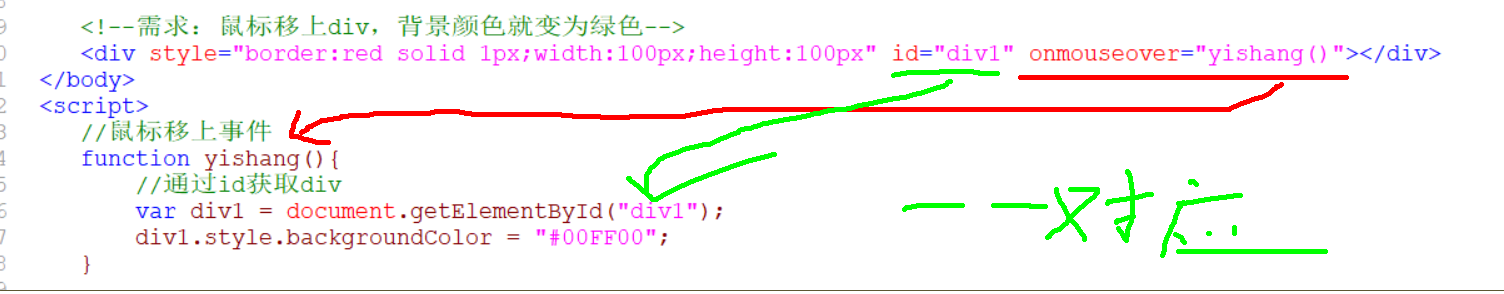
**鼠标事件:**

**[昂可立克]onclick 鼠标单击事件**

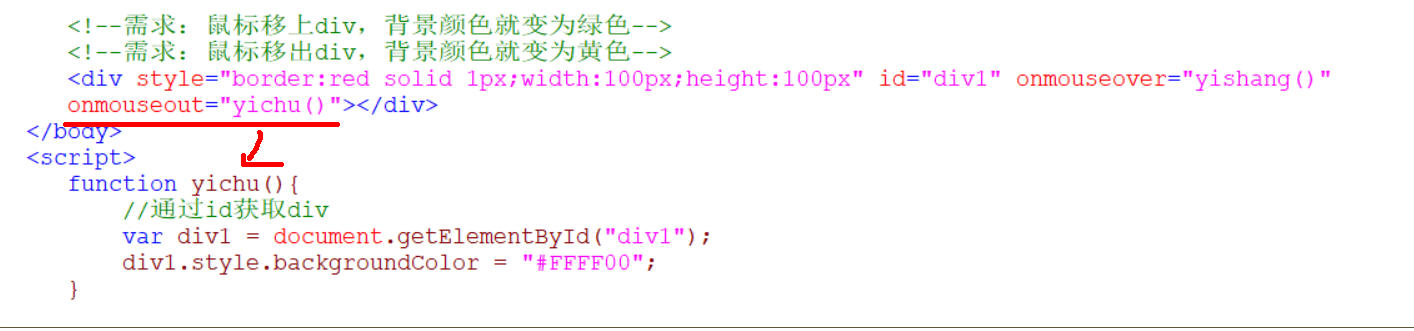
**[昂大bO可立克]ondblclick 鼠标双击事件**



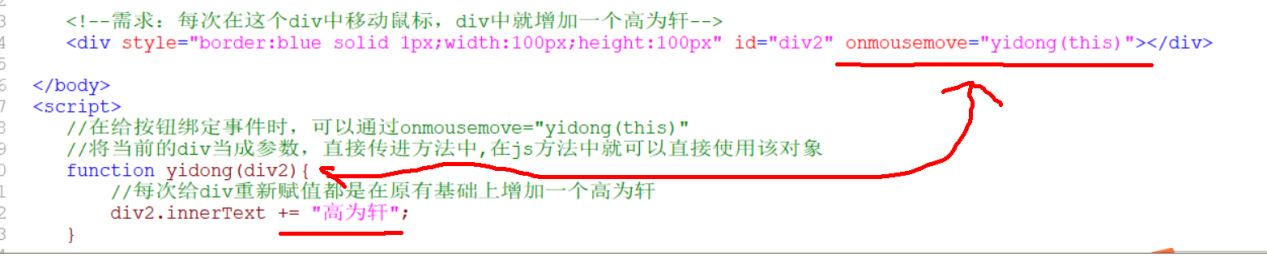
**[昂貌似over]onmouseover 鼠标移上事件**



**[昂貌似奥特]onmouseout 鼠标移出事件**

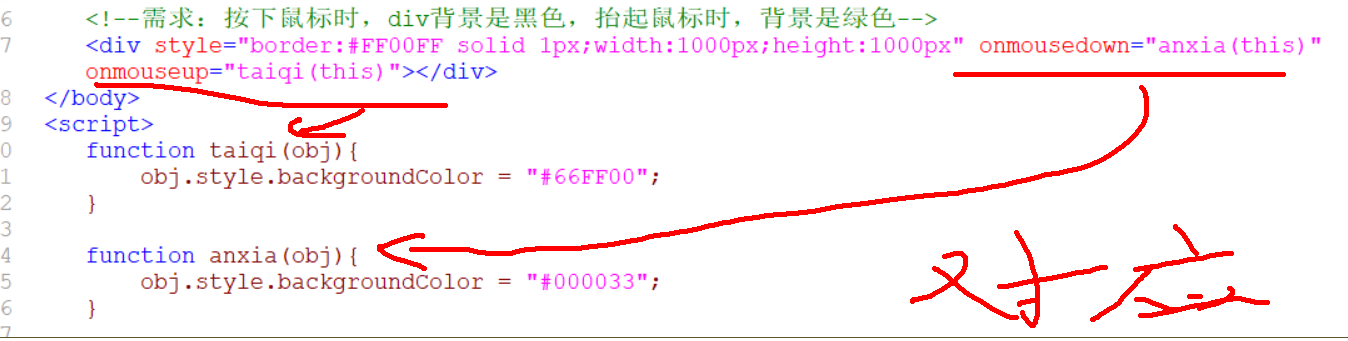


**[昂貌似某vu]onmousemove 鼠标移动事件**



**[昂貌似当]onmousedown 鼠标按下事件**

**[昂貌似阿普]onmouseup 鼠标抬起事件**



**页面事件:**

**[昂露的]onload 页面加载事件 (用于body标签)这个事件是页面加载完成之后才会被触发**

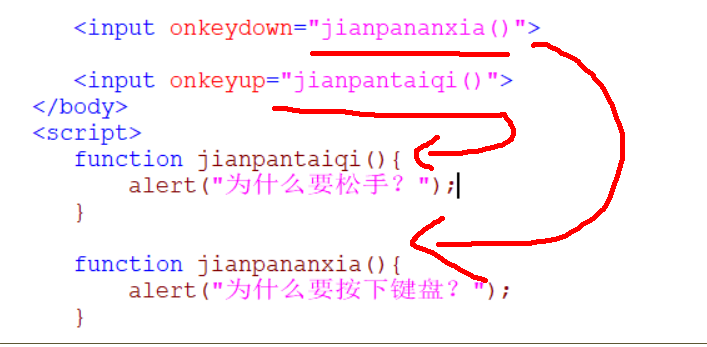
**[昂按露的]onunload 页面离开事件(用于body标签)[没啥用]**



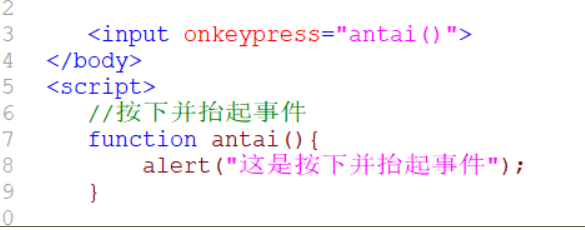
**键盘事件(常用于文本框/密码框/文本域):**

**[昂K当]onkeydown 按键按下事件**

**[昂K阿普]onkeyup 按键抬起事件**



**[昂K普ruai斯]onkeypress 按键按下并抬起事件**



键盘按下事件和按下并抬起事件，都是在按下键盘的一瞬间触发的，但是键盘按下事件允许所有键位，包括功能键。

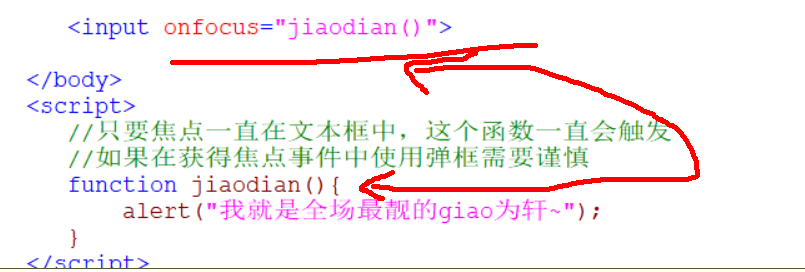
而按下并抬起事件不允许功能键使用。

**表单事件:**

**啥是焦点？**

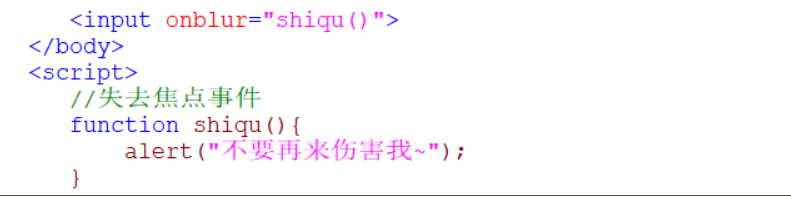
**就是电脑上鼠标点过的地方，一闪一闪的光标，就是焦点。**

**[昂否司]onfocus 得到焦点事件 (常用于文本框和密码框、文本域[按钮、div这些东西即使绑定了得到焦点事件也没有效果])**

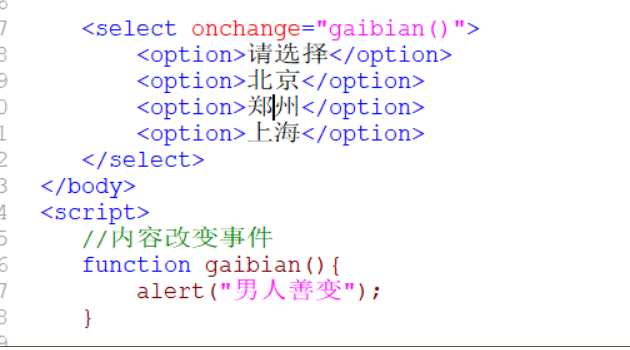


**[昂不乐儿]onblur 失去焦点事件（常用于文本框和密码框、文本域）**

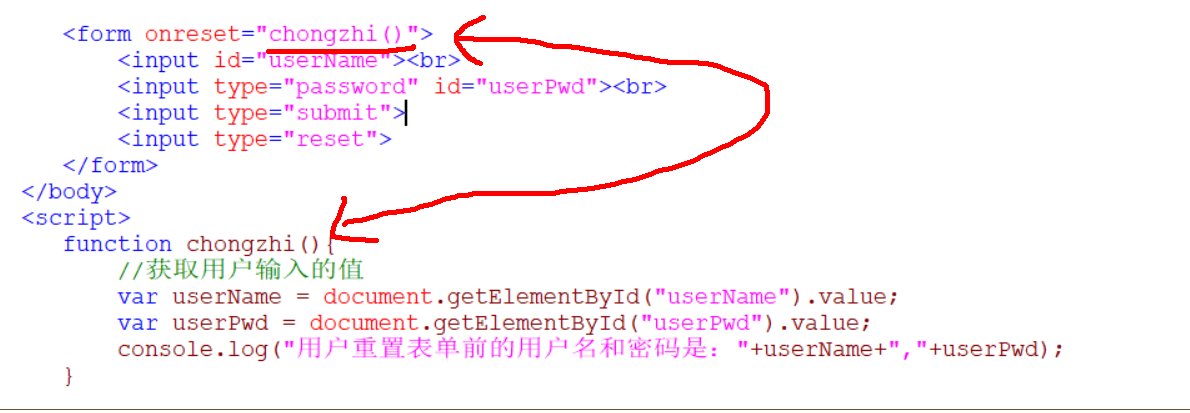
**只会在失去焦点后触发一次**



**[昂陈志]onchange 内容改变事件 (常用于下拉框)**



**[昂瑞赛特]onreset 表单重置事件 (用在表单上面)，点击重置按钮会触发此事件**



**[昂撒不米特]onsubmit 表单提交事件 (用在表单上面)**

## 23JavaWeb开发技术之JSP

1 Jsp是什么

JSP全称是Java Server Pages(java服务器页面)，是SUN公司定义的一种用于开发动态web资源的技术。

JSP这门技术的最大的特点在于，写jsp就像在写html，但它相比html而言，html只能为用户提供静态数据，

而Jsp技术允许在页面中嵌套java代码，为用户提供动态数据。它使用JSP标签在HTML网页中插入Java代码。

JSP标签通常以 <% 开头, 以 %> 结束。JSP文件后缀名为(\*.jsp)。JSP开发的WEB应用可以跨平台使用，

既可以运行在Linux（林牛克斯命令行）操作系统上也能运行在Windows操作系统上。

Linux是一种操作系统，一般用于服务器使用。

JSP的本质是servlet([色ver莱特]另一种java动态web资源开发技术)，主要用于实现Java web应用程序的用户界面部分。

JSP是Java EE不可或缺的一部分。

JSP运行原理(笔试必考，五分，面试必问)

以下步骤表明了Web服务器是如何使用JSP来创建网页(html)的：

1、用户发送一个请求---->服务器(tomcat)

2、服务器根据请求的url(浏览器地址栏的内容)----->找到相应的jsp文件

3、服务器将找到的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(html)------>用户

以上提及到的步骤可以用下图表示:

2 JSP页面元素

1、静态内容

JSP页面中的HTML内容称之为静态内容

2、JSP脚本程序

JSP脚本程序(scriptlet)用于在JSP页面中编写多行Java代码。语法：

　　 <%

多行java代码

%>

　 在<% %>中可以定义变量、编写语句，不能定义方法。

注意:

1. JSP脚本程序中只能出现java代码

2. JSP脚本程序中的Java代码必须严格遵循Java语法，例如，每行语句后面必须用分号（;）结束。

3. 在一个JSP页面中可以有多个脚本程序，在两个或多个脚本程序之间可以嵌入文本、HTML标签和其他JSP元素。

3、JSP表达式

JSP脚本表达式（expression）用于将程序数据输出到客户端(网页)，效果和out.println()方法一样

语法：<%= 变量或表达式 %>

out.print()和out.println()的区别，ln的换行只是源代码(控制台)中换行，给我们的视觉效果只是一个空格。

如果想在网页中换行，只能使用<br>标签

注意:

JSP脚本表达式中的变量或表达式后面不能有分号（;）

如果想要强行写分号是会报错的！！！

4、JSP声明

一个声明语句可以声明一个或多个变量、方法，供后面的Java代码使用。多个变量和方法可以定义在一个JSP声明中，

也可以分别单独定义在多个JSP声明中。

语法: <%! Java代码 %>

注意：如果在jsp的声明中定义了方法，想要调用的话，就得使用jsp的脚本标签<%%>

直接通过方法名(参数)的方式进行调用。

5、JSP指令

JSP指令用来设置与整个JSP页面相关的属性。

语法格式：

<%@ directive attribute="value" %>

指令 描述

<%@ page ... %> 定义页面的依赖属性，比如脚本语言、error(错误)页面、缓存需求等等

图中指定了语言=”java” 内容类型=”文本/HTML; 编码=UTF-8” 整个页面支持utf-8

<%@ include ... %> 包含其他文件

<%@ taglib ... %> 引入标签库的定义，可以是自定义标签

因为jsp给我们提供的标签不能实现某些特定的内容，或者实现起来太麻烦，

所以有很多优秀的开发人员自己进行了补充，我们如果想用别人写好的标签，

就需要taglib来引入别人的标签

6、JSP注释

JSP注释主要有两个作用：增强代码可读性以及将某段代码注释掉。

<%-- 注释 --%> JSP注释，注释内容不会被发送至浏览器甚至不会被编译

<!-- 注释 --> HTML注释，通过浏览器查看网页源代码时可以看见注释内容

--------------------------------------------------------------------------------------------------------------------------------------

3 jsp九大内置对象(隐式对象):

JSP内置对象是JSP事先创建好的一组对象，不使用new关键字就可以直接使用。

1、所有JSP页面均可使用(只要你是JSP页面，就可以使用JSP九大内置对象)。

2、只有在JSP表达式(<%=%>)或JSP脚本(<%%>)中才可使用，

如果是我们自己想创建一个对象，还需要实例化，还需要导包，使用起来特别麻烦，

所以jsp就将使用频繁的几个对象直接作为内置对象。

jsp内置对象的分类(笔试必出面试必问):

输入输出 作用域通信 servlet对象 错误对象

Request[瑞快斯特]请求 session[塞申]会话 page[配制]网页 exception[A克赛普神]异常

Response[瑞思邦斯]响应 application[额普雷K神]应用 config[康飞个]配置

Out[奥特]输出 pageContext[配置康泰克斯特]页面

从功能上来划分的话，jsp九大内置对象可以分为4类，其中request既属于输入/输出对象也属于作用域通信对象。

以上内置对象中常用的是pageContext、request、response、session、application，掌握了这5个，即可进行程序开发

1、Request 请求

request对象是javax.servlet.http.HttpServletRequest 类的实例。

request对象代表了客户端的请求信息，主要用于接收客户端通过传送到服务器的数据。

request内置对象是使用最多的一个对象，每当客户端请求一个JSP页面时，

JSP引擎(tomcat)就会制造一个新的request对象来代表这个请求。其主要作用是接收客户端发送来的请求信息，

如请求的参数、发送的头信息等都属于客户端发来的信息。request对象在实际开发中使用的较多。

//我们将一个表单中的值如果想要提交到后台，就需要使用request对象。

如何向一个JSP页面发起请求?

