

复旦大学计算机科学技术学院



编程方法与技术

B.1. 上节课复习

周扬帆

2021-2022第一学期

线程控制

□ 休眠、把CPU时间给别的线程

■ 调用Thread.sleep(毫秒数)

```
class MyRunnable implements Runnable{
    @Override
    public void run() {
        System.out.println("Hello World!");
        try {
            for(;;) {
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("I am Interrupted");
        }
    }
}

Thread myThread = new Thread(new MyRunnable());
myThread.start();
...
myThread.interrupt();
```

休眠

wait/notify/notifyAll 机制

```
Object lock = new Object();
```

线程1:

```
...  
try {  
    lock.wait(); //等待被notify, 然后继续  
} catch (InterruptedException e) {  
}  
}
```

线程2:

```
...  
try {  
    lock.wait(); //等待被notify, 然后继续  
} catch (InterruptedException e) {  
}  
}
```

线程3:

```
lock.notifyAll(); //让线程1及线程2的lock.wait结束  
//lock.notify(); //让线程1或线程2的lock.wait结束
```

wait/notify/notifyAll 机制

□ wait()

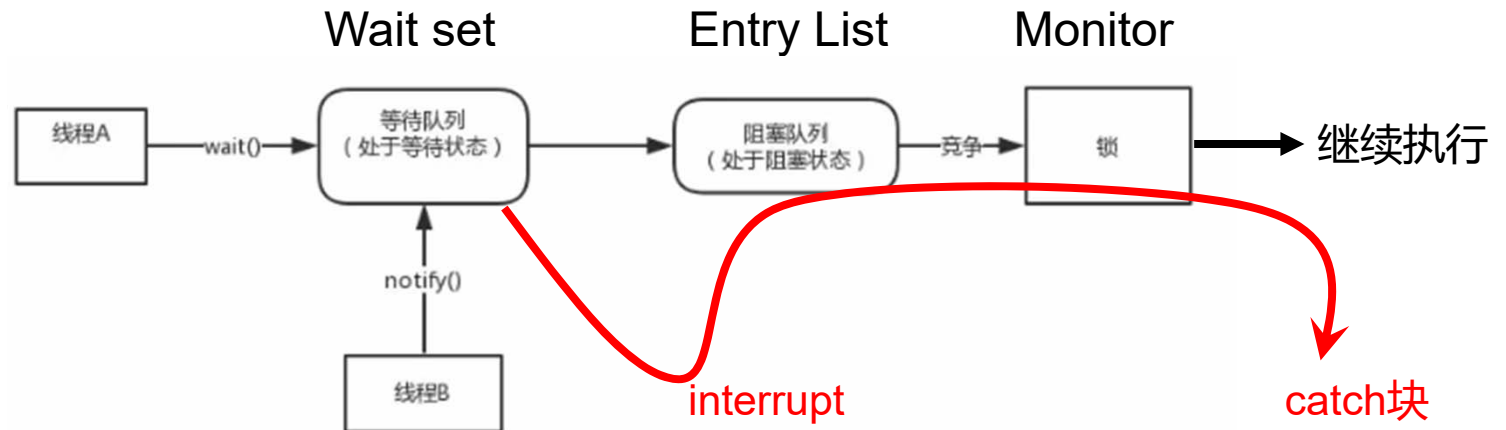
- 必须持有monitor, 否则抛异常
- 交出去monitor
- 然后在wait set上等着被notify
 - 被notify了之后去entry list等重新获得monitor, 继续执行
 - 被interrupt之后去等monitor, 然后进入catch块

□ notify()/notifyAll()

- 必须持有monitor, 否则抛异常
- wait set上的一个/**所有**线程被notify
- 不交出去monitor

```
lock.wait();  
...  
  
synchronized (lock) {  
    ...  
}
```

