# Java基础

## String类为啥是不可变的

因为String类型是final修饰的，而被final修饰的类是不可以被继承的

## 2、java中的队列有哪些，有什么区别？

阻塞队列，当队列是空时，向里面get数据时会阻塞，当队列满时，向里面add数据会阻塞

## 3、ArrayList、LinkedList、Vector的区别？

**ArrayList和Linkedlist区别？**

1、相同点

1.1、都实现了List接口

2、不同点

2.1、ArrayList存储的是一组有顺序的数据，每个元素都有索引，so

在get和set时，效率很高，在add和remove时，效率较低，需移动元素

2.2、LinkedList存储的是链表，元素与元素之间是通过next来连接，

add和remove时，效率高，只需要改变相邻两个元素的连接就行，在get和set效率较低，需要一个一个的依次往下查找

**ArrayList和Vector的区别？**

一样是实现了List接口

1、ArrayList是非线程安全的，而Vector是线程安全的，在效率上，ArrayList高

2、当存储空间不足时，ArrayList大小扩容为原来大小的0.5倍，Vector扩容为原来大小的1倍

## 4、String、StringBuilder、StringBuffer

运行速度来说 String < StringBuffer < StringBuilder()

String是字符常量 其余两个是字符变量 所以在执行效率来说 String < 其它两种

StringBuffer < StringBuilder,因为StringBuilder是非线程安全的

StringBuilder和StringBuffer默认大小是16，当大小不够的时候，自动扩容（其实是重新新建了一个数组，然后将之前数组中的内容copy到新建数组中）

数组在定义好之后，不能更改大小

扩容的大小 = 原数组大小 \* 2 +2

## 5、HashMap和HashTable的区别

都实现了Map接口，都是hash表来存储

1、HashMap是非线程安全，允许null值和null键

2、HashTable是线程安全，不允许有null值和null键

Ps:运行效率上，因为HashMap是非线程安全的，所以相对来说高于HashTable

## 6、String类的常用方法

toString将其它字符转换成字符串

replace 字符串替换

equals 字符串比较

substring 字符串截取

split 字符串分割

toCharArray 将字符串转换成字符数组

length 获取字符串长度

charAt查找字符串中指定字符位置

getBytes 将字符串转换成Byte数组

indexOf 查找指定字符位置

trim 替换字符空格

## 7、java的基本数据类型和引用数据类型有哪几种

基本数据类型

Int（4）、double（8）、float（4）、long（8）、short（2）、char（2）、byte（1）、boolean（1）

引用数据类型

Array（数组）、class（类）、interface（接口 ）

## 8、类的实例化顺序，比如父类的静态方法、构造方法、普通方法，子类的静态方法、构造方法、普通方法执行顺序

一个类的实例化过程:

1. 父类的静态方法
2. 子类的静态方法
3. 父类的构造方法
4. 子类的构造方法
5. 父类的普通方法
6. 子类的普通方法

## 9、抽象类和接口的区别，类可以继承多个类么，接口可以继承多个接口么,类可以实现多个接口么。

抽象类和接口的区别

1. 抽象类中的抽象方法和接口中的方法一样，都不能有方法体
2. 抽象类可以定义非抽象的方法，接口中的方法都是抽象的
3. 抽象类与接口都不能实例化
4. 抽象类要被子类继承，接口要被子类实现
5. 接口中定义的属性默认是final，抽象类中可以包含非fianl的属性

类可以继承多个类吗

不可以，一个类只能继承一个类，但是可以实现多个接口，接口可以继承多个接口

## 10、Overriding和Overloading的区别

1、Overriding是重写，Overloading是重载

2、Overriding是针对父类子类，Overloading针对同一个类

## 11、什么是值传递，什么是引用传递

1. 基本数据类型都是值传递，在传递的过程值直接发生改变
2. 除基本数据类型之外的都是引用传递，在传递的过程中，会不断的创建不同的副本，原对象不会被更改

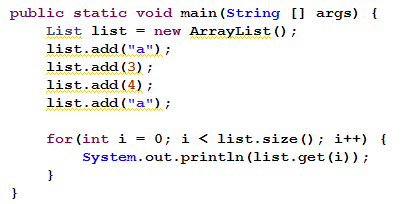
## 12、进程和线程的区别是什么

进程是一个独立的应用程序，包含一个或多个线程

## 18、集合的理解

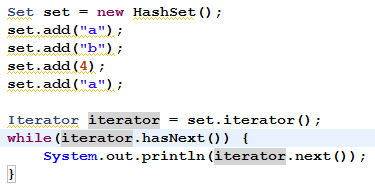
List:有序集合，可以包含重复的元素

实例操作：



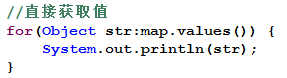
Set：无序集合，不可以包含重复的元素

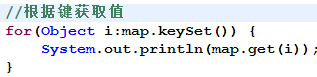
[ArrayList、LinkedList、vecto](#_3、ArrayList和LinkedList的区别？)r

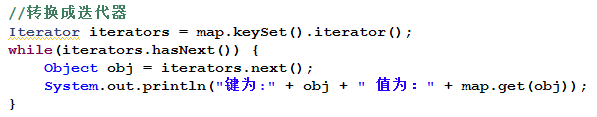


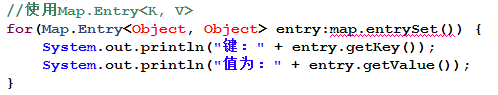
Map：无序集合，不能有重复的key，可以有重复的value











## 19、Array与ArrayList的区别

相同点：都能存储不同类型的数值

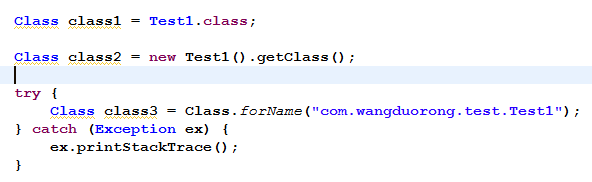
不同点：Array的大小是固定的，ArrayList大小是可变化的，容量不足时，增加原来容量的50%

ArrayList对元素的操作提供了更加丰富的方法

## 20、System.gc()和Runtime.gc()的作用

告诉JVM，进行垃圾回收，但是是立即还是延时要看JVM

## 21、反射的原理，反射创建实例的三种方式



## 22、反射中Class.forName()和ClassLoader()的区别

Class.forName()在执行的时候，除了将.class加载到JVM，还会执行static块

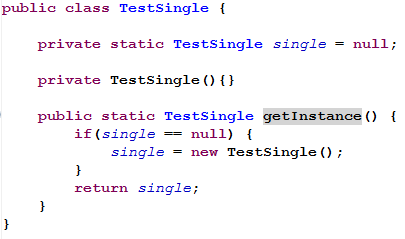
ClassLoader()在执行的时候，只会将.class文件加载到JVM中，不会执行static块，在实例化时才去执行static块

## 24、几种单例模式

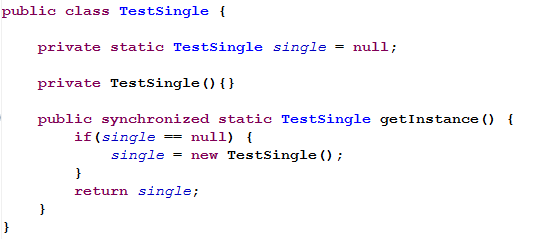
大的来说只有两种（饿汉式、懒汉式），但是为了处理不同的场景可以衍生出不同的单例模式

懒汉式的一个最大的好处就是，懒加载，使用时在初始化

1. 懒汉式，不做任何处理是一种

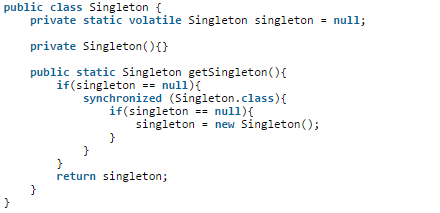


1. 懒汉式，在返回对象的方法中增加synchronized（生亏啦死得）关键词

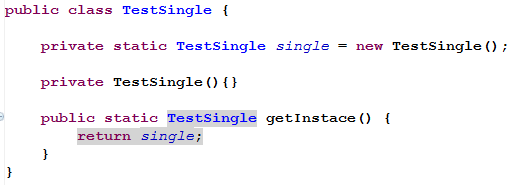


好处就是，线程安全，但性能低，变相的变成了串模式

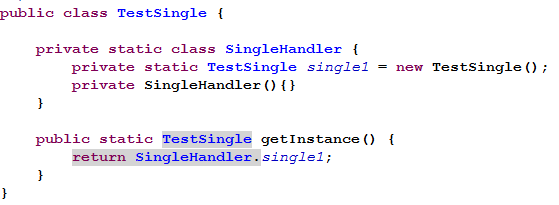
1. 懒汉式，双重检验，同步代码块



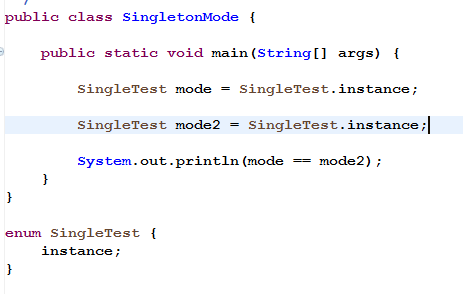
1. 饿汉式，类一加载就进行实例化



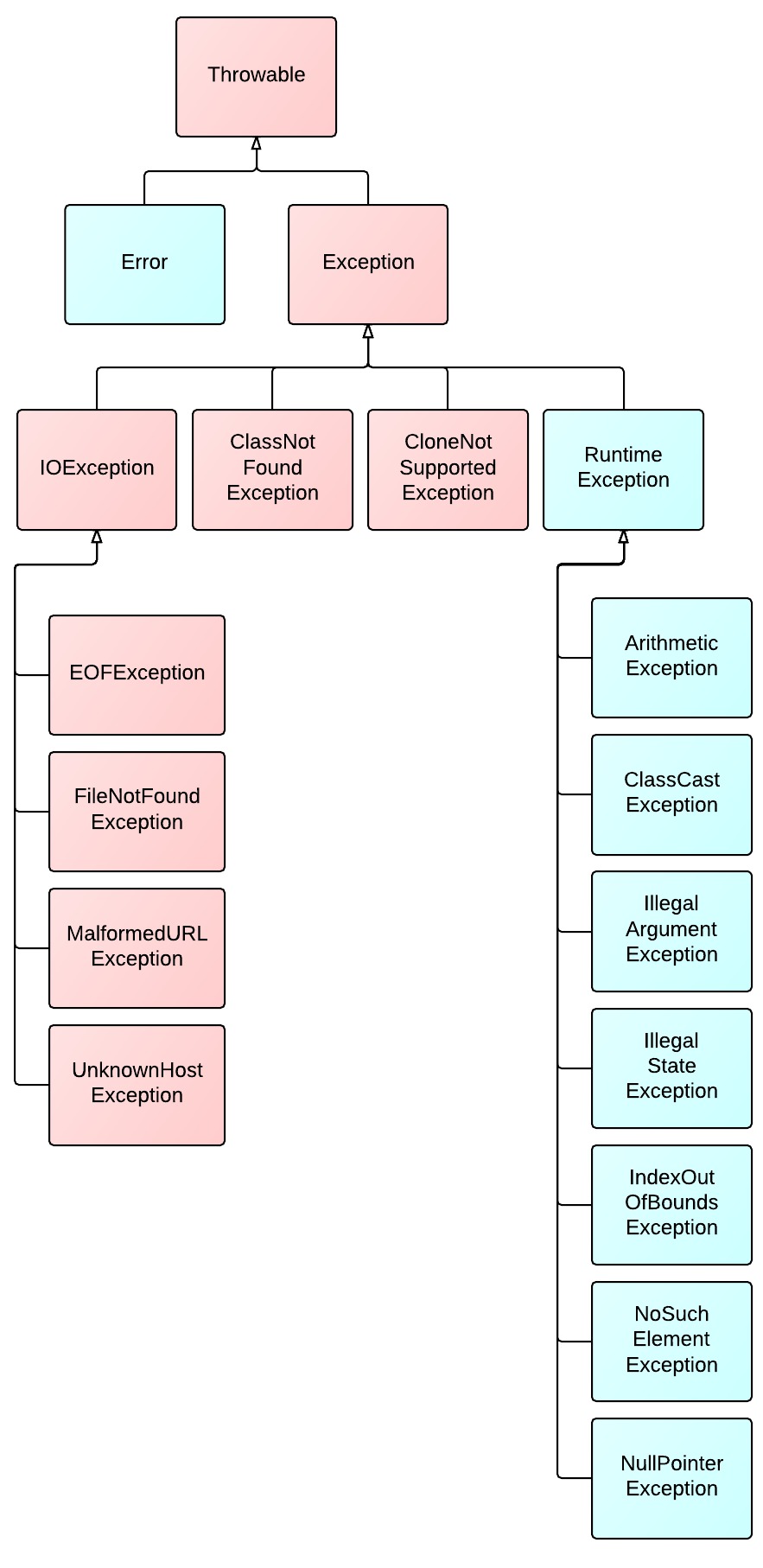
1. 饿汉式，内部类处理



6、枚举，简单方法，避免了反射和序列化，没有懒加载



## 25、异常的理解



从上图中可以看出，有一个顶级父类异常，就是Throwable（穿楼包），下面有两个实现类error、exception

Error一般是由JVM抛出的异常，不做处理，一般终止程序，如内存溢出，内存泄漏

Exception一般是由程序代码引起的异常，需try{}catch(){}或者抛到上一级处理

Error与Exception有一个本质的区别就是一个是程序本身引起的错误，且能处理，一个是JVM的错误，无法处理

Exception下面有四个实现类，一个是IoException、RuntimeException、ClassNotFoundException

IoException主要是操作文件类的错误

RuntimeException 运行异常，比如空指针异常、数组越界、算术错误、类型转换错误、数据格式错误

ClassNotFoundException：类加载不到异常，类加载异常

Exception分为两大类异常，一类是运行时异常，一类是编译异常

## 26、内存溢出和内存泄漏可能发生的情况

内存溢出

1. 程序中存在大量不必要的对象，使用完后没有及时清掉，一直存在内存中
2. 一次操作，在内存中加载了大量的数据，在Hibernate中比较常见

内存泄漏

1. 生命周期长的对象引用生命周期短的对象

全局变量引用局部变量

1. 连接数据，忘记关掉

## 27、Swtich能否用String类型做参数

在jdk1.7之前，只能用int,byte,char,short做参数，之后可以使用String

## 28、equals（）和 == 的区别

==在基本数据类型中比较的是两个对象在内存中存放的值

==在引用类型中比较的是两个对象在内存中存放的地址

Java当中所有的类都是继承自Object这个基类，在该类中定了一个equals方法，该方法也是使用==进行比较，也就是比较两个对象的内存地址，但是在String、Integer、Date这些类中，覆盖了equals方法，比较的是他们的值是否相等

## 29、HashCode与Equals的区别

Equals比较的是两个对象的地址是否相同（重写的equals除外）

为什么重写完equals方法需要在重写hashcode方法？

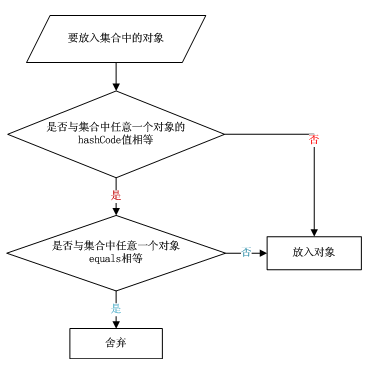
主要目的是：保证是同一个对象

1. 两个对象equals相同，hashCode一定相同
2. 两个对象hashcode相同，equals不一定相同

如果该对象不需要放在需要先计算Hashcode在来存储的集合中，可以不需要重写

HashCode方法只在集合中使用

Set集合中假如有100个元素，重新实例化一个对象存入，如果使用equals比较，会进行100次对比，使用HashCode，首先是判断新增对象的hashCode是否与集合对象中某一个Hashcode相同，如果不相同就直接存入对象，相同则在去判断equals



## 30、解析XML的几种实现方式

1、dom（dom4j、jdom）

2、sax

## 32、多态的原理

面向对象的三大特征：封装、继承、多态

多态的定义就是同一个方法，根据发送的对象不同而采用不同的处理方式（比如重载，或者重写）

## 33、jdk7与jdk8的新特性

Jdk1.7之前，只能支持int,byte，char,short,枚举

Jdk7

1、Swtich中增加了字符串类型

2、集合中增加了语言支持

List<String> list = [“王朵荣”];

List[0]

Set<String> set = {“王朵荣”};

Map<String,Integer> map = {“key”:1};

map[“key”]

3、在try..catch中可以捕获多个异常，用|隔开

Jdk8

接口方法中可以设置默认方法，之前方法不能有方法体

## 34、java的引用类型有哪几种

强引用、弱引用、软应用、虚引用

1. 如果一个对象具有强引用，JVM是不会回收这个对象，所以如果强引用对象过多会导致内存溢出
2. 如果一个对象具有弱引用，JVM在监控到没有其他对象在调用它时会立马进行回收
3. 如果一个对象具有软引用，JVM会先判断内存空间是否足够，如果足够不后回收，不足够，就会回收这个对象
4. 如果一个对象具有虚引用，JVM在任何时候都有可能随时回收对象

## 35、数组的定义

int [] int1 = new int[2];

int [] int2 = {4,6,7};

int [] int3 = new int[] {4}

## 36、排序

1、可以借助Collections中的sort方法进行排序

2、冒泡排序

***~~@Test~~***

**public** **void** **sort**() {

**int** [] ints = **new** **int**[]{2,6,8,3,9,7};

System.out.println("原有数组的排序如下：");

**for**(**int** i = 0; i < ints.length; i++) {

System.out.print(ints[i] + " ");

}

System.out.println();

System.out.println("使用工具Collections类处理后数组的排序如下: ");

List<Integer> list1 = **new** ArrayList<Integer>();

**for**(**int** i = 0; i < ints.length; i++) {

list1.add(ints[i]);

}

Collections.*sort*(list1);

**for**(**int** a = 0; a < list1.size(); a++) {

System.out.print( list1.get(a)+" ");

}

System.out.println();

System.out.println("使用冒泡排序处理后数组的排序如下,从大到小: ");

**for**(**int** i = 0; i < ints.length-1; i++) { //确定大的循环次数

**for**(**int** j = 0; j < ints.length-1-i; j++) {

**if**(ints[j] < ints[j+1]) {

**int** temp = ints[j];

ints[j] = ints[j+1];

ints[j+1] = temp;

}

}

}

**for**(**int** i = 0; i < ints.length; i++) {

System.out.print(ints[i] + " ");

}

System.out.println("获取数组中的最小值和最大值： ");

**int** temp = ints[0];

**for**(**int** i = 0; i < ints.length - 1; i++) {

**if**(temp > ints[i+1]) {

temp = ints[i+1];

}

}

System.out.println("数组最小值为: " + temp);

}

## 37、重写equals方法为什么要重写hashCode方法

1、两个对象equals相同，hashCode一定相同

2、两个对象的hashCode相同，equals不一定相同

重写equals方法之后又重写HashCode就是为了保证是同一个对象

Set集合在添加时，就是用hashcode来判断的

## 41、举例说说几个排序，并说明其排序原理

1、冒泡排序

用第一个元素与其它的元素进行比较，然后进行互换，之后再用第二元素与其它元素比较互换，直至排序完成

2、快速排序

设置一个基准元素，然后与其它元素循环比较，互换，使用递归的调用

3、选择排序

4、插入排序

## 42、序列化的原理和作用

序列化是将对象转换成字节，需要实现Serialization

反序列化是将字节转换成对象需要实现deserialization

主要用在数据的传输中，比如Http的或者webservice的传输中

## 43、jdk1.5后增加了哪些并发处理类

Synchronized和lock

Java.util.Concurrent包

## 44、其它

### 1、Eclipse中如何安装插件

1. 在线安装

打开Eclipse -> Help -> Install New Software

输入<http://download.eclipse.org/releases/juno>点击查询，然后选中自己需要的插件进行安装

1. 将下载好的插件放到tomcat的plugins目录下

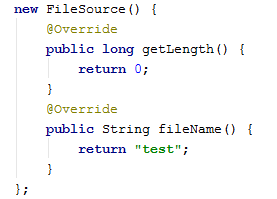
两个都需要重启eclipse

### 2、MyEclipse2017如何激活

https://pan.baidu.com/s/1qZcwN4C 密码：2h5e

<http://blog.csdn.net/qq_34131212/article/details/78156888>

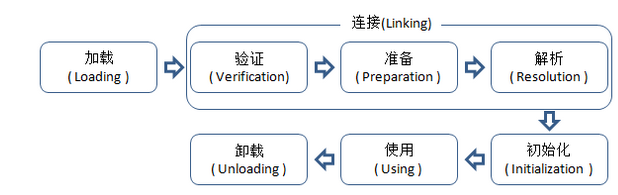
### 2、实例化接口，重写接口中的参数



# Java高级

## 1、java的JVM相关知识点

### 类加载器的机制

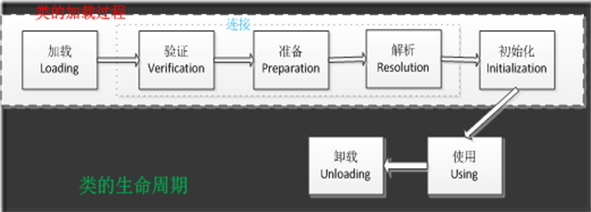


从上图中可以看出java文件，经过javac编译成.class文件的字节码，然后通过类加载器（ClassLoader）加载到JVM中，然后经过链接，初始化，使用，卸载

**整个类的加载过程分为加载、链接(验证、准备、解析)、初始化**，这些步骤是顺序开始，但不一定顺序进行或完成，因此在执行的过程中会有交叉的进行

**Ps:通过ClassLoader将.class文件加载到JVM中，并对该进行验证、准备、解析、初始化，最终形成虚拟机直接调用的java类型**

#### 类的生命周期



#### 加载

在加载过程中，虚拟机需要完成3件事情

1. 通过一个类的全路径名来获取该类的二进制字节流（加载.class的方式有很多种
2. 从java源文件动态编译成.class文件
3. 从本地系统中直接加载
4. 通过网络下载.class
5. 从数据库中获取.class
6. 从jar中获取.class）
7. 将这个字节流代表的静态数据结构转换为方法区的运行时数据结构
8. 在内存的堆区创建一个java.lang.Class的对象，通过这个对象可以访问方法区中对应类的结构

**总结：第一步获取到.class文件，第二步在方法区中针对该类生成一个运行时的数据结构**

**第三步在堆区创建于一个java.lang.Class的对象，通过对象引用方法区中的数据结构**

#### 验证

验证是链接阶段的第一步，主要是确保加载的.class文件中是否符合JVM的要求，是否有危害到JVM自身安全，主要进行的检验有（文件格式、元数据、字节码、符号引用验证）

#### 准备

准备是链接阶段的第二步，为类的静态分配内存并设置类变量的初始值，在这个时候进行内存分配的只有静态变量，初始化的那些值都是‘通常是’（0，null，false等），如下：