1.当用户在浏览器地址栏输入URL

2.单击Web页面中的超链接(a标签)

3.提交表单可以向一个JSP页面发起请求

常用方法:

String getParameter(String参数名)【盖特派瑞梅特】;根据参数名获取参数值

String[] getParameterValues(String参数名)；根据参数名获取参数值数组

void setAttribute(String 属性名,Object 属性值);【赛特额吹biu特】 往request对象中设置一个属性，

请注意，如果多次调用该方法，并且使用相同的name，那么会覆盖上一次的值

Object getAttribute(String 属性名); 从request对象中通过属性名取出对应属性值，如果指定属性名不存在则返回null

void removeAttribute(String 属性名); 【瑞某vu额吹biu特】从request对象中移除指定属性名的属性，

如果指定属性名不存在，那么什么也不做。

void getRequestDispatcher(url).forward(request,response); 对客户端请求时进行转发

【盖特瑞块斯特第四排客车 否沃尔德】forward转发

如果是使用转发来跳转页面，则request(请求)的数据是共享的！！！！！！

接收表单提交过来的数据需要注意的问题:

单一的参数都可以使用getParameter()接收(和name值对应),而一组参数要用getParameterValues()接收(一般用于复选框)。

在表单控件中，像文本框（text）、单选按钮（radio）、密码框（password）、隐藏域（hidden）、下拉框(select)、

文本域(textarea)等，一般都会使用getParameter()方法进行接收，因为这些控件在使用时参数的名称都只有一个不会重复；

而像复选框（checkbox），一般参数的名称都是重复的，是一组参数，所以只能使用getParameterValues()接收，

如果不小心使用了getParameter()方法，则只会接收第一个选中的内容。

2、Response响应

response对象是javax.servlet.http.HttpServletResponse类的实例。

当服务器创建request对象时会同时创建用于响应这个客户端请求的response对象。

response对象的主要作用是对客户端的请求进行回应，将Web服务器(tomcat)处理后的结果发回给客户端(浏览器)。

常用方法:

 void sendRedirect(String url);[森德瑞迪ruai科特]重定向，发送一个响应给浏览器，请求另外一个url

案例:新建登录页面和验证页面，在验证页面对用户名和密码进行验证,如果用户名为admin，密码为123456则验证通过。

如果验证不通过则重定向到登录页面。

重定向(redirect)和转发(forward)的区别？(考试必考5分)

重定向地址栏发生改变，多次请求，运行效率低；[转发(浏览器只需要跑一次，等着就行)：]

转发地址栏不发生改变，一次请求，运行效率高！[重定向(浏览器来回跑)：]

3、Session 会话

javax.servlet.http.HttpSession的实例。

session表示一个会话，我们可以把一个会话内需要共享的数据保存到session对象中！

什么是会话?

可以把会话理解为客户端与服务器之间的一次见面，在一次会晤中可能会包含多次请求和响应。例如你给10086打个电话，

你就是客户端，而10086服务人员就是服务器了。从双方接通电话那一刻起，会话就开始了，

到某一方挂断电话表示会话结束。在通话过程中，你会向10086发出多个请求，那么这多个请求都在一个会话中。

在JavaWeb中，客户向某一服务器发出第一个请求开始，会话就开始了，直到客户关闭了浏览器会话结束。

常用方法:

String getId() 获取当前session会话的唯一id

只要不关浏览器访问同一个项目，在特定时间(30分钟)得到的session是相同的。

void setAttribute(String name,Object value); 往session对象中设置一个属性，请注意，如果多次调用该方法，

并且使用相同的name，那么会覆盖上一次的值。

Object getAttribute(String name); 从session对象中通过属性名取出对应属性值，如果指定属性名不存在则返回null

 Void removeAttribute(String name); 从session对象中移除指定属性名的属性，如果指定属性名不存在，什么也不做

 setMaxInactiveInterval(秒数) 设置session超时时间，注:tomcat默认session超时时间为30分钟

【赛特麦克斯因艾克题vu因特V哦】

4、Out

out是javax.servlet.jsp.JspWriter类的实例化对象，

主要功能就是完成页面的输出操作，使用println()或print()方法输出，但是从实际的开发来看，

直接使用out对象的几率较小，一般使用表达式(<%=%>)完成输出的操作。

常用方法:

print(内容) 向页面中输出内容 out.print();

println(内容) 向页面中输出内容并换行,这里的换行指的是浏览器源代码的换行。

而不是网页中的换行，网页中的换行，都是<br> out.println();

5、Application应用，应用程序

javax.servlet.ServletContext 类的实例。

通过向application中添加属性，则在web应用中的所有JSP文件都能访问到这些属性。

application对象实现了用户间数据的共享，可存放全局变量。application对象开始于web服务器(tomcat)的启动，

结束于web服务器(tomcat)的关闭。

常用方法:

void setAttribute(String name,Object value); 往application对象中设置一个属性，请注意，如果多次调用该方法，

并且使用相同的name，那么会覆盖上一次的值。application.setAttribute(“属性名”,属性值);

Object getAttribute(String name); 从application对象中通过属性名取出对应属性值，如果指定属性名不存在则返回null

Void removeAttribute(String name); 从application对象中移除指定属性名的属性，如果指定属性名不存在，什么也不做

6、pageContext: 页面上下文

pageContext对象是javax.servlet.jsp.PageContext类的实例，

主要表示一个JSP页面的上下文。pageContext对象中放置的属性只能作用于当前页面。

这个对象存储了request对象、application对象、config对象、session对象、out对象、response对象的引用。

 void setAttribute(String name,Object value); 往pageContext对象中设置一个属性，请注意，如果多次调用该方法，

并且使用相同的name，那么会覆盖上一次的值。

 Object getAttribute(String name); 从pageContext对象中通过属性名取出对应属性值，如果指定属性名不存在则返回null

 void removeAttribute(String name); 从pageContext对象中移除指定属性名的属性，如果指定属性名不存在，什么也不做

 void setAttribute(String name,Object value,int scope) 往指定作用域对象中设置一个属性

 Object getAttribute(String name, int scope) 从指定作用域对象中通过属性名获得对应属性值

 removeAttribute(String 属性名,int 作用域范围),从指定作用域对象中移除指定属性名的属性

 Object findAttribute(String name) 在所有作用域对象中通过属性名查找对应的属性值（从小往大的查！）

7、Exception

exception对象是java.lang.Throwable的对象。

exception对象是一个异常对象，当一个页面在运行过程中发生了异常，就产生这个对象。如果一个JSP页面要应用此对象，

就必须把isErrorPage设置为true，否则无法编译。

8、Config(配置)

config是对象是javax.servlet.ServletConfig接口的实例化对象，主要功能是取得一些初始化的配置信息。

9、Page(页面)

page对象就是指向当前JSP页面本身，有点像java类中的this指针，它是java.lang.Object类的实例。

4四大作用域(考试必考5分)：

作用域 作用域对象 作用域范围

page pageContext 在当前页面有效(代号1)

request request 在当前请求中有效（通过forward跳转可以共享request对象中存放的属性,代号2）

session session 在当前会话中有效(只要浏览器不关闭/session不超时，

session中存放的属性就不会消失，代号3)

application application 在当前应用程序中有效（整个项目，只要服务器不停止，

application对象中存放的属性就不会消失，代号4）

1. Page

作用域对应的作用域对象是pageContext：作用域仅限于当前页面对象，离开当前JSP页面（无论是redirect还是forward）,

则pageContext中的所有属性值就会丢失。

1. Request

作用域对应的作用域对象是request：作用域是同一个请求之内，在页面跳转时，

如果通过forward方式跳转,则forward目标页面仍然可以拿到request中的属性值。

如果通过redirect方式进行页面跳转，由于redirect相当于重新发出的请求，此种场景下，request中的属性值会丢失。

1. Session

作用域对应的作用域对象是session：session的作用域是在一个会话的生命周期内，会话失效(关闭浏览器或session超时)，

则session中的数据也随之丢失。

1. Application

作用域对应的作用域对象是application：作用域是最大的，只要服务器不停止，则application对象就一直存在，

并且为所有会话所共享。

## 24数据库

数据库英文为DataBase,简称DB。所谓数据库就是用来存放数据的仓库。数据可以存放到硬盘上也可以存储到内存中。

1 SQL简介（程序员必须掌握）

结构化查询语言 (Structured Query Language 简称SQL) 是用于访问数据库的标准语言，这些数据库包括 SQLServer、

Oracle、MySQL等等。

通过SQL可以对数据库进行读写操作(只要是对数据库中的数据进行了改动，增加、删除、修改都叫”写”操作)。

除了 SQL 标准之外，大部分数据库都拥有它们自己的私有扩展！

2 SQL的分类:

数据定义语言(Data Definition Language )DDL: create【科瑞特】(创建) 、alter【奥特r】(修改/更改)、 drop【灶普】(删除)

（DDL是针对与表和表结构的操作(删除表、新增表、修改某个表字段)）

数据操作语言(Data Manipulation Language)DML: insert【in涩r特】(增加) select(查询) update(更新) delete(删除)

（DML是针对于表中的数据进行操作,如加一行数据，删除一行数据）

事务控制语言(Transaction Control Language)TCL: commit【考米特】(提交)、rollback【肉拜克】(回滚)

（简单理解，提交就相当于是ctrl+s，回滚就相当于是ctrl+z）

数据控制语言(Data Control Language)DCL: grant【格软的】(授权) 、revoke(撤销)

### 3数据库的数据类型

数据类型是为了限制表中列的内容。

1字符类型：

varchar2【瓦查图】 最多4000字节，可变长度，最为常用

nvarchar2 【N 瓦查图】 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

数值类型：

number(总长度,小数点后的位数) 最大精度38位

number(5) 代表一个长度1到5位整数

number(5,2) 代表整数位最多为3位，小数点后最多为2位

number(7,1) 整数位最多6位，小数点后最多1位

2日期类型：

date 年月日时分秒

//注意：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字段/表 的命名规则：

由字母数字下划线组成，

不允许出现特殊字符(-!@#$%^&\*等等)

一般以英文字母开头，每个单词之间由\_分割

不能是oracle关键字(关键字直接打出来会变成绿色)。

没有驼峰命名法，因为oracle不区分大小写。

//打个比方，我们创建的表名如果想叫 prodect\_test\_equipment，如果没有下划线区分，就会变成：PRODUCTTESTEQUIPMENT，

//有的英文单词拼在一起会变成另一个意思，为了避免引起歧义，所以我们在每个单词中间加上下划线。

## 25 HashMap底层原理（面试热点）

**1、HashMap红黑树原理分析**

相比 jdk1.7 的 HashMap 而言，JDK8之后，如果哈希表单向链表中元素超过8个，那么单向链表这种数据结构会变成红黑树数据结构。当红黑树上的节点数量小于6个，会重新把红黑树变成单向链表数据结构。

为什么要这样设计呢？好处就是避免在最极端的情况下链表变得很长很长，在查询的时候，效率会非常慢。

红黑树查询：其访问性能近似于折半查找，性能较高

链表查询：这种情况下，需要遍历全部元素才行，性能较低。

简单的说，红黑树是一种近似平衡的二叉查找树，其主要的优点就是“平衡“，即左右子树高度几乎一致，以此来防止树退化为链表，通过这种方式来查找省时省力。

**2、hashMap运行原理**

\*HashMap的底层实现是一个哈希表即数组+链表；

我们在将数据存入哈希表的时候，要得到hashcode的值，用值去%数组的长度，

得到的余数放入数组段中，此时数组段的位置上已经有值了，那么将通过equals方法比较两个hash值是否相同，如果两个hash值相等，再比较两个键值对key是否相等，如果两个hash值不等，则在此位置，以链表的方式，将新存储的数据放到表头；如果hash值和key都相等，将覆盖原先的value。

这样hashmap不仅具有了数据查询快速的特性，同时有了链表方便插入、删除的特性；因此hashMap对于数据的存储查询具有非常好的特性；

**3、HashMap中的“死锁”的原因？**

HashMap是非线程安全，死锁一般都是产生于并发情况下。我们假设有二个进程T1、T2，HashMap容量为2,T1线程放入key A、B、C、D、E。在T1线程中A、B、C Hash值相同，于是形成一个链接，假设为A->C->B，而D、E Hash值不同，于是容量不足，需要新建一个更大尺寸的hash表，然后把数据从老的Hash表中 迁移到新的Hash表中(refresh)。这时T2进程闯进来了，T1暂时挂起，T2进程也准备放入新的key，这时也 发现容量不足，也refresh一把。refresh之后原来的链表结构假设为C->A，之后T1进程继续执行，链接结构 为A->C,这时就形成A.next=B,B.next=A的环形链表。一旦取值进入这个环形链表就会陷入死循环。

**4、HashMap中能put两个相同key吗？为什么？**

能/不能

能，如果两个key相同，但hash值不等，则在此位置，以链表的方式，将新存储的数据放到表头

不能，如果两个key的hash值和key都相同，就会把相同的value值覆盖。

**5、HashMap中的键值可以为null吗？原理？**

HashMap 支持key=null 但是 Hashtable 不支持 key =null/TreeMap也不支持key=null

HashMap在调用put方法时，其实是调用了putVal方法，而该方法的第一个参数hash值的获取是通过HashMap中自定义的hash方法得到，hash方法中明确指出当key值为null时返回hash值为0，并不会报错，且putVal方法中也没有对value值进行任何约束，故HashMap中的键值都能存放null值。

HashTable在调用put方法时，首先会对put方法中的值value进行null值判断，为null抛出空指针异常；然后会对key值进行Object类的hashCode方法计算hash值，为null值也会抛出空指针异常，故不能在HashTable键、值中放入null值。

**6、HashMap扩容机制？**

当HashMap中的元素越来越多的时候，hash冲突的几率也就越来越高，因为数组的长度是固定的。所以为了提高查询的效率，就要对HashMap的数组进行扩容。

当HashMap中的元素个数超过数组大小(数组总大小length,不是数组中个数size)\*loadFactor时，就会进行数组扩容，负载因子的默认值为0.75，这是一个折中的取值。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这个临界值的时候，就把数组的大小扩展为 2\*16=32，即扩大一倍，然后重新计算每个元素在数组中的位置。

0.75这个值成为负载因子，那么为什么负载因子为0.75呢？这是通过大量实验统计得出来的，如果过小，比如0.5，那么当存放的元素超过一半时就进行扩容，会造成资源的浪费；如果过大，比如1，那么当元素满的时候才进行扩容，会使get,put操作的碰撞几率增加。

# 二、 java框架

## 1、什么是MVC

(MVC是一种设计模式)

MVC思想就是分层思想，使项目结构更清晰，方便管理

### 1 model层

model 模型(持久) 数据处理，一般代表Service和Dao 也可以包括封装类

#### 1、JDBC

JDBC（Java DataBase Connectivity其实就是java程序连接数据库的技术）相关概念

数据库驱动

这里的驱动的概念和平时听到的那种驱动的概念是一样的，比如平时购买的声卡，网卡直接插到计算机上面是不能用的，

必须要安装相应的驱动程序之后才能够使用声卡和网卡，同样道理，我们安装好数据库之后，

我们的应用程序(java)也是不能直接使用数据库的，必须要通过相应的数据库驱动程序，通过驱动程序去和数据库打交道。

JDBC介绍

JDBC全称为java database connectivity,是sun公司指定的java数据库连接技术的简称，JDBC是连通java程序和数据库之间的桥梁。

SUN公司为了简化、统一对数据库的操作，定义了一套Java操作数据库的规范（接口），称之为JDBC。

JDBC就是这套接口由数据库厂商去实现，这样，开发人员只需要学习JDBC接口，并通过JDBC加载具体的驱动，就可以操作数据库。

JDBC中常用类详解

1、DriverManager类

Jdbc程序中的DriverManager用于加载驱动，并创建与数据库的链接，这个类的常用方法：

DriverManager.getConnection(url, user, password) 获取数据库连接

2、Connection类

Jdbc程序中的Connection，它用于代表数据库的连接，Connection是数据库编程中最重要的一个对象，

客户端与数据库所有交互都是通过connection对象完成的，这个对象的常用方法：

prepareStatement(sql) ：创建向数据库发送预编译sql的PreparedSatement对象。

3、PreparedStatement类

Jdbc程序中的Statement对象用于向数据库发送SQL语句， Statement对象常用方法：

executeQuery() ：用于向数据发送查询语句并返回结果集

executeUpdate()：用于向数据库发送insert、update或delete语句并返回受影响的行数

4、ResultSet类

Jdbc程序中的ResultSet用于代表Sql语句的执行结果。Resultset封装执行结果时，采用的类似于表格的方式。

ResultSet 对象维护了一个指向表格数据行的游标，初始的时候，游标在第一行之前，调用ResultSet.next() 方法，

可以使游标指向具体的数据行，进行调用方法获取该行的数据。

ResultSet既然用于封装执行结果的，所以该对象提供的都是用于获取数据的get方法：

获取任意类型的数据

getObject(int index)

getObject(string columnName)

