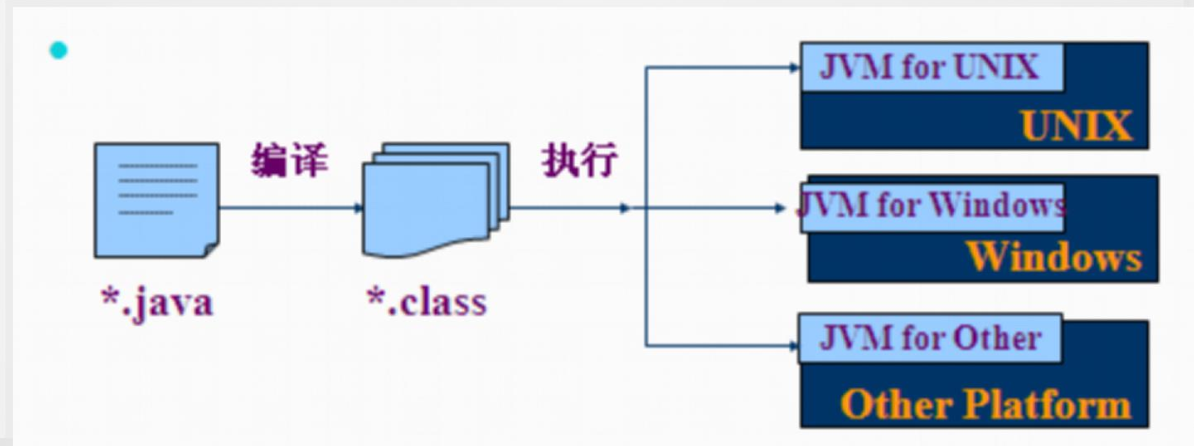
Java SE

# 第一章：java

## 一、java跨平台原理



字节码

源程序

编译器

* Java最大的优势所在
* 一次编译，到处运行
* 编译成格式独立的字节码文件，字节码文件需要虚拟机来解释执行

## 二、Java两种核心机制

JVM 垃圾回收机制

# 第二章：数据类型和标识符

## 一、标识符

* 程序员声明的单词（自己可以起名字的都是标识符）。

## 二、数据类型



占两块内存（堆、栈）

占一块内存

### final常量

* 使用final修饰的变量，只能被初始化一次，变成了常量。
* final常量是有名称的

## 三、基本数据类型之间的转换

### 转换方式：

* 自动转换：容量小 → 容量大
* 强制转换：容量大 ← 容量小

容量小 --------------------------------------------------------容量大

byte —> short,char—>int —>long—>float —>double

## 四、变量

* 本质上是内存中的一小块区域，使用变量名来访问这块区域

# 第三章：流程控制语句

* 顺序、选择、循环结构，跳转，多重循环（略）

## 方法 (形参，实参，返回值，返回值类型)

### 重载：

* 方法名相同
* 参数不同（类型/数量）
* 返回值相同
* 只有返回值不同或者只有形参的名称不同不构成重载

# 第四章：数组

## 一、数组：相同类型数据的有序集合.

### 数组特点:

* + 其长度是确定的。数组一旦被创建，它的大小就是不可以改变的
  + 其元素必须是相同类型,不允许出现混合类型。
  + 数组中的元素可以是任何数据类型，包括基本类型和引用类型。
  + 数组属引用类型

## 二、一维数组初始化

### 动态初始化：

int a[] = null;

a = new int[3];

a[0] = 3;

a[1] = 9;

a[2] = 8;

### 静态初始化

public class Test {

public static void main(String args[]) {

int [] a = { 3, 5, 7 };

}

}

## 三、二维数组初始化

* Declare, create and initiate in the same time ：

• int intA[][] = {{1,2},{2,3},{3,4,5}};

• int intB[3][2] = {{1,2},{2,3},{4,5}}; //非法

* Declare, create and initiate separately ：

• int a[][] = new int[2][5];

• int b[][] = new int[2][] ;

• b[0] = new int[2];

• b[1] = new int[3];

# 第五章：面向对象编程oop

## 一、局部变量和成员变量

* 区别
  + 声明位置不同： 类中 方法中
  + 作用范围不同： 当前类的方法 当前方法
    - 不同的方法中即使有同名的局部变量，没有关系，互不影响，建议相同
  + 内存存放的位置的： 栈内存中 堆内存中
  + 成员变量有默认值 局部变量没有默认值