初始化的值是0，而不是123，把123赋值给类变量是在初始化的时候

在特殊情况下，比如在类变量前面增加一个final，则会直接进行赋值

#### 解析

解析是链接阶段的最后一步，把方法区中常量池内的符号转换为直接引用，解析动作主要是针对类、接口、字段、方法、类型等7类符号

将符号引用转换成直接引用，将符号引用直接指向目标的指针

比如：人与人的位置，从概念化转换成具体

#### 初始化

主要是对类变量赋值，初始化是在类构造器方法中完成，静态变量、静态块只会执行一次

通过类的构造函数对类变量，静态块合并

在初始化该类的时候发现有父类，则先初始化父类

##### 类初始化的时机分为主动引用和被动引用

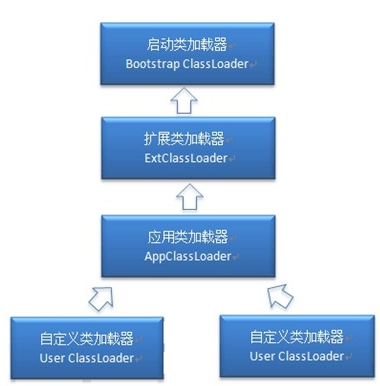
主动引用

1. new一个对象
2. 调用类中的静态变量，除final修饰的常量除外，因为final修饰的变量在连接的准备阶段就已经初始化
3. 调用类的静态方法
4. 反射加载类
5. 初始化某个类的子类，则父类也会被初始化
6. 虚拟机启动后，先初始化main方法对应的类

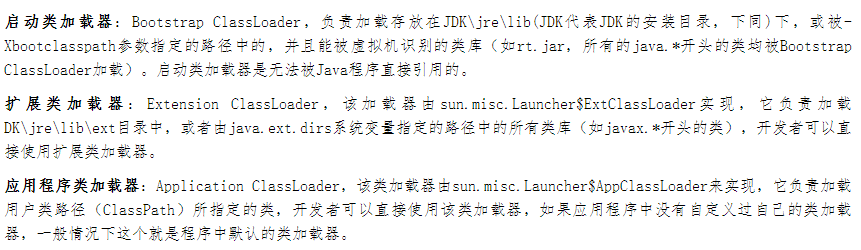
被动引用

1. 通过子类去调用父类的静态变量或者静态方法时，子类不会被初始化
2. 引用常量不会被初始化

#### 类加载器的的层次关系



从上图中可以看出类加载器总共有四类



**自定义类的加载器，继承ClassLoader类，重写findClass**，比如需要通过网络来传输java类的字节码，为了保证安全性，需先对字节码进行加密处理

**package** com.wangduorong.jvm;

**import** java.io.ByteArrayOutputStream;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.io.InputStream;

/\*\*

\* 自定义加载器

\* **@author** wnagduorong

\*

\*/

**public** **class** TestMyClassLoader **extends** ClassLoader {

//定义一个文件根目录

**private** String root;

//重写ClassLoader类中的findClass方法

**protected** Class<?> **findClass**(String name) **throws** ClassNotFoundException {

/\*

\* 查找类加载其中是否已经存在

\* 如果存在直接返回

\* 如果不存在就使用双亲委托模式或者自定义的类加载器

\* 既然是使用自定义类加载器，该方法在加载的时候只使用自定义的加载器进行类加载

\*/

**byte** [] classData = loadClassData(name);

**if**(classData == **null**) {

**throw** **new** ClassNotFoundException();

} **else** {

//使用自定义的类加载

**return** defineClass(name, classData, 0,classData.length);

}

}

//通过类名找到具体位置下对应的.class，然后将其转换为byte数组

**private** **byte**[] **loadClassData**(String className) {

//组成一个完整的文件路径，然后去该类中加载指定的类

String fileName = root + File.separatorChar

+ className.replace('.', File.separatorChar) + ".class";

InputStream input = **null**;

//创建一个字节数组输出流

ByteArrayOutputStream baos = **new** ByteArrayOutputStream();

**try** {

//创建一个文件输入流

input = **new** FileInputStream(fileName);

//定义一个byte数组，大小为1M=1024

**byte** [] buffer = **new** **byte**[1024];

**int** temp = 0;

//从文件流中循环读取字节

**while**((temp = input.read(buffer)) != -1) {

//将读取的字节写入到字节数组流中

baos.write(buffer,0,temp);

}

**return** baos.toByteArray();

} **catch** (IOException ex) {

ex.printStackTrace();

} **finally** {

**if**(input != **null**) {

**try** {

input.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**return** **null**;

}

**public** **static** **void** **main**(String[] args) {

TestMyClassLoader loader = **new** TestMyClassLoader();

loader.setRoot("E:/java");

// D:\Myeclipse\_workspace\java基础知识\src\com\wangduorong\test\com\wangduorong\test

String fileName = "com.wangduorong.test.TestLoader";

Class<?> cls = **null**;

**try** {

cls = loader.findClass(fileName);

Object object = cls.newInstance();

System.out.println(object.getClass().getClassLoader());

} **catch** (Exception ex) {

ex.printStackTrace();

}

}

**public** String **getRoot**() {

**return** root;

}

**public** **void** **setRoot**(String root) {

**this**.root = root;

}

}

从上面代码中可以看出

第一步：继承ClassLoader类，重写findClass方法

第二步：通过类全文件名找到该类，然后通过文件流将该类转换成byte数组

第三步：调用ClassLoader类中defineClass方法，返回一个Class对象

**在自定义类加载器中需要注意几个点**

1. 方法中传递的文件名是全路径名称，因为defineClass方法就是根据这种格式进行处理
2. 最后不要重写loadClass方法，因为会破坏双亲委托模式

最顶层的那个启动类加载器是用C++编写的，主要加载存放在lib下面的类，如rt.jar(java核心包)

扩展类加载器加载的是lib下ext目录下的jar包

**程序中默认的加载器是应用程序类加载器（Application ClassLoader）**

**类加载是使用代理模式来加载指定的类**

双亲委托机制

双亲委托机制的工作流程是：如果一个类加载器收到了一个加载类的请求，首先它自己先不去加载，而是把请求委托给父类加载器去完成，依次向上，直至找到最高级别的类加载器进行加载，如果加载不了，则会退回到下一级类加载器中去完成加载

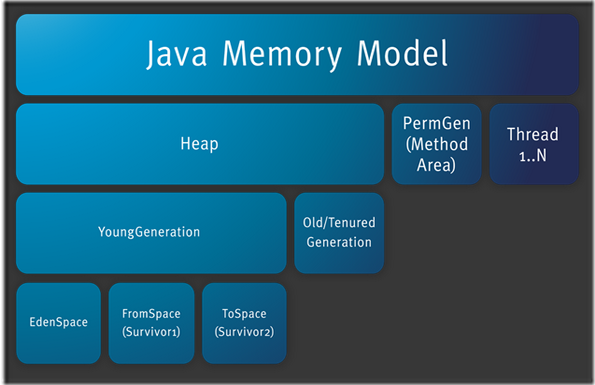
总括：加载类是，父类优先，父类处理不了在交个子类

Ps：**并不是所有的类加载器都是按照双亲委托模式来加载类，tomcat服务器的恰恰相反，先从请求的类加载中完成加载，如果加载不了才会找父类去加载**

双亲委托模式的作用：

保证java核心类的安全性，限制用户不能自定义java.lang.String类的情况

### JVM内存结构



JVM内存结构主要有三大块：堆、栈、方法区

#### Java堆（Heap）

**堆内存是JVM中最大的一块，主要存放的是对象的实例**

**堆内存是垃圾收集管理器的主要区域，因此很多时候也叫”GC堆”**

**从回收的角度来看，垃圾回收器是根据分代收集算法，所以java堆区还可以细分为年轻代和年老代，年轻代又分为Eden空间、From Survivor(s娃吻)空间、To Survivor空间**

**Java堆区的大小可以设置成固定的也可以设置成可扩展的（设置一个范围）,通过-Xmx和-Xms来控制**

#### JVM栈

与堆区和方法区不同，JVM栈是线程私有的

主要存放基本数据类型和对象的引用，为方法服务

每个方法被执行的时候，都会创建一个栈帧，每一个方法从调用到执行完成，对应着一个栈帧的入栈和出栈全过程

在栈操作过程中有两种异常

1. 入栈时，当栈内存已经满了，就会抛出一个StackOverFlowError（栈溢出）-SOF
2. JVM栈的大小是可以扩展的，当扩展时申请的大小超过内存剩余大小，就会抛出OutOfMemoryEroor（内存不足）

#### 方法区

该区和java堆一样，都是线程共享的内存区域，主要存储类的信息、静态变量、常量等数据。根据Java虚拟机规范的规定，当方法区无法满足内存分配需求时，将抛出OutOfMemoryError异常

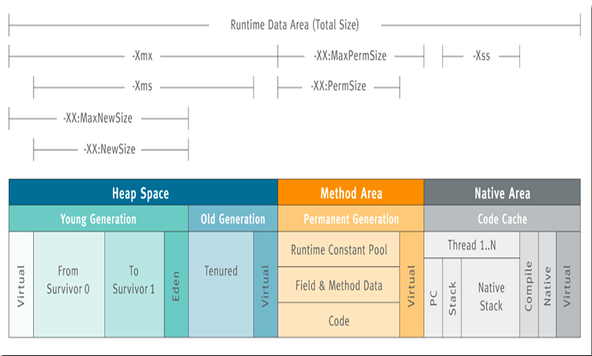
#### 程序计数器

在内存中占有很少的一部分空间

如果现在执行的是java方法，那么计数器记录的是正在执行的虚拟机字节码指令

如果执行的是原生态方法，那么计数器为0

#### 本地方法栈





### GC机制、算法、垃圾回收器

Jvm中虚拟机栈、程序计算器和本地方法栈都会随着线程结束而消失

栈帧随着方法的进入和结束做入栈和出栈操作，实现了自动的内存清理

**所以一个垃圾回收器主要针对堆内存和方法区**

#### 对象存活判断

判断对象是否存活一般有两种方式：

##### 引用计数

每一个对象都有一个引用计数器，新增一个引用，计数器+1，引用释放，计数器-1，当对象引用计数器为0是，GC进行回收，**无法解决对象相互循环引用问题**

##### 可达性分析

从GC Roots开始向下进行搜索，当一个对象与GC Roots中的对象没有任何的引用，则说明这个对象是不可用的，GC进行回收

GC Roots包括：

1. 虚拟机栈中的引用对象
2. 方法区中静态属性实体引用对象
3. 常量池中引用的对象
4. 本地栈中JNI的引用

#### 垃圾收集算法

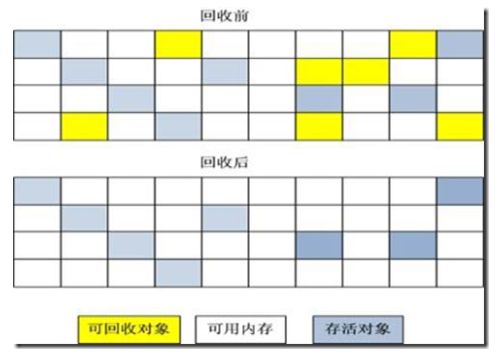
##### 标记-清除算法（基础算法）

该类算法分为两步，第一步是标记，第二步是清除

首先标记哪些对象是需要回收的，在标记完成后统一回收被标记的对象

缺点：

1. 效率问题，不论是标记和清除效率都不高
2. 空间问题，标记清除后会产生大量不连续的空间碎片，会导致某些需要连续分配内存的对象得不到想用的内存(比如ArrayList，因为是顺序存储的，所以在分配空间时也是连续分配内存)，而触发垃圾回收



##### 复制算法

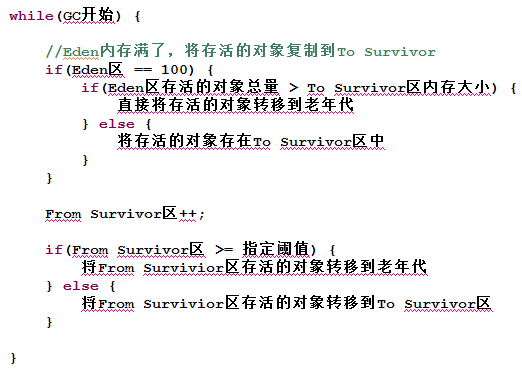
将可用内存分为两块大小相等的区域，每次只是用其中的一块，当这一块的内存用完了，将内存中还存活的对象复制到另一块内存中，然后把无用的对象进行回收

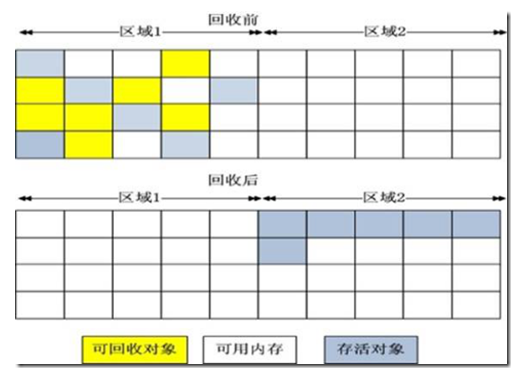
在一块内存中进行回收，不需要考虑空间碎片的场景，只需要移动指针，按顺序分配内存即可，效率高

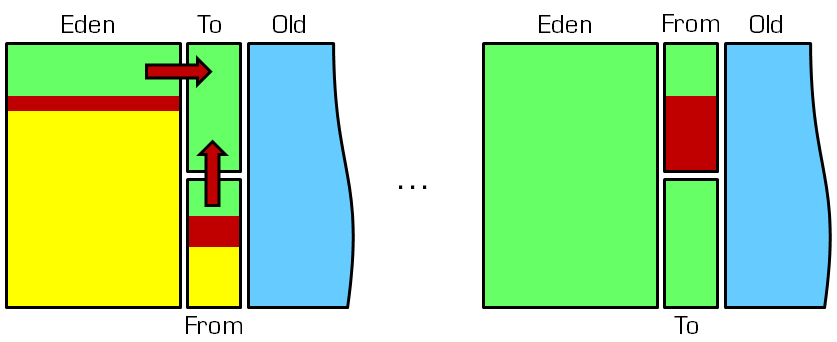
缺点是将可用内存缩小到原来的一半，持续复制生命周期长的对象会导致效率低

现在很多虚拟机已经改进了这个算法，分配比例上更加细化，8:1:1，较大一块内存交给Eden区，另外两块分别交给Survior(死W稳)区。

新建的对象只会存在Eden区和From Survivor区，To Survivor是空的

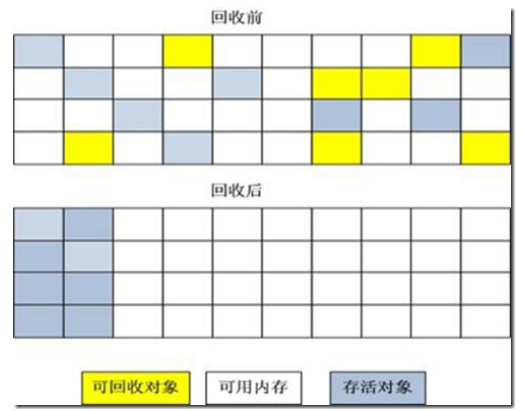






##### 标记-压缩算法

延续标记清除算法，先标记哪些对象需要清理的，标记完后暂不进行统一回收，先将可用的对象存放在一个连续的可用内存中，然后回收除这块之外的标记对象



##### 分代收集算法

将堆内存分为新生代和老年代，根据各个年代的特点进行不同的回收机制

在新生代中，每次垃圾回收前都会有大量的对象消失，只有少量存活的，一般选用复制算法，只需要移动少量的指针就可以完成收集，而对于老年代，对象存活率长高，一半选用标记清除或者标记压缩的回收机制进行收集

Ps：根据不同的场景进行不同的收集算法，从而达到高效

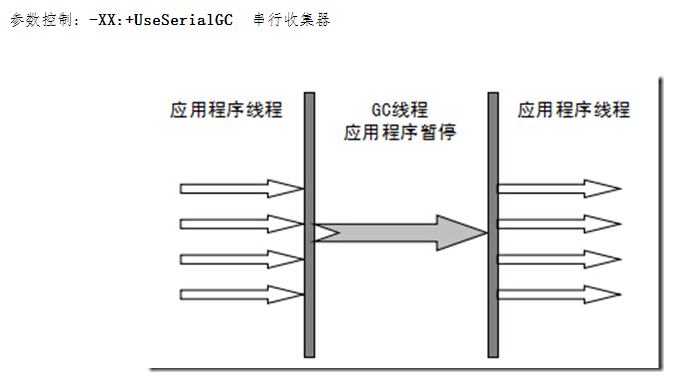
为什么要进行分代？

在不分代的情况下，GC一样能进行回收，但是每次在回收前都要搜索整个堆内存中的对象，效率低，而且大部分的对象都是很快就无用，如果分代的话，先将高频率回收的对象放到一个区域，可以腾出很大的内存来做其它事情

#### 垃圾收集器

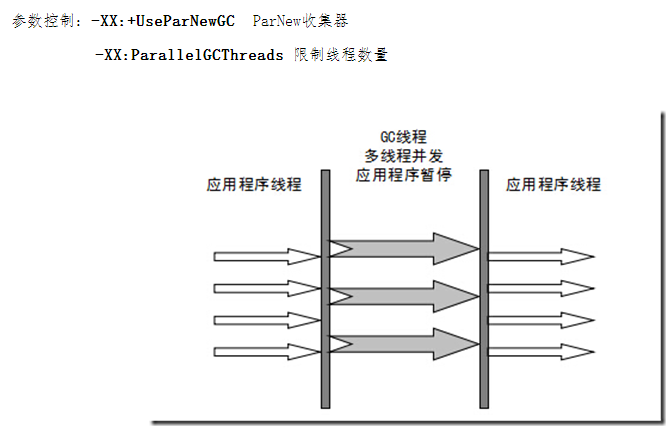
##### Serial收集器

是一种串行的收集器，只用一个线程去处理，所以在收集过程中可能会产生比较长的停顿



##### ParNew收集器

多线程去处理垃圾回收



#### GC如何判断对象是否可以被回收了

1. 对象没有引用（通过引用计数法和可达性算法来判断该对象是否有引用）
2. 执行了System.exit()方法
3. 程序正常执行完成
4. 程序在执行中遇到异常或错误而异常终止

在java程序中我们可以将一个对象显式的设置成null，然后调用System.gc()方法，告诉GC尽快来回收这个对象，但是这个方法尽量避免使用，因为使用System.gc()方法，会出发Full GC，影响性能

#### GC什么时候执行

GC分为Minor GC 和 Full GC

##### 触发Minror GC的条件

Eden区域内存空间不足的时候，会触发Minor（嘛二冷） GC

##### 触发Full GC的条件

1. 调用System.gc()
2. 老年代空间不足
3. 永久代空间不足，也就是方法区空间不足

#### GC在什么时候对什么东西做了什么事情-面试题

在Eden区和老年代内存不足时分别触发Minor GC和Full GC

通过引用计数法和可达性算法 找到无用对象

清理对象，整理内存

#### GC中的新生代如何转移到老年代-面试题

**堆区分为新生代和老年代，而新生代又分为Eden区、From Survivor区和To Survivor区**

新创建的对象都是存放在Eden区和From Survivor区，如果Eden区满了，会触发Minor GC进行回收，存活下来的对象复制到To Survivor区（特殊情况，如果存活的对象占用的内存大于To Survivor区内存大小，直接将对象复制到老年代）From Survivor区中存活的对象，年龄+1，达到阈值之后，复制到老年代，没有达到阈值，复制到to Survivor，经过GC之后，Eden区和From Survivor区全部清空，最后在把 To Survivor区的对象复制到From Survivor区中

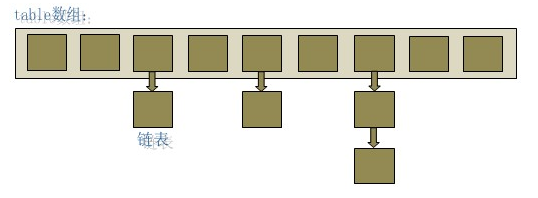
**总的一句话来概况：新创建的对象存放在Eden区（大对象直接存放在老年代），经过第一次Minor GC之后，存活下来的对象，转移到Survivor区。Survivor中的对象每熬过一次GC，年龄增长一次，当达到一定的阈值，将对象转移到老年代**

## 2、java数据结构和原理



### 1、Map

#### HashMap的实现原理是什么



**HashMap的底层就是一个数组+链表的结合体**

创建一个HashMap，就会初始化一个Entry数组，然后数组的每一项都是一个链表，Entry是一个静态类，类中有四个重要属性，key、value、next、hash，通过next获取下一个元素的引用，从而构成一个链表



Entry是一个静态类，包含三个重要的元素，键、值和一个next(烂kst)指针



**HashMap通过get和put来获取和存储数据**

**当我们向HashMap中put元素时，调用键对象的HashCode()方法来计算hash值，根据hash值确定这个元素在数组中的位置（return** hash % (数组长度-1)**）**

**如果该数组中不存在这个元素，则将这个元素添加到这个位置的链头**

**如果该数组中已经存在这个元素，首先判断它们的hash值和key是否相同，如果相同，则覆盖掉之前的那个key，翻盖完之后，循环继续遍历，假如还有相同的值，则继续翻盖，如果没有相同的，则新增一个Entry对象，放入链头，后加的那个放在链尾**

**当我们向HashMap中get元素时，调用键对象的HashCode()方法来计算hash值，根据hash值获取元素的对象getEntry()，然后返回元素的值**

#### HashMap在什么时候扩容？

**HashMap大小默认是16，HashMap不会等待数组全部存满后在去扩容，**

**在添加元素后，HashMap如果发现元素总和大小（数组的长度\*加载因子） >= 阈值，则会调用resize()进行扩容，扩容为原来大小的两倍，HashMap的加载因子默认是0.75**

#### HashMap如何解决扩容后带来的问题

在高并发的情况下，会存在条件竞争，而导致死锁的出现

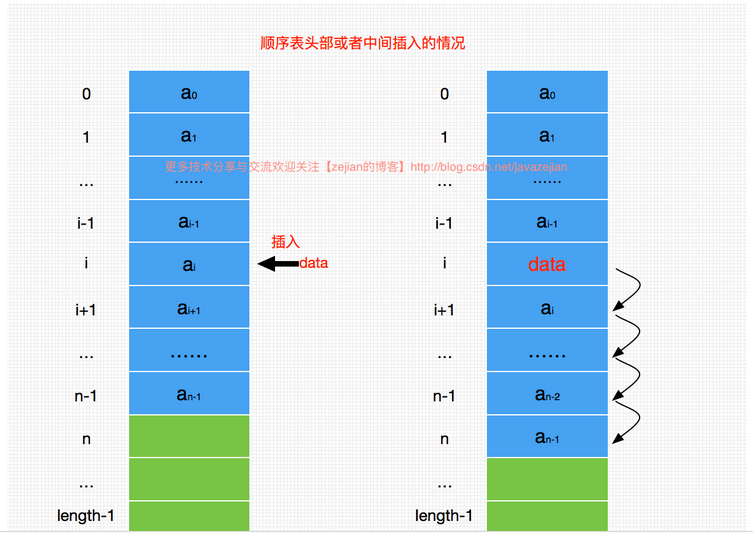
后续在补上

**HashMap通过链表的形式来解决冲突或者碰撞**

### 2、List原理

#### ArrayList的实现原理

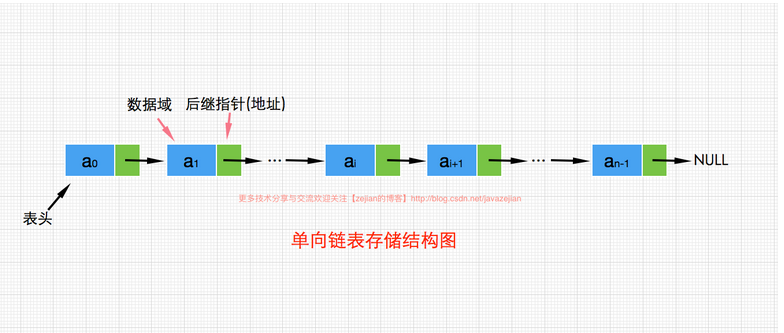
ArrayList存储的是一组有顺序的动态数组，**实例化后JVM会为该数组分配一块连续的内存块**，数组默认的大小是10，在数组容量不足的情况下，扩容1.5倍（int newCapacity = oldCapacity + (oldCapacity >> 1)），在get()和set()时，效率很高，但是在新增和删除的时候，因为是顺序存储的，所以需要移动数组，导致效率会比较慢（有一种特殊情况，就是新增和删除的数据假如是最后一个元素，则不会影响性能，相反会很高效）