获取指定类型的数据，例如：

getString(int index)

getString(String columnName)

jdbc的操作步骤及主要代码：

1、将jar包导入项目

2、加载驱动(Class.forName)

3、连接数据库(DriverManager.getConnection())[拽ver买你这 盖特康耐科神]

4、编写SQL语句

5、创建数据库操作对象(prepareStatement())[普瑞派尔斯dei特门特]

6、执行SQL语句(查询使用executeQuery/增删改使用executeUpdate)

7、关闭连接，释放资源

#### 2、MyBatis

0什么是MyBatis

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。

Hibernate 和 MyBatis 的区别

MyBatis 是一个小巧、方便、高效、简单、直接、半自动化的持久层框架

Hibernate 是一个强大、方便、高效、复杂、间接、全自动化的持久层框架。

MyBatis 需要手动匹配 POJO、SQL 和映射关系，Hibernate只需提供 POJO 和映射关系即可

parameterType（接值）和resultType（传值）

$和#的区别（作用都是取值：基本数据类型、JavaBean、map）

$相当于是拼接，会有SQL注入的问题

#相当于是占位符，有效的防止SQL注入问题

拼接SQL的取值如果是关键字必须用$，如果是条件值用 #

MyBatis的优缺点？

优点：便于维护管理，接近JDBC，比较灵活

缺点：对SQL语句依赖程度很高

MyBatis里配置的信息(数据库信息(数据源JDBC DBCP C3P0 ODBC DRUID处理)、框架自身的信息、映射文件)

具体参考 <https://mybatis.org/mybatis-3/zh/index.html>

具体参考 <http://c.biancheng.net/mybatis/>

##### 1、什么是Mybatis？

（1）Mybatis是一个半ORM（对象关系映射）框架，它内部封装了JDBC，开发时只需要关注SQL语句本身，不需要花费精力去处理加载驱动、创建连接、创建statement等繁杂的过程。程序员直接编写原生态sql，可以严格控制sql执行性能，灵活度高。

（2）MyBatis 可以使用 XML 或注解来配置和映射原生信息，将 POJO映射成数据库中的记录，避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。

（3）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）。

##### 2、Mybaits的优缺点？

（1）sql语句与代码分离，存放于xml配置文件中：

优点：便于维护管理，不用在java代码中找这些语句；

缺点： JDBC方式可以用用打断点的方式调试，但是Mybatis不能，需要通过log4j日志输出日志信息帮助调试，然后在配置文件中修改。

（2）用逻辑标签控制动态SQL的拼接：

优点：用标签代替编写逻辑代码；

缺点：拼接复杂SQL语句时，没有代码灵活，拼写比较复杂。不要使用变通的手段来应对这种复杂的语句。

（3）查询的结果集与java对象自动映射：

优点：保证名称相同，配置好映射关系即可自动映射或者，不配置映射关系，通过配置列名=字段名也可完成自动映射。

缺点：对开发人员所写的SQL依赖很强。

（4）编写原生SQL：

优点：接近JDBC，比较灵活。

缺点：对SQL语句依赖程度很高；并且属于半自动，数据库移植比较麻烦，比如mysql数据库编程Oracle数据库，部分的sql语句需要调整。

##### 3、MyBatis框架适用场合？

（1）MyBatis专注于SQL本身，是一个足够灵活的DAO层解决方案。

（2）对性能的要求很高，或者需求变化较多的项目，如互联网项目，MyBatis将是不错的选择。

##### 4、MyBatis与Hibernate有哪些不同？

（1）Mybatis和hibernate不同，它不完全是一个ORM框架，因为MyBatis需要程序员自己编写Sql语句。

（2）Mybatis直接编写原生态sql，可以严格控制sql执行性能，灵活度高，非常适合对关系数据模型要求不高的软件开发，因为这类软件需求变化频繁，一但需求变化要求迅速输出成果。但是灵活的前提是mybatis无法做到数据库无关性，如果需要实现支持多种数据库的软件，则需要自定义多套sql映射文件，工作量大。

（3）Hibernate对象/关系映射能力强，数据库无关性好，对于关系模型要求高的软件，如果用hibernate开发可以节省很多代码，提高效率。

##### 5、#{}和${}的区别是什么？

{} 是预编译处理，KaTeX parse error: Expected 'EOF', got '#' at position 21: …串替换。

Mybatis在处理#̲{}时，会将sql中的#{}替…{}时，就是把${}替换成变量的值。

使用#{}可以有效的防止SQL注入，提高系统安全性。

##### 6、当实体类中的属性名和表中的字段名不一样 ，怎么办 ？

第1种： 通过在查询的sql语句中定义字段名的别名，让字段名的别名和实体类的属性名一致。

第2种： 通过 来映射字段名和实体类属性名的一一对应的关系。

##### 7、 模糊查询like语句该怎么写?

第1种：在Java代码中添加sql通配符。

第2种：在sql语句中拼接通配符，会引起sql注入

##### 8、通常一个Xml映射文件，都会写一个Dao接口与之对应，请问，这个Dao接口的工作原理是什么？Dao接口里的方法，参数不同时，方法能重载吗？

Dao接口即Mapper接口。接口的全限名，就是映射文件中的namespace的值；接口的方法名，就是映射文件中Mapper的Statement的id值；接口方法内的参数，就是传递给sql的参数。

Mapper接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为key值，可唯一定位一个MapperStatement。在Mybatis中，每一个 select、insert、update、delete 标签，都会被解析为一个MapperStatement对象。

举例： com.mybatis3.mappers.StudentDao.findStudentById ，可以唯一找到namespace为 com.mybatis3.mappers.StudentDao 下面 id 为 findStudentById 的 MapperStatement。

Mapper接口里的方法，是不能重载的，因为是使用 全限名+方法名 的保存和寻找策略。Mapper 接口的工作原理是JDK动态代理，Mybatis运行时会使用JDK动态代理为Mapper接口生成代理对象proxy，代理对象会拦截接口方法，转而执行MapperStatement所代表的sql，然后将sql执行结果返回

##### 9、Mybatis是如何进行分页的？分页插件的原理是什么？

Mybatis使用RowBounds对象进行分页，它是针对ResultSet结果集执行的内存分页，而非物理分页。可以在sql内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

分页插件的基本原理是使用Mybatis提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的sql，然后重写sql，根据dialect方言，添加对应的物理分页语句和物理分页参数。

##### 10、Mybatis是如何将sql执行结果封装为目标对象并返回的？都有哪些映射形式？

第一种是使用 标签，逐一定义数据库列名和对象属性名之间的映射关系。

第二种是使用sql列的别名功能，将列的别名书写为对象属性名。

有了列名与属性名的映射关系后，Mybatis通过反射创建对象，同时使用反射给对象的属性逐一赋值并返回，那些找不到映射关系的属性，是无法完成赋值的。

##### 11、在mapper中如何传递多个参数?

(1) 第一种：使用Java Bean传递多个参数；

(2) 第二种：使用 注解传递多个参数，如@param 注解；

(3) 第三种：使用map接口传递参数，即将多个参数封装成map；

##### 12、Mybatis动态sql有什么用？执行原理？有哪些动态sql？

Mybatis动态sql可以在Xml映射文件内，以标签的形式编写动态sql，执行原理是根据表达式的值 完成逻辑判断并动态拼接sql的功能。

Mybatis提供了9种动态sql标签： trim、where、set、foreach、if、choose、when、otherwise、bind。

元素

作用

备注

if

判断语句

单条件分支判断

choose（when，otherwise）

相当于Java中的switch和case语句

多条件分支判断

trim(where,set)

辅助元素，用于处理特定的SQL拼装问题，比如去掉多余的and,or等

用于处理SQL拼装的问题

foreach

循环语句

在in语句等列举条件常用

动态SQL实际使用的元素并不多，但是它们带来了灵活性，减少了许多工作量的同时，也很大程度上提高了程序的可读性和可维护性。

##### 13、Xml映射文件中，除了常见的select|insert|updae|delete标签之外，还有哪些标签？

resultMap、parameterMap、sql、include、selectKey，加上动态SQL的9个标签，其中 sql 为sql片段标签，通过 include 标签引入sql片段， selectKey 为不支持自增的主键生成策略标签。

##### 14、Mybatis的Xml映射文件中，不同的Xml映射文件，id是否可以重复？

不同的Xml映射文件，如果配置了namespace，那么id可以重复；如果没有配置namespace，那么id不能重复；

原因就是namespace+id是作为Map <String,MapperStatement> 的key使用的，如果没有namespace，就剩下id，那么，id重复会导致数据互相覆盖。有了namespace，自然id就可以重复，namespace不同，namespace+id自然也就不同。

##### 15、为什么说Mybatis是半自动ORM映射工具？它与全自动的区别在哪里？

Hibernate属于全自动ORM映射工具，使用Hibernate查询关联对象或者关联集合对象时，可以根据对象关系模型直接获取，所以它是全自动的。而Mybatis在查询关联对象或关联集合对象时，需要手动编写sql来完成，所以，称之为半自动ORM映射工具。

##### 16、MyBatis实现一对一有几种方式?具体怎么操作的？

有联合查询和嵌套查询,联合查询是几个表联合查询,只查询一次, 通过在resultMap里面配置association节点配置一对一的类就可以完成；

嵌套查询是先查一个表，根据这个表里面的结果的 外键id，去再另外一个表里面查询数据,也是通过association配置，但另外一个表的查询通过select属性配置。

##### 17、MyBatis实现一对多有几种方式,怎么操作的？

有联合查询和嵌套查询。联合查询是几个表联合查询,只查询一次,通过在resultMap里面的collection节点配置一对多的类就可以完成；嵌套查询是先查一个表,根据这个表里面的 结果的外键id,去再另外一个表里面查询数据,也是通过配置collection,但另外一个表的查询通过select节点配置。

##### 18、什么是MyBatis的接口绑定？有哪些实现方式？

接口绑定，就是在MyBatis中任意定义接口,然后把接口里面的方法和SQL语句绑定, 我们直接调用接口方法就可以,这样比起原来了SqlSession提供的方法我们可以有更加灵活的选择和设置。

接口绑定有两种实现方式,一种是通过注解绑定，就是在接口的方法上面加上 @Select、@Update等注解，里面包含Sql语句来绑定；另外一种就是通过xml里面写SQL来绑定, 在这种情况下,要指定xml映射文件里面的namespace必须为接口的全路径名。当Sql语句比较简单时候,用注解绑定, 当SQL语句比较复杂时候,用xml绑定,一般用xml绑定的比较多

##### 19、使用MyBatis的mapper接口调用时有哪些要求？

（1）Mapper接口方法名和mapper.xml中定义的每个sql的id相同；

（2）Mapper接口方法的输入参数类型和mapper.xml中定义的每个sql 的parameterType的类型相同；

（3）Mapper接口方法的输出参数类型和mapper.xml中定义的每个sql的resultType的类型相同；

（4）Mapper.xml文件中的namespace即是mapper接口的类路径。

##### 20、Mapper编写有哪几种方式？

第一种：接口实现类继承SqlSessionDaoSupport：使用此种方法需要编写mapper接口，mapper接口实现类、mapper.xml文件。

1、在sqlMapConfig.xml中配置mapper.xml的位置

2、定义mapper接口

3、实现类集成SqlSessionDaoSupport mapper方法中可以this.getSqlSession()进行数据增删改查。

4、spring 配置

第二种：使用 org.mybatis.spring.mapper.MapperFactoryBean ：

1、在sqlMapConfig.xml中配置mapper.xml的位置，如果mapper.xml和mappre接口的名称相同且在同一个目录，这里可以不用配置

2、定义mapper接口：

3、mapper.xml中的namespace为mapper接口的地址 2、mapper接口中的方法名和mapper.xml中的定义的statement的id保持一致 3、Spring中定义

第三种：使用mapper扫描器：

1、mapper.xml文件编写：

mapper.xml中的namespace为mapper接口的地址； mapper接口中的方法名和mapper.xml中的定义的statement的id保持一致； 如果将mapper.xml和mapper接口的名称保持一致则不用在sqlMapConfig.xml中进行配置。

2、定义mapper接口：

注意mapper.xml的文件名和mapper的接口名称保持一致，且放在同一个目录

3、配置mapper扫描器：

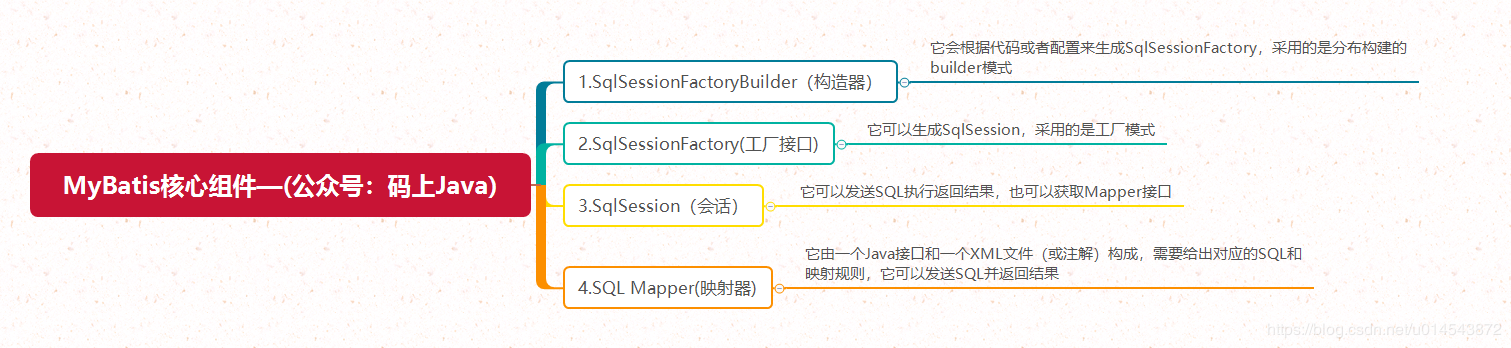
4、使用扫描器后从spring容器中获取mapper的实现对象。

21、简述Mybatis的插件运行原理，以及如何编写一个插件？

Mybatis仅可以编写针对ParameterHandler、ResultSetHandler、StatementHandler、Executor这4种接口的插件，Mybatis使用JDK的动态代理，为需要拦截的接口生成代理对象以实现接口方法拦截功能，每当执行这4种接口对象的方法时，就会进入拦截方法，具体就是InvocationHandler的invoke()方法，当然，只会拦截那些你指定需要拦截的方法。

编写插件：实现Mybatis的Interceptor接口并复写intercept()方法，然后在给插件编写注解，指定要拦截哪一个接口的哪些方法即可，记住，别忘了在配置文件中配置你编写的插件

##### MyBatis的核心组件有哪些？



#### 3、JPA

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。

Spring Data JPA是什么

是一个基于ORM的框架(规范)，简化了持久层的开发

Spring Data JPA的核心接口

 1、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。

 2、CrudRepository：是Repository的子接口，提供CRUD(增删改查)的功能。

 3、Paging(培经)AndSorting(消停)Repository：是CrudRepository的子接口，添加分页和排序的功能。

 4、JpaRepository：是PagingAndSortingRepository的子接口，增加了一些实用的功能，例如批量操作。

 5、JpaSpecification(丝拍谁非K鑫)Executor(A克丝Q特儿)：用来做负责查询的接口。

6、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。

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//

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

4 Hibernate

**1.hibernate运行原理**

1.通过configuration【康佛瑞可神】加载了hibernate.cfg.xml文件。【还播奶特】

2.加载后创建sessionFactory【塞申反克瑞】

3.然后通过sessionFactory获得session。【塞申】

4.通过session操作数据库，最后通过transaction来进行事物的控制.【春赛克神】

**2.hibernate的五大核心接口**

1.Configuration(类):【康佛瑞可神】

加载配置文件hibernate.cfg.xml文件中的配置信息，从而得到：【还播奶特】

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)

2.sessionFactory（接口）

通过configuration创建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.