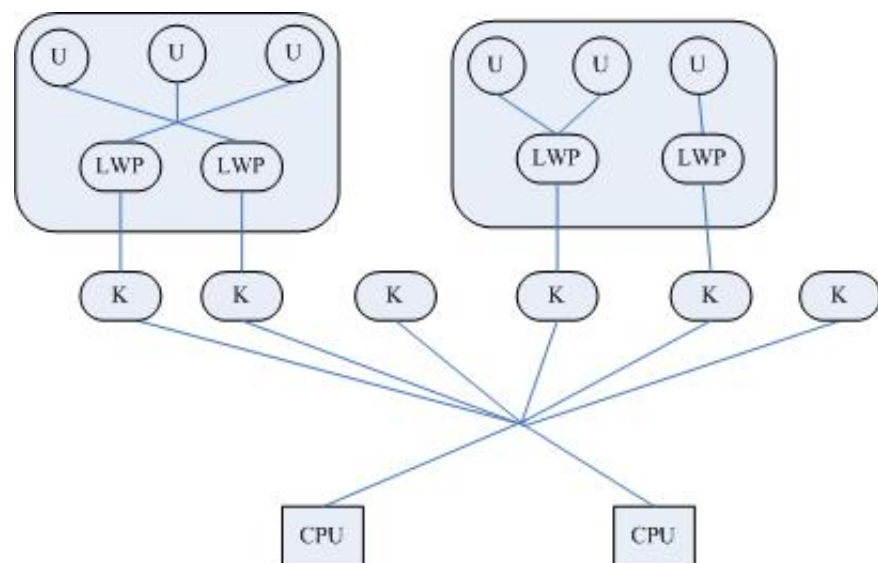
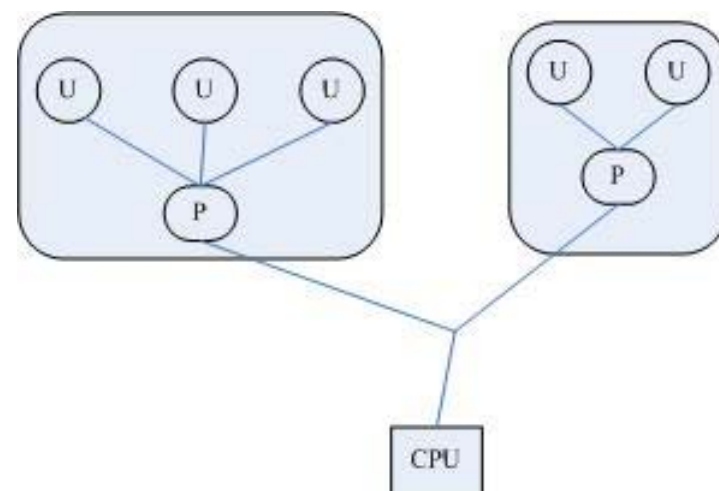
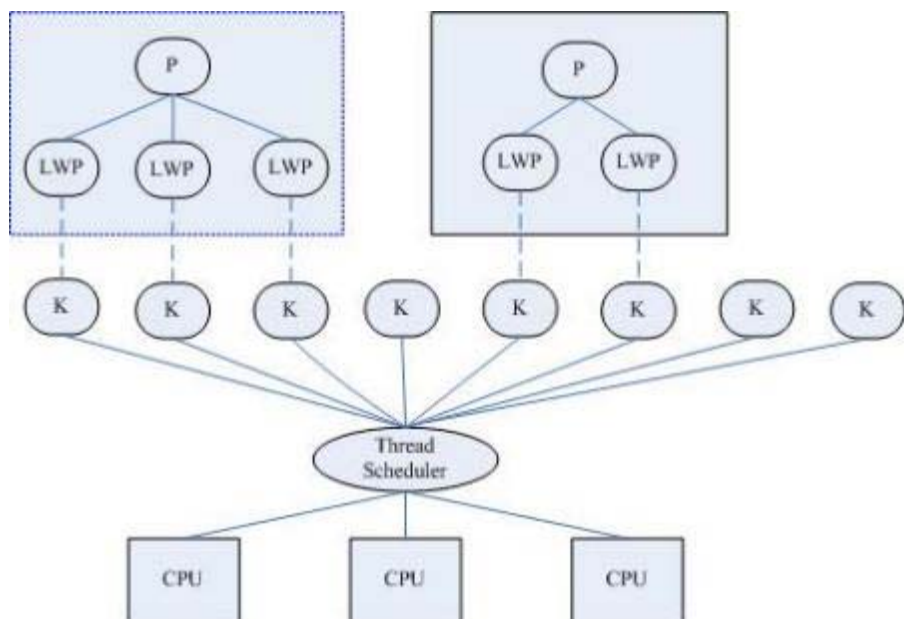
wait/notify/notifyAll 机制



```
new Thread() {  
    public void run() {  
        synchronized (lock) {  
            try {  
                lock.wait();  
            } catch (InterruptedException e) {  
                System.out.println("Interrupted");  
            }  
        }  
    }  
}.start();
```