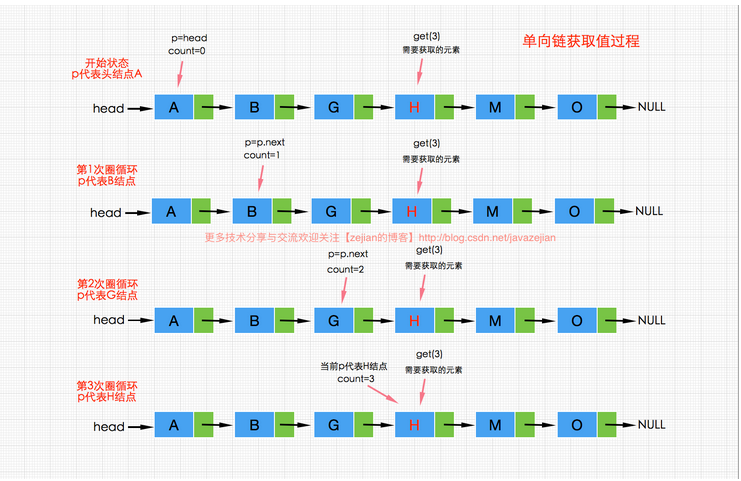


删除的方法和新增类似，不做演示

#### LinkedList的实现原理

LinkedList存储的是链表形式的数组，在内存中不是依次排序的存储，而是诺干个地址散列的存储，每一个存储的单元中都会有一个地址指向下一个元素，在新增和删除效率高  


在get值时的流程图，获取的元素越往后，效率越低，因为是一个个的一次往下找



## 3、java设计模式

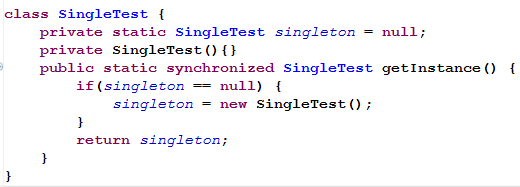
我个人用的比较多的：单例模式、工厂模式、抽象工厂模式、代理模式、适配器模式

### 单例模式

保证一个类只有一个实例

#### 实例

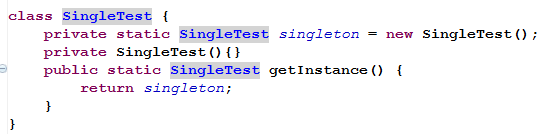
1. 懒汉式：线程安全、支持懒加载、效率不高（线程同步）



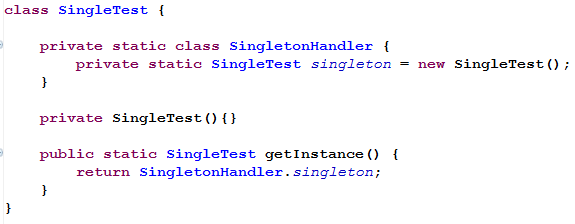
1. 懒汉式-双重检验：线程安全、支持懒加载，但由于JVM底层内部模型原因，偶尔会出错



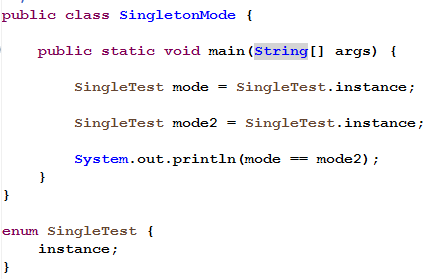
1. 饿汉式：线程安全，不支持懒加载，效率高



1. 饿汉式-内部类：线程安全，支持懒加载，效率高



1. 枚举类



#### 通过反射和反序列化破解单例(不包含枚举)和解决方法

反射最后通过newInstance()方法获取真正的对象，相当于重新new了一个对象

通过抛出异常在处理

将对象写入硬盘，然后在从硬盘读取，读取的是另一个对象

通过readRObject方法

#### 应用的场景

1、电脑中的任务管理器

2、项目中的一些配置文件，一般也是只有一个对象，不需要创建多个对象去读取

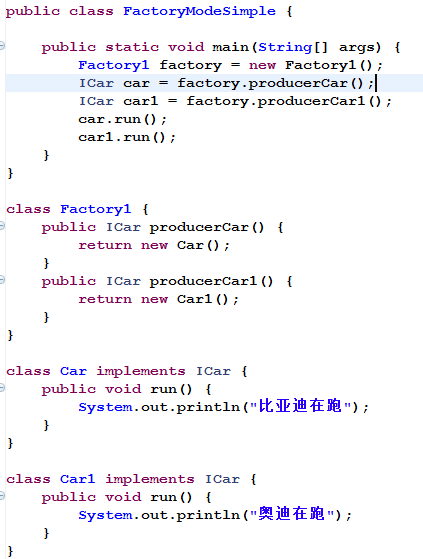
3、spring容器中，每个bean默认是单例的

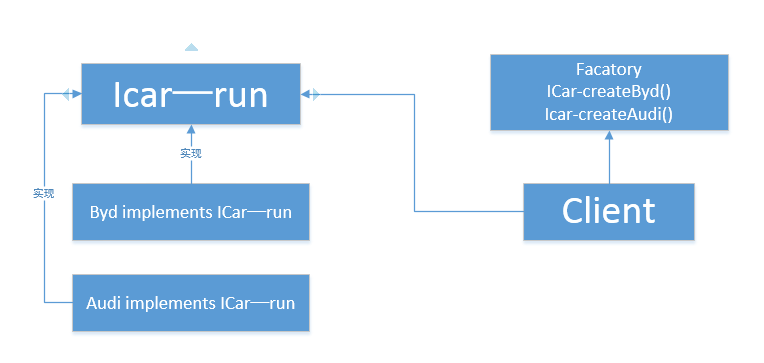
4、springmvc中的Controller方法默认是单例，可以通过注解@Scope(“”)设置成多列

### 工厂模式

#### 实例

1. 简单工厂模式-增加了一个对象需要修改代码，违背了开闭原则



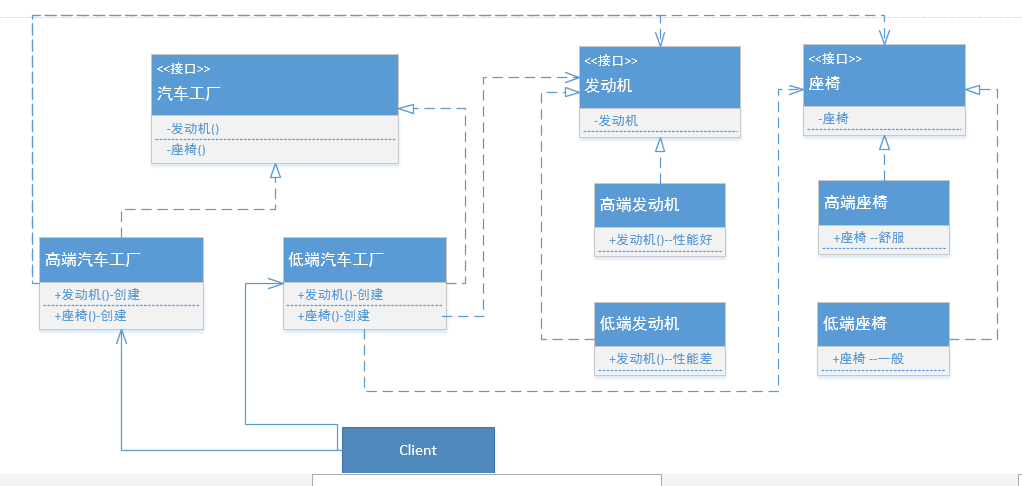


1. 工厂方法模式-一个对象一个工厂





1. 抽象工厂模式-生产不同产品



有一个抽象汽车工厂，下面有两个具体的汽车工厂（高端、低端）

有一个发动机接口

发动机下面分为高端发动机和低端发动机

有一个座椅接口

座椅下面分为高端座椅和低端座椅

#### 应用场景

1. spring中IOC容器创建管理Bean对象
2. 只要文件名以Factory结尾的一般都是用工厂模式

### 适配器模式

### 代理模式-抽象接口角色、代理角色、代理目标角色

#### 静态代理

#### 动态代理



1、jdk的动态代理，主要是实现InvocationHandler类，重写Invoke()方法

2、cglib动态代理

#### 应用场景

**SpringAOP，**

**1、在方法执行前进行相关操作，比如**

**（记录用户信息（session、Cookie、memcache、redis）、创建事务、数据检验（必填、格式、签名）、日志记录、权限限制、异常处理）**

**2、方法执行中，进行相关操作，比如**

**进行业务逻辑处理**

**3、在方法执行后进行相关操作，比如:关闭连接、清理缓存、提交或回滚事务、监控日志**

### 面向对象的设计原则

1. 开闭原则（OCP），对扩展开放，对修改关闭
2. 依赖倒转原则（DIP），针对接口编程，不要针对实现编程
3. 迪米特原则（LOD），只与你直接的朋友联系，不和陌生人联系

## 4、多线程相关知识点

### 1、多线程理论和知识点

#### 1、对多线程的理解

多个任务同时处理

多线程的目的是为了最大限度的利用CPU来提高程序的处理效果

**JDK1.5之后提供了java.util.Concurrent（坑抗t）包**

**静态Synchronized 方法和非静态synchronized方法锁定的对象不是同一个，静态方法属于类，所以锁定的对象是类，而非静态的方法属于类对象，所以锁定的是类的对象**

**同步代码块，锁的是任意对象**

#### 2、线程的状态

分为四个大状态，创建-就绪（启动线程）-运行（获取线程锁）-阻塞（睡眠、同步、等待）-结束

Sleep:线程处于睡眠状态，不释放锁，睡眠时间过后不一定马上执行该线程，因为CPU有可能在执行其它的任务

Yield：线程让步，进入可运行池中，只针对优先级相同的线程，而且不一定能达到让步效果，CPU可能重新从可运行池中获选中它

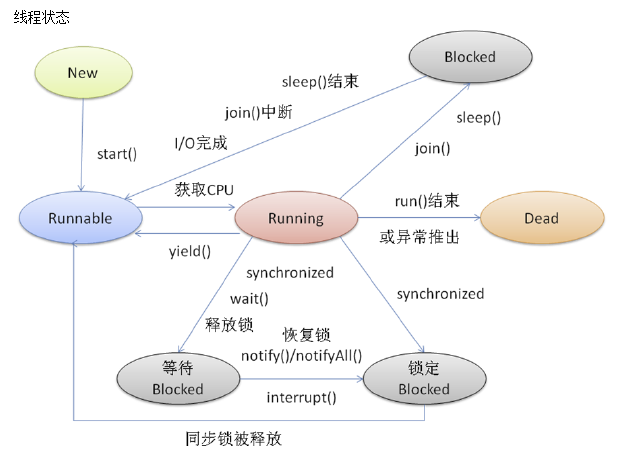
Join:使当前线程完成后在执行其它线程，主要使用的场景一般是放在子线程，比如子线程需要操作一些前置条件，这个时候需子线程先执行完

Wait:线程等待，释放线程锁，进入等待池中，等到唤醒，唤醒后进入可运行池中

Notify和notifyAll：唤醒线程或者唤醒所有的线程

Sleep与wait的区别

1. sleep来自Threrd（思维的）类，wait来自于Object
2. sleep不释放锁，wait释放锁



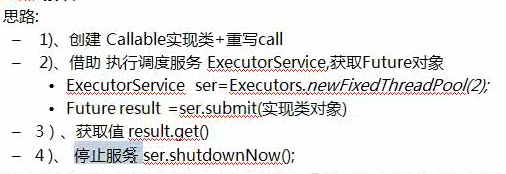
1. 创建一个线程，未启动前，状态是new
2. 启动线程，线程进入可运行状态，不一定马上执行，需要看CPU的调度
3. 线程获取CPU资源，进入运行状态
4. 线程经过sleep、wait、join、synchronized会进入阻塞状态，

调用sleep、join，线程不释放锁，会使线程进入阻塞状态，sleep睡眠时间到了，join完成该线程进入可运行状态

调用wait，线程释放锁，进入等待池中，通过notify或者notifyAll唤醒，该线程进入同步锁中

#### 3、创建线程有哪几种方式

1. 实现Runable接口，实现run方法
2. 继承Threrd类，重写run方法
3. 实现Callable接口，然后与Executor和Future结合，如下



Ps：在创建线程的时候可以为线程指定名称，会默认生成一个名称，使用哪种方式根据场景来决定，如果该类已经有继承类则需要使用实现Runable接口来创建线程

**在java中每个程序至少启动了2个线程，一个是main主线程，一个是垃圾收集线程（优先级低，需要触发相关条件才会执行）**

Thread和Runnable的区别？

1. Runnable避免了java的单继承限制
2. 线程池中只能放入实现了Runnable或者Callable类的线程，不能直接放入继承了Thread类的线程

#### 4、同步方法和同步代码块的区别

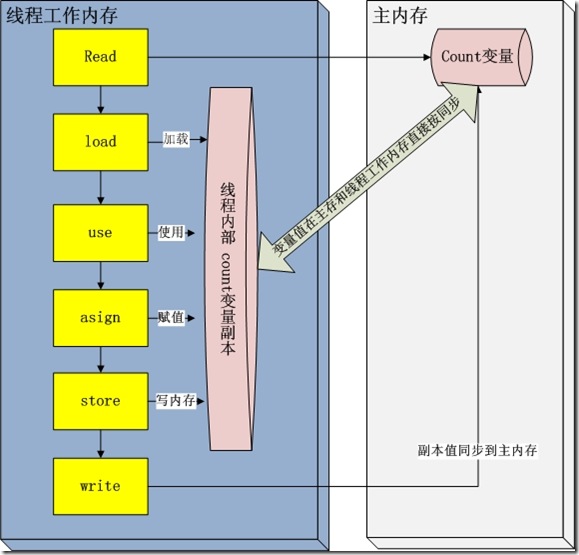
每个对象都有一个代码锁，线程可以使用关键词synchronized来获取对象上的锁，使用在方法上就是同步方法，使用在代码块上就是同步代码块

**sychronized关键字只和一个对象实例绑定**

### 2、多线程高级

#### 1、volatile 关键字

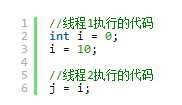
Volatile一般是和内存模型挂钩，下面 是普通线程操作的一个步骤，如下图：



CPU中有一个高速缓存的概率，也就是说程序在运行过程中，除了第一读取与内存交互，之后的都会从缓存区中获取数据，程序执行完后将数据写入到内存，在多线程下会出现一个致命问题：缓存不一致的问题

假设执行线程1的是CPU1，执行线程2的是CPU2，当线程1执行i=10这句的时候，会先把i的初始值加入到CPU1的缓存中，然后将10赋值给i，但并不会写入到主内存

当线程2执行的是，将i值赋值给j，因为CPU2缓存中并没有i的值，所以会去直接读取内存中的值，也就是0，而不是10



通过Volatile可以解决这个问题

2、

## 5、web项目的性能优化

项目优化总共分为两个大的方向，浏览器端和服务器端

### 浏览器端

1. 页面静态化、有些场景下css放在头部，js放在底部
2. 对上传的图片进行压缩，展示合理的尺寸（又拍云）
3. 减少http请求，比如session、cookie或者memcache框架保存用户的基本信息，下次登录不需要访问请求数据层

### 服务器端

1、合理使用缓存框架，

2、减少对数据库的请求，对于复杂的SQL语句，可以使用存储过程来执行

3、控制文件上传的大小

4、数据查询较多的进行分页，不要查询所有字段

5、借助jproFiter性能分析工具，找到代码性能损耗的瓶颈

6、分布式部署，使用nginx进行负载均衡，来统一输出路径分发到不同的服务器

7、业务分离

## 6、分布式项目如何实现session共享

网上有很多种实现方式

1. nginx+redis+session
2. nginx+tomcat+session
3. 直接用tomcat本身内部的一些机制然后加上一些配置文件

### 服务器层来解决

<http://blog.sina.com.cn/s/blog_8022ad020102we7h.html>

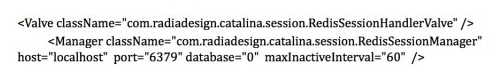
之前跟团游后台管理系统是使用tomcat+redis来处理session共享问题的

1、需要导入几个jar，其中有一个就是tomcat-redis-session-manager

commons-pool2-2.4.2.jar、jedis-2.6.1.jar、tomcat-redis-session-manager.jar

将上面的包复制到tomcat的lib下面

2、修改tomcat目录下conf目录下的context.xml，增加配置文件



大致的实现原理是：

用户发送请求，服务器先解析请求中获取到的sessionid，然后将sessionid存在request的参数列表中，然后在从reqeust中获取session的信息，如果存在sessionid，直接从session池中获取，如果不存在sessionid或者session失效，则新建一个session，并且将信息存入到session池中，供下次使用。

### 代码层来解决

1、配置一个filter（非二T）对需要共享的数据存入到缓存，第二次访问直接读取，验证通过正常访问

2、如果用户数量少的情况下都可以直接存入数据库

### Session的原理

Session是一个会话，是浏览器与服务器交互的一个重要桥梁

http请求是无状态的，通过session或者Cookie来保存用户浏览器与服务器之间的一个连接来达到有状态的场景

用户浏览器向服务器发送请求，服务器通过request.getSession()方法获取一个session对象，这个对象中有一个唯一的sessionid，然后将以sessionId为键，session对象为值存入Map集合中，最后将sessionId以Cookie的形式返回给客户端，存放在客户端浏览器的缓存中，当客户端再次请求时会将sessionId以请求头的方式发送给服务器，服务器根据sessionId获取对象来跟踪用户的相关信息

### Cookie的原理

用户客户端向服务器发送请求，服务器调用response.addCookie()方法，响应的时候，会产生一个set-cookie响应头，将Cookie以文本得形式发送给客户端，客户端获取Cookie（比如jsp通过request.getCookies()方法获取Cookie），当客户端再次向服务器发送请求时，会产生一个Cookie请求头，服务器获取Cookie后，就可以定位到用户的相关信息

Cookie的分类：

1. 存放在浏览器缓存中，浏览器在不关闭前，Cookie对象一直存在，当浏览器关闭后，Cookie对象消失
2. 存放在客户端的文件中，并且设置过期时间，在时间未过期前，用户关闭浏览器，在此打开重新请求，任然能降Cookie信息发送到服务，时间过期后，Cookie自动消失

Ps：因为Cookie是以文件的形式存放在用户客户端，所以会有安全隐患，可以通过加密的方式来进行防范，后者不要将隐秘的信息存入到Cookie中

Cookie禁用的场景下，使用哪些方式来解决session

**URL路径重写**，将sessionid与用户的请求地址绑定，下次请求时，服务器会从URL中找是否有sessionid的cookie

### Session和Cookie的区别

相同点：

1. 都能存储信息，都可以解决http无状态下引发的一些问题
2. 都可以设置过期时间

不同点：

1. session是存放在服务器中，安全性高，Cookie是存放在客户端中，安全性较低
2. 因为session是保存在服务器中，当用户的访问量增多，会占用一定的性能
3. 单个Cookie保存的信息不能超过4K，不同的浏览器支持的Cookie数稍有不同

## 7、前后端如何防范用户的重复提交

### 前端-通过js来判断

1. 用户点击完提交按钮后，置灰按钮
2. 设置一个变量，提交完一次后，将变量值更改为true

### 后端-令牌机制(token)

1. 用户请求到服务器时，通过服务器创建的session生成一个唯一的token，然后返回给前端，下次请求时，两者进行比较
2. Springmvc,在配置文件中配置一个拦截器，拦截需要检验的请求，用户访问后，进入拦截器，生成一个token令牌，存入到redis或者memcache中，将token返回给前端，放到隐藏域中，下次请求时，将token发送到拦截器进行判断

# Io流

## Io流分为两大类

1. 按操作数据类型，可以分为字符流和字节流
2. 按流向分，可以分为输入流和输出流

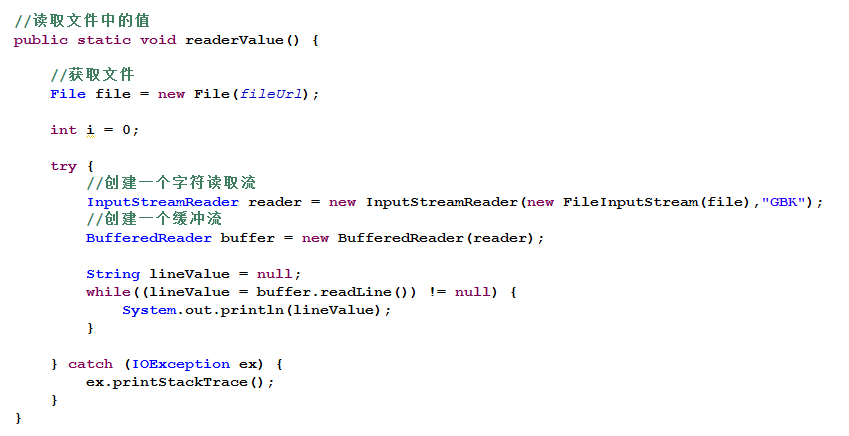
字符流：Reader、Writer

字节流：InputStream、OutPutStream

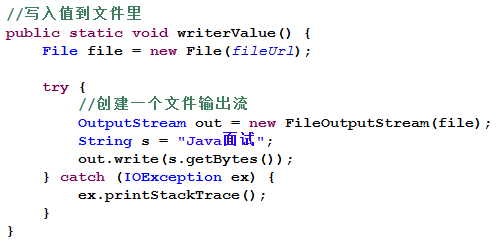


## 实战

### 1、读取文件中的值（字符读取）



### 2、写入到文件



### 3、将图片文件复制到指定路径

**public static void** writeImage(File sourceFile,String saveImagePath) {  
  
 *//创建一个输入流* FileInputStream fis = **null**;  
 FileOutputStream fos = **null**;  
  
 **try** {  
  
 *//创建一个输入流* fis = **new** FileInputStream(sourceFile);  
  
 *//创建一个输出流* fos = **new** FileOutputStream(saveImagePath);  
  
 **byte**[] b = **new byte**[3073];  
  
 **int** len = 0;  
 **while**((len = fis.read(b)) != -1) {  
 fos.write(b,0,len);  
 }  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 } **finally** {  
 **try** {  
 fis.close();  
 fos.close();  
 } **catch** (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
}

# JavaWeb

## 1、Session和cookie的区别，session的生命周期，

1. cookie数据是存放在客户浏览器中，session数据是存放在服务器中
2. cookie是保存在内存中，关闭浏览器cookie消失，但如果设置了过期时间，就会将cookie保存到硬盘上，关闭浏览器下次再打开依然有效
3. 当用户访问服务器，服务器先判断这个请求是否已经包含sessionid，如果有没有，新建一个sessionid
4. Session是会占用服务器内存的，如果访问量大的的情况下，是会影响系统的性能，如果