3.Session(接口)

线程不安全。

相当于jbdc的connection.用来操作数据库.【康奶可深】

4.Transaction（接口）【春赛克神】

进行事物的控制 实现事物的commit（提交)和rollback（回滚）

5.Query(接口)

用来执行数据库的查询操作。

**3.hibernate主键生成策略:**

1）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

2）sequence:oracle数据库会自行处理，会增加一个序列配置

3）assigned(爱字三恩的)：人工指派不给的话会报错。

4）native（通用多个数据库，数据库的本地生成策略）【nai 特否】

5）uuid：生成一个32位的，不重复的字符串，可以达到跨数据库。

**4.jdbc和hibernate的区别。**【还播奶特】

hibernate是JDBC的轻量级封装，hibernate基于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是全自动的，跨数据，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

**5.hibernate中query的常用方法。**

1）excuteUpdate()用于hql语句的执行修改和删除

2）uniqueResult()返回单条记录

3）setFirstResult()设置分页时的查询开始下标

4）setMaxResults()设置每页条数

5）list()方法返回查询结果集

**6.hibernate中session中的常用方法**

save():保存【谁我】

saveOrUpdate()保存或者修改【谁我奥】

delete()删除

update()修改

get()立即加载

load()延迟加载

**7.get和load的区别**

1)load是延迟加载，查询是不会立即发送sql语句，当访问是对象时发送sql语句， 访问实体不存在时返回null,如果设置了lazy=“false”,load就不再进行延迟加载

2)get立即加载 查询是立即发送sql语句，当访问实体不存在是返回null

8.hibernate三种状态

临时状态、持久化状态、托管状态/游离状态

Transient【春z恩特】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【排赛ten特】(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【地忒拆的】(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

### 2 controller层

controller 控制层 一般用来接收请求和响应(接参、传参、跳转页面)

1 springmvc

**1.Springmvc是如何搭建的**

1. 导入jar包
2. 在web中配置核心控制器
3. 把spring-mvc.xml文件导入到resources目录下【瑞骚死】
4. 配置开启注解和扫描注解以及视图解析器
5. 在controller目录下创建\*controller类 再在类上加@Controller注解 @RequestMapping注解可加可不加 在方法上一定要加上@RequestMapping 前台请求是根据requestmapping中的值来找到对应的方法的

**2.Springmvc接收值得几种方式？**

1.HttpServletRequest接收

2.对象接收

3.属性接收

4.@RequestParam接收【帕瑞米】

**3.Springmvc返回页面的几种方式**

1. 通过modelandview【为哦】 构造函数 可以指定返回的页面 也可以通过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. 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modelMap中
4. 通过request.getRequestDispartcherServlet返回【第四排车】
5. 如果返回json的话 可以加@responsebody【瑞思胖死报得】注解 把数据返回到当前页面

6.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

7.Springmvc生命周期(运行原理)：

用户发起请求到前端控制器

前端控制器请求HandlerMapping【】查找 Handler 【韩的乐】

可以根据xml配置、注解进行查找

处理器映射器HandlerMapping向前端控制器返回Handler

前端控制器调用处理器适配器去执行Handler

Handler执行完成给适配器返回ModelAndView

处理器适配器向前端控制器返回ModelAndView

ModelAndView是springmvc框架的一个底层对象，包括 Model和view

前端控制器 请求 视图解析器 去进行视图解析 根据 逻辑视图名 解析成真正的视图(jsp)

视图解析器向前端控制器返回View

前端控制器进行视图渲染 视图渲染将模型数据(在ModelAndView对象中)填充到request域

前端控制器向用户响应结果

SpringMVC运行原理(生命周期)

1用户发送一个请求给核心控制器

2核心控制器去映射器找到相对应的hanlder 并返回给核心控制器

3核心控制器找适配器执行方法，返回一个modelandview 给核心控制器

4核心控制器将jsp进行解析发送给视图解析器渲染数据 在返回给核心控制器

5核心控制器给用户一个响应

#### 2 Spring Boot

1什么是Spring Boot

微服务(一组相对较小且独立的功能单元)

Spring Boot 是由 Pivotal 团队提供用来简化 Spring 的搭建和开发过程的全新框架。

Spring Boot 去除了大量的 xml 配置文件，简化了复杂的依赖管理，配合各种starter 使用，基本上可以做到自动化配置。

2 接口开发

接口开发，可以理解成面向接口开发(接口实际上就是定义的一组规范，并且接口开发具有解耦和可扩展的优点)

定义接口文档：请求路径+请求方式+请求参数+返回值

**3设计目的：** 用来简化新 Spring 应用的初始搭建以及开发过程。

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

4常用注解

1、@SpringBootApplication

Spring Boot最核心的注解, 用在 Spring Boot的主类上，标识这是一个 Spring Boot 应用，用来开启 Spring Boot 的各项能力。 @Configuration,@EnableAutoConfiguration,@ComponentScan三个注解的组合。

2、@PathVariable

路径变量。参数与大括号里的名字一样要相同。

3、@RestController

用于标注控制层组件(如struts中的action)，表示这是个控制器bean,并且是将函数的返回值直接填入HTTP响应体中,是REST风格的控制器；它是@Controller和@ResponseBody的合集。

**@EnableAutoConfiguration**

允许 Spring Boot 自动配置注解，开启这个注解之后，Spring Boot 就能根据当前类路径下的包或者类来配置 Spring Bean。

**@Configuration**

用于定义配置类，指出该类是 Bean 配置的信息源，相当于传统的xml配置文件，一般加在主类上。如果有些第三方库需要用到xml文件，建议仍然通过@Configuration类作为项目的配置主类——可以使用@ImportResource注解加载xml配置文件。

**@ComponentScan**

组件扫描。让spring Boot扫描到Configuration类并把它加入到程序上下文。@ComponentScan注解默认就会装配标识了@Controller，@Service，@Repository，@Component注解的类到spring容器中。

具体参考 <http://c.biancheng.net/view/4630.html>

具体参考 <https://blog.csdn.net/qq_40147863/article/details/84194493>

具体参考 <https://www.yiibai.com/spring-boot>

面试问题 <https://blog.csdn.net/zhoushimiao1990/article/details/99713372>

注解参考 <https://www.php.cn/faq/417047.html>

注解参考 <https://www.cnblogs.com/caizhaokai/p/10982727.html>

springboot+mybatis的注解

3 view 层

view 视图 一般代表JSP页面，可视化视图

1 jquery

jquery是一个轻量级的js框架，具有跨浏览器的特性，兼容性好，并且封装了很多工具，方便使用。

常用的有: 选择器 ，dom操作 ，ajax(ajax不能跨域) ，特效，工具类

常用的选择器 :

1）$("#id") id选择器

2）$(".class")class选择器

3) $("[属性=属性值]")属性选择器

4) $("标签名称")标签选择器

常用的属性 :

attr():【奥特】给标签增加属性

prop():【普若不】获取元素属性（可以赋值）

css():给标签增加样式

append()：【额喷的】追加子标签

html():获得指定文本内容，也可赋值

val（）:获取表单中的文本框的值，也可赋值.

#### 2 bootstart

1 什么是bootstarp？

Bootstrap是目前最受欢迎的前端框架。

Bootstrap 是基于 HTML、CSS、JAVASCRIPT 的，它简洁灵活，使得 Web 开发更加快捷。

2 环境搭建

下载资源包（官网）

引入js css（引入jquery bootstrap基于jquery）

全局css

container （div 都要加 ）

按钮 btn primary success info warning danger

表单 结构 form div lable input class

表格 几种样式

图片 几种样式

网格系统 div class row 12列

下拉菜单 div class="dropdown" buttton ul

导航栏 结构 标签 nav 一般情况 两个div 第一个div class="nav-header"公司log 网站的log

面板 结构 标签 div class="panel " 一般情况下 div头 div 内容

字体图标

3 layui

1 什么是layui？

是一款采用自身模块规范编写的前端 UI 框架。

2 环境搭建

下载资源（js ，css）

按照语法

3 记住关键点

<div 声明网格系统>

<div 占多少分 ></div>

<div 占多少分 ></div>

</div>

form结构

<form>

<div>

<lable></lable>

<div>

</div>

</div>

</form>

js 语法

模块化语法

layer form table laydate upload

//非模块化加载

；！function（）{

}（）；

//模块化加载

layui.use('',function(){

})

关于属性的作用

#### 4、Vue

**什么是Vue**(所有东西都是**响应式的**)

Vue是一套用于构建用户界面的**渐进式框架。**

Vue与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。

Vue的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。

另一方面，当与**[现代化的工具链](https://cn.vuejs.org/v2/guide/single-file-components.html)**以及各种**[支持类库](https://github.com/vuejs/awesome-vue" \l "libraries--plugins" \t "_blank)**结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

**Vue的优点**

轻量级框架：只关注视图层，是一个构建数据的视图集合；

简单易学：国人开发，中文文档，不存在语言障碍 ，易于理解和学习；

组件化：实现了html的封装和重用，在构建单页面应用方面有着独特的优势；

双向数据绑定：在数据操作方面更为简单；

视图、数据、结构分离：使数据的更改更为简单，不需要进行逻辑代码的修改，只需要操作数据就能完成相关操作；

运行速度更快

**Vue常用属性**

**el属性**

用来指示Vue编译器从什么地方开始解析Vue的语法，可以说是一个占位符

**data属性**

用来组织从view中抽象出来的属性，可以说将视图的数据抽象出来存放在data中(设置值)

**methods属性**

放置页面中的业务逻辑，JS方法一般都放置在methods中

**computed属性**

用来计算(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。)

**watch属性**

监听data中数据的变化；两个参数，一个返回新值，一个返回旧值

**template属性**

用来设置模板，会替换页面元素，包括占位符

Vue常用指令

**(1)v-text**  
v-text主要用来更新textContent，可以等同于JS的text属性。

**(2)v-html**  
它等同于JS的innerHtml属性。

**(3)v-if**  
v-if可以实现条件渲染，Vue会根据表达式的值的真假条件来渲染元素。

如果属性值为true，则显示。否则，不会渲染这个元素。

**(4)v-else**  
v-else是搭配v-if使用的，它必须紧跟在v-if或者v-else-if后面，否则不起作用。

**(5)v-else-if**  
v-else-if充当v-if的else-if块,可以链式的使用多次.可以更加方便的实现switch语句。

**(6)v-show**

也是用于根据条件展示元素。和v-if不同的是，如果v-if的值是false，则这个元素被销毁，不在Dom中。但是v-show的元素会始终被渲染并保存在Dom中，它只是简单的切换CSS的dispaly属性

**(7)v-for**  
用v-for指令根据遍历数组来进行渲染

(注意：当v-for和v-if同处于一个节点时，v-for的优先级比v-if更高。这意味着v-if将运行在每个v-for循环中)

**(8)v-bind**  
v-bind用来**动态的绑定一个或者多个特性**。没有参数时，可以绑定到一个包含键值对的对象。常用于动态绑定class和style。以及href等。简写为一个冒号 “ : ”

**(9)v-model**  
这个指令用于在表单上创建**双向数据绑定**。  
v-model会忽略所有表单元素的value、checked、selected特性的初始值。因为它选择Vue实例数据做为具体的值。

**(10)v-on**  
v-on主要用来监听dom事件，以便执行一些代码块。表达式可以是一个方法名。  
简写为 @

常用修饰符(事件修饰符)

.stop 阻止点击事件冒泡

.once 只执行一次，只要点击按钮只会执行一次。

生命周期  钩子函数

一个组件从创建到销毁的过程就是生命周期。

**beforeCreate:创建前**

1、当前vue实例化的时候会做一个初始化的操作，在这个生命周期函数里面我们可以做初始化的loading

2、在当前函数里面是访问不到data中的属性，但是可以通过vue的实例对象进行访问

**created:创建后**

1、当beforeCreate执行完毕以后，会执行created. 在当前函数中我们可以访问到data中的属性

2、当前生命周期函数执行的时候会将data中所以的属性和methods身上所以的方法添加到vue的实例身上，同时会将data中所有的属性添加一个getter/setter方法

3、如果需要进行前后端上数据交互(ajax请求的时候) 需要在当前生命周期中使用

**beforeMount:挂载前 (渲染)**

render函数初次被调用---->数据和模板没有进行相结合，同时还没有渲染到html页面上可以在此做渲染前data中数据最后的修改

**mounted:挂载后**

1、数据和模板进行相结合，渲染成真实的DOM结构

2、在当前生命周期函数里面我们可以访问到真实的DOM结构,

3、在vue中我们可以通过$refs来访问到真实的DOM结构

4、ref类似与id一样 值必须是唯一的   访问的时候我们可以通过this.$refs.属性

**beforeDestroy:销毁前**

销毁之前还可以访问到DOM结构  以及相关的数据(data)

在这个生命周期函数中我们可以将绑定的事件进行移除

**destroyed:销毁后**

在这个生命周期函数中会将数据和模板之间的关系断开(不是你的做的)

在这个生命周期函数中我们还是可以访问到data中的属性，但是访问不到真实的DOM结构了

**beforeUpdate:更新前**

只要data中的属性发生了改变,那么这个生命周期就会执行,render函数再次会执行。

在这个生命周期函数中我们可以对数据进行最后的修改，同时也可以访问到最新的DOM结构和数据

**updated:更新后**

在当前生命周期函数中我们可以访问到最新的DOM结构(数据更新后最新的DOM结构)和数据

路由概念

路由用于设定访问路径，并将路径和组件映射起来。简单来说就是来做导航栏的。

路由实际上就是可以理解为指向，就是我在页面上点击一个按钮需要跳转到对应的页面，这就是路由跳转

组件详解(Component)

组件是Vue.js最强大的功能之一。组件可以扩展HTML元素，封装可以重用的代码。

**Prop**

prop 是子组件用来接受父组件传递过来的数据的一个自定义属性。

父组件的数据需要通过 props 把数据传给子组件，子组件需要显式地用 props 选项声明 "prop"

脚手架(node.js)环境搭建命令

**1、下载安装** <https://nodejs.org/en/download/>

**验证是否安装成功** node -v npm (相当于maven，Js的仓库，仓库在国外)

**2、两种修改仓库方法**

(1)修改npm的仓库地址 npm config set registry https://registry.npm.taobao.org

配置后可通过下面方式来验证是否成功

npm config get registry 或 npm info express

(2) 安装cnpm(设置国内的)

npm install cnpm -g --registry=https://registry.npm.taobao.org

**3**、**安装vue-cli** cnpm install -g vue-cli

**验证** vue -V

**4、创建项目**

命令：vue init webpack my-project

显示结果

1、? Project name shiro-web 项目名

2、? Project description a shop project 项目名描述

3、? Author xijunge 作者

4、? Install vue-router? Yes 是否需要路由选择Y

5、? Use ESLint to lint your code? No 是否需要ESlist语法检查

6、? Set up unit tests No 是否需要单元测试

7、? Setup e2e tests with Nightwatch? No 是否需要e2e测试

8、? Should we run `npm install` for you after the project has been created? (recommended) no 这个地方需要选择最后一个自己完成启动

**5、启动项目**

cnpm(npm) install

cnpm(npm) run dev

localhost:8080

脚手架中的vue代码模块(格式)

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

引入组件方式 import 变量 from 组件路径

**什么是跨域**

简单来说：一个域请求另一个域的数据

跨域问题的出现是因为浏览器的同源策略问题，浏览器请求必须遵循同源策略：同协议、同域名、同端口

三种解决跨域方法

1. CORS跨域：服务端设置，前端直接调用，后台允许前端某个站点进行访问

加注解

2、JSONP跨域：前端适配，后台配合，前后台同时改造

JSONP 是服务器与客户端跨源通信的常用方法。最大特点就是简单适用，兼容性好(兼容低版本IE)，缺点是只支持get请求，不支持post请求。

3

钩子函数参考 <https://blog.csdn.net/qq_35585701/article/details/81216704>

组件参考 <https://www.jianshu.com/p/f5eb996f5e2a>

路由参考 <https://blog.csdn.net/m0_45899013/article/details/106680880>

路由参考 <https://www.cnblogs.com/zwnsyw/p/12307208.html#autoid-1-6-0>

跨域参考 <https://blog.csdn.net/qq_26652185/article/details/109238544>

跨域参考 <https://blog.csdn.net/qq_38128179/article/details/84956552>

具体参考 <https://www.runoob.com/vue2/vue-tutorial.html>

具体参考 <https://cn.vuejs.org/v2/guide/index.html>

面试问题 <https://zhuanlan.zhihu.com/p/92407628>

## 2、Spring

Spring就是一个粘合剂，就是用来融合其他框架

DI依赖注入： 处理对象间的依赖关系

依赖注入的方式有:

常用的有：属性注入（也就是set方法注入），构造函数注入。

不常用的：接口注入

IOC的概念:

IOC是一个概念,并不是一项技术

IOC也称控制反转，就是把原先控制代码对象的生产由代码转换到IOC容器中去实现。作用是为了解耦，降低类之间的耦合度。

其设计思想就是设计模式的工厂模式，我们并不需要知道其生产的具体过程，我们只要其产出的对象即可。

其工作流程就是：在Spring容器启动的时候，Spring会把你在applicationContext.xml中配置好的bean都初始化，在你需要调用的时候，把已经初始化的bean分配给你要调用这些bean的类，而不用去创建一个对象的实例。

AOP的理解

AOP：面向切面编程是软件编程思想发展到一定阶段的产物，是面向对象编程(OOP)的有益补充。

AOP一般适用于具有横切逻辑的场合，如安全控制、事务管理、日志记录、性能统计等。

面向切面编程简单地说就是在不改变源程序的基础上为代码段增加新的功能，对代码段进行增强处理。

AOP的代理通过JDK动态代理，也可以通过cglib（谁个来波）实现，默认是通过JDK动态代理实现的。JDK动态代理需要接口的支持，如果没有接口只有类，则使用cglib实现。

AOP的术语

1、通知：切面的工作被称为通知。通知定义了切面是什么以及何时使用。除了描述切面要完成的工作，通知还解决了何时执行这个工作的问题。  
5种通知类型：

**前置通知（Before）**：在目标方法被调用之前调用通知功能  
**后置通知（After）**：在目标方法完成之后调用通知，此时不会关心方法的输出是什么  
**返回通知（After-returning）**：在目标方法成功执行之后调用通知  
**异常通知（After-throwing）**：在目标方法抛出异常后调用通知  
**环绕通知（Around）**：通知包裹了被通知的方法，在被通知的方法调用之前和之后执行自定义的行为

2、切点：一个切面并不需要通知应用的所有连接点，切点有助于缩小切面所通知的连接点范围。如果说通知定义了切面的“什么”和“何时”的话，那么切点就定义了“何处”。因此，切点其实就是定义了需要执行在哪些连接点上执行通知。

3、切面：切面是通知和切点的结合。通知和切点共同定义了切面的全部内容——它是什么，在何时和在何处完成其功能。

详情参考：<https://www.cnblogs.com/sandea/p/11175834.html>

spring 的优点

1.依赖注入(DI)方法使得JavaBean中的依赖关系一目了然

2.IOC容器更轻量级使用了单例，节约内存。

3.iOC控制反转技术实现了解耦。对象通过spring容器管理，不用再程序中实例化。

4.集成了相应的组件，让测试更方便。

5.aop可以在不修改业务代码的情况下，增加新的功能。如日志，1登录拦截，事务管理，统计分析，安全认证.