## 二、内存分析

* 栈：
  + 存放：局部变量
  + 先进后出，自下而上存储
  + 方法执行完毕，自动释放空间
* 堆：
  + 存放new出来的对象
  + 需要垃圾回收器来回收
* 方法区：
  + 存放：类的信息(代码)、 static变量、字符串常量等.（只有一份的）

## 三、构造器

* 构造器定义： constructor 构造方法
  + 一个在创建对象时被自动调用的特殊方法。
* 构造器作用：
  + 为对象进行初始化（成员变量）工作。
* 构造器是一种特殊的方法：
  + 构造器的方法名必须和类名一致！
  + 构造器虽然有返回值，但是不能定义返回类型(返回值的类型肯定是本类)，不能在构造器里调用return。
  + 通过new关键字调用！！
  + 如果我们没有定义构造器，则系统会自动定义一个无参的构造方法。如果已定义则编译器不会
  + 添加无参数构造方法！
  + 与普通方法一样，构造方法也可以重载

## 四、this

* 构造方法中的this
  + 使用this来调用其它构造方法
  + 位置：必须是第一条语句
* 不能用于static方法

## 五、static

* 在类中，用static声明的成员变量为静态成员变量 ,或者叫做： 类属性，类变量.
  + 它为该类的公用变量，属于类，被该类的所有实例共享，在类被载入时被显式初始化，
  + 对于该类的所有对象来说，static成员变量只有一份。被该类的所有对象共享！！
  + 可以使用”对象.类属性”来调用。不过，一般都是用“类名.类属性”
  + static变量置于方法区中！
* 用static声明的方法为静态方法
  + 不需要对象，就可以调用(类名.方法名)
  + 在调用该方法时，不会将对象的引用传递给它，所以在static方法中不可访问非static的成员。
  + 静态方法不能以任何方式引用this和super关键字

public class TestStatic {

int a;

static int width;

static void gg(){

System.out.println("gg");

}

void tt(){

System.out.println("tt");

}

public static void main(String[] args){

TestStatic hi = new TestStatic();

TestStatic.width = 2;

TestStatic.gg(); //gg();

hi.gg();

gg();

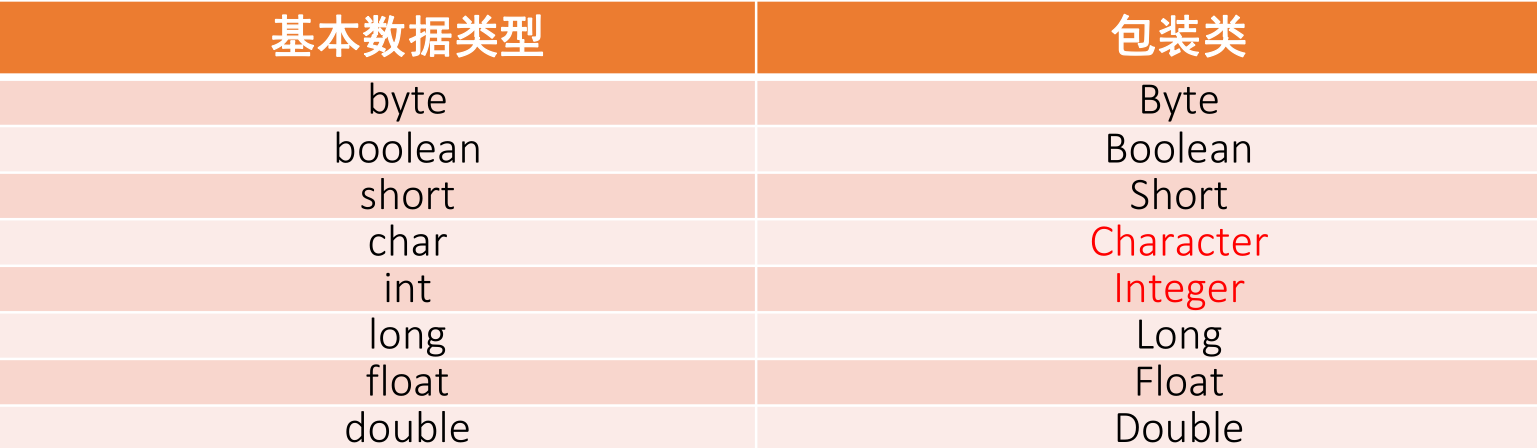
}

}

# 第六章：java常用类

## 一、基本数据类型的包装类

* 包装类均位于java.lang包，包装类和基本数据类型的对应关系：



## 二、字符串相等的判断(一般使用equals方法)

* equals判断字符串值相等，==判断字符串对象引用相等

public class StringTest3 {

public static void main(String[] args) {

String s1 = "abc";