## 2、servlet的生命周期

初始化、响应客户请求、销毁

Servlet的请求流程

用户发送servlet请求，进入service（）方法，根据用户设置的请求方式跳转到doget或者dopost方法中，进行业务逻辑的判断，处理完之后响应客户（重定向或者跳转）

## 3、servlet与jsp的区别

1、servlet和jsp都可以生成动态的网页

2、servlet是java语言，擅长处理业务流程，缺点是生成动态网页不直观，不容易调试

3、jsp擅长制作动态网页，生成网页直观，缺点是不易调试，只能逐行逐行定位

## 4、jdbc连接步骤

1、加载数据库驱动

2、创建数据连接

3、打开连接

4、创建一个Statement   （s德t梦的）

5、执行SQL

6、处理获取的结果

7、关闭连接

## 5、web程序中web.xml的作用

一般看一个项目都是先从web.xml中开始i

比如

1、项目名称、监听器、过滤器、spring配置文件、全局变量的设置、编码的设置、servlet的访问配置、struts的访问配置

## 6、jsp的内置函数

Request：请求对象

Response：响应对象

Session 会话对象

Application 应用程序对象

Exception 错误对象

Out 输出对象

Config：配置对象

pageContext：页面上下文对象

# 框架

## 1、Hibernate

## 2、mybatis

Mybatis的实现原理

Mybatis根据XML配置文件创建SqlSesionFactory，然后SqlSessionFatory在根据配置获取一个SqlSession，在这个里面有操作数据库的所有方法，然后通过SqlSession实例直接调用映射的SQL语句，完成对数据的操作和事务提交，用完之后关闭SqlSession

Mybatis的优缺点

优点：

1. 简单易学，容易上手
2. 实现了SQL语句与程序代码的分离
3. 提供了丰富的标签，使程序操作数据库更加方法
4. 灵活，对需求变化大的项目特别适用

缺点：

1. 编写SQL语句工作量大，尤其是在对象大，字段比较多的情况下，很容易出错
2. SQL语句依赖数据库，导致数据库的移植性差，不能更改数据库

3、SQL语句全是自己编写，对后期维护比较麻烦

### 1、#{}和${}的区别

#传递的参数，mybatis会将它转换成String类型

$传递的参数，mybait不会做任何转换，容易造成SQL的注入

### 2、xml映射中，除了常见的CRUD标签之外，还有哪些标签分别有什么作用

<select>、<insert>、<update>、<delete>

Select查询中可能需要参数< parameterMap >、返回值：resultMap

返回主键<selectKey><sql><include>

动态SQL条件判断语句<where><if><foreach>等

### 3、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，会重载吗？除了接口方式与xml交互还有哪种方式可以

Dao接口就是Mapper接口，接口的全路径名名称就是映射文件中的namespace的值，接口的方法名要与映射文件中的sqlID保持一致，服务层在调用dao接口时，mybatis会自动根据接口名+方法名定位到映射文件中具体的执行SQL

不会重载

还可以使用sqlSession来与映射文件交互，第一个参数是映射文件中的namespace的值.要执行的SQLId,第二参数是传递的参数

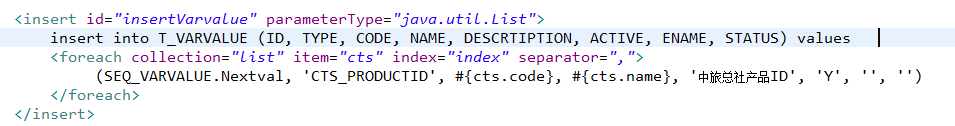
### 4、有哪两种like方法

1、第一种是直接在SQL语句中拼接，使用$

2、第二种是在java代码中拼接，以参数的形式来传递,使用#

### 5、如何执行批量插入

Mysql的批量新增最简单，如下：



Oracle执行这个SQL语句会抛出错误

### 6、如果获取自动生成的主键

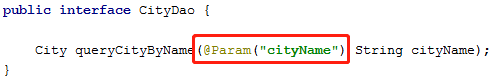
在插入语句中，可以配置相关属性来获取主键值

### 7、传参

1、单参数

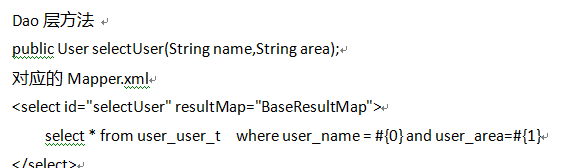
注意点：如果需要对单参数进行判断，不能直接写参数的名称，而应该是\_paramter，在3.2版本中会出现这个错误

或者在dao层使用注解



2、多参数

1、使用占位符

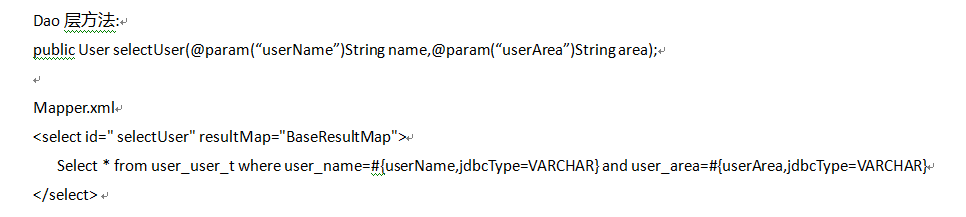


2、使用集合

如果是单个参数，Array、List、Map都可以

如果参数中存储的是对象，使用List，将对象存入List集合中，使用迭代器进行输出

1. 使用注解



### 8、mybatis能执行一对一、一对多、多对一、多对多的关联查询吗

能，可以使用selectOne（）和selectList（）

### 10、mybatis实战

#### 10.1、配置文件详解

##命名空间主要是dao层地址

<**mapper namespace="dao.CityDao"**>

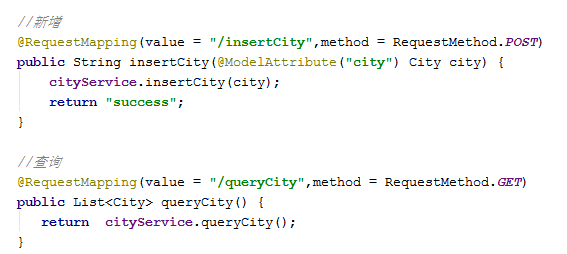
<**select id="queryCityByName" resultMap="cityInfo" parameterType="" parameterMap=""**></**select**>

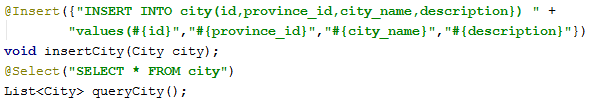
**Mybatis中resultType已经不用，使用resultMap**

**单个单数使用parameterType，多个参数或者集合使用parameterMap**

#### 10.2、注解形式来操作SQL

<http://blog.csdn.net/abc709272013/article/details/52609310>





### 11、mybatis与ibatis的区别

最大的变化就是与映射文件的交互

1. mybatis直接写一个Dao接口，在写一个对应的映射文件就可以执行，ibatis需要些Dao实现类
2. mybatis的核心处理类是SqlSession，ibatis是使用SqlMapClient

### 12、hibernate跟Mybatis 的区别，为什么选择？

1. Hibernate功能强大，与数据库无关性好，数据库移植性好，mybatis对数据库的依懒性强，不同的数据库写不同的SQL，移植性差
2. Hibernate在查询上需要返回所有的表字段属性，mybatis能指定返回字段
3. Hibernate门槛较高，mybaits容易上手，灵活

选择哪种开源框架主要是根据项目的需求来决定，如果需求变化大，复杂SQL比较多，使用mybatis开发会快，且容易管理，但是如果只是相对简单的SQL，可以使用hibernate，因为框架都已经帮你封装好了

## 3、spring

### Spring的原理

#### 1、主要体现在AOP和IOC

##### 1.1、IOC

控制反转，主要的实现方式有两种：1、依赖注入（DI） 2、依赖查找

更多的是采用依赖注入，注入的方法有3种(set、构造器、接口)

好处是：

1. 对象交给容器来管理
2. 降低了对象与对象之间的依赖关系，也就是耦合度

##### 1.2、AOP：面向切面编程

包含的关键点（方面、连接点、切入点、通知、引入、目标对象、织入）

主要是在处理业务逻辑的前后增加相关的操作（事务管理、权限控制、异常处理、懒加载、日志记录、数据检验、持久化）

### Spring事务

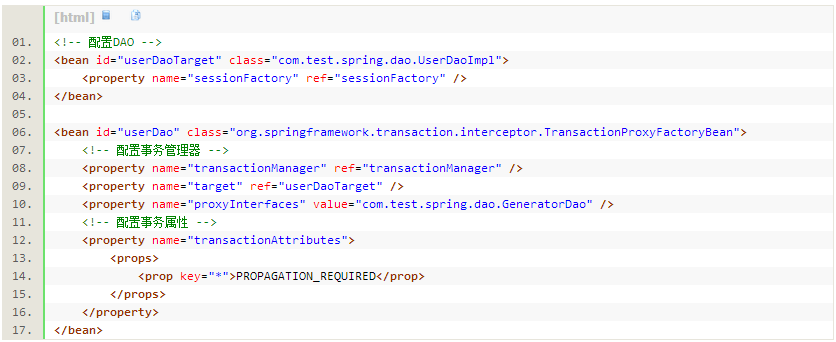
#### 1、声明式事务

声明式事务主要是建立在AOP的基础之上的，本质是在执行方法前，创建事务或者加入一个事务，在方法执行完后，根据执行结果进行事务的提交或者回滚

声明式事务最大的一个优点就是不需要通过编程式事务来管理事务，这样也避免了在业务逻辑代码中编写事务相关的代码，只需要在配置文件中配置或者在具体需要使用的类，方法上添加注解形式的事务

5种配置方式

1. 每一个bean都有一个事务代理



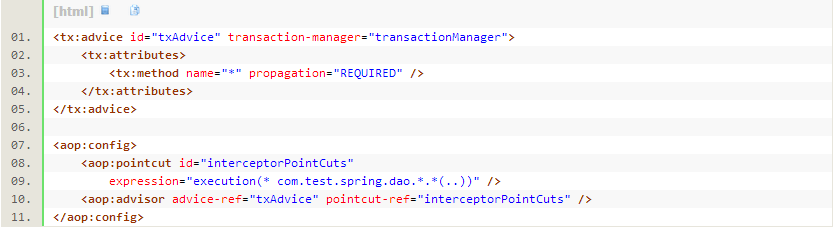
1. 所有bean都有一个事务代理



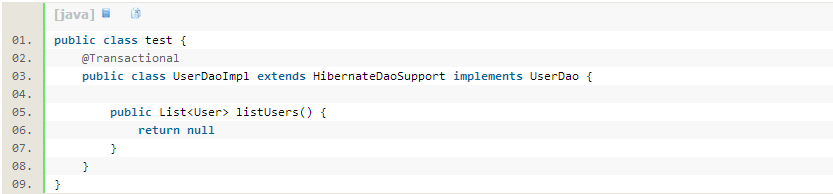
1. 使用拦截器



1. 使用tx标签配置，哪些类，哪些方法需要事务



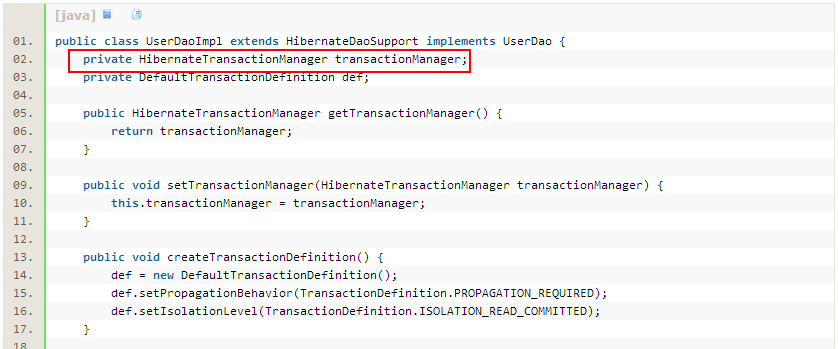
1. 注解



#### 2、编程式事务

编程式事务有两种方式

1. 使用TransactionManager(穿撒少英买了狗)-事务管理



1. 使用TransactionTemplate(穿撒少英疼普列特)-事务模板



#### 3、spring事务的特性、传播属性和隔离级别

##### 4种特性

1. 原子性：包含的事务要么全部成功，要么全部回滚
2. 一致性：事务执行前和执行后都必须保证数据的一致性状态，

举例:银行转账，A账号和B账号两者的钱都是1000元，那不论怎么操作，最后两者的钱+起来还是1000

1. 隔离性：当前事务在操作数据之前，其它事务不能干扰
2. 持久性：事务提交后，数据永久的写入了数据库

##### 6种传播属性

PROPAGATION\_REQUIRED：支持当前事务，如果当前没有事务，新建一个事务

PROPAGATION\_SUPPORTS：支持当前事务，如果当前没有事务，就以非事务去执行

PROPAGATION\_MANDATORY：支持当前事务，如果当前没有事务，抛出异常

PROPAGATION\_REQUIRES\_NEW：新建事务，如果当前事务存在，则将它挂起

PROPAGATION\_NOT\_SUPPORTED：以非事务方式执行，如果当前事务存在，则将它挂起

PROPAGATION\_NEVER：以非事务方式执行，如果当前事务存在，则抛出异常

##### 5种隔离级别

1、ISOLATION\_DEFAULT：默认的隔离级别，使用数据库默认的隔离级别

2、ISOLATION\_READ\_UNCOMMITTED：这是事务最低的一种隔离级别，允许另一个事务查看当前事务未提交的数据，容易造成数据的脏读和不可重复性读

3、ISOLATION\_READ\_COMMITTED：保证一个事务修改的数据提交后才能被另一个事务读取

4、ISOLATION\_REPEATABLE\_READ：保证一个事务不能读取另一个未提交事务的数据，还保证了避免出现不可重复读

5、ISOLATION\_SERIALIZABLE ：使事务按照顺序执行，防止了脏读，不可重复度，和幻像读

什么是脏读，什么是不可重复性读，幻读

当前事务在未提交数据修改前，被另一个事务访问了

在同一个事务中，第一次读取的数据和第二次读取的数据不一致

### Spring实战

#### 1、Spring定时任务

<http://cron.qqe2.com/>

#### 2、Bean的作用

注解分为两类

1. 一类是使用bean

把已经配置好的bean拿来使用，比如@Autowired(噢T乱D)、@Resources，分别根据byType和byName来获取bean

1. 一类是注册bean，比如@Component(啃破了特)、@Repository(r破自 ture) 、@Controller、@Service、@Configration(啃飞哥red 小)这些注解都是将要实例化的对象转换成一个Bean放入到IOC容器中

## 4、springmvc

Springmvc默认是单例的，Controller中不适合定义成员变量，如果一定要使用，使用@Scope(“prototype”)标明为多例

### 1、Springmvc的工作原理

1. 用户发送请求，进入前端控制器
2. 前端控制收到请求后调用HandlerMapping处理器映射器，返回一个具体的处理器
3. 前端控制器根据处理器调用HandlerAdapter适配器
4. 适配器在调用具体的Handler处理器，也就是Controller方法
5. 在Controller中完成业务逻辑的操作，返回一个ModelAndView
6. 适配器将Controller的执行结果返回给前端控制器
7. 前端控制器调用视图解析器，将ModelAndView解析成真正的view视图
8. 前端控制器将视图进行渲染（数据填充），响应客户

### 2、Springmvc和struts2的区别

1. springmvc的请求入口是servlet，struts2的请求入口是一个过滤器
2. springmvc是基于方法，默认是单例，可以通过注解设置成多例，struts2是基于类，每一次请求都会实例化一个Action对象，所以默认是多例
3. springmvc支持jsr303，处理Ajax请求更加方法，只需要一个注解就可以返回结果文本
4. Struts2有自己的拦截器，SpringMvc用的独立的AOP动态代理

### 3、springmvc中的类一般是用哪个注解来表示控制器

@Controller来表示该是控制器

### 4、springmvc中的requestMapping用在类上和方法上分别代表什么

一个是作用于整个类，一个是作用于具体的某个方法

标注在类上，表示类中所有的请求URL都是以这个为父路径

### 5、在controller中获取参数的方式有哪几种

总共有两种

1. Request获取
2. 通过注解@requestParam

### 6、怎么样把某个请求映射到特定的方法上面

在指定的方法上使用注解@RqeustMapping，在设置拦截的URL

### 7、如果在拦截请求中,我想拦截get方式提交的方法,怎么配置

在@RequestMapping的注解中加上method=RequestMethod.GET

### 8、如果在拦截请求中,我想拦截提交参数中包含"type=test"字符串,怎么配置

直接在注解RqeustMapping的值中写上包含的参数

### 9、拦截的方法里面得到从前台传入的参数,怎么得到

直接在形参中声明这个方法就可以，参数名称与传入的参数名称保持一致

### 10、如果前台有很多个参数传入,并且这些参数都是一个对象的,那么怎么样快速得到这个对象

直接在方法中声明这个对象，springmvc会自动将前端参数赋值给这个声明的对象

### 11、SpringMvc中函数的返回值是什么.

String和modelandview

### 12、Springmvc如何处理重定向和跳转的

在方法的路径前面加上forward，跳转

加上redirect，表示重定向

### 13、SpringMvc用什么对象从后台向前台传递数据的

通过ModelMap对象，前端使用el表达式直接获取

### 14、springmvc如何与ajax交互

通过jackson框架，直接把java对象转换成js能识别的json对象

### 15、@ComponentScan、@ Configuration的作用

Springmvc除了本身的配置文件外，几乎可以实现全注解的形式开发

**@ComponentScan就是告诉spring哪些packages中有注解的类，然后会被spring自动扫描并且装载到bean容器，而不需要再spring文件中去配置**

比如，你有一个类，使用了@Controller，如果你不在配置文件中扫描到这个类，spring容器不会扫描到这个类，这个时候在@Controller上使用该@ComponentScan注解，就能让spring自动扫描到

**@Configuration表示这个类中定义的Bean会装载到Spring容器中，等同于XML中的配置**

### 16、spring mvc与struts的区别

1、springmvc的前端控制器是servlet，struts2的是Fliter

2、springmvc是方法级别的拦截，一个方法对应一个reqeust和response，struts2是类级别的拦截，一个类对应一个reqeust和respose

3、springmvc默认是单例，Struts2默认是多例，都可以通过注释来更改他们的默认

Springmvc是在类或者方法上使用注释@Scope(“prototype”)

Struts2是在每个bean的配置文件中增加scope="prototype"



1. springmvc支持JSR303，在数据验证和与前端Ajax交互上很方便

17、springmvc常用的注解

<https://www.cnblogs.com/leskang/p/5445698.html>

## 微服务开发-spring boot和spring cloud

### 1、springboot

<https://www.zhihu.com/question/53729800>

<https://gitee.com/didispace/SpringBoot-Learning>

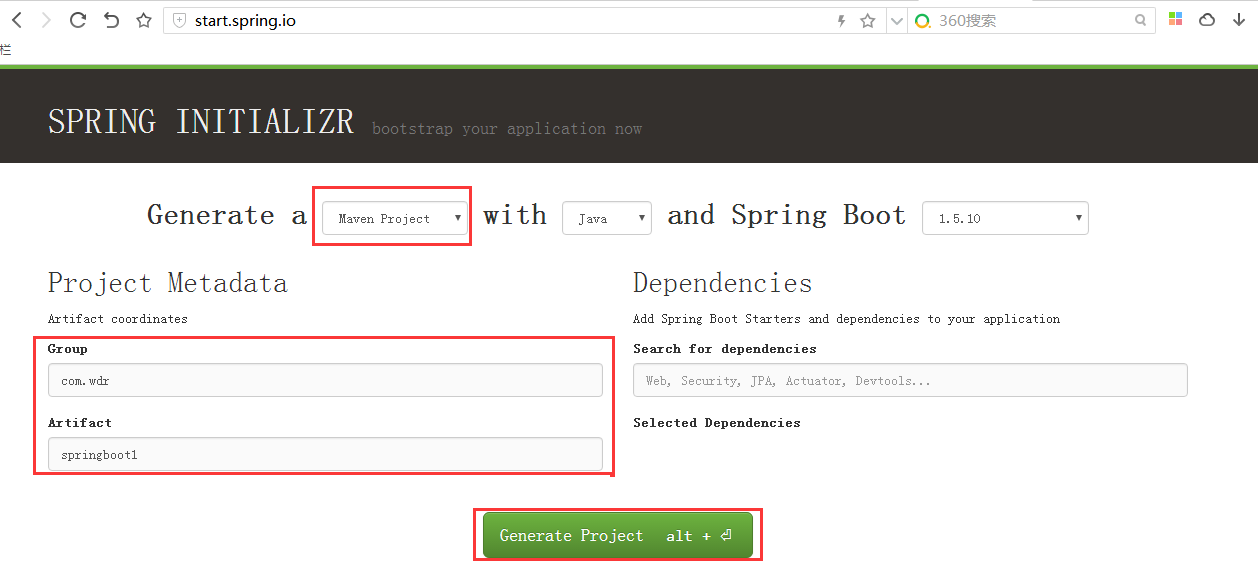
<http://springboot.fun/>

<http://springcloud.fun/>

#### 1.1、springboot知识点

##### 1.1.1、通过spring initializr生成springboot项目

<http://start.spring.io/>



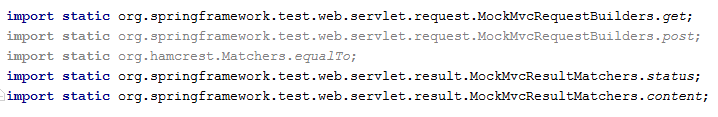
##### 1.1.2、springboot如何在类中模拟http请求测试

@RunWith(SpringJUnit4ClassRunner.**class**)  
@SpringBootTest(classes = MockServletContext.**class**)//新版用这个，老版用@springapplicationconfiguration这个注解  
@WebAppConfiguration



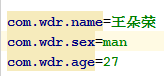
@Test  
**public void** testResult() **throws** Exception {  
  
 RequestBuilder request = **null**;  
  
 *//获取用户列表* request = *get*(**"/users/"**);  
 **mockMvc**.perform(request)  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().string(*equalTo*(**"[]"**)));  
  
 *//创建一个用户* request = *post*(**"/users/"**)  
 .param(**"id"**,**"1"**)  
 .param(**"name"**,**"wangduorong"**)  
 .param(**"age"**,**"27"**);  
 **mockMvc**.perform(request)  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().string(*equalTo*(**"success"**)));  
  
 *//根据指定id获取用户信息* request = *get*(**"/users/1"**);  
 **mockMvc**.perform(request).andExpect(*status*().isOk())  
 .andExpect(*content*()  
 .string(*equalTo*(**"{\"id\":1,\"name\":\"wangduorong\",\"age\":27}"**)));  
  
 *//根据指定id更新对应的用户信息* request = *put*(**"/users/1"**)  
 .param(**"name"**,**"zenglingzhi"**)  
 .param(**"age"**,**"28"**);  
 **mockMvc**.perform(request)  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().string(*equalTo*(**"success"**)));  
  
 *//删除指定ｉｄ的用户信息* request = *delete*(**"/users/1"**);  
 **mockMvc**.perform(request)  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().string(*equalTo*(**"success"**)));  
}