## 3、Spring MVC

Spring MVC是一个控制层框架(主要是做接参、传参、页面跳转)

配置文件(扫描包路径、视图解析器、文件解析器、拦截器、放开静态资源)

接参的方式有： request、基本数据类型、对象(JavaBean)

传参的方式有： 转发(model)传值、json传值(@ResponseBody)

跳转页面： 重定向(redirect)、转发(forward)

常用的注解有

@Controller 标明为控制层

@RequestMapping 请求路径、地址

@ResponseBody 返回的是json数据

@RequestParam 接参、定义值的属性名

@ControllerAdvice 全局异常处理

@ExceptionHandler 标明是什么方法异常

@DataTimeFormat 时间格式的转换(前后到后台)

Spring MVC的运行原理

用户发送一个请求到核心控制器DispatcherServlet

DispatcherServlet根据HandlerMapping找到相应的Handler(根据xml的配置、注解进行查找)

映射器将找到的Handler返回给DispatcherServlet

DispatcherServlet通过适配器(handlerAdapter)去执行Handler

执行成功后得到一个ModelAndView并返回给适配器(HandlerAdapter)

适配器再将ModelAndView返回给DispatcherServlet

DispatcherServlet通过视图解析器解析JSP，解析后返回一个view给核心控制器

然后在DispatcherServlet里进行视图渲染，最后DispatcherServlet向用户响应结果

知识点 参考<https://blog.csdn.net/striner/article/details/80442500>

面试题 参考<https://blog.csdn.net/cx521600/article/details/90241944>

## 4、maven概念

(Maven是用来构建项目、管理依赖和打包的工具)。

打包命令：mvn clean package

优点： 1、使用maven创建出来的项目具有统一的结构，节约项目的空间

2、规范jar包的下载，Maven可以把一个jar包所依赖的jar包也一并下载下来

3、通过maven可以对项目进行编译、测试、打包、部署操作

缺点： 耦合度高，系统复杂

Maven中Setting.xml、pom.xml文件

settings.xml是maven的全局配置文件，里面包括：本地仓库位置、远程仓库服务器

pom.xml主要描述了项目的maven坐标，依赖关系

## 5、什么是json

(一种轻量级的数据交换格式)

json格式：就是在{}中存储键值对，键和值之间用冒号分隔，键值对之间用逗号分隔;

XML特点：有且只有一个根节点；数据传输的载体；所有的标签都需要自定义；是纯文本文件

区别：传输同样格式的数据，xml需要使用更多的字符进行描述，流行的是基于json的数据传输；xml的层次结构比json更清晰

共同点：xml和json都是数据传输的载体，并且具有跨平台跨语言的特性。(为了统一规范)

XML的优缺点：

优：格式统一，符合标准；容易与其他系统进行远程交互，数据共享比较方便。

缺：XML文件庞大，文件格式复杂，传输占带宽；

JSON的优缺点：

优：数据格式比较简单，易于读写，格式都是压缩的，占用带宽小；易于解析，支持多种语言

缺：没有XML格式这么推广的深入人心和喜用广泛，没有XML那么通用性

具体参考 <https://www.cnblogs.com/songyoulian/p/10944205.html>

具体参考 <https://www.runoob.com/json/json-tutorial.html>

## 8、Vue

**什么是Vue**(所有东西都是**响应式的**)

Vue是一套用于构建用户界面的**渐进式框架。**

Vue与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。

Vue的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。

另一方面，当与**[现代化的工具链](https://cn.vuejs.org/v2/guide/single-file-components.html)**以及各种**[支持类库](https://github.com/vuejs/awesome-vue" \l "libraries--plugins" \t "_blank)**结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

**Vue的优点**

轻量级框架：只关注视图层，是一个构建数据的视图集合；

简单易学：国人开发，中文文档，不存在语言障碍 ，易于理解和学习；

组件化：实现了html的封装和重用，在构建单页面应用方面有着独特的优势；

双向数据绑定：在数据操作方面更为简单；

视图、数据、结构分离：使数据的更改更为简单，不需要进行逻辑代码的修改，只需要操作数据就能完成相关操作；

运行速度更快

**Vue常用属性**

**el属性**

用来指示Vue编译器从什么地方开始解析Vue的语法，可以说是一个占位符

**data属性**

用来组织从view中抽象出来的属性，可以说将视图的数据抽象出来存放在data中(设置值)

**methods属性**

放置页面中的业务逻辑，JS方法一般都放置在methods中

**computed属性**

用来计算(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。)

**watch属性**

监听data中数据的变化；两个参数，一个返回新值，一个返回旧值

**template属性**

用来设置模板，会替换页面元素，包括占位符

Vue常用指令

**(1)v-text**  
v-text主要用来更新textContent，可以等同于JS的text属性。

**(2)v-html**  
它等同于JS的innerHtml属性。

**(3)v-if**  
v-if可以实现条件渲染，Vue会根据表达式的值的真假条件来渲染元素。

如果属性值为true，则显示。否则，不会渲染这个元素。

**(4)v-else**  
v-else是搭配v-if使用的，它必须紧跟在v-if或者v-else-if后面，否则不起作用。

**(5)v-else-if**  
v-else-if充当v-if的else-if块,可以链式的使用多次.可以更加方便的实现switch语句。

**(6)v-show**

也是用于根据条件展示元素。和v-if不同的是，如果v-if的值是false，则这个元素被销毁，不在Dom中。但是v-show的元素会始终被渲染并保存在Dom中，它只是简单的切换CSS的dispaly属性

**(7)v-for**  
用v-for指令根据遍历数组来进行渲染

(注意：当v-for和v-if同处于一个节点时，v-for的优先级比v-if更高。这意味着v-if将运行在每个v-for循环中)

**(8)v-bind**  
v-bind用来**动态的绑定一个或者多个特性**。没有参数时，可以绑定到一个包含键值对的对象。常用于动态绑定class和style。以及href等。简写为一个冒号 “ : ”

**(9)v-model**  
这个指令用于在表单上创建**双向数据绑定**。  
v-model会忽略所有表单元素的value、checked、selected特性的初始值。因为它选择Vue实例数据做为具体的值。

**(10)v-on**  
v-on主要用来监听dom事件，以便执行一些代码块。表达式可以是一个方法名。  
简写为 @

**常用修饰符(事件修饰符)**

.stop 阻止点击事件冒泡

.once 只执行一次，只要点击按钮只会执行一次。

**生命周期  钩子函数**

一个组件从创建到销毁的过程就是生命周期。

**beforeCreate:创建前**

1、当前vue实例化的时候会做一个初始化的操作，在这个生命周期函数里面我们可以做初始化的loading

2、在当前函数里面是访问不到data中的属性，但是可以通过vue的实例对象进行访问

**created:创建后**

1、当beforeCreate执行完毕以后，会执行created. 在当前函数中我们可以访问到data中的属性

2、当前生命周期函数执行的时候会将data中所以的属性和methods身上所以的方法添加到vue的实例身上，同时会将data中所有的属性添加一个getter/setter方法

3、如果需要进行前后端上数据交互(ajax请求的时候) 需要在当前生命周期中使用

**beforeMount:挂载前 (渲染)**

render函数初次被调用---->数据和模板没有进行相结合，同时还没有渲染到html页面上可以在此做渲染前data中数据最后的修改

**mounted:挂载后**

1、数据和模板进行相结合，渲染成真实的DOM结构

2、在当前生命周期函数里面我们可以访问到真实的DOM结构,

3、在vue中我们可以通过$refs来访问到真实的DOM结构

4、ref类似与id一样值必须是唯一的 访问的时候我们可以通过this.$refs.属性

**beforeDestroy:销毁前**

销毁之前还可以访问到DOM结构  以及相关的数据(data)

在这个生命周期函数中我们可以将绑定的事件进行移除

**destroyed:销毁后**

在这个生命周期函数中会将数据和模板之间的关系断开(不是你的做的)

在这个生命周期函数中我们还是可以访问到data中的属性，但是访问不到真实的DOM结构了

**beforeUpdate:更新前**

只要data中的属性发生了改变,那么这个生命周期就会执行,render函数再次会执行。

在这个生命周期函数中我们可以对数据进行最后的修改，同时也可以访问到最新的DOM结构和数据

**updated:更新后**

在当前生命周期函数中我们可以访问到最新的DOM结构(数据更新后最新的DOM结构)和数据

路由概念

路由用于设定访问路径，并将路径和组件映射起来。可以用来做导航栏的。

路由实际上就是可以理解为指向，就是我在页面上点击一个按钮需要跳转到对应的页面，这就是路由跳转

格式：<router-link/> <router-view/>

组件详解(Component)

组件是Vue.js最强大的功能之一。组件可以扩展 HTML元素，封装可以重用的代码。

**Prop**

prop 是子组件用来接受父组件传递过来的数据的一个自定义属性。

父组件的数据需要通过props 把数据传给子组件，子组件需要显式地用 props 选项声明 "prop"

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

脚手架(node.js)环境搭建命令

**1、下载安装** <https://nodejs.org/en/download/>

**验证是否安装成功** node -v npm (相当于maven，Js的仓库，仓库在国外)

**2、两种修改仓库方法**

(1)修改npm的仓库地址 npm config set registry https://registry.npm.taobao.org

配置后可通过下面方式来验证是否成功

npm config get registry 或 npm info express

(2) 安装cnpm(设置国内的)

npm install cnpm -g --registry=https://registry.npm.taobao.org

**3**、**安装vue-cli** cnpm install -g vue-cli

**验证** vue -V

**4、创建项目(推荐进入到项目目录CMD)**

命令：vue init webpack my-project

显示结果

1、? Project name shiro-web 项目名

2、? Project description a shop project 项目名描述

3、? Author xijunge 作者

4、? Install vue-router? Yes 是否需要路由选择Y

5、? Use ESLint to lint your code? No 是否需要ESlist语法检查

6、? Set up unit tests No 是否需要单元测试

7、? Setup e2e tests with Nightwatch? No 是否需要e2e测试

8、? Should we run `npm install` for you after the project has been created? (recommended) no 这个地方需要选择最后一个自己完成启动

**5、启动项目**

cnpm(npm) install

cnpm(npm) run dev

localhost:8080

脚手架中的vue代码模块(格式)

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

引入组件方式 import 变量 from 组件路径

**什么是跨域**

简单来说：一个域请求另一个域的数据

跨域问题的出现是因为浏览器的同源策略问题，浏览器请求必须遵循同源策略：同协议、同域名、同端口

解决跨域方法

1. CORS跨域：服务端设置，前端直接调用，后台允许前端某个站点进行访问

@CrossOrigin

2、JSONP跨域：前端适配，后台配合，前后台同时改造

JSONP 是服务器与客户端跨源通信的常用方法。最大特点就是简单适用，兼容性好(兼容低版本IE)，缺点是只支持get请求，不支持post请求。

请求传值失败解决方案(原因：)

1、前端解决：@RequestBody

2、后端解决：import qs from “qs”; qs.stringify()

钩子函数参考 <https://blog.csdn.net/qq_35585701/article/details/81216704>

组件参考 <https://www.jianshu.com/p/f5eb996f5e2a>

路由参考 <https://blog.csdn.net/m0_45899013/article/details/106680880>

路由参考 [https://www.cnblogs.com/zwnsyw/p/12307208.html#autoid-1-6-0](https://www.cnblogs.com/zwnsyw/p/12307208.html" \l "autoid-1-6-0)

跨域参考 <https://blog.csdn.net/qq_26652185/article/details/109238544>

跨域参考 <https://blog.csdn.net/qq_38128179/article/details/84956552>

具体参考 <https://www.runoob.com/vue2/vue-tutorial.html>

具体参考 <https://cn.vuejs.org/v2/guide/index.html>

面试问题 <https://zhuanlan.zhihu.com/p/92407628>

## 9、Element-UI

Element，一套为开发者、设计师和产品经理准备的基于 Vue 2.0 的桌面端组件库

使用方式

npm命令安装 npm i element-ui -S

引入 Element

import ElementUI from 'element-ui'

import 'element-ui/lib/theme-chalk/index.css'

Vue.use(ElementUI)

使用请参考官网

## 10、GIT

Git是分布式的版本控制系统(SVN是集中式控制系统)

安装下载地址：<https://git-scm.com/download/win>

安装完成在桌面点击右键，打开git bash here，在创建的文件夹目录下执行以下命令

git config --global user.name "Your Name"

git config --global user.email "email@example.com"

基本命令

1. git init 把目录变成Git可以管理的仓库
2. git add 文件名 告诉Git，把文件添加到仓库
3. git **commit** -m "提示语" 告诉Git，把文件提交到仓库
4. git status 命令查看修改结果
5. git diff 文件名

查看具体修改了什么内容，查看difference，显示的格式正是Unix通用的diff格式

1. git log 显示从最近到最远的提交日志(可加参数--pretty=oneline)
2. git re**set** --hard HEAD^ 或 git re**set** --hard 1094a

回退到上一个版本

1. git reflog 查看记录你的每一次命令
2. git checkout - 文件 把文件在工作区的修改全部撤销

一种是文件自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；

一种是文件已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。

10、git re**set** HEAD readme.txt

可以把暂存区的修改撤销掉（unstage），重新放回工作区

远程仓库推送与克隆命令

推送

1、ssh-keygen -t rsa -C "邮箱" 创建SSH Key

2、git remote add origin git@github.com:dxl-200087/DXL.git(远程仓库路径)

把一个已有的本地仓库和GitHub仓库与之关联

3、git push origin master 把本地master分支的最新修改推送至GitHub

当你第一次使用Git的clone或者push命令连接GitHub时，会得到一个警告：

The authenticity of host 'github.com (xx.xx.xx.xx)' can't be established.

RSA key fingerprint is xx.xx.xx.xx.xx.

Are you sure you want to continue connecting (yes/no)?