String s2 = "abc";

String s3 = new String("abc");

String s4 = new String("abc");

System.out.println(s1==s2); //true

System.out.println(s1==s3); //false

System.out.println(s3==s4); //false

}

}

## 三．字符串区别

* String：不可变字符序列
* StringBuilder：可变字符序列、效率高、线程不安全
* StringBuilder：可变字符序列、效率低、线程安全

# 第七章：容器

## 一、list

* List
  + 特点：有序 不唯一（可重复）
* ArrayList 线性表中的顺序表
  + 在内存中分配连续的空间，实现了长度可变的数组
  + 优点：遍历元素和随机访问元素的效率比较高
  + 缺点：添加和删除需大量移动元素效率低，按照内容查询效率低，
* LinkedList 线性表中双向链表
  + 采用双向链表存储方式。
  + 缺点：遍历和随机访问元素效率低下
  + 优点：插入、删除元素效率比较高（但是前提也是必须先低效率查询才可。如果插入删除发生在头尾可以减少查询次数）
* 集合中内容是否相同
* 通过equals进行内容比较，而是==引用比较

## 二、SET

* Set

• 特点：无序 唯一（不重复）

* HashSet

• 采用Hashtable哈希表存储结构（神奇的结构）

• 优点：添加速度快 查询速度快 删除速度快

• 缺点：无序

* LinkedHashSet

• 采用哈希表存储结构，同时使用链表维护次序

• 有序（添加顺序）

* TreeSet

• 采用二叉树（红黑树）的存储结构

• 优点：有序 查询速度比List快（按照内容查询）

• 缺点：查询速度没有HashSet快

## 三、map

* Map

• 特点 key-value映射

* HashMap

• Key无序 唯一 （Set）

• Value 无序 不唯一 （Collection）

* LinkedHashMap

• 有序的HashMap 速度快

* TreeMap

• 有序 速度没有hash快

* 问题：Set和Map有关系吗？

• 采用了相同的数据结构，只用于map的key存储数据，就是Set

* Vector和ArrayList的联系和区别
  + 实现原理相同，功能相同，都是长度可变的数组结构，很多情况下可以互用
  + 两者的主要区别如下
    - Vector是早期JDK接口，ArrayList是替代Vector的新接口
    - Vector线程安全，效率低下；ArrayList重速度轻安全，线程非安全
    - 长度需增长时，Vector默认增长一倍，ArrayList增长50%
* HashMap和Hashtable的联系和区别
  + 实现原理相同，功能相同，底层都是哈希表结构，查询速度快，在很多情况下可以互用
  + 两者的主要区别如下
    - Hashtable是早期JDK提供的接口，HashMap是新版JDK提供的接口
    - Hashtable继承Dictionary类，HashMap实现Map接口
    - Hashtable线程安全，HashMap线程非安全
    - Hashtable不允许null值，HashMap允许null值