在测试中可能需要用到的方法，需要手动导入jar包

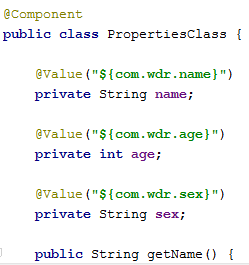


##### 1.1.3、如何读取配置文件中的值

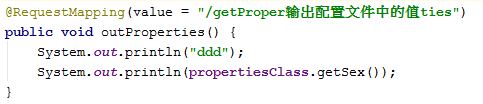
1、在配置文件中定义配置信息

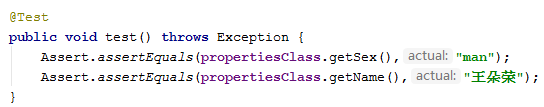


2、创建一个读取配置文件信息的类



3、在测试类或者控制器中获取指定的配置信息

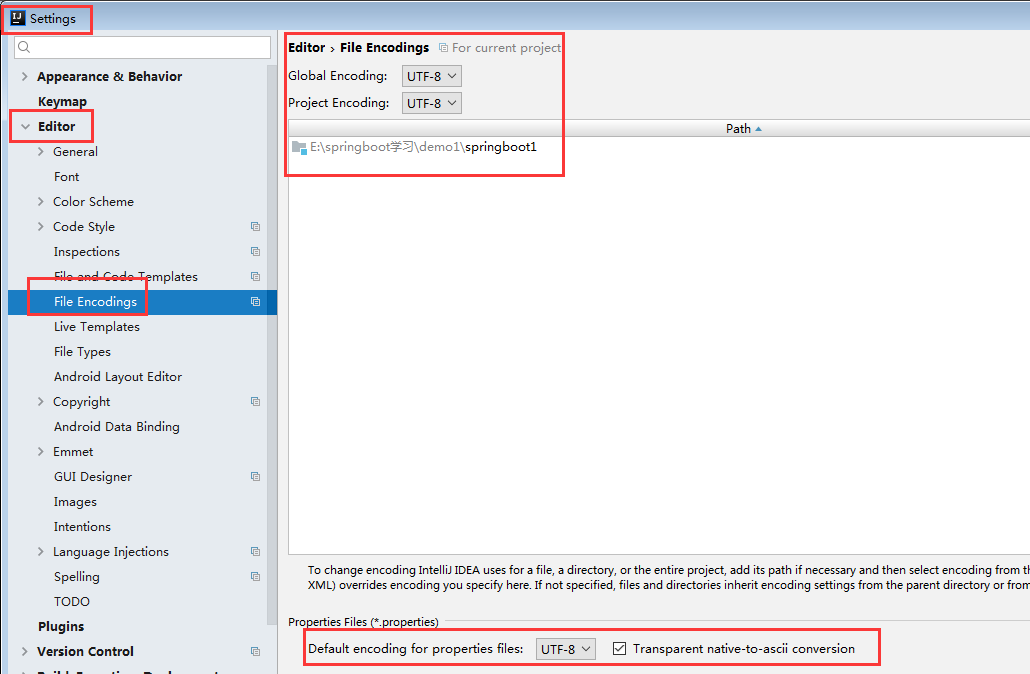




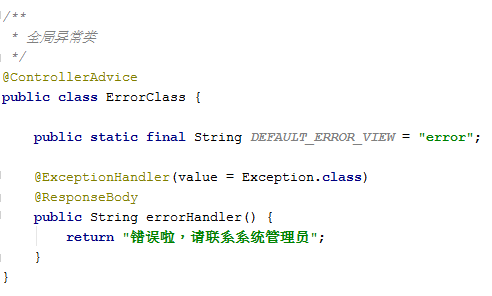
Ps：在读取配置文件时，有可能会出现中文乱码，解决方法有两种

1、在配置文件最上面增加**spring.http.encoding.charset**=**UTF-8**

2、将配置文件的编码格式修改成UTF-8

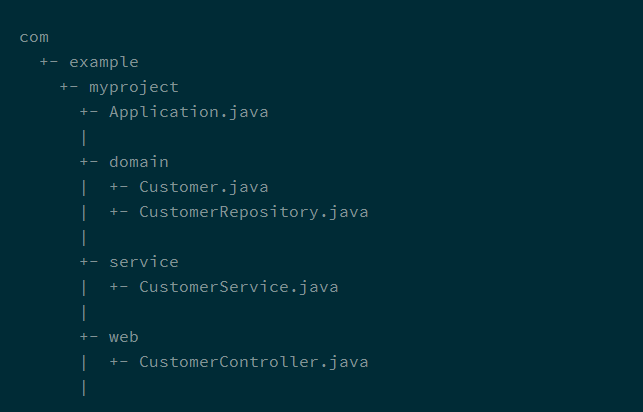


##### 1.1.4、全局捕获异常



##### 1.1.5、springboot工程目录结构

<http://blog.didispace.com/springbootproject/>



##### 1.1.6、springboot配置文件详解

<https://www.bysocket.com/?p=1786>

<https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>

##### 1.1.7、如何修改springboot端口号

Springboot默认端口是8080

在application.propertie或者application.yml中加上如下语句

server.port=9090（端口号）

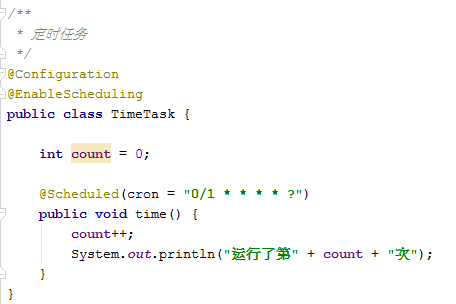
##### 1.1.8、ComponentScan和MapperScan的区别

两者都是程序在启动时要扫描的包，避免出现bean找不到

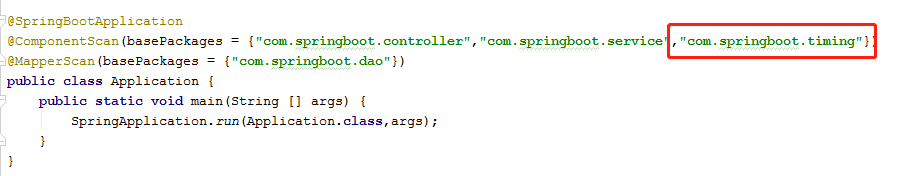
第一个主要是针对controller和service层

MapperScan主要是针对dao层

##### 1.1.9、springboot启动定时器



Ps:需要注意的时，在启动应用程序的时候，需要将这个类对应的包放入到扫描中



##### 1.1.10、springboot打包

1、使用mvn clean package

2、进入target目录，输入java –jar 包名

#### 1.2、springboot实战

##### 1.2.1、springboot如何将图片上传到服务器，且获取到图片URL

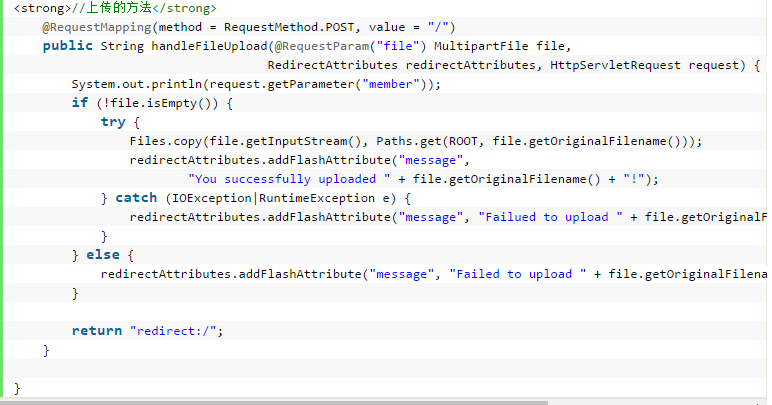
一般图片存储都时候放在特定的存储器中，或者云服务上（图片量大的情况下）

将图片上传到Linux服务器中的某一个文件夹中（少量图片下）

正常情况，外部不能直接访问到Linux中的图片，解决方案

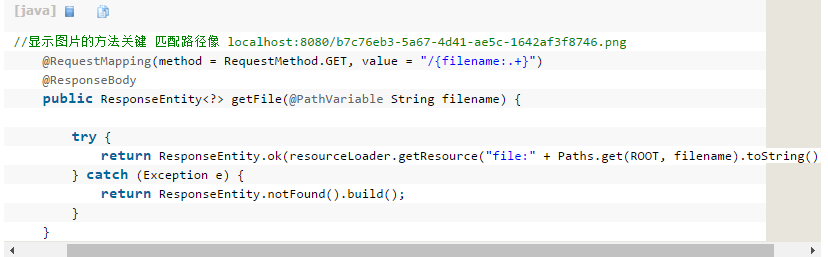
1. 在Linux服务中添加一个tomcat容器，将图片存储在tomcat容器中
2. 通过SpringBoot自带的一种方式

<http://blog.csdn.net/a625013/article/details/52414470>



1. 这个路径会在本地的工程根目录上创建，不应用部署里的目录，所以一般的访问http访问不可能 ，所以它提供了ResourceLoader,利于这个类可以加载非应用目录的里文件然后返回

通过如下方法获取图片URL



##### 1.2.2、springboot整合webservice

###### 1、使用原生的jdk注解实现webservice

<http://blog.csdn.net/u010235716/article/details/77988844>

1.1、使用注解创建一个service接口

package com.webservice.service;  
  
import javax.jws.WebMethod;  
import javax.jws.WebService;  
  
@WebService  
public interface TestService {  
 @WebMethod  
 public String yuan(String param);  
}

1.2、实现接口

package com.webservice.service.impl;  
  
import com.webservice.service.TestService;  
  
import javax.jws.WebService;  
  
*/\*\*  
 \* targetNamespace:反射后的包名空间  
 \* name：通过工厂生成一个MyTestWS实例  
 \* serviceName：实例工厂  
 \*/*@WebService(targetNamespace="http://localhost/client",name="MyTestWS",  
 serviceName="MyWebService")  
public class TestServiceImpl implements TestService {  
  
 public String yuan(String param) {  
 return param;  
 }  
}

1.3、发布服务

package com.webservice;  
  
import com.webservice.service.impl.TestServiceImpl;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.boot.builder.SpringApplicationBuilder;  
  
import javax.xml.ws.Endpoint;  
  
@SpringBootApplication  
public class Application {  
 public static void main(String[] args) {  
  
 new SpringApplicationBuilder(Application.class).web(true).run(args);  
  
 //webservice 接口暴露  
 String address = "http://localhost:8088/MyWebService";  
 //使用Endpoint类提供的publish方法发布WebService，发布时要保证使用的端口号没有被其他应用程序占用  
 Endpoint.*publish*(address, new TestServiceImpl());  
 System.*out*.println("发布webservice成功!");  
 }  
}

1.4、启动服务，访问

浏览器访问http://localhost:8088/MyWebService?wsdl

1.5、将wsdl反编译成java代码

命令：wsimport -d bin -s src <http://localhost:8088/MyWebService?wsdl>

1. wsimport -d bin -s src http://localhost:8088/MyWebService?wsdl
2. -keep：是否生成java源文件
3. -d：指定.class文件的输出目录
4. -s：指定.java文件的输出目录
5. -p：定义生成类的包名，不定义的话有默认包名
6. -verbose：在控制台显示输出信息
7. -b：指定jaxws/jaxb绑定文件或额外的schemas
8. 最后面是webservice的访问地址

实例:

在idea中的terminal中输入，如下：

wsimport -keep -d E:\springboot学习\demo1\webservice\_demo\target\classes\com\wsdl -s E:\springboot学习\demo1\webservice\_demo\src\main\java\com\webservice\wsdl -verbose <http://localhost:8088/MyWebService?wsdl>

1.6、异常处理

存在相同的response；这是自动生成时，默认构建的问题。请将文件修改为

**XmlType 的name不要和类名一样即可**

###### 2、使用springboot集成的webservice框架

<http://blog.csdn.net/u011410529/article/details/68063541?winzoom=1>

<https://www.cnblogs.com/learnhow/p/7205524.html?utm_source=itdadao&utm_medium=referral>

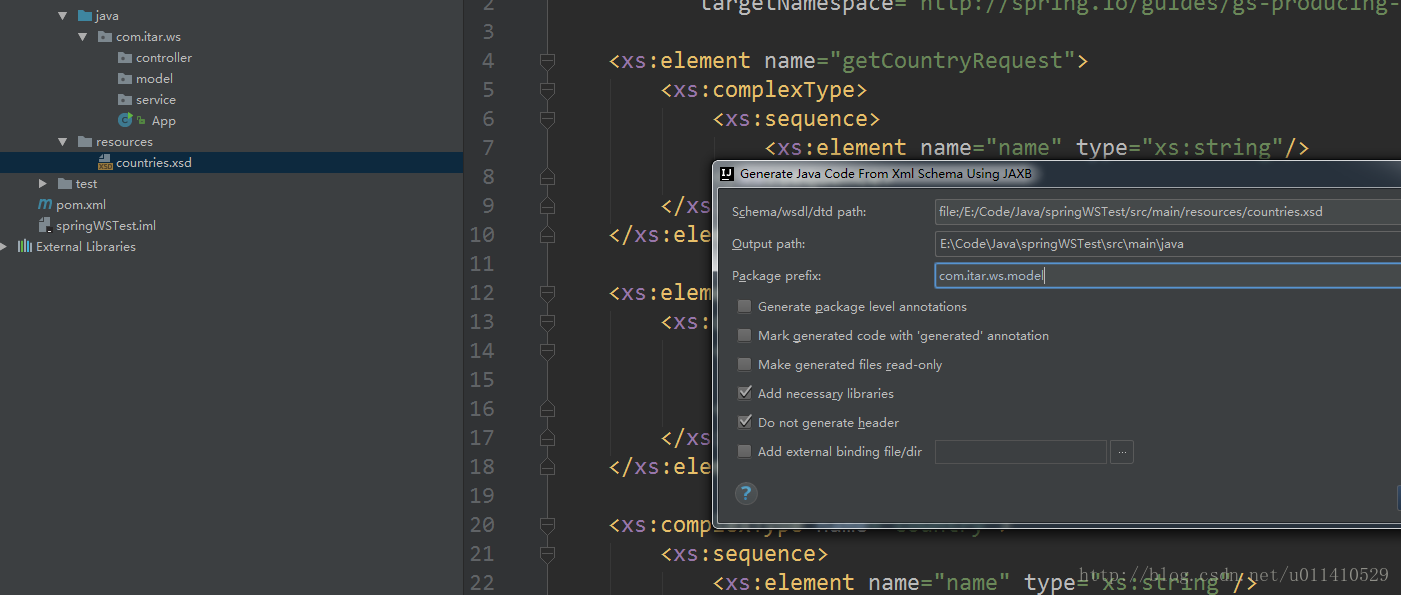
2.1、添加依赖jar

2.2、根据对象编写XSD

<http://blog.csdn.net/xumesang/article/details/50546851>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
 xmlns:tns="http://www.yourcompany.com/webservice"  
 targetNamespace="http://www.yourcompany.com/webservice"  
 elementFormDefault="qualified">  
  
 <!-- 定义一个获取Country的元素 -->  
 <xs:element name="getCountryRequest">  
 <xs:complexType>  
 <xs:sequence>  
 <xs:element name="name" type="xs:string"/>  
 </xs:sequence>  
 </xs:complexType>  
 </xs:element>  
  
 <!-- 定义一个返回Country的元素 -->  
 <xs:element name="getCountryResponse">  
 <xs:complexType>  
 <xs:sequence>  
 <xs:element name="country" type="tns:country"/>  
 </xs:sequence>  
 </xs:complexType>  
 </xs:element>  
  
 <!-- 定义复杂类型 -->  
 <xs:complexType name="country">  
 <xs:sequence>  
 <xs:element name="name" type="xs:string"/>  
 <xs:element name="population" type="xs:int"/>  
 <xs:element name="capital" type="xs:string"/>  
 <xs:element name="currency" type="tns:currency"/>  
 </xs:sequence>  
 </xs:complexType>  
  
 <!-- 定义简单类型 -->  
 <xs:simpleType name="currency">  
 <xs:restriction base="xs:string">  
 <xs:enumeration value="GBP"/>  
 <xs:enumeration value="EUR"/>  
 <xs:enumeration value="PLN"/>  
 </xs:restriction>  
 </xs:simpleType>  
</xs:schema>

2.3、根据xsd生成java代码



2.4、数据填充

package com.webservice.test.wsdl;  
  
import org.springframework.stereotype.Component;  
import org.springframework.util.Assert;  
  
import javax.annotation.PostConstruct;  
import java.util.HashMap;  
import java.util.Map;  
  
*/\*\*  
 \* 数据填充类  
 \*/*@Component  
public class CountryRepository {  
  
 private static final Map<String, Country> *countries* = new HashMap<String, Country>();  
  
 //构造函数后调用该初始方法进行数据填充  
 @PostConstruct  
 public void initData() {  
 Country spain = new Country();  
 spain.setName("Spain");  
 spain.setCapital("Madrid");  
 spain.setCurrency(Currency.*EUR*);  
 spain.setPopulation(46704314);  
  
 *countries*.put(spain.getName(), spain);  
 }  
  
 //根据名称获取对应的Country对象  
 public Country findCountry(String name) {  
 Assert.*notNull*(name, "The country's name must not be null");  
 return *countries*.get(name);  
 }  
}

ps:真实开发中，是从service中获取数据

2.5、配置对外发布的类

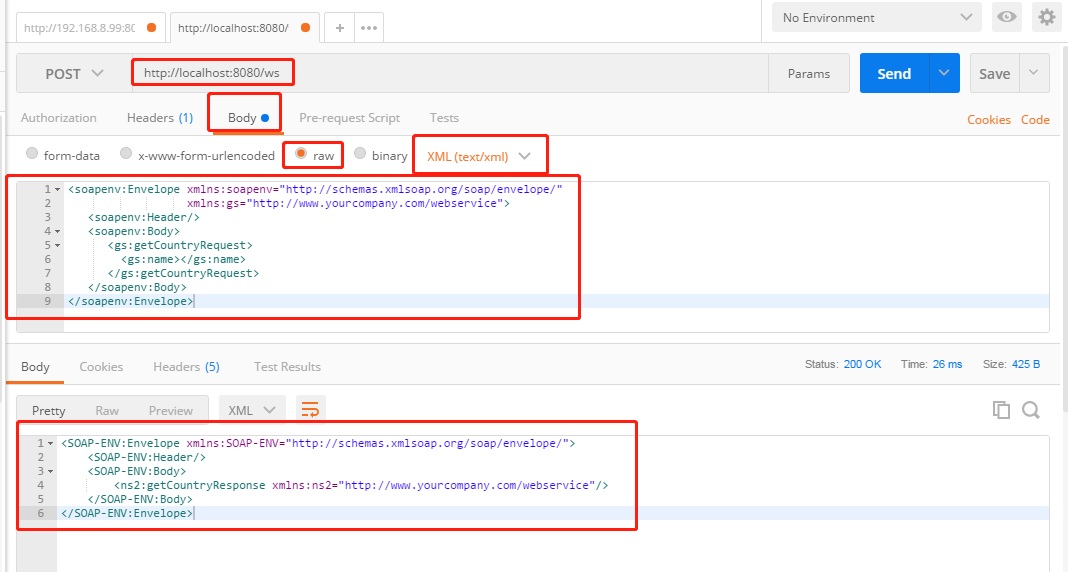
package com.webservice.test.webservice;  
  
import org.springframework.boot.web.servlet.ServletRegistrationBean;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.core.io.ClassPathResource;  
import org.springframework.ws.config.annotation.EnableWs;  
import org.springframework.ws.config.annotation.WsConfigurerAdapter;  
import org.springframework.ws.transport.http.MessageDispatcherServlet;  
import org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition;  
import org.springframework.xml.xsd.SimpleXsdSchema;  
import org.springframework.xml.xsd.XsdSchema;  
  
@Configuration  
@EnableWs  
public class WebServiceConfig extends WsConfigurerAdapter {  
  
 @Bean  
 public ServletRegistrationBean messageDispatcherServlet(ApplicationContext applicationContext) {  
 MessageDispatcherServlet servlet = new MessageDispatcherServlet();  
 servlet.setApplicationContext(applicationContext);  
 servlet.setTransformWsdlLocations(true);  
 //配置对外服务根路径  
 return new ServletRegistrationBean(servlet, "/ws/\*");  
 }  
  
 @Bean(name = "countries")  
 public DefaultWsdl11Definition defaultWsdl11Definition(XsdSchema countriesSchema) {  
 DefaultWsdl11Definition wsdl11Definition = new DefaultWsdl11Definition();  
 wsdl11Definition.setPortTypeName("CountriesPort");  
 wsdl11Definition.setLocationUri("/ws");  
 wsdl11Definition.setTargetNamespace("http://www.yourcompany.com/webservice");  
 wsdl11Definition.setSchema(countriesSchema);  
 return wsdl11Definition;  
 }  
  
 @Bean  
 public XsdSchema countriesSchema() {  
  
 return new SimpleXsdSchema(new ClassPathResource("countries.xsd"));  
 }  
}

2.6、创建对外访问的服务类

该方法本质是获取一个request，然后处理完后返回一个response，返回的是XML

package com.webservice.test.controller;  
  
import com.webservice.test.wsdl.CountryRepository;  
import com.webservice.test.wsdl.GetCountryRequest;  
import com.webservice.test.wsdl.GetCountryResponse;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.ws.server.endpoint.annotation.Endpoint;  
import org.springframework.ws.server.endpoint.annotation.PayloadRoot;  
import org.springframework.ws.server.endpoint.annotation.RequestPayload;  
import org.springframework.ws.server.endpoint.annotation.ResponsePayload;  
  
@Endpoint  
public class CountryEndpoint {  
  
 private static final String *NAMESPACE\_URI* = "http://www.yourcompany.com/webservice";  
 private CountryRepository countryRepository;  
 @Autowired  
 public CountryEndpoint(CountryRepository countryRepository) {  
 this.countryRepository = countryRepository; }  
 //配置对外接口  
 @PayloadRoot(namespace = *NAMESPACE\_URI*, localPart = "getCountryRequest")  
 @ResponsePayload  
 public GetCountryResponse getCountry(@RequestPayload GetCountryRequest request) {  
 GetCountryResponse response = new GetCountryResponse();  
 response.setCountry(countryRepository.findCountry(request.getName()));  
  
 return response;  
 }  
}

2.7、使用postman测试



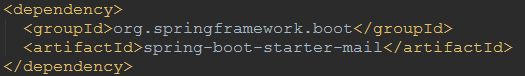
###### 3、webservice请求和返回实例

<http://eduboss.vanke.com/Peas/WebService/K2WebService.asmx>

##### 1.3、springboot发送邮件

<http://www.ityouknow.com/springboot/2017/05/06/springboot-mail.html>

###### 1.3.1、引入jar包



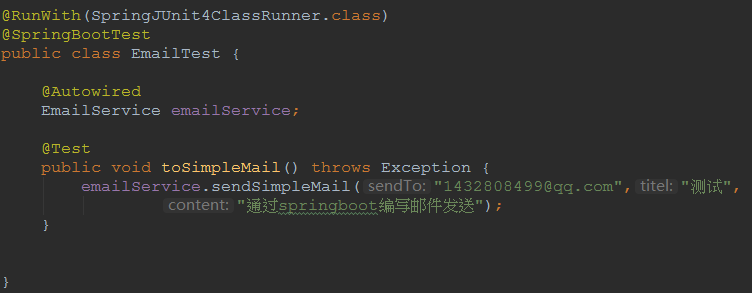
###### 1.3.2、配置连接



###### 1.3.3、编写service类简单版

package com.springboot.service.impl;  
  
import com.springboot.service.EmailService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.mail.SimpleMailMessage;  
import org.springframework.mail.javamail.JavaMailSender;  
import org.springframework.stereotype.Service;  
  
  
@Service  
public class EmailServiceImpl implements EmailService {  
  
 @Value("${spring.mail.fromMail.addr}")  
 private String email;  
  