这是因为Git使用SSH连接，而SSH连接在第一次验证GitHub服务器的Key时，需要你确认GitHub的Key的指纹信息是否真的来自GitHub的服务器，输入yes回车即可。

注：git push origin master 命令最先推送一次即可，以后可用命令git push进行推送

克隆

1. git clone [git@github.com:dxl-200087/DXL.git](mailto:git@github.com:dxl-200087/DXL.git) 克隆仓库
2. git pull 更新克隆的数据文件

# 三、架构演变

## 1、MongoDB

MongoDB简介

MongoDB是一个非关系型数据库

MongoDB 是一个基于分布式文件存储的数据库。由C++语言编写。旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。

MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。

MongoDB 是一个面向文档存储的数据库，操作起来比较简单和容易。

你可以在MongoDB记录中设置任何属性的索引 (如FirstName="Sameer",Address="8 Gandhi Road")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(需要更多的存储空间和更强的处理能力)，它可以分布在计算机网络中的其他节点上这就是所谓的分片。

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。

操作数据的命令

1、数据库相关操作

show dbs（查看所有数据库（有数据的数据库））

use（创建或者切换数据库）

db（查看当前数据库）

2、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) 查看所有集合

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

3、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()

参考地址<https://www.runoob.com/mongodb/mongodb-create-database.html>

Spring Boot整合<https://blog.csdn.net/zhangcongyi420/article/details/91406843>

## 2、SpringCloud

### Eureka注册中心

#### 什么是Eureka

Eureka是基于REST（代表性状态转移）的服务，主要在AWS云中用于定位服务，以实现负载均衡和中间层服务器的故障转移。我们称此服务为Eureka服务器。Eureka还带有一个基于Java的客户端组件Eureka Client，它使与服务的交互变得更加容易。客户端还具有一个内置的负载平衡器，可以执行基本的循环负载平衡。在Netflix，更复杂的负载均衡器将Eureka包装起来，以基于流量，资源使用，错误条件等多种因素提供加权负载均衡，以提供出色的弹性。

Spring Cloud 封装了 Netflix 公司开发的 Eureka 模块来实现**服务注册和发现**。Eureka 采用了 C-S 的设计架构。Eureka Server 作为服务注册功能的服务器，它是**服务注册中心**。而系统中的其他微服务，使用 Eureka 的客户端连接到 Eureka Server，并维持心跳连接。这样系统的维护人员就可以通过 Eureka Server 来监控系统中各个微服务是否正常运行。Spring Cloud 的一些其他模块（比如Zuul）就可以通过 Eureka Server 来发现系统中的其他微服务，并执行相关的逻辑。

另可参考<https://www.jianshu.com/p/1016cae4fc29>

另可参考<https://blog.csdn.net/MZ_22172217/article/details/82864886>

#### 环境搭建(服务端)

**Eureka Server提供服务注册功能**，各个节点启动后，会在EurekaServer中进行注册，这样EurekaServer中的**服务注册表**中将会存储所有可用服务节点的信息，服务节点的信息可以在界面中直观的看到

声明springcloud

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

引入eureka

<!-- eureka服务端的启动器-->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

application配置文件

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

server.port=8000

eureka.client.serviceUrl.defaultZone=http://localhost:${server.port}/eureka/

App启动器加对应的注解

@EnableEurekaServer //针对注册中心为eureka

或

@EnableDiscoveryClient //相同的注解 注册中心 可以为zookeeper eureka

#### 环境搭建（客户端）

EurekaClient是一个Java客户端，用于简化Eureka Server的交互，客户端同时也具备一个内置的、使用轮询(round-robin)负载算法的负载均衡器。在应用启动后，将会向Eureka Server发送心跳(默认周期为30秒)。如果Eureka Server在多个心跳周期内没有接收到某个节点的心跳，EurekaServer将会从服务注册表中把这个服务节点移除（默认90秒

声明springcloud

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

Pom中引入eureka

<!—eureka客户端的启动器-->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

application配置文件

server.port=8080

spring.application.name=服务名

eureka.client.serviceUrl.defaultZone=http://localhost:${server.port}/eureka/

app启动器加对应的注解

@EnableEurekaClient

服务端客户端搭建参考: <https://www.jianshu.com/p/0395cc887d82>

服务端客户端搭建参考: <https://www.cnblogs.com/yxth/p/10845640.html>

#### eureka的自我保护机制 参考

默认情况下，如果Eureka Server在一定时间内没有接收到某个微服务实例的心跳，Eureka Server将会注销该实例（默认90秒）。但是当网络分区故障发生时，微服务与Eureka Server之间无法正常通信，以上行为可能变得非常危险了——因为微服务本身其实是健康的，此时本不应该注销这个微服务。Eureka通过“自我保护模式”来解决这个问题——当Eureka Server节点在**短时间内丢失过多客户端时**（可能发生了网络分区故障），那么这个节点就会进入自我保护模式。一旦进入该模式，Eureka Server就会保护服务注册表中的信息，不再删除服务注册表中的数据（也就是不会注销任何微服务）。当网络故障恢复后，该Eureka Server节点会自动退出自我保护模式。

在自我保护模式中，Eureka Server会保护服务注册表中的信息，不再注销任何服务实例。当它收到的心跳数重新恢复到阈值以上时，该Eureka Server节点就会自动退出自我保护模式。它的设计哲学就是宁可保留错误的服务注册信息，也不盲目注销任何可能健康的服务实例。一句话讲解：**好死不如赖活着**。

综上，自我保护模式是一种应对网络异常的安全保护措施。它的架构哲学是宁可同时保留所有微服务（健康的微服务和不健康的微服务都会保留），也不盲目注销任何健康的微服务。使用自我保护模式，可以让Eureka集群更加的健壮、稳定。

在Spring Cloud中，可以使用eureka.server.enable-self-preservation = false 禁用自我保护模式。

<https://www.cnblogs.com/cnsec/p/13286632.html>

<https://www.cnblogs.com/xishuai/archive/2018/04/20/spring-cloud-eureka-safe.html>

### Ribbon负载均衡

#### 什么是Ribbon

Spring Cloud Ribbon是基于Netflix Ribbon实现的一套**客户端负载均衡**的工具。

简单的说，Ribbon是Netflix发布的开源项目，主要功能是提供客户端的软件负载均衡算法，将Netflix的中间层服务连接在一起。Ribbon客户端组件提供一系列完善的配置项如连接超时，重试等。简单的说，就是在配置文件中列出Load Balancer（简称LB）后面所有的机器，Ribbon会自动的帮助你基于某种规则（如简单轮询，随机连接等）去连接这些机器。我们也很容易使用Ribbon实现自定义的负载均衡算法。

参考地址<https://blog.csdn.net/kinghmj01/article/details/89446772>

#### 什么是负载均衡

LB，即负载均衡(Load Balance)，在微服务或分布式集群中经常用的一种应用。

负载均衡简单的说就是将用户的请求平摊的分配到多个服务上，从而达到系统的HA。

常见的负载均衡有软件Nginx，LVS，硬件 F5等。

相应的在中间件，例如：dubbo和SpringCloud中均给我们提供了负载均衡，**SpringCloud的负载均衡算法可以自定义**。

**集中式LB**

即在服务的消费方和提供方之间使用独立的LB设施(可以是硬件，如F5, 也可以是软件，如nginx), 由该设施负责把访问请求通过某种策略转发至服务的提供方；

**进程内LB**

将LB逻辑集成到消费方，消费方从服务注册中心获知有哪些地址可用，然后自己再从这些地址中选择出一个合适的服务器。

Ribbon就属于进程内LB，它只是一个类库，集成于消费方进程，消费方通过它来获取到服务提供方的地址。

#### Ribbon环境搭建、接口之间的调用(httpclient)

Spring cloud有两种服务调用方式，一种是ribbon+restTemplate（spring提供的对httpclien进行封装），另一种是feign(详解看Feign)。Feign默认集成了ribbon。

引入ribbon的启动器 需要eureka的依赖：

<!-- eureka Client 启动器-->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-ribbon</artifactId>

</dependency>

配置文件 注意：提供的服务名称不能有下划线（\_）

spring.application.name=服务名称  
eureka.client.serviceUrl.defaultZone=http://localhost:注册中心端口/eureka/

启动类上加注解

@EnableDiscoveryClient （EurekaClient声明是注册中心的客户端）

启动类中注入bean

@Bean

@LoadBalanced //声明 ribbon （并对resttemmplate进行负载均衡，调用的时候直接写服务名就行 不用写ip和端口）

RestTemplate restTemplate() {

return new RestTemplate(); // RestTemplate对httpclient发送请求 进行封装了

}

在调用的地方注入resttemplate，调用其里面的方法。例：get请求 getforobject

Ribbon+RestTemplate使用参考<http://c.biancheng.net/view/5348.html>

### Feign

#### Feign是什么

**官网解释**：[http://projects.spring.io/spring-cloud/spring-cloud.html#spring-cloud-feign](http://projects.spring.io/spring-cloud/spring-cloud.html" \l "spring-cloud-feign)

Feign是一个声明式WebService客户端。**使用Feign能让编写Web Service客户端更加简单**, 它的使用方法是**定义一个接口**，然后**在上面添加注解**，同时也支持JAX-RS标准的注解。Feign也支持可拔插式的编码器和解码器。Spring Cloud对Feign进行了封装，使其支持了Spring MVC标准注解和HttpMessageConverters。**Feign可以与Eureka和Ribbon组合使用以支持负载均衡**。

Feign是一个声明式的Web服务客户端，使得**编写Web服务客户端**变得非常容易，

**只需要创建一个接口，然后在上面添加注解即可**。

#### Feign能干什么

Feign 是一种声明式、模板化的 HTTP 客户端(仅在 consumer 中使用)，Feign旨在使编写Java Http客户端变得更容易。**声明式调用就像调用本地方法一样调用远程方法**。Spring Cloud 的声明式调用, 可以做到使用 HTTP 请求远程服务时能就像调用本地方法一样的体验，开发者完全感知不到这是远程方法，更感知不到这是个 HTTP 请求。**它解决了让开发者调用远程接口就跟调用本地方法一样**，无需关注与远程的交互细节，更无需关注分布式环境开发。

前面在使用Ribbon+RestTemplate时，利用RestTemplate对http请求的封装处理，形成了一套模版化的调用方法。但是在实际开发中，由于对服务依赖的调用可能不止一处，往往一个接口会被多处调用，所以通常都会针对**每个微服务自行封装一些客户端类来包装这些依赖服务的调用**。所以，Feign在此基础上做了进一步封装，由他来帮助我们定义和实现依赖服务接口的定义。在Feign的实现下，我们**只需创建一个接口并使用注解的方式**来配置它(以前是Dao接口上面标注Mapper注解,现在是一个微服务接口上面标注一个Feign注解即可)，即可完成对服务提供方的接口绑定，简化了使用Spring cloud Ribbon时，自动封装服务调用客户端的开发量。

#### Feign集成了Ribbon

Feign也是运行在消费者端的，使用 Ribbon 进行负载均衡，所以 Feign 直接内置了 Ribbon。而与Ribbon不同的是，通过feign只需要定义服务绑定接口且以声明式的方法，优雅而简单的实现了服务调用。

#### 环境搭建

导入 pom的依赖

<!-- 引入feign 封装了ribbon+ restTemplate hystrix-->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-openfeign</artifactId>

</dependency>

启动类添加注解

@EnableFeignClients//声明fegin

定义接口（和远程服务作对应）

//声明这是feign ，value是服务提供者的服务名；fallback=接口的实现类的类对象，实现方法是熔断处理(熔断详解看Hystrix)

@FeignClient(value = "productapi",fallback =ProductExcetion.class )

public interface ProductApi {

@RequestMapping("product/queryProduct")

public String productApi();

}

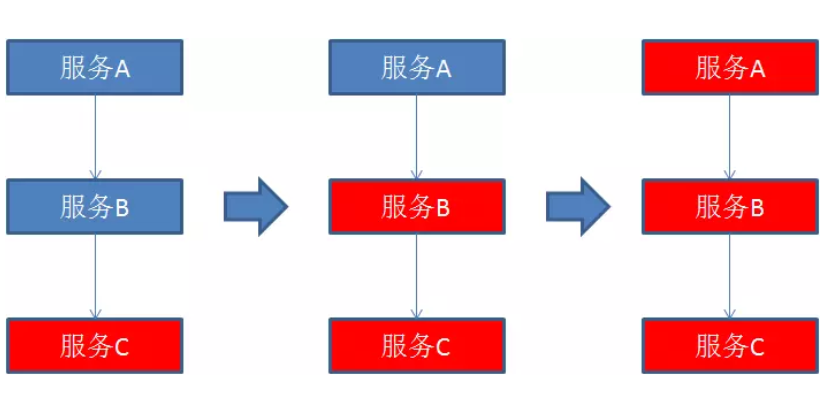
Feign具体参考: <https://blog.csdn.net/qq_45730819/article/details/107815919>

Feign具体参考: <https://blog.csdn.net/wo18237095579/article/details/83343915>

### Hystrix熔断器

#### 为什么使用Hystrix

在微服务架构中，根据业务来拆分成一个个的服务，服务与服务之间可以相互调用（RPC），在Spring Cloud**可以用Feign+Ribbon来进行服务之间的调用**。为了保证其高可用，单个服务通常会集群部署。由于网络原因或者自身的原因，服务并不能保证100%可用。如客户端访问A服务，而A服务需要调用B服务，B服务需要调用C服务，由于网络原因或者自身的原因，如果B服务或者C服务不能及时响应，A服务将处于阻塞状态，直到B服务C服务响应。此时若有大量的请求涌入，容器的线程资源会被消耗完毕，导致服务瘫痪。**服务与服务之间的依赖性，故障会传播，造成连锁反应，会对整个微服务系统造成灾难性的严重后果，这就是服务故障的“雪崩”效应。**



#### 服务雪崩解决方案

**熔断模式**：这种模式主要是参考电路熔断，如果一条线路电压过高，保险丝会熔断，防止火灾。放到我们的系统中，如果某个目标服务调用慢或者有大量超时，此时，熔断该服务的调用，对于后续调用请求，不在继续调用目标服务，直接返回，快速释放资源。如果目标服务情况好转则恢复调用。

**隔离模式**：这种模式就像对系统请求按类型划分成一个个小岛的一样，当某个小岛被火少光了，不会影响到其他的小岛。例如可以对不同类型的请求使用线程池来资源隔离，每种类型的请求互不影响，如果一种类型的请求线程资源耗尽，则对后续的该类型请求直接返回，不再调用后续资源。这种模式使用场景非常多，例如将一个服务拆开，对于重要的服务使用单独服务器来部署，再或者公司最近推广的多中心。

**限流模式**：上述的熔断模式和隔离模式都属于出错后的容错处理机制，而限流模式则可以称为预防模式。限流模式主要是提前对各个类型的请求设置最高的QPS阈值，若高于设置的阈值则对该请求直接返回，不再调用后续资源。这种模式不能解决服务依赖的问题，只能解决系统整体资源分配问题，因为没有被限流的请求依然有可能造成雪崩效应。漏斗算法，令牌桶算法。

#### 服务熔断

当下游的服务因为某种原因突然变得不可用或响应过慢，上游服务为了保证自己整体服务的可用性，不再继续调用目标服务，直接返回，快速释放资源。如果目标服务情况好转则恢复调用。

#### 服务降级

1、当下游的服务因为某种原因响应过慢，下游服务主动停掉一些不太重要的业务，释放出服务器资源，增加响应速度！(**弃车保帅，壮士断腕，刮骨疗伤**)

2、当下游的服务因为某种原因不可用，上游主动调用本地的一些降级逻辑，避免卡顿，迅速返回给用户！

#### 服务熔断和服务降级的区别和联系

服务降级有很多种降级方式！如开关降级、限流降级、熔断降级!**服务熔断属于降级方式的一种**！**因为从实现上来说，熔断和降级必定是一起出现**。因为当发生下游服务不可用的情况，这个时候为了对最终用户负责，就需要进入上游的降级逻辑了。因此，**将熔断降级视为降级方式的一种**，也是可以说的通的！

#### Hystrix(黑丝锤科丝)是什么

Hystrix中文是豪猪的意思，豪猪浑身长满了长刺，可以保护自己。**Hystrix是一个用于处理分布式系统的延迟和容错的开源库，在Spring Cloud中防雪崩的利器就是Spring Cloud Hystrix**，Spring Cloud Hystrix是基于Netflix Hystrix实现的。在分布式系统里，许多依赖不可避免的会调用失败，比如超时、异常等，Hystrix能够保证在一个依赖出问题的情况下，不会导致整体服务失败，避免级联故障，以提高分布式系统的弹性。

#### Hystrix能干什么

**1、服务熔断**

当Hystrix Command请求后端服务失败数量超过一定比例(默认50%), 断路器会切换到开路状态(Open). 这时所有请求会直接失败而不会发送到后端服务. 断路器保持在开路状态一段时间后(默认5秒), 自动切换到半开路状态(HALF-OPEN).

**2、服务降级**

Fallback相当于是降级操作, 我们可以实现一个fallback方法, 当请求后端服务出现异常或者服务熔断后的时候, 直接调用本地fallback方法，告知后面的请求服务不可用了，不要再来了**。**

**3、服务容错**

**4、线程和信号隔离**

在Hystrix中, 主要通过线程池来实现资源隔离. 通常在使用的时候我们会根据调用的远程服务划分出多个线程池.比如说，一个服务调用两外两个服务，你如果调用两个服务都用一个线程池，那么如果一个服务卡在哪里，资源没被释放。后面的请求又来了，导致后面的请求都卡在哪里等待，导致你依赖的A服务把你卡在哪里，耗尽了资源，也导致了你另外一个B服务也不可用了。这时如果依赖隔离，某一个服务调用A B两个服务，如果这时我有100个线程可用，我给A服务分配50个，给B服务分配50个，这样就算A服务挂了，我的B服务依然可以用。

**5、请求缓存**

比如一个请求过来请求我userId=1的数据，你后面的请求也过来请求同样的数据，这时我不会继续走原来的那条请求链路了，而是把第一次请求缓存过了，把第一次的请求结果返回给后面的请求。

**6、请求合并**

我依赖于某一个服务，我要调用N次，我发了N条请求发然后拿到一堆结果，这时候我们可以把多个请求合并成一个请求，这样我们只需调用一次远程服务，提升了效率。

**7、近乎实时的服务监控**

#### 环境搭建

导入pom依赖

<!-- hystrix 熔断器-->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-hystrix</artifactId>

</dependency>

启动类舔加两个注解

@EnableHystrix// 熔断器 需要两个注解

@EnableCircuitBreaker

对你要熔断的方法

// 此注解标识是要熔断 fallbackMethod 熔断后进行执行的方法

@HystrixCommand(fallbackMethod = "errorMethod",commandProperties = {

@HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds",value = "3000")

}) //对此方法进行熔断

Feign+Hystrix需要开启feign.hystrix.enabled=true

使用可参考<https://www.jianshu.com/p/6f6ac6433d5e>

#### Hystrix监控平台

1、搭建服务端

Pom添加依赖

<!-- hystrix dashboard 监控平台的启动器-->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>

</dependency>

启动类加注解 @EnableHystrixDashboard

2、搭建客户端

Pom添加依赖

<!-- 将此服务加入hystrix bashboard 监控平台 -->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-hystrix</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

启动类中提供访问的监控路径

/\* 提供监控平台的路径\*/

@Bean

public ServletRegistrationBean getServlet() {

HystrixMetricsStreamServlet streamServlet = new HystrixMetricsStreamServlet();

ServletRegistrationBean registrationBean = new ServletRegistrationBean(streamServlet);

registrationBean.setLoadOnStartup(1);

registrationBean.addUrlMappings("/hystrix.stream");

registrationBean.setName("HystrixMetricsStreamServlet");

return registrationBean;