同一个对象的monitor对应的同步块/方法，不可能有两条线程同时在跑

线程实现



ThreadPool: 线程池

线程调度

□ 优先级

- Thread类的setPriority/getPriority方法
- Thread.currentThread()获取当前的Thread的引用

Thread.currentThread().setPriority(Thread.MAX_PRIORITY)

主线程也可以用

- Java支持10种优先级
 - 最小: Thread.MIN_PRIORITY
 - 最大: Thread.MAX_PRIORITY
- Windows支持7种优先级, Solaris支持 2^{32} 种优先级
- 操作系统可能有对LWP的相应的优先级动态调整策略
- 优先级设置并不能严格保证的优先级次序

线程的执行过程

□ 等待线程结束: 调用join方法

```
class MyRunnable implements Runnable {
    public void run() {
        try {
            Thread.sleep(3000);
            System.out.println("thread ends");
        } catch (InterruptedException e) { }
    }
}

public class ThreadDemo {
    public static void main(String[] args) {
        Thread myThread = new Thread(new MyRunnable());
        myThread.start();
        try {
            myThread.join();
        } catch (InterruptedException e) { }
        System.out.println("main ends");
    }
}
```

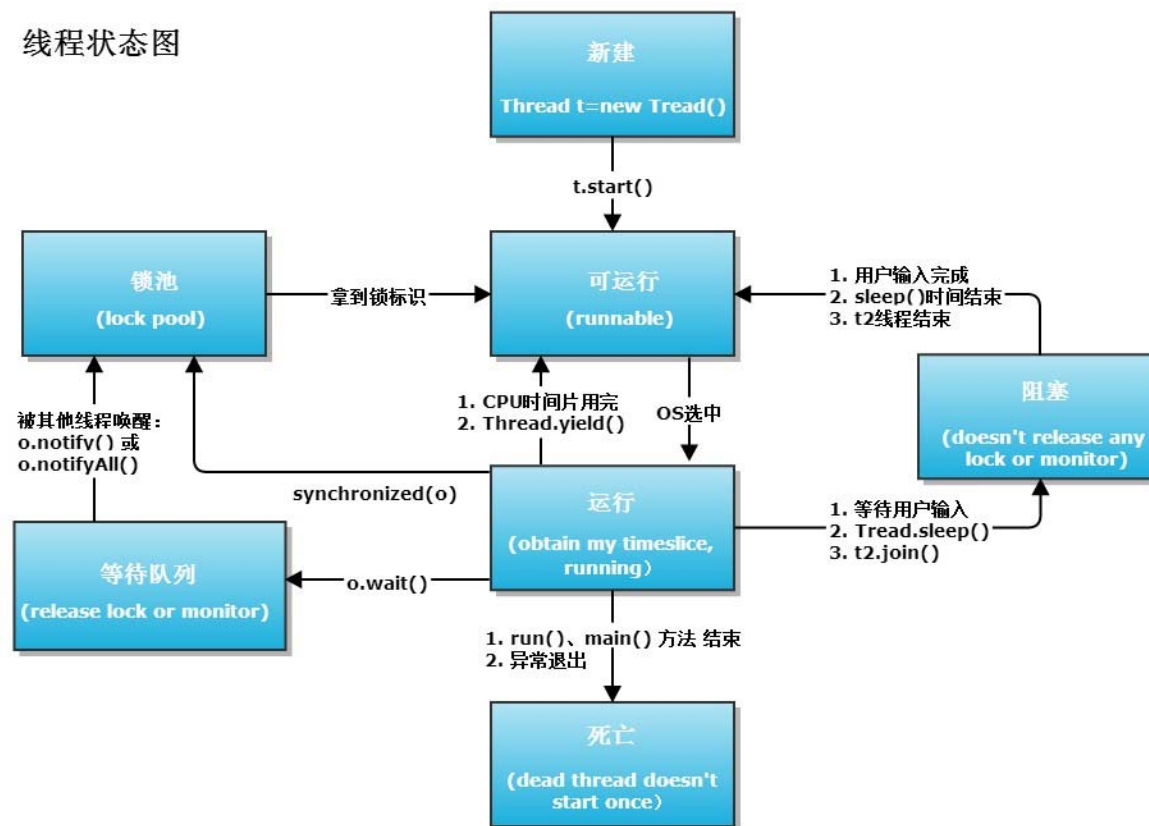
本线程开始等待
myThread线程结束

线程的执行过程

□ 线程的状态

- New
- Runnable
- Running
- Timed-waiting
- Blocked
- Waiting
- Terminated

线程状态图

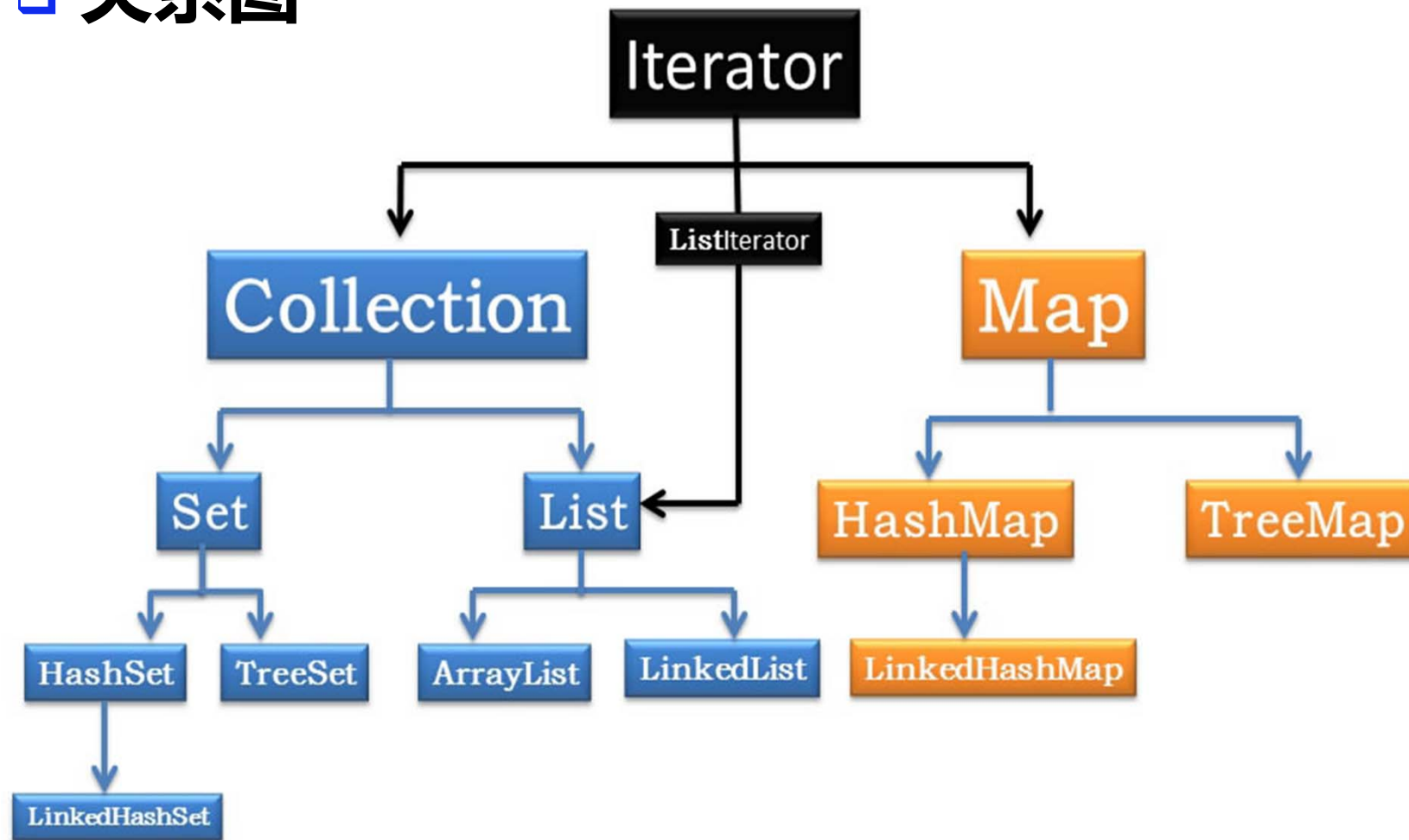


图片来源

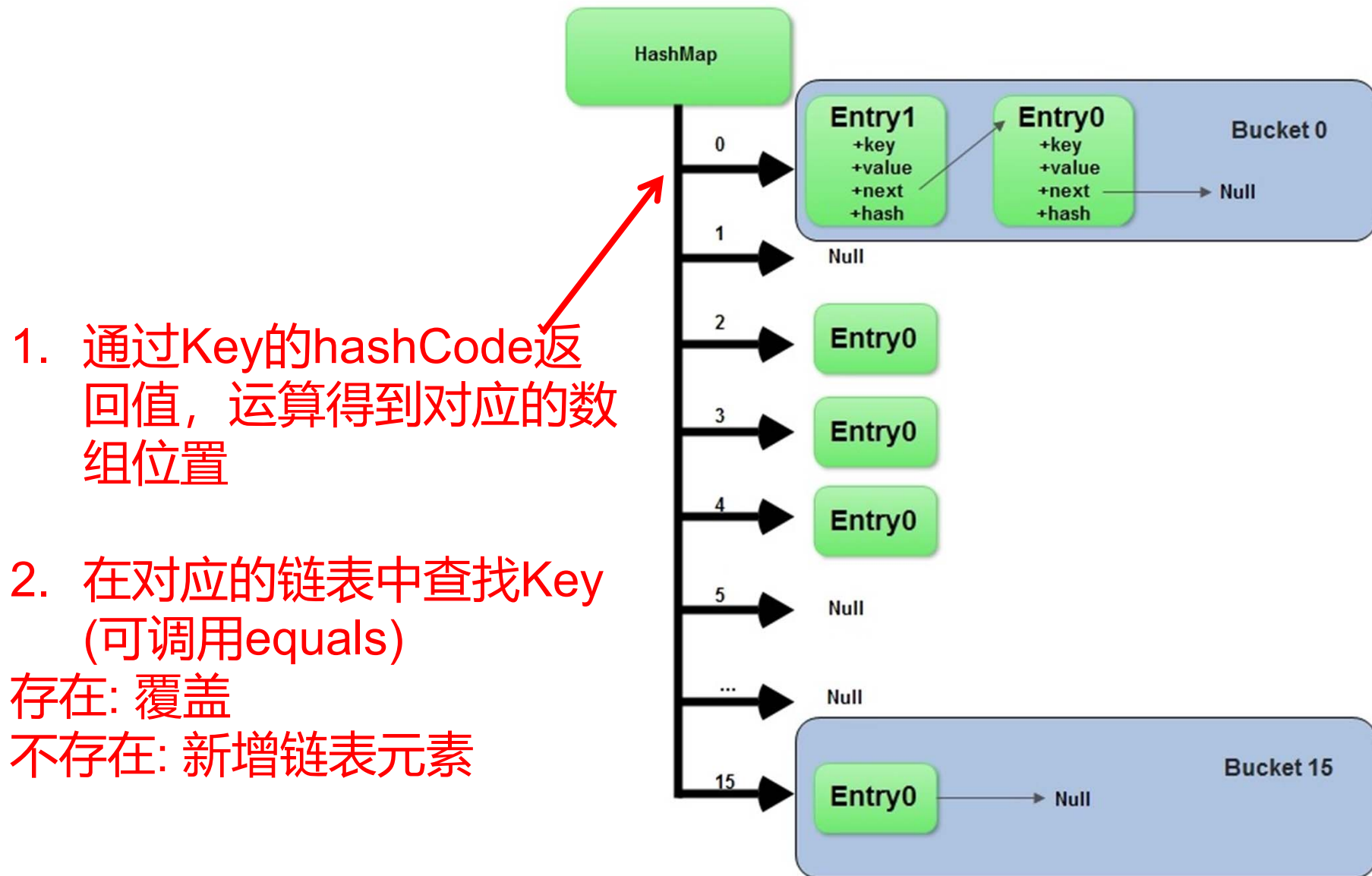
<https://my.oschina.net/mingdongcheng/blog/139263>