# 第八章：多线程

## 一、程序、进程与线程

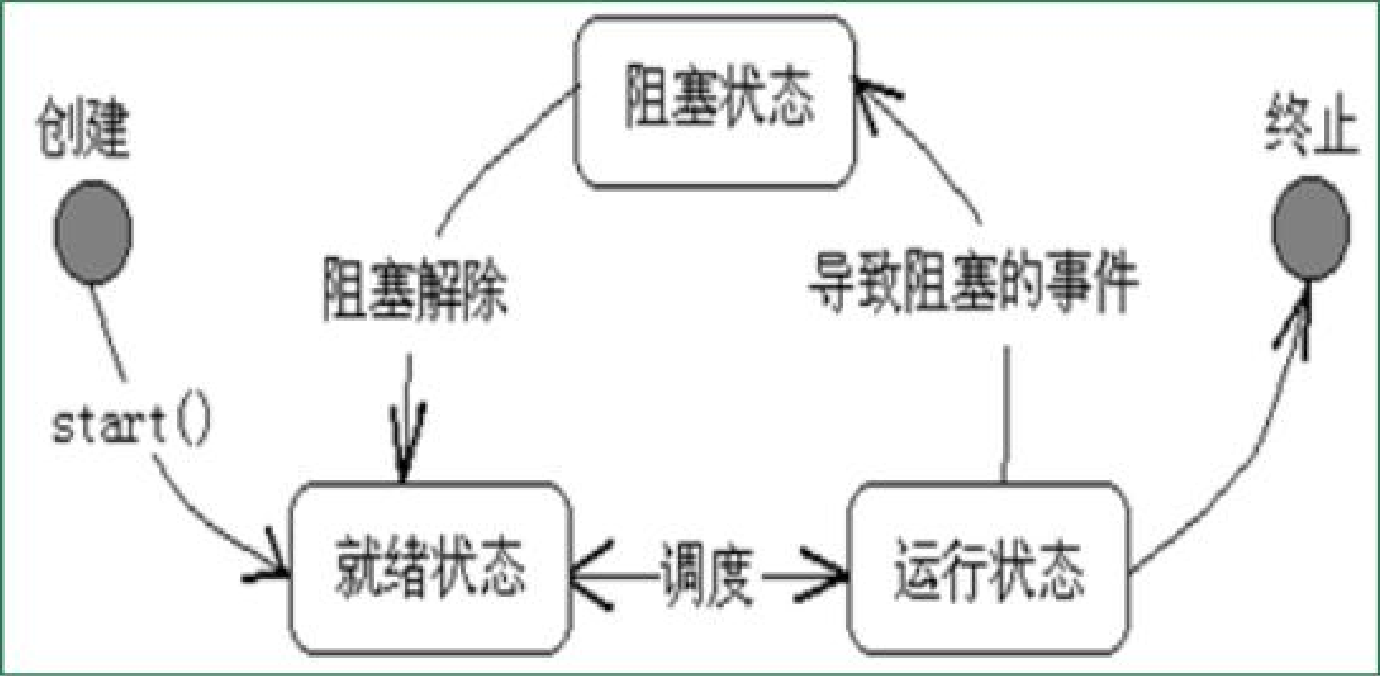
* 程序Program
  + 程序是一段静态的代码，它是应用程序执行的蓝本
* 进程Process
  + 进程是指一种正在运行的程序，有自己的地址空间
  + 进程的特点
    - 动态性
    - 并发性
    - 独立性
* 并发和并行的区别
  + 多个CPU同时执行多个任务
  + 一个CPU（采用时间片）同时执行多个任务
* 线程Thread
  + 进程内部的一个执行单元，它是程序中一个单一的顺序控制流程。
  + 线程又被称为轻量级进程(lightweight process)
  + 如果在一个进程中同时运行了多个线程，用来完成不同的工作，则称之为 多线程