}

3、访问根目录 /hystrix 例：<http://localhost:8086/hystrix>

熔断和监控使用参考<https://www.cnblogs.com/progor/p/12894684.html>

### Zuul网关

#### 什么是服务网关

服务网关是介于我们客户端和各个服务中间的一个流量入口，因为高度集中所以他还充当服务调用的中央策略执行点，因为他的入口性，和集中性，那么我们**横切服务的关注点就很容易在一个地方实现**，而不需要在各个开发团队来实现这些关注点。

服务网关=路由转发+过滤器

**路由转发**：接收一切外界(客户端)请求，转发到后端的微服务上去；

**过滤器**：在服务网关中可以完成一系列的横切功能，例如权限校验、限流以及监控等，这些都可以通过过滤器完成（其实路由转发也是通过过滤器实现的）。

#### Zuul(如欧)是什么

ZUUL是Netflix开源的微服务网关，Spring Cloud Zuul对Zuul进行了整合和增强，它可以和Eureka、Ribbon、Hystrix等组件配合使用，Zuul组件包含了对**请求的路由**和**过滤**两个最主要的功能：其中路由功能负责将外部请求转发到具体的微服务实例上，是实现外部访问统一入口的基础。而过滤器功能则负责对请求的处理过程进行干预，是实现请求校验、服务聚合等功能的基础。通过将Zuul和Eureka进行整合，将Zuul自身注册为Eureka服务治理下的应用，同时从Eureka中获得其他微服务的消息，也即**以后通过客户端(浏览器，APP)访问微服务都是通过Zuul跳转后获得**。

注意：Zuul服务最终还是会注册进Eureka Server

#### 路由转发环境搭建

引入pom依赖

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-zuul</artifactId>

</dependency>

启动类 加上注解 @enableZuulProxy

配置配置文件

zuul.routes.名称.path=/名称/\*\*

zuul.routes.名称.service-id=服务名称

#### 过滤器环境搭建

写个类，继承zuulFilter抽象类，重写里面的四个方法

#### Zuul Filter简介

ZuulFilter是过滤器的顶级父类。在这里我们看一下其中定义的4个最重要的方法

public class ZuulCeptor extends ZuulFilter {  
 @Override  
 public String filterType() {return "pre";}  
 @Override  
 public int filterOrder() {return 0;}  
 @Override  
 public boolean shouldFilter() { return true;}  
 @Override  
 public Object run() throws ZuulException {return null;}  
}

**shouldFilter**:返回一个 Boolean 值，判断该过滤器是否需要执行。返回true执行，返回false不执行。

**run**:过滤器的具体业务逻辑。

**filterType**:返回字符串，代表过滤器的类型。包含以下4种：

**pre:请求在被路由之前执行**

routing:在路由请求时调用微服务的时候执行

post:在routing和errror过滤器之后调用

error:处理请求时发生错误调用

**filterOrder**:通过返回的int值来定义过滤器的执行顺序，数字越小优先级越高。

## 3、Redis

什么是Redis(数据库 关系型数据库mysql、orale、db2、sqlserver，非关系型数据库（缓存）redis、mogodb )

Redis 是完全开源的，遵守 BSD 协议，是一个高性能的非关系型数据库。

可基于内存亦可持久化的日志型、Key-Value数据库

Redis 使用场景：

数据高并发的读写

海量数据的读写

对扩展性要求高的数据

Redis 与其他 key - value 缓存产品有以下三个特点：

Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。

Redis不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。

Redis支持数据的备份，即master-slave模式的数据备份。

Redis优势（rdb(快照) aof(基于日志)）

性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s 。

丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。(String、list、set、zset、hash)

原子 – Redis的所有操作都是原子性的，要么成功执行要么失败完全不执行。单个操作是原子性的、多个操作也支持事务。

丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性。

## 4、Linux命令

ls 查看目录

pwd 查看当前路径

cd / 切换目录 /根目录 ./当前目录 ../上一级

mkdir 创建目录

rm -rf（参数） 删除（慎用）

ps -ef |grep软件名 看是否启动，查看服务

kill -9 进程号 干掉服务

systemctl status firewalld (status 状态、stop关闭、start 开启) 防火墙

cat 文件名 查看文件

vi 路径 按i进入编辑模式、摁 esc退出编辑模式、：wq！保存强制退出

tar -zxvf 解压

上传文件 （linux安装一个上传插件）

yum -y install lrzsz

rz 上传命令

sz 下载命令

## 5、Docker

Docker是一款可以解决运行环境和配置问题的软件容器引擎

1 什么是docker？

Docker是一款可以解决运行环境和配置问题的软件容器引擎，

方便做持续集成并有助于整体发布的容器虚拟化技术。

2 docker的安装？

环境准备

Docker 运行在 CentOS 7 上，

要求系统为64位、系统内核版本为 3.10 以上。

Docker 运行在 CentOS-6.5 或更高的版本的 CentOS 上，

要求系统为64位、系统内核版本为 2.6.32-431 或者更高版本。

确认环境

怎么查看centos版本

cat /etc/redhat-release

怎么查看内核

uname -r

安装docker

centos7安装docker

a 安装gcc（c语言的环境 ）

linux 安装有很多命令

yum （有外网） 命令（在线安装 一步到位 ）

yum -y install gcc

yum -y install gcc-c++

b 卸载旧版本 docker

yum -y remove docker docker-common docker-selinux docker-engine

c 安装docker

设置yum下载docker的安装包

安装需要的软件包

yum install -y yum-utils device-mapper-persistent-data lvm2

设置Docker的yum源

yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo

更新yum软件包索引

yum makecache fast

安装Docker CE

yum -y install docker-ce

启动Docker

systemctl start docker

测试

docker version

d 修改docker的仓库地址

（默认地址是国外的 网速比较慢 建议修改 阿里云 网易云）

科大镜像：https://docker.mirrors.ustc.edu.cn/

网易：https://hub-mirror.c.163.com/

阿里云：https://<你的ID>.mirror.aliyuncs.com

七牛云加速器：https://reg-mirror.qiniu.com

具体参考 阿里云的命令 （控制台搜索 容器镜像 得到执行命令 如下）

创建此文件 /etc/docker/daemon.json 内容如下

{

"registry-mirrors": ["https://reg-mirror.qiniu.com"]

}

systemctl daemon-reload

systemctl restart docker

验证 通过 docker info 看到配置信息就对了

docker的命令

docker的组成部分（容器，镜像（image），仓库）

docker version 查看docker版本

docker info 查看docker的详细信息（cpu 内存 容器 镜像）

镜像的操作命令：

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:tag 下载一个centos 的一个镜像 docker pull centos 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

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）

进入后台运行容器的终端 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 如果想把运行中的容器也删掉 加参数 -f）

停止容器

docker stop 容器id（名称）

启动容器

docker start 容器id（名称）

拷贝 宿主机向容器中拷贝文件

docker cp 宿主机的文件 容器id:目录

拷贝 容器中的文件 拷贝到宿主机上

docker cp 容器id:目录 宿主机的目录

（ 先创建容器了 ）

数据卷怎么创建 (创建一个容器 并和宿主机共享目录)

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

数据卷容器 （多个容器共享一个目录 （修改一个目录 多个容器跟着改变））

先创建一个共享的容器

docker run -v 宿主名录:容器目录 --name 容器名称 镜像 /bin/bash

创建容器指定共享的容器 实现共享目录

docker run -itd --volumes-from 共享容器名 --name容器名称 镜像 /bin/bash

常用的一些镜像（安装软件）

安装mysql

之前在虚拟机中安装mysql （安装一个centos 系统 下载mysql的安装包 安装mysql（需要n步））

docker中安装mysql

搜索mysql的镜像 创建一个容器 运行mysql的镜像

下载mysql

docker pull mysql:5.7

创建容器 启动mysql

docker run -id --name 容器名 -p 端口映射 port:port -e MYSQL\_ROOT\_PASSWORD=密码 镜像

安装tomcat

下载tomcat 镜像

创建容器 启动tomcat

docker run -id --name 容器名 -p 端口映射 port:port 镜像

制作镜像 两种方式：

commit （）

build（Dockerfile）

编写Dockerfile文件

语法 ：

FROM 指定基础镜像

MAINTAINER 指定镜像维护者信息

RUN 用于执行指定脚本命令

CMD 指定启动容器时执行的命令

EXPOSE 指定容器暴露的端口

ENV 指定环境变量

ADD 将文件从宿主机复制到容器指定位置，同时对压缩文件有自动解压功能

COPY 将文件从宿主机复制到容器指定位置

ENTRPOINT 设置容器启动时需要运行的命令

WORKDIR 为后续的如RUN、CMD、ENTRYPOINT、COPY、ADD指定工作目录

加深对语法的理解和制作镜像 （制作tomcat的镜像 依赖那些环境 jdk） 参考地址 https://blog.csdn.net/qq\_36538012/article/details/80839361

制作镜像命令 进入Dockfile目录

docker build -f Dockerfile的路径 -t 镜像名称:版本号 .

制作一个springboot项目的镜像 （作业： springboot项目 打成 jar包 java -jar 项目的jar.jar）

创建私有仓库

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":["192.168.200.134:5000"]}

# 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 centos:7 私有仓库服务器IP:5000/centos:7

2、上传标记的镜像 docker push 私有仓库服务器IP:5000/centos:7

从私有仓库拉镜像

docker pull 私有仓库服务器ip:5000/centos:7

查看私服镜像所有仓库

curl http://localhost:5000/v2/\_catalog

天天会写代码（工作中更多的事）

想一个问题 代码到底怎么部署 （部署环境中 也需要jdk tomcat idea ）

真正工作的流程

需求 开发（java程序员） 测试阶段（部署测试环境 jdk tomcat 项目的代码（jar包或者war包）） 上线

springboot项目maven打jar包 pom的配置

<!-- 打包的配置 -->

<build>

<!-- 生成包的名称 不配置 默认是项目名加版本号

<finalName>demo</finalName>-->

<!-- 打包的插件 -->

<plugins>

<plugin>

<!-- springboot 打包的插件-->

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<!-- 重要 指定jar包的启动类-->

<mainClass>com.jgix.App</mainClass>

</configuration>

</plugin>

</plugins>

</build>

jar包的项目怎么启动 java -jar jar包

制作一个镜像 Dockerfile文件 制作的

为了统一方便管理 宿主机创建了一个目录 /xijunge/iso/镜像名/dockerfile

linux 命令中 有一个上传的命令

rz 上传文件 如果没有安装

yum安装

yum install -y lrzsz

阿里云开通镜像服务的时候 让你设置密码 不是咱的登陆密码

rabbitmq-plugins enable rabbitmq\_management

数据雪崩击穿穿透: <https://blog.csdn.net/Longtermevolution/article/details/107916056>

Redis雪崩击穿穿透: <https://blog.csdn.net/qq_27409289/article/details/85885121>

面试问题：<https://blog.csdn.net/weixin_43495390/article/details/86533482>

文档：实训面试冲刺准备.note

链接：<http://note.youdao.com/noteshare?id=e8ed0257d9be5b7aaba8a2f67ff3cd97>

文档：1910A----面试宝典.note

链接：[http://note.youdao.com/noteshare?id=72496e3995cbb15d27fda4f992195036&sub=53 0E74BF070149C0AF83F93955D74DC6](http://note.youdao.com/noteshare?id=72496e3995cbb15d27fda4f992195036&sub=530E74BF070149C0AF83F93955D74DC6)

文档：实训四面试必备技术点.note

链接：<http://note.youdao.com/noteshare?id=a34ae470ebeb59644c6a7cb46f091bf0>

文档：nacos的搭建集群和单机.note

链接：<http://note.youdao.com/noteshare?id=c3767478c0a44ee353dd6097520d8ebd&sub=41B82B9A5FD8429EB3AD678A7B675650>

## 6、单体架构(all in one)

单体架构是最简单的架构风格，所有的代码全都在一个项目中。这样研发团队的任何一个人都可以随时修改任意的一段代码，或者增加一些新的代码。开发人员也可以只在自己的电脑上就可以随时开发、调试、测试整个系统的功能。也不需要额外的一些依赖条件和准备步骤，我们就可以直接编译打包整个系统代码，创建一个可以发布的二进制版本。这种方式对于一个新团队的创立初期，需要迅速开始从 0 到 1，抓住时机实现产品最短时间推向市场，可以省去各种额外的设计，直接上手干活，争取了时间，因而是非常有意义的。单体架构的应用比较容易部署、测试，在项目的初期，单体应用可以很好地运行。然而，随着需求的不断增加，越来越多的人加入开发团队，代码库也在飞速地膨胀。慢慢地，单体应用变得越来越臃肿，可维护性、灵活性逐渐降低，维护成本越来越高。这种方式对于一个系统的长期稳定发展确实有很多缺点的：

|  |
| --- |
| * **项目复杂性高，代码严重耦合，代码维护困难**   单体架构的系统存在代码严重耦合的问题，以一个百万行级别的单体应用为例，整个项目包含的模块非常多、模块的边界模糊、依赖关系不清晰、代码质量参差不齐、混乱地堆砌在一起。可想而知整个项目非常复杂。每次修改代码都心惊胆战，甚至添加一个简单的功能，或者修改一个Bug都会带来隐含的缺陷，修改一处代码，可能会影响一大片的功能无法正常使用。   * **欠下一屁股技术债务**   随着时间推移、需求变更和人员更迭，会逐渐形成应用程序的技术债务，并且越积越多。“ 不坏不修”，这在软件开发中非常常见，在单体应用中这种思想更甚。已使用的系统设计或代码难以被修改，因为应用程序中的其他模块可能会以意料之外的方式使用它。   * **部署频率低**   随着代码的增多，构建和部署的时间也会增加。而在单体应用中，**每次功能的变更或缺陷的修复都会导致需要重新部署整个应用**。全量部署的方式耗时长、影响范围大、风险高，这使得单体应用项目上线部署的频率较低。而部署频率低又导致两次发布之间会有大量的功能变更和缺陷修复，出错率比较高。   * **局部功能的修改需要整个系统重新发布， 牵一发而动全身**   单体架构的系统变更对部署影响大，并且这个问题是所有的单体架构系统都存在的问题。系统作为一个单体部署，每次发布的部署单元就是一个新版本的整个系统，系统内的任何业务逻辑调整都会导致整个系统的重新打包，部署、停机、再重启，进而导致了系统的停机发布时间较长。每次发布上线都是生产系统的重大变更，这种部署模式大大提升了系统风险，降低了系统的可用性。   * **可靠性差**   某个应用Bug，例如死循环、内存溢出等，可能会导致整个应用的崩溃   * **扩展能力受限**   单体应用只能作为一个整体进行扩展，无法根据业务模块的需要进行伸缩。例如，应用中有的模块是计算密集型的，它需要强劲的CPU；有的模块则是IO密集型的，需要更大的内存。如果任何一个业务存在性能问题，那么都需要**考虑多部署几个完整的实例的集群**，或者再加上负载均衡设备，才能保证整个系统的性能可以支撑用户的使用。   * **影响开发效率**   单体架构的系统影响开发效率。如果一个使用 Java 的简单单体项目代码超过 100 万行，那么在一台笔记本电脑上修改了代码后执行自动编译，可能需要等待十分钟以上，并且内存可能不够编译过程使用，这是非常难以忍受的。或者说因为别的模块出现问题，导致整个系统无法运行，这也是非常难以忍受的。   * **系统启动慢**   单体架构模式打包后的部署结构可能过于庞大，导致业务系统启动很慢，进而也会影响系统的可用性。这一条也是所有单体架构的系统都有的问题。   * **阻碍技术创新，技术栈受限**   单体应用往往使用统一的技术平台或方案解决所有的问题，团队中的每个成员都必须使用相同的开发语言和框架，要想引入新框架或新技术平台会非常困难。 |

综上所述，单体架构模式比较适用于规模较小的系统和规模较小的团队，特别是需要快速推出原型实现，以质量换速度的场景。

## 7、垂直架构(分布式架构)

**分布式：多个人完成一件事情**

**集群：多个人做同样的事情**

**分布式和集群的关系**

**比如说你开了一家饭店，饭店里有一个厨师，这个厨师又得配菜又得炒菜，这时候万一这个厨师生病了来不了了干不了活，这个时候你的饭店是不是就不可用了，对外提供不了吃饭服务了。这个时候呢你又招了一个厨师，这两个厨师都做的是一模一样的菜，那这两个厨师之间的关系就是集群。为了让厨师能专注的去炒菜，这个时候呢你又招了一个配菜员，配菜员和厨师之间的关系就是分布式。**

针对单体架构的不足，为了适应大型项目的开发需求，许多公司将一个单体系统按业务垂直拆分为若干个子系统，系统之间通过网络交互来完成用户的业务处理，每个系统可分布式部署，这种架构称为分布式架构。也就说将一个大系统拆分为多个子系统。