 @Autowired  
 private JavaMailSender javaMailSender;  
  
 @Override  
 public void sendSimpleMail(String sendTo, String titel, String content) {  
  
 //实例化一个简单的消息模板  
 SimpleMailMessage sm = new SimpleMailMessage();  
  
 sm.setFrom(email);  
 sm.setTo(sendTo);  
 sm.setSubject(titel);  
 sm.setText(content);  
  
 javaMailSender.send(sm);  
 }  
}

测试类-纯中文



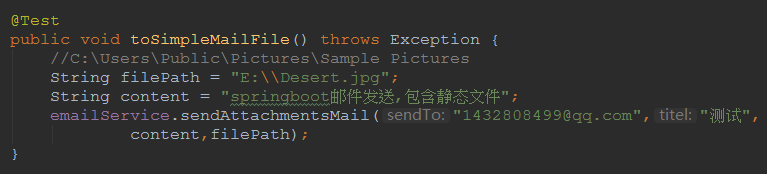
测试类-带html



###### 1.3.4、编写service-带静态文件

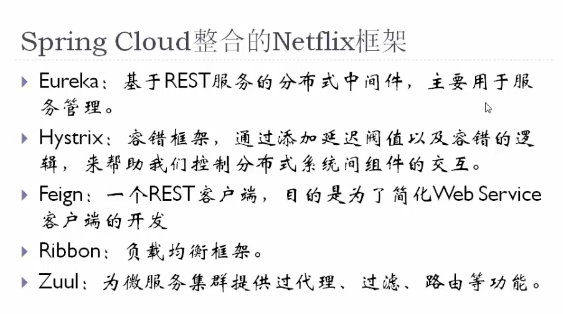
@Override  
public void sendAttachmentsMail(String sendTo, String titel, String content,  
 String filePath) {  
  
 MimeMessage message = javaMailSender.createMimeMessage();  
  
 try {  
  
 MimeMessageHelper helper = new MimeMessageHelper(message,true);  
  
 helper.setFrom(email);  
 helper.setTo(sendTo);  
 helper.setSubject(titel);  
 helper.setText(content);  
  
 FileSystemResource file = new FileSystemResource(filePath);  
 //获取文件名  
 String fileName = filePath.substring(filePath.lastIndexOf(File.*separator*));  
  
 System.*out*.println("文件名称: " + fileName);  
  
 helper.addAttachment(fileName,file);  
  
 javaMailSender.send(message);  
  
 } catch (Exception ex) {  
 ex.printStackTrace();  
 }  
}

测试类



###### 1.3.5、编写service-发送模板

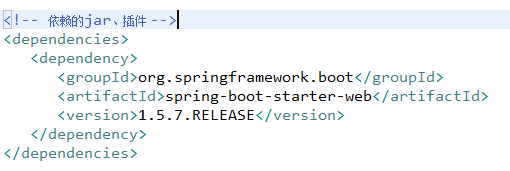
### 2、SpringCloud



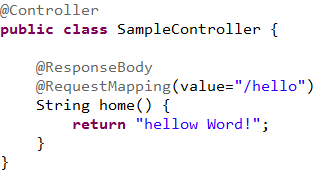
#### 1、搭建一个简单的spring Cloud项目

1.1、创建一个maven工程

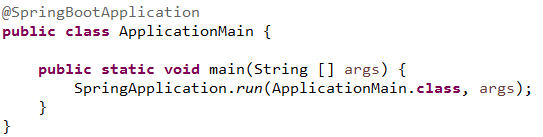
1.2、在pom.xml文件中写入依赖的jar包



1.3、创建一个Controller文件



* 1. 运行该java项目



#### 2、读取配置文件

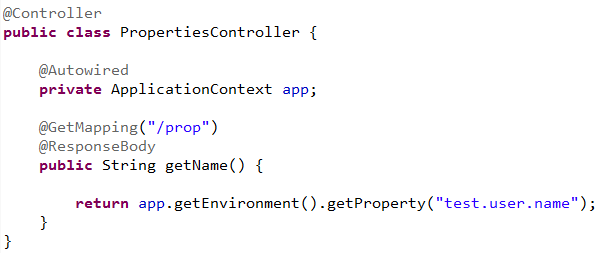
##### 1、读取默认配置文件

###### 1.1、从四个地方读取application.properties



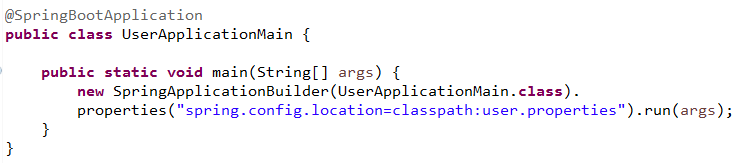
1. 项目的根目录下的config下
2. 项目的根目录下
3. ClassPath目录下的config下
4. Classpath目录下

###### 1.2、读取配置文件的实例



###### 2、读取自定义的配置文件

##### 2.1、通过java程序来指定获取的配置文件



#### 3、创建一个Eureka服务

## 5、struts2

### 1、struts2的工作原理

用户发送请求，进入拦截器中，执行完后，有可能进入下一个拦截器，如果没有，就进入action类中，根据用户的请求和struts.xml的配置，进入到具体的方法当中，默认是excute方法，处理完业务逻辑后，有可能在次进入其他的拦截器中进行处理，处理完成后返回结果

### 2、请求到struts2中有哪些方式

1、默认进入execute（）方法

2、通过指定方法和使用占位符都可以跳转到具体的某一个方法中

## 6、Maven

### 基础知识点

<http://blog.csdn.net/u012225679/article/details/73740785>

<https://www.cnblogs.com/hzg110/p/6936101.html>

#### Maven的相关概念

**Maven两大核心**

1. **项目构建：项目编码完成后，对项目进行编译、测试、打包、部署、发布的一系列操作都可以通过命令来实现**

**部署到服务器（tomcat），如下图：**

****

1. **依赖管理(jar包的管理)**

在一个项目中，jar包都会用到很多，所以jar所占的空间也会很大

Maven项目需要使用一个jar包只需要在配置文件pom.xml中配置它的坐标信息，

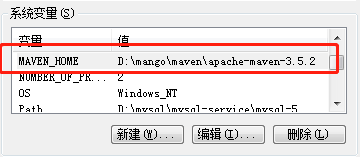
Maven程序会自动去maven仓库查找相对应的jar包

**Maven安装需要依赖jdk，不同的maven版本需要不同的jdk版本**

|  |  |
| --- | --- |
| **Maven版本** | **Jdk版本** |
| **3.3.1-3.5.0** | **1.7+** |
| **3.2.1-3.2.5** | **1.6+** |
| **3.0-3.1.1** | **1.5+** |

#### Maven如何安装和配置

1. 下载相关maven包
2. 解压到相关目录（不要包含中文和空格）
3. 配置环境变量



然后将%MAVEN\_HOME%\bin加入到path中

通过如下步骤，就可以在本地完成maven的安装和配置

在命令窗口可以通过来查看是否已经配置好

Maven下面的文件标配是

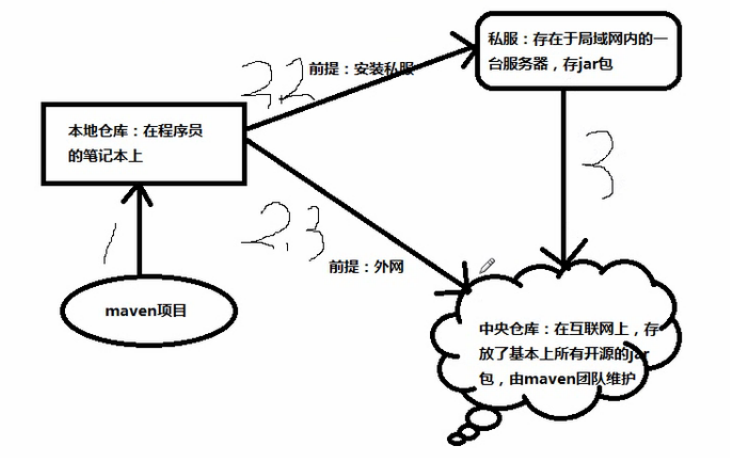
Java

Test

resource

#### Maven配置本地仓库

1. 下载jar包解压到相关目录
2. 找到maven下conf下的settings配置文件
3. 修改配置文件中指定本地仓库的路径



三个仓库的概念

1、本地仓库(存放项目中所需要用到的jar)

2、局域网仓库(存放项目中基本都会用到的jar包)

3、中央仓库(maven团队维护)

三个仓库的依赖搜索顺序

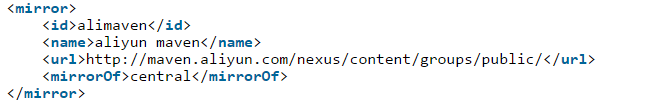
步骤1、在本次仓库中查找需要的jar包，如果找不到执行步骤2

步骤2、在中央仓库搜索，如果找不到，就执行步骤3

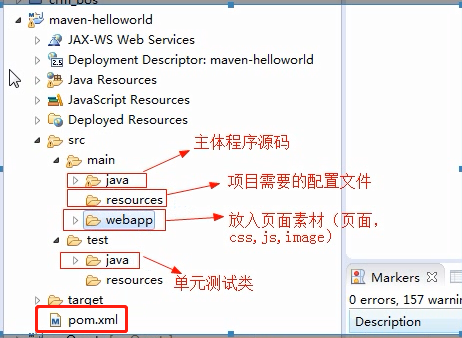
步骤3、在局域网仓库中搜索jar，如果还是没找到则抛出异常

在真实的项目开发中，可以不配置中央仓库，可以手动配置成阿里云仓库

在maven的conf/settings.xml配置文件中修改如下：



#### Maven的标准目录结构



#### Maven的常用命令

#### clean:清除

清除根目录下的target文件（清除编译后的class文件）

#### 2、compile(坑怕了)：编译

对maven项目重新进行编译，在根目录下生成一个target的文件，存放class文件

#### 3、test:进行单元测试

#### 4、package：打包

Web项目---生成一个war包

Java项目—生成一个jar包

将项目打包到target目录下

#### 5、install(in s to)：安装

#### Maven的生命周期

**Maven有三套独立的生命周期，互不影响，执行后面的命令，前面的命令也会跟着执行**

**第一阶段：Clean**

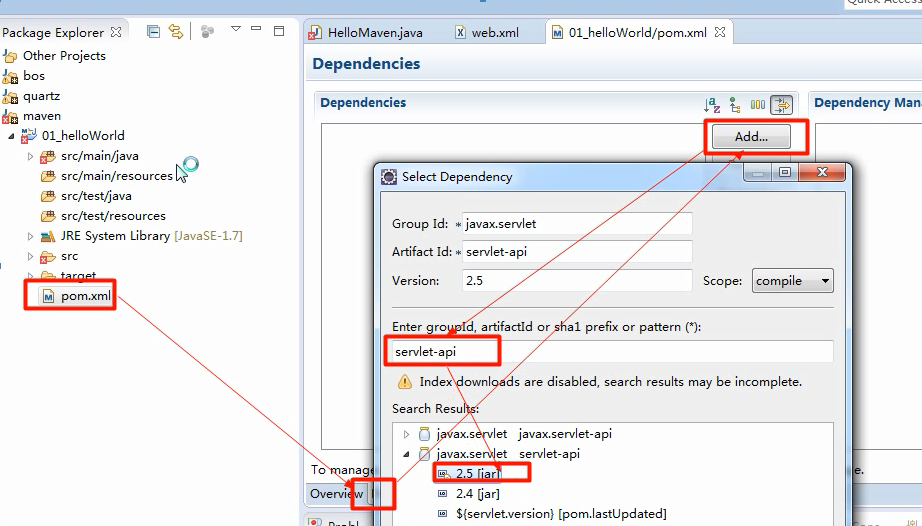
**第二阶段：Compile(编译-坑怕二恩) 🡪 test(测试) 🡪package(打包) 🡪install(安装-英思当)🡪 deploy(部署-低破儿)**

**第三阶段：site(发布-撒二T)**

#### Maven缺失jar如何导入

1、在核心配置文件pom.xml中手动引入

2、jar包搜索



3、在添加核心包的时候，maven会将相关的一依赖包也导入

#### Maven-pom.xml核心配置文件详解

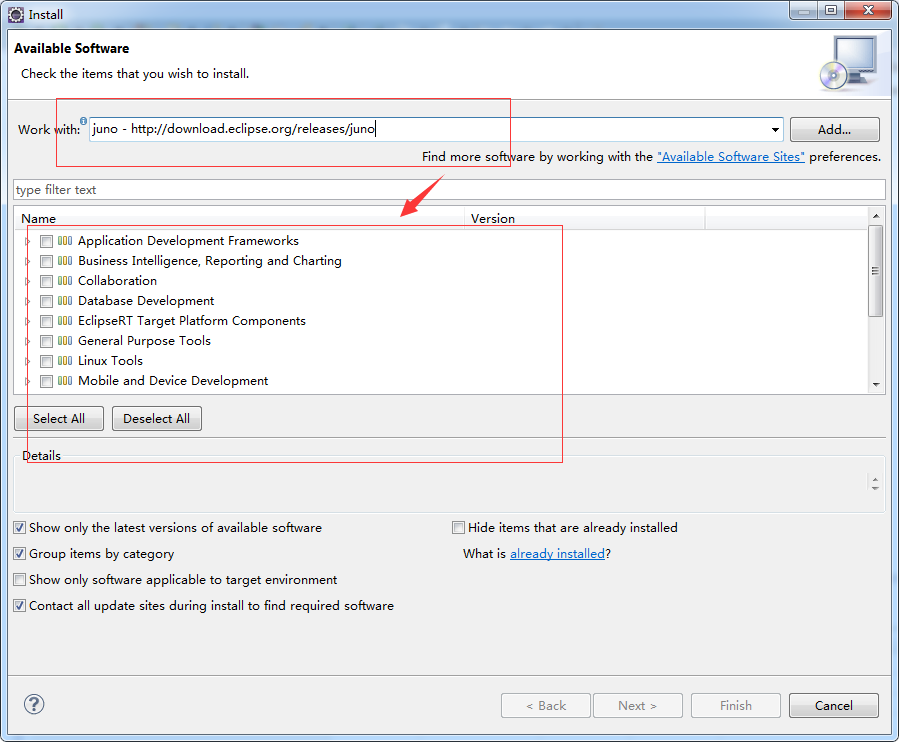
一个程序中应该有且有一个pom.xml配置文件

该配置文件中有一个**project**元素和三个必须的字段**groupId、artifactId、version**

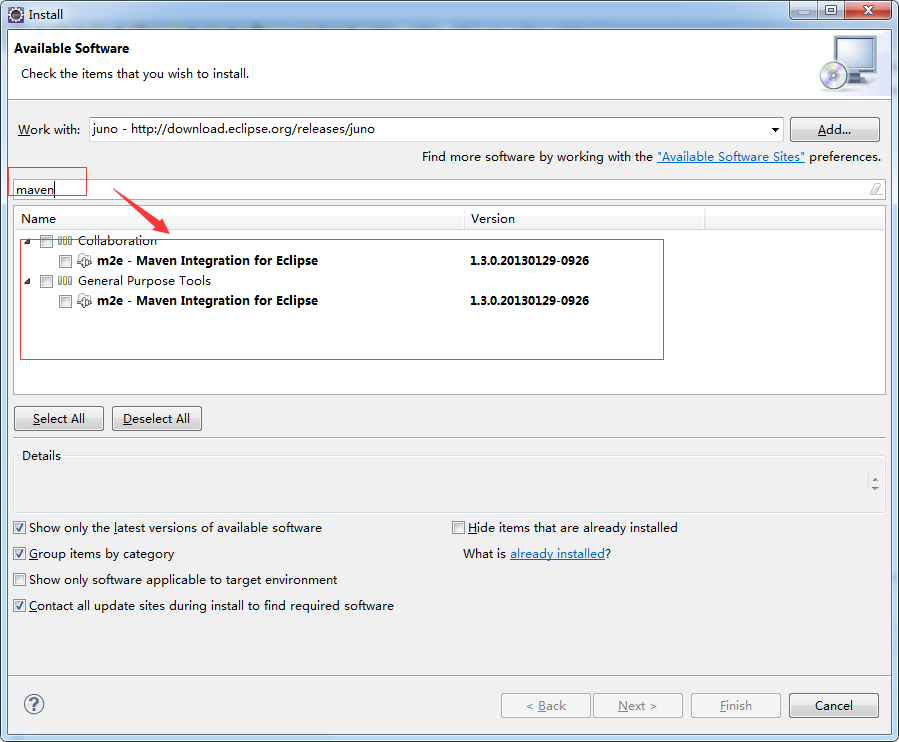
### Maven实战

#### Eclipse中如何安装Maven插件

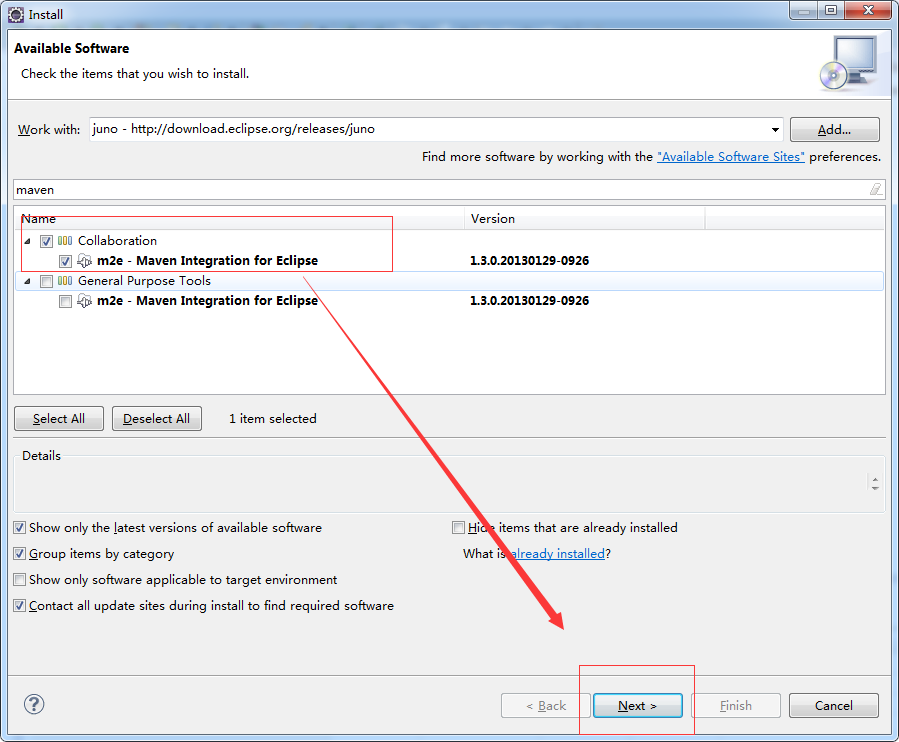
1. 打开eclipse，菜单“Help”-“Install New Software...”
2. 在Work with 地址栏输入：http://download.eclipse.org/releases/**juno**  （注意：红字部分是eclipse对应的版本名称），稍等片刻：



1. 在filter框中输入maven便能定位要安装的插件，如图：



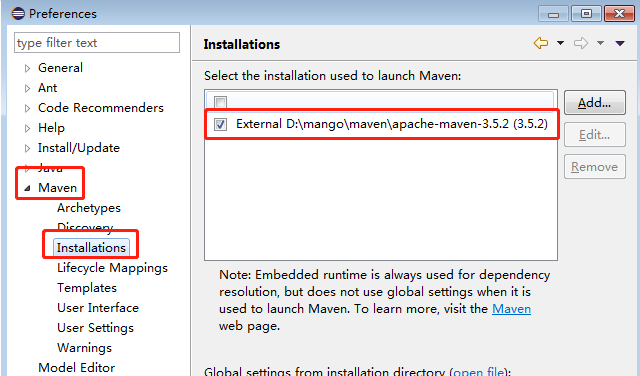
1. 选择“Collaboration”-“m2e - Maven Integration for Eclipse"，并点击next按钮进行安装步骤。图：



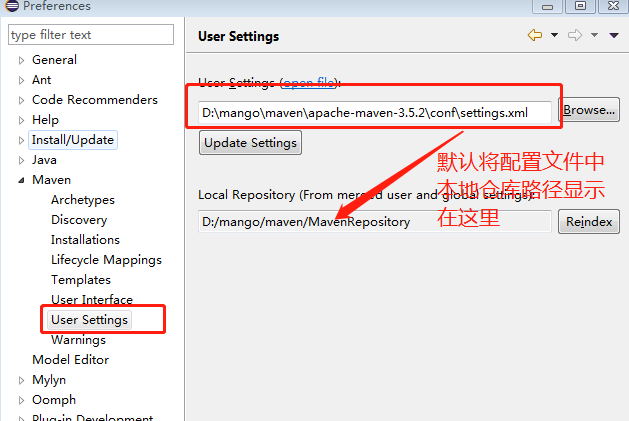
1. 安装完成重启eclipse，菜单：Window --> preferences ，输入maven进行相关设置。