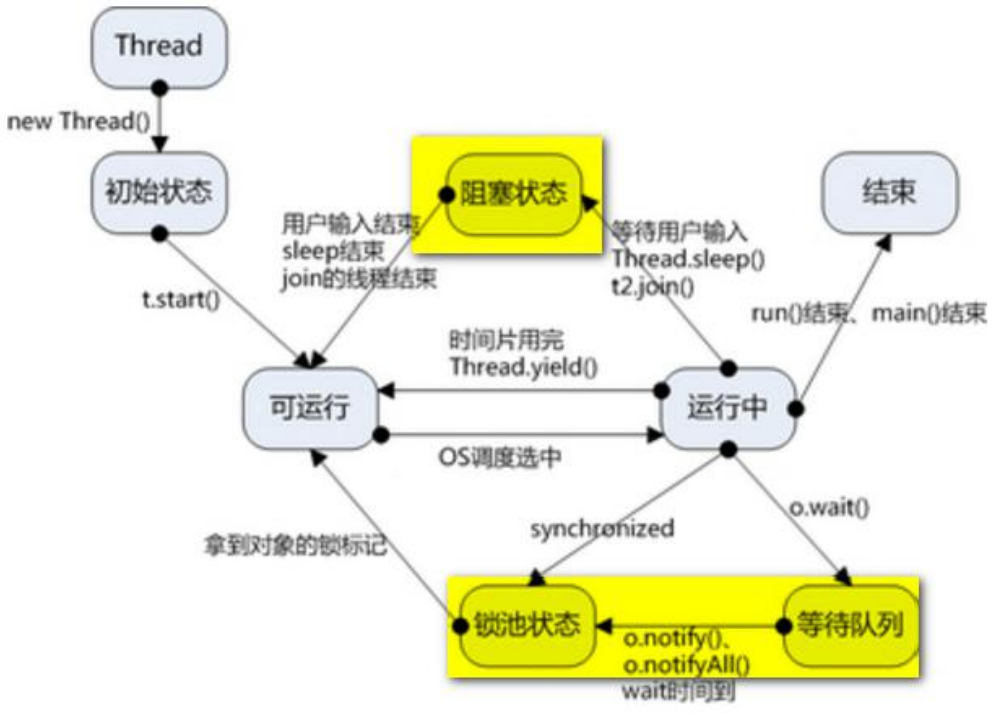


## 二、线程的创建

* 两种线程创建方式的比较
  + 继承Thread类方式的多线程
    - 优势：编写简单
    - 劣势：无法继承其它父类
  + 实现Runnable接口方式的多线程
    - 优势：可以继承其它类，多线程可共享同一个Runnable对象
    - 劣势：编程方式稍微复杂，如果需要访问当前线程，需要调用Thread.currentThread()方法
  + 实现Runnable接口方式要通用一些

## 三、线程的生命周期





* 新生状态：
  + 用new关键字建立一个线程对象后，该线程对象就处于新生状态。
  + 处于新生状态的线程有自己的内存空间，通过调用start进入就绪状态
* 就绪状态：
  + 处于就绪状态线程具备了运行条件，但还没分配到CPU，处于线程就绪队列，等待系统为其分配CPU
  + 当系统选定一个等待执行的线程后，它就会从就绪状态进入执行状态，该动作称之为“cpu调度”。
* 运行状态：
  + 在运行状态的线程执行自己的run方法中代码，直到等待某资源而阻塞或完成任务而死亡。
  + 如果在给定的时间片内没有执行结束，就会被系统给换下来回到等待执行状态。
* 阻塞状态：
  + 处于运行状态的线程在某些情况下，如执行了sleep（睡眠）方法，或等待I/O设备等资源，将让出CPU并暂时停止自己的运行，进入阻塞状态。
  + 在阻塞状态的线程不能进入就绪队列。只有当引起阻塞的原因消除时，如睡眠时间已到，或等待的I/O设备空闲下来，线程便转入就绪状态，重新到就绪队列中排队等待，被系统选中后从原来停止的位置开始继续运行。
* 死亡状态：
  + 死亡状态是线程生命周期中的最后一个阶段。线程死亡的原因有三个。一个是正常运行的线程完成了它的全部工作；另一个是线程被强制性地终止，如通过执行stop方法来终止一个线程[不推荐使用]，三是线程抛出未捕获的异常

## 四、线程池

* 什么是线程池
  + 创建和销毁对象是非常耗费时间的
  + 创建对象：需要分配内存等资源
  + 销毁对象：虽然不需要程序员操心，但是垃圾回收器会在后台一直跟踪并销毁
  + 对于经常创建和销毁、使用量特别大的资源，比如并发情况下的线程，对性能影响很大。
  + 思路：创建好多个线程，放入线程池中，使用时直接获取引用，不使用时放回池中。可以避免频繁创建销毁、实现重复利用
* 线程池的好处
  + 提高响应速度（减少了创建新线程的时间）
  + 降低资源消耗（重复利用线程池中线程，不需要每次都创建）
  + 提高线程的可管理性：避免线程无限制创建、从而销耗系统资源，降低系统稳定性，甚至内
  + 存溢出或者CPU耗尽

# 第九章：反射、类加载机制与注解

## 一、反射机制

* 指的是可以于运行时加载、探知、使用编译期间完全未知的类。
* 程序在运行状态中，可以动态加载一个只有名称的类，对于任意一个已加载的类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；

Class c = Class. forName ("com.whitecat.test.User");

## 获取Class对象

* 运用getClass()
* 运用Class.forName()（最常被使用）
* 运用.class 语法

## 三、动态编译

* 动态编译的两种做法：
  + 通过Runtime调用javac，启动新的进程去操作

Runtime run = Runtime.getRuntime();