**特点**：

1、按业务垂直拆分成一个一个的单体系统，此架构也称为垂直架构。

2、系统与系统之间的存在数据冗余，耦合性较大，如上图中三个项目都存在客户信息。

3、系统之间的接口多为实现数据同步，如上图中三个项目要同步客户信息。

**优点**：

1、通过垂直拆分，每个子系统变成小型系统，功能简单，前期开发成本低，周期短。

2、每个子系统可按需伸缩。

3、每个子系统可采用不同的技术。

**缺点**：

1、子系统之间存在数据冗余、功能冗余，耦合性高。

2、按需伸缩粒度不够，对同一个子系统中的不同的业务无法实现，比如订单管理和用户管理。

## 8、SOA架构

SOA是一种面向服务的架构，基于分布式架构，它将不同业务功能按服务进行拆分，并通过这些服务之间定义良好的接口和协议联系起来。

**特点**：

1、基于SOA的架构思想，将重复公用的功能抽取为组件，以服务的方式向各各系统提供服务。

2、各个系统与服务之间采用webservice、rpc等方式进行通信。

3、ESB企业服务总线作为系统与服务之间通信的桥梁。

**优点**：

1、将重复的功能抽取为服务，提高开发效率，提高系统的可重用性、可维护性。

2、可以针对不同服务的特点按需伸缩。

3、采用ESB减少系统中的接口耦合。

**缺点**：

1. 系统与服务的界限模糊,会导致抽取的服务的**粒度过大**,系统与服务之间耦合性高。

2、虽然使用了ESB，但是服务的接口协议不固定，种类繁多，不利于系统维护。

## 9、微服务架构

基于SOA架构的思想，为了满足移动互联网对大型项目及多客户端的需求，对服务层进行**细粒度**的拆分，所拆分的每个服务只完成某个特定的业务功能，比如订单服务只实现订单相关的业务，用户服务实现用户管理相关的业务等等，**服务的粒度很小，所以称为微服务架构**。

**特点**：

1、服务层按业务拆分为一个一个的微服务。

2、微服务的职责单一。

3、微服务之间采用RESTful、RPC等轻量级协议传输。

<https://www.jianshu.com/p/2accc2840a1b>

4、有利于采用前后端分离架构。

**优点**：

1、服务拆分粒度更细，有利于资源重复利用，提高开发效率。

2、可以更加精准的制定每个服务的优化方案，按需伸缩。

3、适用于互联网时代，产品迭代周期更短。

**缺点**：

1、开发的复杂性增加，因为一个业务流程需要多个微服务通过网络交互来完成。

2、微服务过多，服务治理成本高，不利于系统维护。

## 10、分布式事务

1、什么是分布式事务

分布式事务就是指事务的参与者、支持事务的服务器、资源服务器以及事务管理器分别位于

不同的分布式系统的不同节点之上。以上是百度百科的解释，简单的说，就是一次大的操作

由不同的小操作组成，这些小的操作分布在不同的服务器上，且属于不同的应用，分布式事

务需要保证这些小操作要么全部成功，要么全部失败。本质上来说，分布式事务就是为了保

证不同数据库的数据一致性。

2、分布式事务的产生的原因

2.1、数据库分库分表

当数据库单表一年产生的数据超过1000W，那么就要考虑分库分表，具体分库分表的原理在

此不做解释，以后有空详细说，简单的说就是原来的一个数据库变成了多个数据库。这时

候，如果一个操作既访问01库，又访问02库，而且要保证数据的一致性，那么就要用到分布

式事务。

2.2、应用SOA化

所谓的SOA化，就是业务的服务化。比如原来单机支撑了整个电商网站，现在对整个网站进

行拆解，分离出了订单中心、用户中心、库存中心。对于订单中心，有专门的数据库存储订

单信息，用户中心也有专门的数据库存储用户信息，库存中心也会有专门的数据库存储库存信息。这时候如果要同时对订单和库存进行操作，那么就会涉及到订单数据库和库存数据

库，为了保证数据一致性，就需要用到分布式事务。

以上两种情况表象不同，但是本质相同，都是因为要操作的数据库变多了！

3、事务的ACID特性

3.1、原子性(A)

所谓的原子性就是说，在整个事务中的所有操作，要么全部完成，要么全部不做，没有中间

状态。对于事务在执行中发生错误，所有的操作都会被回滚，整个事务就像从没被执行过一

样。

3.2、一致性(C)

事务的执行必须保证系统的一致性，就拿转账为例，A有500元，B有300元，如果在一个事务

里A成功转给B50元，那么不管并发多少，不管发生什么，只要事务执行成功了，那么最后A账户一定是450元，B账户一定是350元。

3.3、隔离性(I)

所谓的隔离性就是说，事务与事务之间不会互相影响，一个事务的中间状态不会被其他事务

感知。

3.4、持久性(D)

所谓的持久性，就是说一单事务完成了，那么事务对数据所做的变更就完全保存在了数据库

中，即使发生停电，系统宕机也是如此。

4、分布式事务的应用场景

4.1、支付

最经典的场景就是支付了，一笔支付，是对买家账户进行扣款，同时对卖家账户进行加钱，

这些操作必须在一个事务里执行，要么全部成功，要么全部失败。而对于买家账户属于买家

中心，对应的是买家数据库，而卖家账户属于卖家中心，对应的是卖家数据库，对不同数据

库的操作必然需要引入分布式事务。

4.2、在线下单

买家在电商平台下单，往往会涉及到两个动作，一个是扣库存，第二个是更新订单状态，库

存和订单一般属于不同的数据库，需要使用分布式事务保证数据一致性。

5、常见的分布式事务解决方案

5.1、基于XA协议的两阶段提交

XA是一个分布式事务协议，由Tuxedo提出。XA中大致分为两部分：事务管理器和本地资源管

理器。其中本地资源管理器往往由数据库实现，比如Oracle、DB2这些商业数据库都实现了

XA接口，而事务管理器作为全局的调度者，负责各个本地资源的提交和回滚。XA实现分布式

事务的原理如下：总的来说，XA协议比较简单，而且一旦商业数据库实现了XA协议，使用分布式事务的成本也

比较低。但是，XA也有致命的缺点，那就是性能不理想，特别是在交易下单链路，往往并发

量很高，XA无法满足高并发场景。XA目前在商业数据库支持的比较理想，在mysql数据库中

支持的不太理想，mysql的XA实现，没有记录prepare阶段日志，主备切换回导致主库与备库

数据不一致。许多nosql也没有支持XA，这让XA的应用场景变得非常狭隘。

5.2、消息事务+最终一致性

所谓的消息事务就是基于消息中间件的两阶段提交，本质上是对消息中间件的一种特殊利

用，它是将本地事务和发消息放在了一个分布式事务里，保证要么本地操作成功成功并且对

外发消息成功，要么两者都失败，开源的RocketMQ就支持这一特性，具体原理如下：1、A系统向消息中间件发送一条预备消息

2、消息中间件保存预备消息并返回成功

3、A执行本地事务

4、A发送提交消息给消息中间件

通过以上4步完成了一个消息事务。对于以上的4个步骤，每个步骤都可能产生错误，下面一

一分析：

步骤一出错，则整个事务失败，不会执行A的本地操作

步骤二出错，则整个事务失败，不会执行A的本地操作

步骤三出错，这时候需要回滚预备消息，怎么回滚？答案是A系统实现一个消息中间件的回

调接口，消息中间件会去不断执行回调接口，检查A事务执行是否执行成功，如果失败则回滚预备消息

步骤四出错，这时候A的本地事务是成功的，那么消息中间件要回滚A吗？答案是不需要，其

实通过回调接口，消息中间件能够检查到A执行成功了，这时候其实不需要A发提交消息了，

消息中间件可以自己对消息进行提交，从而完成整个消息事务

基于消息中间件的两阶段提交往往用在高并发场景下，将一个分布式事务拆成一个消息事务

(A系统的本地操作+发消息)+B系统的本地操作，其中B系统的操作由消息驱动，只要消息事

务成功，那么A操作一定成功，消息也一定发出来了，这时候B会收到消息去执行本地操作，

如果本地操作失败，消息会重投，直到B操作成功，这样就变相地实现了A与B的分布式事

务。原理如下：

虽然上面的方案能够完成A和B的操作，但是A和B并不是严格一致的，而是最终一致的，我们

在这里牺牲了一致性，换来了性能的大幅度提升。当然，这种玩法也是有风险的，如果B一

直执行不成功，那么一致性会被破坏，具体要不要玩，还是得看业务能够承担多少风险。

5.3、TCC编程模式

所谓的TCC编程模式，也是两阶段提交的一个变种。TCC提供了一个编程框架，将整个业务逻

辑分为三块：Try、Confirm和Cancel三个操作。以在线下单为例，Try阶段会去扣库存，Confirm阶段则是去更新订单状态，如果更新订单失败，则进入Cancel阶段，会去恢复库

存。总之，TCC就是通过代码人为实现了两阶段提交，不同的业务场景所写的代码都不一

样，复杂度也不一样，因此，这种模式并不能很好地被复用。

6、总结

分布式事务，本质上是对多个数据库的事务进行统一控制，按照控制力度可以分为：不控

制、部分控制和完全控制。不控制就是不引入分布式事务，部分控制就是各种变种的两阶段

提交，包括上面提到的消息事务+最终一致性、TCC模式，而完全控制就是完全实现两阶段提

交。部分控制的好处是并发量和性能很好，缺点是数据一致性减弱了，完全控制则是牺牲了

性能，保障了一致性，具体用哪种方式，最终还是取决于业务场景。作为技术人员，一定不

能忘了技术是为业务服务的，不要为了技术而技术，针对不同业务进行技术选型也是一种很

重要的能力

## 11消息队列

**什么是RabbitMQ？为什么使用RabbitMQ？**

RabbitMQ是一款开源的，Erlang编写的，消息中间件；

可以用它来：解耦、异步、削峰。

**RabbitMQ 使用场景**

解耦（为面向服务的架构（SOA）提供基本的最终一致性实现）

场景说明：

用户下单后，订单系统需要通知库存系统。传统的做法是，订单系统调用库存系统的接口。

缺点：假如库存系统无法访问，则订单减库存将失败，从而导致订单失败

引入消息队列后

用户下单后，订单系统完成持久化处理，将消息写入消息队列，返回用户订单下单成功

订阅下单的消息，采用拉/推的方式，获取下单信息，库存系统根据下单信息，进行库存操作

在下单时库存系统不能正常使用。也不影响正常下单，因为下单后，订单系统写入消息队列就不再关心其他的后续操作了。实现订单系统与库存系统的应用解耦

（如果问下订单成功之后，等到访问库存时候，这时发现库存不足怎么办，可以在下订单时候加判断，如果订单数量大于内存，就不让下订单成功，这样就解决了。）

异步提升效率

场景说明：用户注册后，需要发注册邮件和注册短信。

传统的做法有两种 1.串行的方式；2.并行方式(多线程)

（1）串行方式：将注册信息写入数据库成功后，发送注册邮件，再发送注册短信。以上三个任务全部完成后，返回给客户端。这种做法让客户端等待没有必要等待的东西。影响用户体验度

（2）并行方式(使用多线程)：将注册信息写入数据库成功后，发送注册邮件的同时，发送注册短信。以上三个任务完成后，返回给客户端。与串行的差别是，并行的方式可以提高处理的时间

引入消息队列后

引入消息队列，将发送邮件,短信不是必须的业务逻辑异步处理

引入消息队列后，用户的响应时间就等于写入数据库的时间+写入消息队列的时间(可以忽略不计),引入消息队列后处理后,响应时间是串行的3倍,是并行的2倍。

流量削峰

流量削锋也是消息队列中的常用场景，一般在秒杀或团抢活动中使用广泛

应用场景：

系统其他时间A系统每秒请求量就100个，系统可以稳定运行。系统每天晚间八点有秒杀活动，每秒并发请求量增至1万条，但是系统最大的处理能力只能每秒处理1000个请求，于是系统崩溃，服务器宕机。

之前架构：大量用户（100万用户）通过浏览器在晚上八点高峰期同时参与秒杀活动。大量的请求涌入我们的系统中，高峰期达到每秒钟5000个请求，大量的请求打到MySQL上，每秒钟预计执行3000条SQL。但是一般的MySQL每秒钟扛住2000个请求就不错了，如果达到3000个请求的话可能MySQL直接就瘫痪了，从而系统无法被使用

引入消息队列后 ：

100万用户在高峰期的时候，每秒请求有5000个请求左右，将这5000请求写入MQ里面，系统A每秒最多只能处理2000请求，因为MySQL每秒只能处理2000个请求。系统A从MQ中慢慢拉取请求，每秒就拉取2000个请求，不要超过自己每秒能处理的请求数量即可。MQ，每秒5000个请求进来，结果只有2000个请求出去，所以在秒杀期间（将近一小时）可能会有几十万或者几百万的请求积压在MQ中。这个短暂的高峰期积压是没问题的，因为高峰期过了之后，每秒就只有50个请求进入MQ了，但是系统还是按照每秒2000个请求的速度在处理，所以说，只要高峰期一过，系统就会快速将积压的消息消费掉。我们在此计算一下，每秒在MQ积压3000条消息，1分钟会积压18万，1小时积压1000万条消息，高峰期过后，1个多小时就可以将积压的1000万消息消费掉。

作用:

1.用户的请求,服务器收到之后,首先写入消息队列,加入消息队列长度超过最大值,则直接抛弃用户请求或跳转到错误页面.

2.秒杀业务根据消息队列中的请求信息，再做后续处理.

**RabbitMQ基础概念**

消息队列服务通常有三个概念： 提供者、消息队列、消费者者

RabbitMQ 在这个基本概念之上在提供者和队列之间,加入了交换器 (Exchange)。这样提供者和消息队列就没有直接联系转而变成提供者把消息发给交换器，交换器根据调度策略再把消息转发给消息队列

常用的交换机有以下三种

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

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：

\* (星号) 用来表示一个单词 (必须出现的)

# (井号) 用来表示任意数量（零个或多个）单词

通配的绑定键是跟队列进行绑定的，举个小例子

队列Q1 绑定键为 \*.TT.\* 队列Q2绑定键为 TT.#

如果一条消息携带的路由键为 A.TT.B，那么队列Q1将会收到；

如果一条消息携带的路由键为TT.AA.BB，那么队列Q2将会收到；

**消息确认机制**

什么是消息确认ACK

如果在处理消息的过程中，消费者的服务器在处理消息的时候出现异常，那么可能这条正在处理的消息就没有完成消息消费，数据就会丢失。为了确保数据不会丢失，RabbitMQ支持消息确定-ACK。

ACK的消息确认机制

ACK机制是消费者从RabbitMQ收到消息并处理完成后，反馈给RabbitMQ，RabbitMQ收到反馈后才将此消息从队列中删除。

1.如果一个消费者在处理消息出现了网络不稳定、服务器异常等现象，那么就不会有ACK反馈，RabbitMQ会认为这个消息没有正常消费，会将消息重新放入队列中。

2.如果在集群的情况下，RabbitMQ会立即将这个消息推送给这个在线的其他消费者。这种机制保证了在消费者服务端故障的时候，不丢失任何消息和任务。

3.消息永远不会从RabbitMQ中删除，只有当消费者正确发送ACK反馈，RabbitMQ确认收到后，消息才会从RabbitMQ服务器的数据中删除。

4.消息的ACK确认机制默认是打开的。

ACK机制的开发注意事项

如果忘记了ACK，那么后果很严重。当Consumer退出时候，Message会一直重新分发。然后RabbitMQ会占用越来越多的内容，由于RabbitMQ会长时间运行，因此这个"内存泄漏"是致命的

12Shiro安全框架

**shor是一个安全框架 可以用来做登录认证和权限控制常用的方法有**

**login();进行认证的**

**isAuthenticated() 判断是否需要认证**

**getSubject() 获取主题**

**getInstance(); 安全管理器**

**hasRole(); 获取角色**

**isPermitted(); 获取权限**

**在项目中我们实现权限控制是由五张表来完成的一张用户表 一张权限表 一张角色表 一张角色权限表 一张用户权限表 不同的用户登上可以看到不同的页面和功能**

**在接触shor框架之前 权限框架和登录拦截都是通过拦截器来完成的 后来在网上看到了shor框架就开始学习起来 用起来还不错 在项目中我是这么使用shor框架的**

**实现这一系列功能是通过shor框架来完成的 要在web 中配置shor的过滤器 同时配置一个shor框架的配置文件 在配置文件中开启安全认证 ，以及要注入的方法，配置要过滤的方法 还要在springmvc 中开启shor的注解 新建一个类继承AuthorizingRealm这个类 得到两个方法 一个可以用来做登录认证 一个可以用来做权限控制**

**先说一下登录认证吧 在用户登录的时候 通过 SecurityUtils.getSubject(); 获取Subject对象 把用户的账号和密码放入UsernamePasswordToken()对象 通过Subject的login的方法把用户的账号和密码传到我们新建的这个类中，这doGetAuthenticationInfo的方法中的参数获取认证对象 然后获取要认证的对象 通过账号去数据查询获取信息 判断用户是否存在 如果不存在的话就返回一个状态码 存在的话新建一个SimpleAuthenticationInfo对象把 这个对象返回出去 在务层 根据返回的值 去提示用户是否登陆成功。**

**权限控制**

**创建一个SimpleAuthorizationInfo对象再去获取方法中的参数获取**

**需要认证的凭证根据凭证的信息去数据库中查询数据 同时每个方法上要加上一个@RequiresPermissions注解用来做权限控制 根据每条信息的类型来判断是按钮还是目录啥的 最后把认证的信息放入SimpleAuthorizationInfo这个对象中最后作为返回值返回回去。**

详情网址：https://blog.csdn.net/nglhaeg/article/details/108322824