#### Eclipse中配置maven和创建maven实例

1、选中解压后的文件

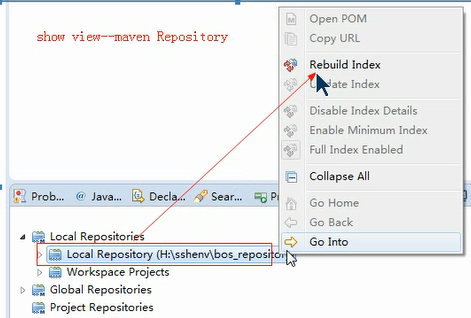


1. 配置settings文件

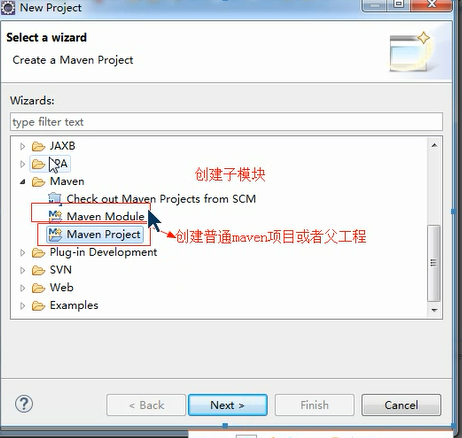


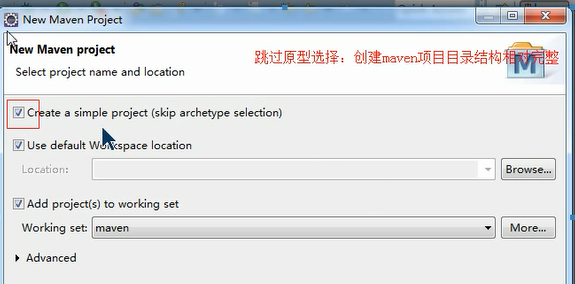
1. 构建索引

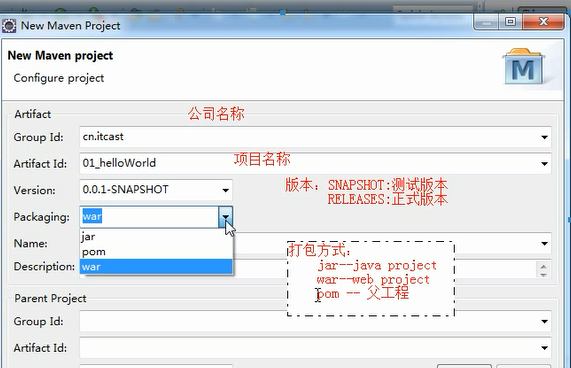
打开maven视图，选中Rebuild Index



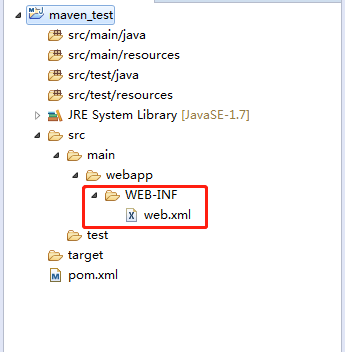
1. 创建一个maven项目



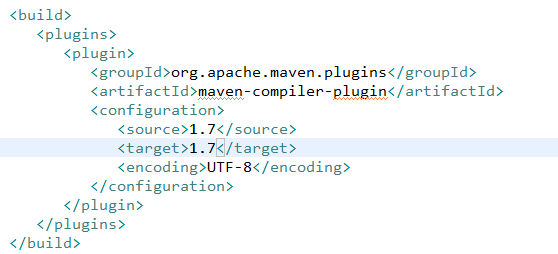




1. 创建好maven项目后，会缺失web.xml文件，需手动创建



1. 在pom.xml中生成一个jdk1.7的插件



#### Maven如何解决依赖冲突

**直接依赖 优先级高于 传递依赖**

##### 1、maven有自己的调解原则

###### 1.1、第一声明原则

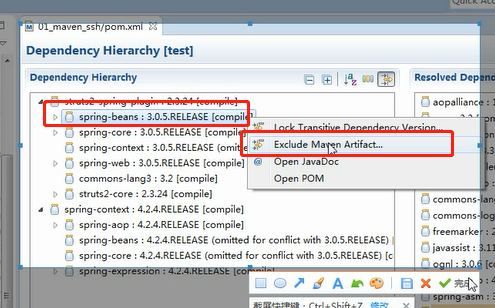
谁先导入包就用谁的jar，如果要更改在pom.xml更改顺序就行

###### 1.2、路径优先原则

简单来说就是将冲突的包手动导入进来，变成直接依赖

##### 2、排除依赖

找到冲突的包，然后将不需要的包排除掉

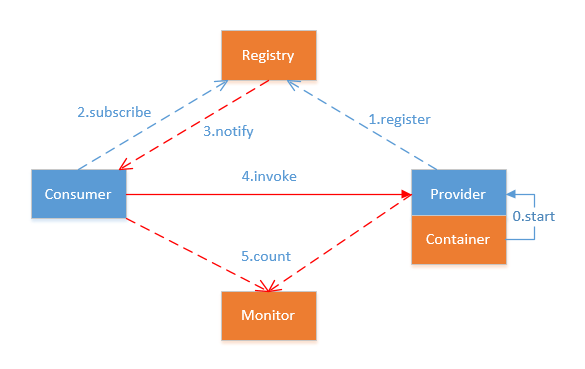


##### 3、版本锁定

在pom.xml文件中使用dependencyManagement(第盼等己 卖你几每T)来指定项目中依赖的版本



## 7、dobbo-zookeeper



1. 服务程序启动后，Provider向Registry注册中心暴露服务
2. 消费者程序启动后，向Registry注册中心订阅自己需要的服务
3. Registry向消费者返回服务商提供的地址
4. 消费者调用url
5. 服务者和消费者将内存中调用的次数和时间定时向Monitor（监控中心）发送

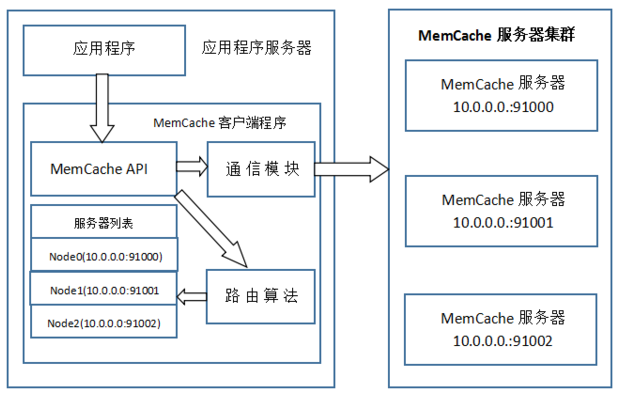
## 8、Memcache

Memcache的作用：缓存各类数据，减少用户与数据层的交互，从而达到快速响应客户

Memcache的引用场景：

1. 作为数据库的前端缓存应用
2. 完整缓存，事先将数据加入到缓存，用时直接从缓存中获取，我们称这种现实方式为预热
3. 执点缓存，需要前端配合，只缓存经常被访问的数据，

#### Memcache的工作流程



1. 应用程序调用memcacheAPI进行写数据
2. Api首先计算key的hash值，然后找到具体某一台服务器的位置，然后就可以获取到memcache对象及Ip地址和端口号
3. 将数据写入到具体的服务器上，完成分布式缓存写的操作

读取缓存的步骤和写的步骤类似

#### 分布式部署中，Memcache如何处理某台服务器宕机的情况

假设集群中有三台服务器，其中的某一台宕机后，用户是访问不到该服务器中的缓存数据，通过一致性Hash算法会自动在数组中循环其它可用的节点，选择与之前节点近的节点存储数据

#### 服务扩容后，如何进行映射？

1. 通过%hash的算法，命中概率不高
2. 一致性Hash算法，找到离自身近的节点进行存储

#### Memcache存放在内存中需要注意的几个点

1. **访问数据的速度比关系型数据要块，因为oracle、Mysql这些传递的关系型数据库为了保存数据的持久性，将数据保存在硬盘中，Io读取速度慢**
2. **Memcache中的数据是保存在内存中，只要Memcache重启，数据就会消失，而且缓存的数据大小受系统位数的限制，32位的系统最多只能缓存2GB的数据，64位的可以没有限制**
3. **key最大为250个字节，超过长度无法存储**
4. **保存的元素个数是没有限制的，但是单个元素的大小不能超过1M**
5. **元素的最大失效时间是30天，超过失效时间，元素自动消失**

#### Memcache的优缺点

优点：

1. 存储在内存中，读取速度快
2. 支持分布式部署，避免一台服务器宕机后，获取不到缓存数据
3. 均衡请求，避免同一台服务器受理所用的请求

缺点：

1. 数据不能持久化
2. 存储的数据大小限制，单个key不能超过250个字节，单个元素大小不能超过1M
3. 存储的数据结构单一，只能存储String类型的key

## 9、Redis

缓存数据和缓存页面

Redis是存在内存中，可以定期持久化数据到硬盘，保证了数据的安全性

### 本地安装完redis后，如何启动redis服务器端

1. 在cmd窗口中找到redis的安装路径
2. 输入redis-server.exe redis-windows.conf
3. 出现该界面说明启动成功



1. 不要关闭当前窗口，重启启动一个cmd命令窗口，启动客户端，如下



1. 可以进行编写了

### Redis key、value的注意点

1. key的命名不能包含空格和换行符(\n)
2. key长度不要太长，占内存，查询慢，也不要太短，可读性不好
3. value的大小不能超过1G

### Redis五种类型

String、set、list、hash、sorted Set

用的比较多的是String、hash

Flushall（父啦是熬）--清空所有数据

FlushDB—清除一个数据库

#### String

1. 设置值、获取值(get、set)和设置多个值和获取多个值

**语法:set key value\get key—>单个**

**mset key value key value …**

**mget key key1**







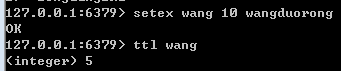
1. 获取值在设置值(getset)、设置key和值的同时设置key的过期时间

**语法:getset key value**

**语法:setex key seconds(秒数) value**

**语法:keys 正则表达式**



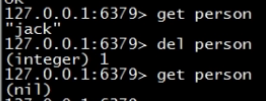




ttl key 可以获取该key还剩多少秒存活 如果已经失效返回-2

1. 删除值(del)

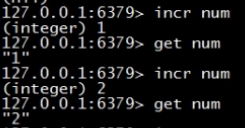
**语法:del key..可以删除一个 可以删除多个**





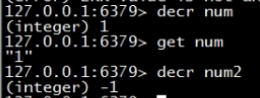
1. 递增(incr-如果不存在就先新增一个对象，默认为0，然后在+1，如果存在则在原有基础上+1)

**语法:incr key**



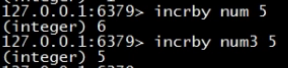
1. 递减(decr-如果递减的对象不存在，新建一个默认为0，在-1)

**语法:decr key**



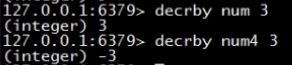
1. 指定增加的数值(incrby,如果不存在，新建一个，默认为0)

**语法:incrby key increment(要增加的值)**



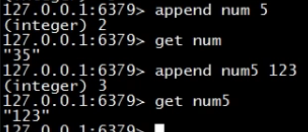
1. 指定减少的数值(decrby)

**语法:decrby key increment(要减少的值)**



1. 字符串拼接(append)

**语法:append key value**



#### Set-不允许有重复key，无序集合

1. 添加元素

**语法:sadd key member1 member2 – 可以添加一个或者多个**

****

****

1. 获取集合中成员或个数

**语法:smembers key**

**语法:scard key**

****

****

1. 判断元素是否是指定集合中的成员

**语法:sismember key member –存在返回1 不存在返回0**



1. 删除一个或者多个成员

**语法:srem key member1 member2**



#### sorted Set-有序集合，不运行有重复值，每一个元素都会有一个double分数

1. 向集合添加成员和分数

**语法:zadd key score member1 scroe2 member2**





1. 查询在指定分数范围的成员个数

**语法:zcount key min max**



1. 移除集合中指定的成员一个或者多个

**语法:zrem key member1 member2**



1. 查询集合中某个成员的分数

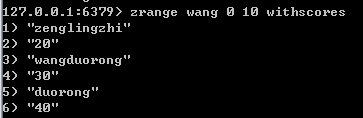
**语法:zscore key member**



1. 返回指定索引范围内的成员和分数

**语法:zrange key start stop [withscores]**

Withscores可填可不填，填了，分数一直查询出来，没填，分数不会查出来



#### List-按照插入的顺序排序

1、向集合中添加值

**语法:lpush key value1 value2 –从左到右插入数据 查询时返回的数据相反**

**rpush key value1 value2 –从右到左插入数据**

****

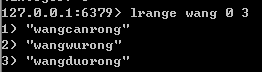
****

1. 输入指定范围内的元素值

**语法:lrange key start end**

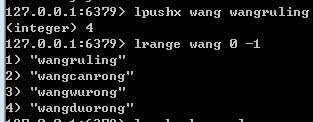
**0表示第一个元素**

**-1表示最后的一个元素 -2表示倒数第二个元素**



1. 将一个或者多个值插入到已存在的列表头部

**语法:lpushx key value**

****

1. 获取集合的长度

**语法:llen key**

****

1. 指定索引设置值

**语法:lset key index value**

****

1. 删除元素

**语法:lrem key count value**

**Count > 0 从表头开始 <0从表尾开始 = 0 删除表中所有相同值**

**语法:lpop key 移除第一个元素**

**语法:rpop key 移除最后一个元素**

****





#### Hash-主要用于存储对象

1. 设置一个属性值(hset key field value)

如果没有field域,直接添加,如果有,则覆盖原field域的值



1. 设置多个属性值

**语法:hmset key field value field value**

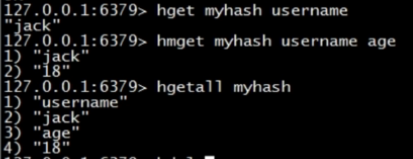


1. 获取一个、多个、或者对象的全部属性值

**语法:hget key field – 获取一个**

**hmget key field1 field2 –获取多个**

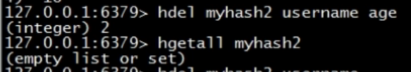
**hgetall key –获取key下面所有的属性和值**



1. 删除一个或者多个

**语法:hdel key field –删除一个**

**hdel key field field –删除多个直接在后面拼接field**



1. 查找对应的key值是否存在

**语法:hexists key field,存在返回1 不存在返回0**



1. 获取所有属性

**语法:hkeys key**



### Redis中在什么情况下会将数据保存到硬盘中

在redis中并不是所有的数据都保存在内存中，如果redis发现内存的使用量超过了某一个阈值，将触发swap的操作，计算出那些key对应的值需要持久化到硬盘，然后将内存中对应的值清空，注意的是持久化的是key对应的值，而不是key本身

RDB

AOF

### Redis分布式

Redis支持主从的模式

主表会将数据同步到从表，但从表不会将数据同步到主表。

从表启动的时候自动从主表中同步数据

可以利用redis的主从关系来实现读写分离

### Redis事务

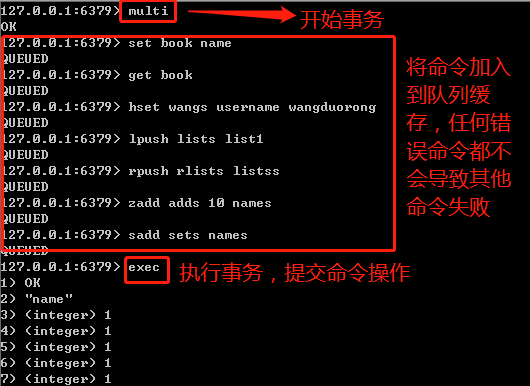
**Redis一次可以执行多个命令**

1. **批量操作在发送exec命令前放入队列缓存**
2. **收到exec命令后进入事务执行，事务中任何命令失败，其它命令依然执行**
3. **在事务执行过程中，其它客户端提交的命令请求不会插入到事务执行命令中(一旦执行不能干预)**

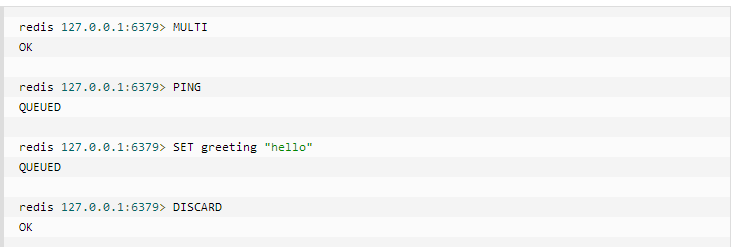
**一个事务从开始到执行有三个步骤**

1. **开始事务**
2. **命令进入队列缓存**
3. **执行事务**

#### 事务执行（multi 木题)）



#### 事务取消（multi-命令-discard（的思高得））



### Redis数据备份和恢复

**在redis根目录下的redis-windows.conf的配置文件**

****

**持久化的一些触发条件，比如多少秒 key变化的个数**

**RDB默认在配置文件中是打开的，AOF默认是关闭**

#### RDB

1. 创建数据的备份-默认rdb，可以使用AOF

Save该命令会在redis安装目录的根目录下生成一个dump.rdb文件

Bgsave也可以执行，





1. 恢复数据

将备份文件dump.rdb文件拷贝到redis的安装目录，重启redis自动恢复

#### AOF

##### 备份

在使用AOF之前需在redis-window.conf中开启



持久化策略

1. appendfsync always（哦为死）-每修改一次都会写入到硬盘中
2. appendsync everyec（）
3. 不同步到硬盘



##### 恢复

1、将生成的appendonly.conf文件打开，然后将最后的那个

### Redis的应用场景

1. 排行榜、计数器
2. 数据缓存
3. 队列，通过List和Set实现

### Redis的优缺点

优点：

1. 存储的数据结构丰富，支持字符串、List、Set、Zset、Hash等数据结构
2. 支持主从复制，主机自动将数据复制到从机，从而实现读写分离业务
3. 读写速度快，存储在内存中
4. 数据持久化，将内存中的数据保存在硬盘中

缺点：

1、不支持自动容灾和恢复功能，主机或者从机的宕机都会导致前端的读写请求失败，需要等待服务器重启或者手动切换前端的IP才能恢复

2、主机宕机，会导致主机和从机的一部分数据不同步

3、key的命名不要过长(不超过1024字节)，也不要过短

## 10、github

Github官网

<https://github.com/>

account:QQ邮箱

password:accpT110

密钥密码也是accpT110

<https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/>

### 1、git入门

#### git init

使用git需要先创建一个仓库(repository)，你可以使用一个已经存在的目录或者

一个空目录，**使用当前目录作为Git仓库，只需要初始化它即可**

#### git init newFile

使用指定的目录作为Git仓库

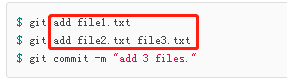
#### git add filename

在git仓库中创建一个新的文件

#### 1.4、git commit –m “Adding files”

提交创建的文件到git仓库

可以一次性提交多个文件，如：



-m后面的描述可以是任意描述，最好是能描述提交信息的一个概述

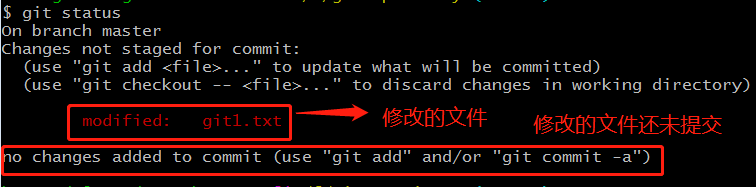
#### 1.5、git commit –a –m “change some file”

如果不想每个文件都使用add，可以使用-a将所有被修改或者已删除的文件（前提是在git仓库中）提交到仓库中，**这里的-a只针对update，add的文件不会提交**

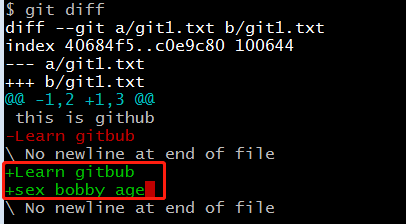


#### 1.6、git status、git diff

查看修改的文件 git status

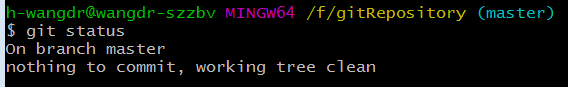


查看修改文件到底修改了哪些内容 git diff

 修改的内容

将修改的文件提交到git仓库和上面步骤一样，add和commit

提交完后，在使用git status看不到刚才修改的文件了

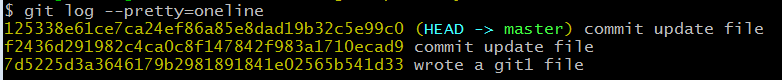




#### 1.7、git log

命令显示从最近到最远的提交日志，如果觉得太多，可以使用

git log --pretty=oneline



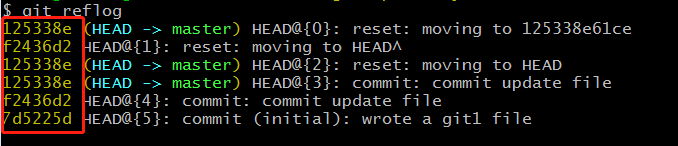
#### 1.8、git reset –hard HEAD^/commit id

默认返回上一个版本



如果误操作导致回到上一个版本，只要你能记录上一个版本号，一样也可以回退回来，

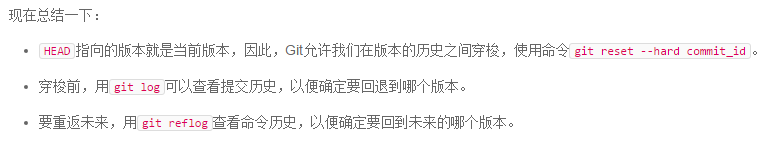
假如版本号忘记了，可以使用git reflog记录每一次操作的命令



版本号不需要全部写好，git会自动去查找，但也不要只写前面几个，不然git很难进行精准的定位

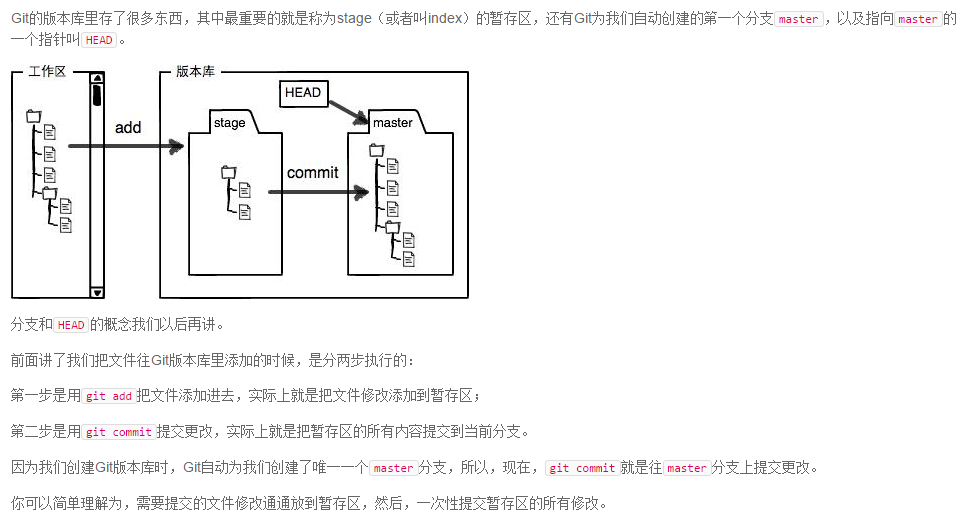


**版本回退的速度很快，git对每一个版本都会有一个指针，回退到某一个版本只需要将head指向对应的指针就行，顺便把工作区的内容更新到指定的版本**



#### 1.9、工作区和暂存区

Git和其他版本（如SVN）有一定的区别，就是有暂存区的概念



**git add只会将文件添加到暂存区，git commit只会将暂存区的文件提交到分支上**

**1.10、git checkout – filename**

撤销当前文件中所有修改的内容，前提是该文件还是未提交到暂存区，或者分支，不然只能使用git reset --hard commit id



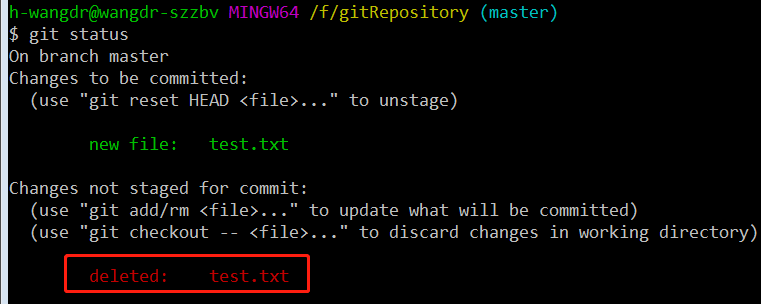
#### 1.11、删除文件

删除文件并且通过git去恢复的前提是该文件已经添加到暂存区或者分支上，也就是已经和git产生了关系

删除指定文件



通过git status查看哪些文件进行了修改(在git来说，删除也是修改的一种)