Process process = run.exec("javac -cp d:/myjava/ HelloWorld.java");

* + 通过JavaCompiler动态编译

public static int compileFile(String sourceFile){

// 动态编译

JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();

int result = compiler.run(null, null, null,sourceFile);

System.out.println(result==0?" 编译成功 ":" 编译失败 ");

return result;

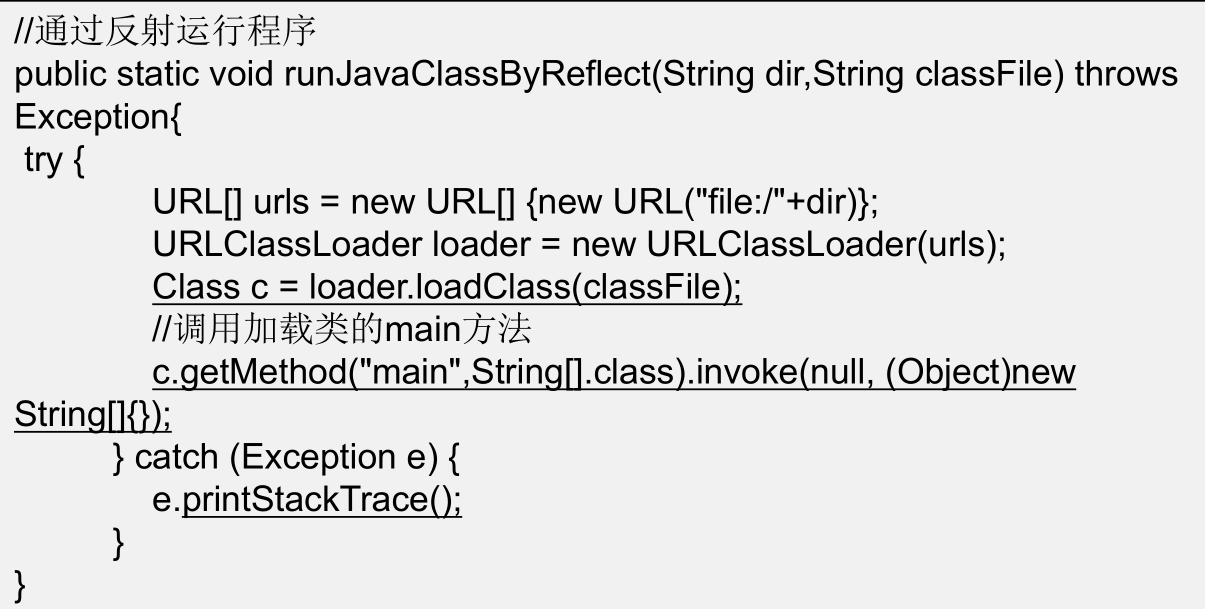
}

* 第一个参数： 为java编译器提供参数
  + - 第二个参数： 得到 Java 编译器的输出信息
    - 第三个参数： 接收编译器的 错误信息
    - 第四个参数： 可变参数（是一个String数组）能传入一个或多个 Java 源文件
    - 返回值： 0表示编译成功，非0表示编译失败
* 通过Runtime. getRuntime() 运行启动新的进程运行

Runtime run = Runtime.getRuntime();

Process process = run.exec("java -cp d:/myjava HelloWorld");

* 通过反射运行编译好的类



## 四、类加载器及其核心机制

### 类加载器的作用

* + 将class文件字节码内容加载到内存中，并将这些静态数据转换成方法区中的运行时数据结构，在堆中生成一个代表这个类的java.lang.Class对象，作为方法区类数据的访问入口。

### java.class.ClassLoader 类

* + 作用
    - java.lang.ClassLoader类的基本职责就是根据一个指定的类的名称，找到或者生成其对应的字节代码，然后从这些字节代码中定义出一个Java 类，即 java.lang.Class类的一个实例。
    - 除此之外，ClassLoader还负责加载 Java 应用所需的资源，如图像文件和配置文件等

### TOMCAT服务器的类加载机制

* 一切都是为了安全！
  + TOMCAT不能使用系统默认的类加载器。
    - 如果TOMCAT跑你的WEB项目使用系统的类加载器那是相当危险的，你可以直接是无忌惮是操作系统的各个目录了。