Java容器(集合)类

□ 关系图



HashMap的存储



关于hashCode和equals

- ❑ Object类的方法，可以被子类覆盖
 - `int hashCode()`
 - `boolean equals(Object o)`
- ❑ 为什么要用hashCode+equals方法判断重复
- ❑ 判断两个对象是否一样: equals方法
 - 设想一个对象管理的数据量很大，“一样”定义为各变量的值相等
 - equals方法需逐一比较各个域
- ❑ 要用hashCode+equals方法判断重复
 - 逐一调用equals → 如果对象复杂，equals会比较耗时
 - hashCode() → 可提取对象特征 → 大大加速查找

关于hashCode和equals

- 因为先比较的是hashCode，所以hashCode方法要和equal一致
 - 不能equal了，但hashCode不一样
 - 覆盖equals方法实现自己的比较方法，须记得覆盖hashCode方法
 - 只要equals方法的比较操作作用到的数据没有被修改，那么hashCode方法必须返回同一个整数

JavaScript Set

□ Set: 不能重复值的集合

```
var a = new Set([1,2]);  
a.add(3);  
console.log(a);  
a.add(3);  
console.log(a);
```

Set { 1, 2, 3 }

?? Set {1, 2, 3}

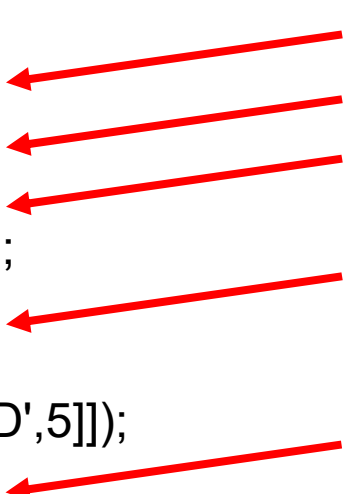
```
var b = new Set([1,2,2,2,3]);  
console.log(...a);
```

?? Set {1, 2, 3}

JavaScript Map

- ❑ 键值对集合
- ❑ 提供键值一一对应的结构

```
var a = new Map([['ID1',2], ['ID2',2]]);  
a.set('Name','YZ');  
console.log(a.has('Name'));  
console.log(a.get('Name'));  
console.log(a.get('ID1'));  
a.set('Name','A New Name');  
console.log(a.get('Name'));
```



true
YZ
2
A New Name


```
var b = new Map([['ID',4], ['ID',5]]);  
console.log(b.get('ID'));
```

5

复旦大学计算机科学技术学院



编程方法与技术

B.2. Java编译工具集

周扬帆

2021-2022第一学期

Java的编译、打包、运行

□ ./src/

- .java文件: 源文件

□ ./bin/

- .class文件: bytecodes编译结果

□ 编译

- javac命令: `javac Test.java`

打包成Windows程序?

□ 打包

- `jar cvf Test.jar Test.class`

□ 运行 (jvm)

- `java Test`
- `java -cp Test.jar Test`

```
public class Test {  
    public static void main(String argc[]) {  
        System.out.println("Hello!");  
    }  
}
```

Jar包

□ Jar

- Java Archive
- 基于zip格式的class文件压缩包

□ 运行

- `java -cp Test.jar Test`
 - -cp: 指定class path为Test.jar包
- `java -jar Test.jar`

□ 包配置

- Jar包可以包含一个META-INF目录，存放配置信息
- 其中包含MANIFEST.MF文件
 - 指定main函数位置
 - 版本信息等



Main-Class: Test

Jar包

□ 打包

- `jar cvf jarfile [classfiles]`
- `jar cvf jarfile [directory]`
 - 注意目录和package名的对应关系 (?)
- `jar cvfm jarfile manifestfile [classfiles]/[directory]`
 - 将manifestfile打包进去,
 - 自动存于META-INF目录下的MANIFEST.MF