可以通过git checkout – 文件名 进行恢复



### 2、远程仓库

#### 2.1、创建远程仓库

1、一种是先在本地创建一个仓库，但不使用Initialize this repository with a README

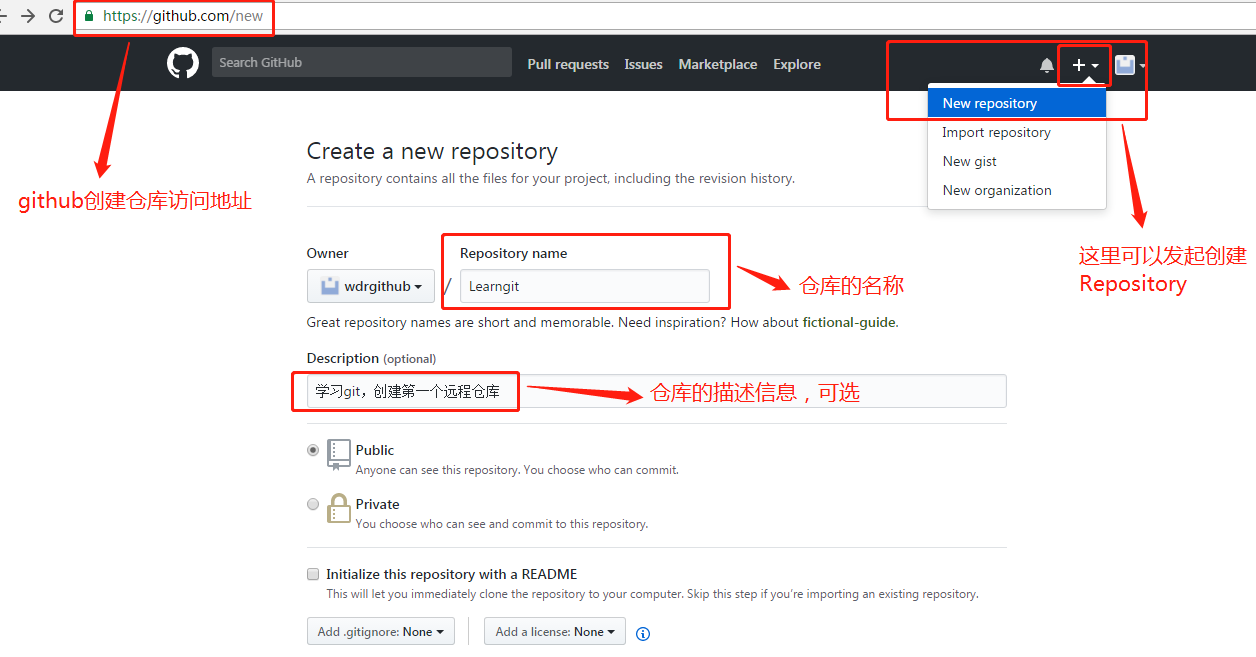
创建成功后，将本地仓库的内容推送到远程仓库

2、创建远程仓库，勾上Initialize this repository with a README然后将远程仓库克隆到本地

**Ps:如果要与远程进行交互，需要为每一个项目添加一个公钥，不然本地无法推送到远程仓库，且每一个公钥都需要重新生成一次，两个或多个项目不能使用相同的公钥**

##### 1、访问<https://github.com/new>

##### 2、如图，最后点击Create Repository



##### 3、创建完仓库后，可以做如下操作

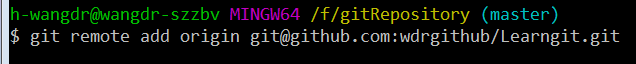
3.1、在线提交Repository中的Content

3.2、将Local Repositort Content commit to remote repository

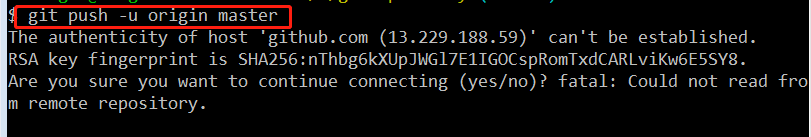
<https://www.zhihu.com/question/21402411>

<http://blog.csdn.net/happystarycy/article/details/52490183>

与远程仓库建立关联



将本地仓库内容推送到remote Repository



抛出的错误，绝大部分是因为github中没有设置SSH，本地设置密钥

1. 跳转到本地，且查看本地是否有密钥



1. 输入ssh-keygen –t rsa –C “邮箱” 点击回车



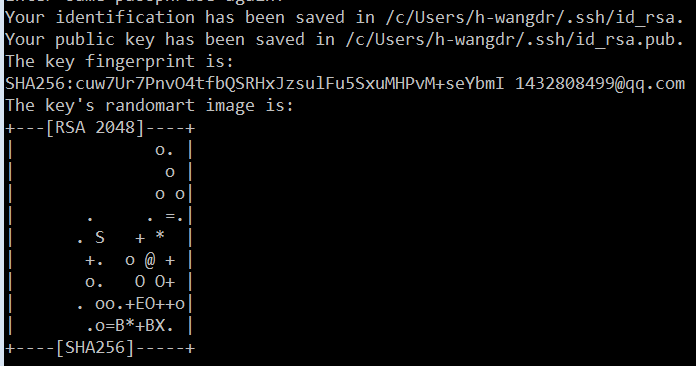
1. 根据邮箱创建一对密钥



1. 根据提示输入密钥密码和确认密码，且不会有文字输出

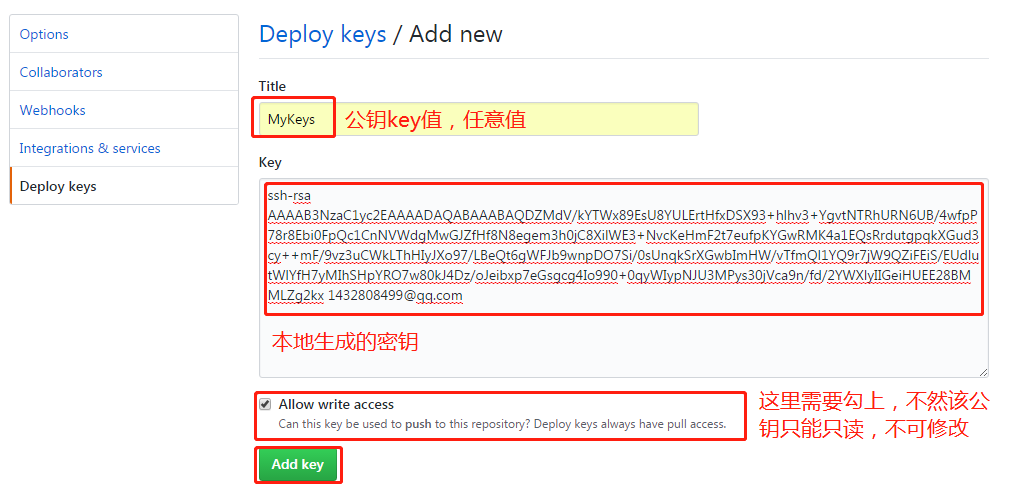


1. 本地密钥生成

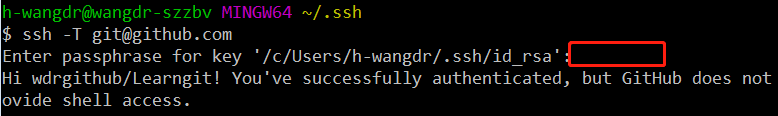


在~/.ssh目录下有 id\_rsa和 id\_rsa.pub两个文件，其中id\_rsa.pub文件里存放的即是公钥key。

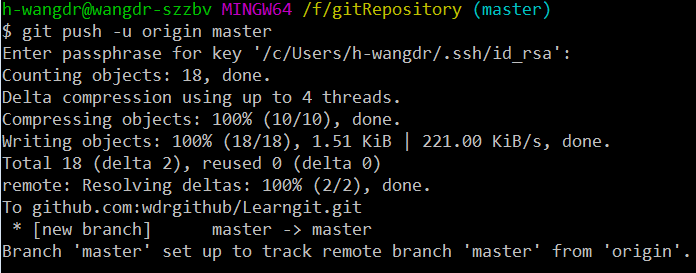
1. 登录到GitHub，点击右上方的头像，选择settings ，点击Add SSHss key，把id\_rsa.pub的内容复制到里面即可。



1. 测试是否配置成功 ssh –T [git@github.com](mailto:git@github.com),红色框中输入密钥（accpT110）



8、推送信息成功



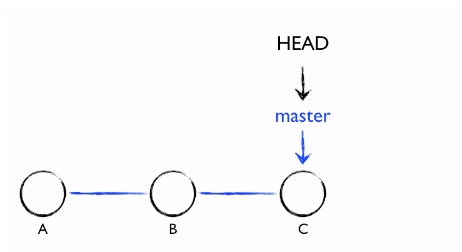
第一次推送需要使用 **–u** 后面，后面就不在需要

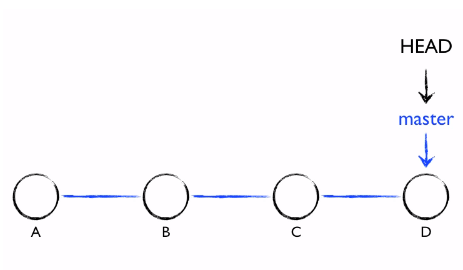
3.3、可以将另一个仓库的内容Import到Repository

### 3、分支管理

#### 3.1、创建与合并分支

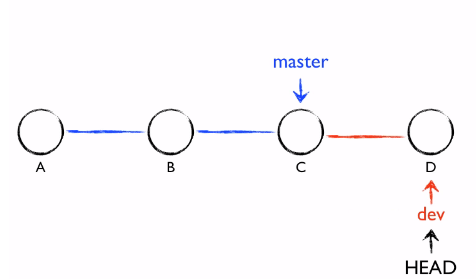
git中的操作，最后基本都是针对主分支（master）来操作，没更新一个文件，master分支就会向前移动一下指针

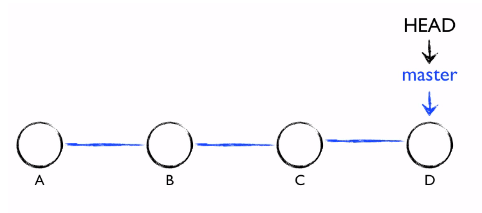




我们可以新建其他分支，这个时候你做的修改就会提交到你创建的分支，别人看不到，也不会影响到master，最后全部完成后，合并到master上

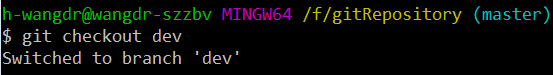




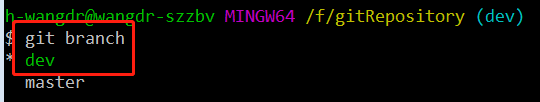


##### 3.1.1、创建分支命令且切换到分支



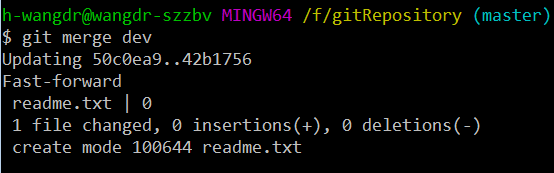


查看当前HEAD指向哪个分支

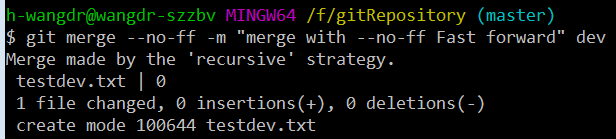


##### 3.1.2、合并分支

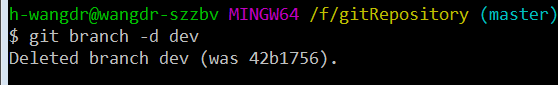
将指定分支上的内容合并到当前分支上



可以不使用Fast forward模式进行合并



##### 3.1.3、删除分支



##### 3.1.4、总结

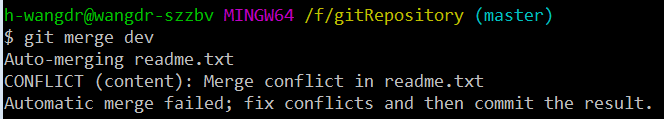


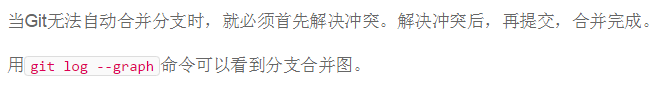
强制删除分支:git branch –D <name>

#### 3.2、解决冲突

开发中最好的方式是，在自己分支上修改，切换到主分支后不要再去进行修改，不然容易导致冲突（修改了相同的内容）

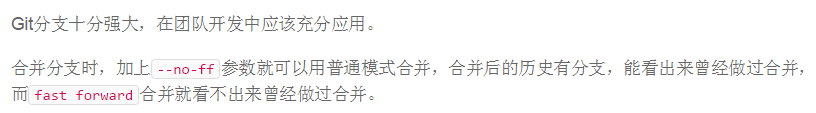
##### 3.2.1、合并文件发生冲突





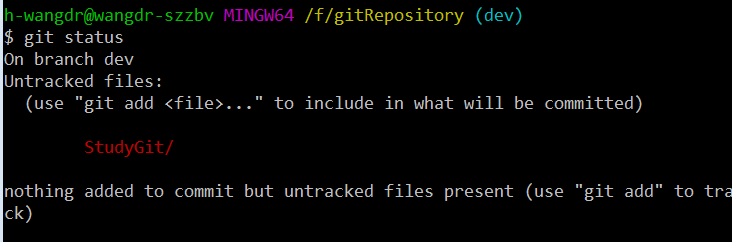
#### 3.3、分支管理策略

一般开发中都是使用Fast forward这种快速模式合并，也可以不使用Fast forward来进行合并

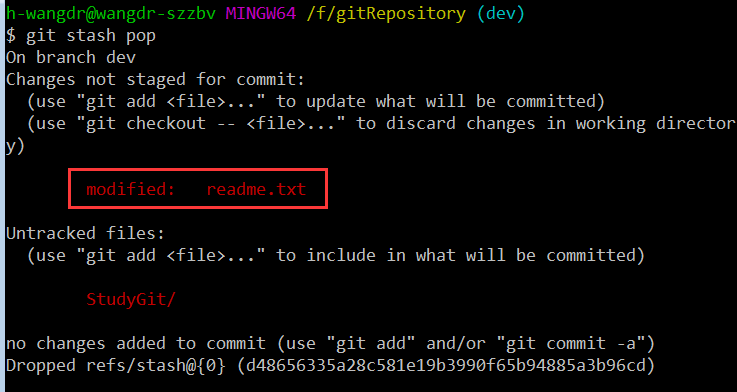


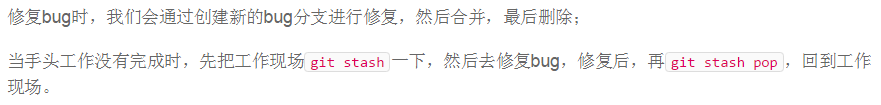
#### 3.4、Bug分支

在开发中很容易碰到bug，假如master突然发生bug且要求马上去解决，这个时候你本身的分支新增了内容，但只完成了一半，不能提交，也不想直接放弃重新来，这个时候可以使用git stash来将当前未提交的信息隐藏



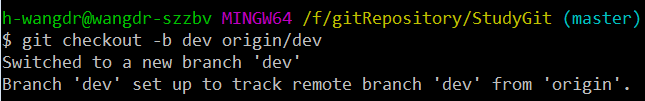
切换到其他分支修复完bug后，在回到dev分支上，在将当前隐藏的文件显示出来



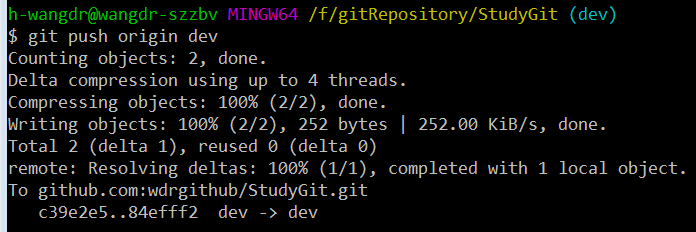


#### 3.5、多人协作开发

项目经理在github上创建一个远程项目，小组成员克隆到本地仓库，但是只能克隆master分支，其他分支，比如dev克隆不了，这个时候需在本地创建一个远程Dev分支，自动会将Dev分支内容推送到本地



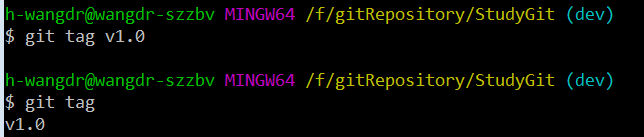
创建成功后，就可以将里面对应的代码提交到远程仓库



### 4、标签管理

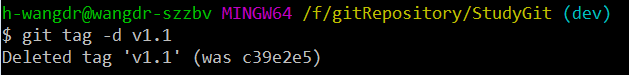
#### 4.1、创建标签

首先切换到需要打标签的分支上，然后使用git tag tagName，通过git tag可以当前分支所有的tag



#### 4.2、操作标签

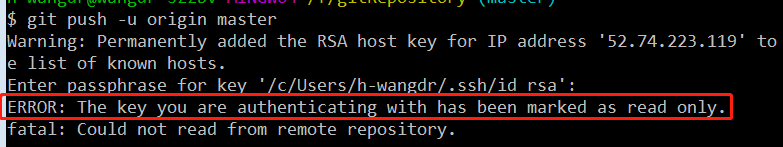
如果标签打错了，可以删除



### 4、错误信息合集

<http://blog.csdn.net/dengjianqiang2011/article/details/9260435>

3.1、将Local master push to remote throws



说明github公钥是只读

### 5、需要注意的地方

4.1、在本地创建密码和在github官网创建公钥

<http://blog.csdn.net/fenghuibian/article/details/73350890>

4.2、可以将https改成ssh

<https://www.cnblogs.com/superGG1990/p/6844952.html>

4.3、可以为每个命令配置别名

<https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/001375234012342f90be1fc4d81446c967bbdc19e7c03d3000>

# 数据库

## Oracle

### 基础

#### 1、列举查询中使用的表连接

内连inner join 找出两个表中相等的数据

左连left join 左表数据全部查询出来，右表中与左表中数据相等的数据查询出来

右连right join右表数据全部查询出来，左表中与右表中数据相等的数据查询出来

全连full join 将两个表中的数据查询出来

在oracle中表连接有一种特殊的写法

Select .. from table1 t1,table t2 where t1.id = t2.id（+）； 左连接

Select .. from table1 t1,table t2 where t1.id(+) = t2.id; 右连接

Select .. from table1 t1,table t2 where t1.id(+) = t2.id(+) 全连接

#### 2、常用函数

##### 1、字符串函数

|  |  |
| --- | --- |
| 字符函数名 | 作用 |
| Length() | 获取字符串长度 |
| Trim()、ITrim()、RTrim() | 删除空格或者从什么地方开始删除空格 |
| Replace() | 替换字符串 |
| Substr() | 截取字符串 |
| Instr() | 查找字符串 |
| Lower() --罗文 | 转换成小写 |
| Upper() --啊宝 | 转换成大写 |

##### 2、时间函数

|  |  |
| --- | --- |
| 时间函数名 | 作用 |
| sysdate | 获取当前系统时间 |

##### 3、转换函数

|  |  |
| --- | --- |
| 转换函数名 | 作用 |
| To\_date() | 转换成日期 |
| To\_char() | 转换成字符串 |
| To\_number() | 转换成数字类型 |
| To\_TIMESTAMP(汤死打普) | 转换成时间戳 |

##### 4、聚合函数、分组函数

|  |  |
| --- | --- |
| 聚合函数名 | 作用 |
| Sum() | 求和 |
| Max() | 求最大值 |
| Min() | 求最小值 |
| Avg() | 求平均值 |
| Count() | 求个数 |
| Group by | 分组函数 |
| Having | 用来判断group by查询出来的数据 |

#### 3、SQL操作符

##### 算术操作符

+-\*/

##### 比较操作符

=、!=、 >、>=、<、<=、between..and

In、not in

Like、not like

Is null、not is null

##### 逻辑操作符

And、or、not

##### 集合操作符

Union：两个SQL语句的总和，去重

Union all：两个SQL语句的总和，不去重

intersect ：两个SQL语句结果的交集

minus ：两个SQL语句，取查询出来的两个集合差

##### 连接操作符

||，将多个字段拼接在一起，或者将多个字符连接在一起

#### 4、SQL的常用数据类型

|  |  |
| --- | --- |
| 常用数据类型 | 作用 |
| char | 最大2000，默认为1字节 |
| Varchar2 | 最大4000，定义时需指定大小 |
| Number | 数据类型，可以存储整数 |
| Date |  |
| Timestamp |  |
| Clob | 存储大对象 |

#### 5、伪列（rownum、rowid）

rowid：物理存在的，长度是18个字符来确定唯一，可以使用它来进行去重

delete from table1 where rowid not in ( //删除重复数据中的最大值

select min(rowid) from table1 group by name //子查询，先查询出最小的那条

)

rownum：rownum是动态的，必须得先有结果集，然后在每个列中增加一个rownum

select rownum,name from table1 t where rownum >5

当产生结果集的时候，oracle会为这个列分配一个rownum，值为1，显然1是不大于5的，继续产生第二条数据，rownum值仍然是1

Select \* from (select rownum nums,name from table1 t) where nums > 5

子查询里查询出所有的结果集，然后oracle为所有的列分配完rownum，之后就可以通过rownum对数据进行筛选

#### 6、oracle常用的SQL语句

##### Insert

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

insert into 表名(字段名1,字段名2,...) select 字段名1,字段名2,.... from 另外的表名;

##### Delete

DELETE FROM表名 WHERE 条件;

##### Update

UPDATE表名 SET 字段名1=值1, 字段名2=值2, …… WHERE 条件;

##### Select

SELECT字段名1, 字段名2, …… FROM 表名1, [表名2, ……] WHERE 条件;

分组查询  
SELECT字段名1, 字段名2, …… FROM 表名1, [表名2, ……] GROUP BY字段名1   
[HAVING 条件] ;

两个以上表之间的连接查询  
SELECT 字段名1, 字段名2, ……   
FROM   表名1, [表名2, ……]   
WHERE 表名1.字段名 = 表名2. 字段名

[ AND ……] ;

查询结果集的排序操作, 默认的排序是升序ASC, 降序是DESC  
SELECT字段名1, 字段名2, …… FROM 表名1, [表名2, ……]   
ORDER BY字段名1, 字段名2 DESC;

### 高级

#### 1、存储过程

存储过程的好处

1. 在处理较复杂的SQL时，可以使用存储过程来代替，如果交个java来处理业务逻辑，可能会涉及到多次与数据库交互，而且存储过程是预编译的，用时直接使用
2. 可以更好的对用户进行权限的控制

存储过程的缺点

1. 不能面向对象编程，很多业务流程全部放在数据库处理对开发人员来说，要求比较高，维护困难
2. 高度耦合，业务与数据库融合在一起

#### 2、SQL语句优化

在查询数据时，哪些情况下会导致全表扫描

**进行优化时，应尽量避免查询整个表，在where及order by 涉及的列上建立索引**

1、在where字句后面，尽量避免使用null值判断，不然会导致引擎放弃使用索引而进行全表扫描，所以在数据表中，字段值尽可能的不要使用null值，尤其是操作比较频繁的字段

2、在where字句后面，尽量避免使用!=或者<>操作符，不然会导致引擎放弃使用索引而进行全表扫描

3、in、not in也要尽量避免使用，会导致全表扫描，对于连续的数值可以使用between，或者使用exists

# 开发工具

## 1、IDEA

## 1.1、IDEA中如何创建一个Project和多个Module

<http://blog.csdn.net/sinat_30160727/article/details/78109769>

<http://blog.csdn.net/qiuyinthree/article/details/63262107>

## 1.2、快捷键

<http://blog.csdn.net/shijiebei2009/article/details/44726433>

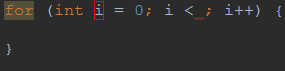
### 1.2.1、main方法快捷键

psvm



### 1.2.2、for循环

输入fori，选中第一个

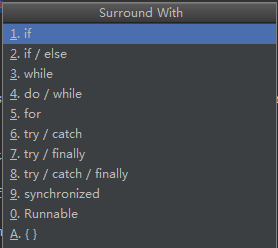


### 1.2.3、System.out.println

sout



1.2.4、组合快捷键(Ctrl+Alt+t)



# 其它知识点

## 1、什么是物联网