• 对于运行在 Java EE™容器中的 Web 应用来说，类加载器的实现方式与一般的 Java 应用有所不同。

* + - 每个 Web 应用都有一个对应的类加载器实例。该类加载器也使用代理模式(不同于前面说的双亲委托机制)，所不同的是它是首先尝试去加载某个类，如果找不到再代理给父类加载器。这与一般类加载器的顺序是相反的。但也是为了保证安全，这样核心库就不在查询范围之内。
* 为了安全TOMCAT需要实现自己的类加载器。
  + - 我可以限制你只能把类写在指定的地方，否则我不给你加载！

## 五、注解

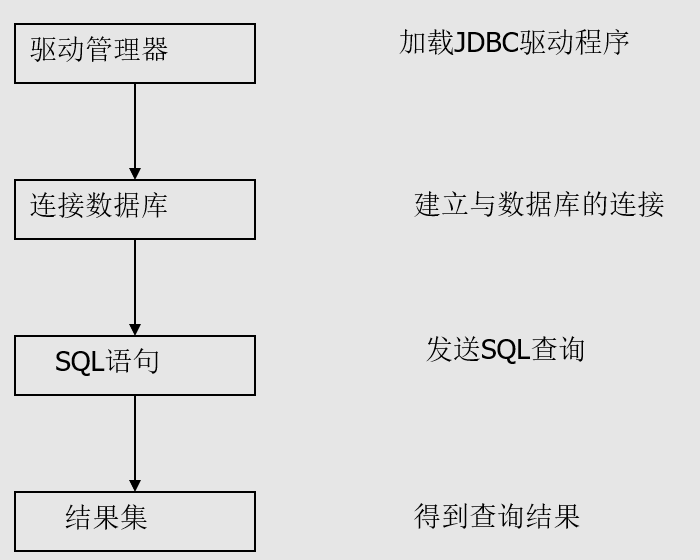
* Annotation的作用：

– 不是程序本身，可以对程序作出解释。(这一点，跟注释没什么区别)

– 可以被其他程序(比如：编译器等)读取。(注解信息处理流程，是注解和注释的重大区别。如果没有注解信息处理流程，则注解毫无意义)

# 第十章：jdbc

## 一、Jdbc访问数据库流程



* Driver接口
  + 在编程中要连接数据库，必须先装载特定厂商的数据库驱动程序。不同的数据库有不同的装载方法。
  + 驱动：就是各个数据库厂商实现的Sun公司提出的JDBC接口。 即对Connection等接口的实现类的jar文件
  + 装载MySql驱动
    - Class.forName("com.mysql.jdbc.Driver");
* DriverManager接口
  + DriverManager是JDBC的管理层，作用于用户和驱动程序之间。
  + DriverManager跟踪可用的驱动程序，并在数据库和相应的驱动程序之间建立连接。
* Connection**接口**
* Connection与特定数据库的连接（会话）,在连接上下文中执行 SQL 语句并返回结果。
* DriverManager的getConnection()方法建立在JDBC URL中定义的数据库Connection连接上
* 连接MYSQL数据库：

Connection con =DriverManager.getConnection("jdbc:mysql://host:port/database","user","password");

* Statement接口
* 用于执行静态 SQL 语句并返回它所生成结果的对象。
* 三种Statement类：
  + - * Statement：由createStatement创建，用于发送简单的SQL语句。(不带参数的)
      * PreparedStatement：继承自Statement接口，由prepareStatement创建，用于发送含有一个或多个输入参数的sql语句。PreparedStatement对象比Statement对象的**效率更高**，并且**可以防止SQL注入**。我们一般都用PreparedStatement.
      * CallableStatement：继承自PreparedStatement 。由方法prePareCall创建，用于调用**存储过程**。
* 常用的Statement方法：
  + - * execute()：运行语句，返回是否有结果集。
      * executeQuery()：运行select语句，返回ResultSet结果集。
      * executeUpdate()：运行insert/update/delete操作，返回更新的行数。
* ResultSet接口
  + Statement执行SQL语句时返回ResultSet结果集。
  + ResultSet提供的检索不同类型字段的方法，常用的有：
    - getString():获得在数据库里是varchar、char等数据类型的对象。
    - getFloat():获得杂数据库里是Float类型的对象。
    - getDate():获得在数据库里面是Date类型的数据。
    - getBoolean():获得在数据库里面是Boolean类型的数据