Main-Class: com.ydroid.bigdata.LDATest

com/ydroid/bigdata/LDATest.class的main方法

□ 运行

- `java -jar XXX.jar` (windows下可直接点击jar打开)
- 自动找到META-INF/MANIFEST.MF从指定入口执行

思考

- 理解Java的编译运行流程，与你熟悉的语言，如C++比较。
- 思考Jar包与C++的静/动态链接的异同点

复旦大学计算机科学技术学院



编程方法与技术

B.3. 配置文件读写

周扬帆

2021-2022第一学期

配置文件

- **配置文件：定义软件系统动态加载的配置**
 - 环境设置: 如程序需要访问的远程数据库地址/端口等
 - 用户特定的设置: 如字体、颜色等
- **常见扩展名: .conf .ini .cfg .properties**
- **常见格式: 文本文件**
 - 每行为 Key = Value 格式
- **xml格式的配置文件**

```
### This file controls the configuration of
### the svnserve daemon, if you use it
### to allow access to this repository.
### (If you only allow access through
### http: and/or file: URLs, then this file
### is irrelevant.)
[general]
anon-access = none
auth-access = write
password-db = passwd
authz-db = authz
```

Java读取配置文件

□ 使用Properties类

■ **import** java.util.Properties;

```
FileReader fr;
```

```
try {
```

```
    String fileName = "test.conf";
```

```
    fr = new FileReader(fileName);
```

```
    Properties props = new Properties();
```

```
    props.load(fr);
```

```
    if (props.containsKey("name")) {
```

```
        System.out.println("name = " + props.get("name"));
```

```
    }
```

```
    else {
```

```
        ...
```

```
    }
```

```
} catch (IOException e) {
```

```
    ...
```

```
}
```

把文件流读入props中

name = XXX

props.containsKey("name") 返回true

props.get("name")返回name=后面的内容

配置文件


❑ 软件可以根据配置改变行为

■ 如：根据配置的地址访问服务器

```
...  
String serverAddress = props.get("server");  
...  
//connect to server  
...
```

■ 如：根据配置改变字体大小

```
...  
if(((String)props.get("fontsize")).compareToIgnoreCase("large") == 0) {  
    ...  
}  
else {  
    ...  
}
```



❑ 行为必须预先编程实现

运行时加载

- 能不能不预先编程，在运行时再加载适合的软件行为？
 - 插件 (plugin)
- Java反射机制 (reflection)

复旦大学计算机科学技术学院



编程方法与技术

B.4. 反射

周扬帆

2021-2022第一学期

反射

- 程序在**运行时**取得、甚至修改一个已知**名称**的类的内部信息
 - Modifiers: 如public, static 等
 - 父类
 - 实现的接口
 - 域和方法的所有信息，并可于运行时改变fields内容或调用methods。
- 程序在**运行时加载、使用编译期间未知的类**
 - 可加载一个运行时才知名称的类，获得其完整结构，并使用它

最简单的理解：

?

String className (类名) = "MyClass"; → 创建MyClass类对象myclass;

Java类

- ❑ 方法 (methods)
- ❑ 域 (fields)
- ❑ 父类 (super class)
- ❑ 修饰符(modifiers)
- ❑ 实现的接口 (interface)
- ❑ 包名 (package)
- ❑ ...

类本身有很多描述这个类特征的数据

JVM如何管理?

Class类

- **Class类是一种Object**
 - Object是所有类的继承根源
- **当一个类被加载，JVM 便自动产生一个Class对象**

//获取Test这个类的Class对象的引用

```
Class myClass = Test.class;
```

//获取Test对象test对应的Class对象的引用

```
Test test = new Test();
```

```
Class myClass = test.getClass();
```

- **Class类的构造函数是私有的**
 - 只允许JVM创建
 - 程序**不能** `Class a = new Class();`

Class类

- 当一个类被加载, JVM 便自动产生一个 Class对象

```
public class Reflection {  
    public static void main(String args[]) {  
        Reflection r1 = new Reflection();  
        Reflection r2 = new Reflection();  
        Class clazz1 = r1.getClass();  
        Class clazz2 = r2.getClass();  
        Class clazz3 = Reflection.class;  
        System.out.println(clazz1 == clazz2);  
        System.out.println(clazz1 == clazz3);  
    }  
}
```

反射

- 当一个类被加载，JVM 便自动产生一个 Class 对象

```
public class Reflection {  
    public static void main(String args[]) {  
        Reflection r1 = new Reflection();  
        Reflection r2 = new Reflection();  
        Class clazz1 = r1.getClass();  
        Class clazz2 = r2.getClass();  
        Class clazz3 = Reflection.class;  
        System.out.println(clazz1 == clazz2);  
        System.out.println(clazz1 == clazz3);  
    }  
}
```

只有 `Class clazz3 = Reflection.class;`
`Reflection` 的静态块会不会被执行？

Class类

- 当一个类被加载, JVM 便自动产生一个 Class对象

```
public class Reflection {  
    public static void main(String args[]) {  
        Reflection r1 = new Reflection();  
        Reflection r2 = new Reflection();  
        Class clazz1 = r1.getClass();  
        Class<? extends Reflection> clazz2 = r2.getClass();  
        Class<Reflection> clazz3 = Reflection.class;  
        System.out.println(clazz1 == clazz2);  
        System.out.println(clazz1 == clazz3);  
    }  
}
```


Class类

- **Class类有很多获取对应的class的信息的方法**
 - myclass.getPackage() / getSuperClass()
 - myclass.getFields() / getMethods / getModifiers
- **通过返回的信息，可以**
 - 构造类的对象
 - 修改域和对象的访问控制权限
 - setAccessible(true/false)
 - 修改域的数值
 - 调用相应的方法

反射例子1

```
interface TestInterface {
    void printInfo(String info);
}
class TestClass implements TestInterface {
    public void printInfo(String info) {
        System.out.println(info);
    }
}
public class Test {
    public static void main(String argc[]) {
        Class<?> clazz = TestClass.class;
        try {
            TestInterface test = (TestInterface)clazz.newInstance();
            test.printInfo("Hello");
        }
        catch(IllegalAccessException | InstantiationException e) {
            System.out.println(e);
        }
    }
}
```

反射例子2

```
import java.lang.reflect.Method;
public static void main(String argc[]) {
    try {
        Class<?> clazz = Class.forName ("TestClass");
        TestInterface test = (TestInterface)clazz.newInstance();
        //Parameter list
        Class<?> parameterTypes[] = new Class[] {String.class };
        Method m = clazz.getMethod("printInfo", parameterTypes);
        m.invoke(test, "Hello");
    }
    catch(InstantiationException |
           InvocationTargetException |
           NoSuchMethodException |
           IllegalAccessException |
           ClassNotFoundException e) {
        System.out.println(e);
    }
}
```

为什么try - catch

getMethod 公有, 包括继承来的

getDeclaredMethod 所有, 不包括继承来的

反射例子3

如果函数的参数是基本数据类型怎么办?

`void printInfo(int i, String info);`

```
import java.lang.reflect.Method;
public static void main(String argc[]) {
    try {
        Class<?> clazz = Class.forName ("TestClass");
        TestInterface test = (TestInterface)clazz.newInstance();
        //Parameter list
        Class<?> parameterTypes[] = new Class[]
            {int.class, String.class };
        Method m = clazz.getMethod("printInfo", parameterTypes);
        m.invoke(test, new Object[]{1, "Hello"});
        //m.invoke(test, 1, "Hello"); 也可以
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

反射例子4

```
public static void main(String argc[]) {  
    try {  
        URL [] urls = new URL[]{new URL("file:test.jar")};  
        URLClassLoader loader = new URLClassLoader(urls);  
        Class<?> clazz = loader.loadClass("TestClass");  
        TestInterface test = (TestInterface)clazz.newInstance();  
        //Parameter list  
        Class<?> parameterTypes[] = new Class[] {String.class };  
        Method m = clazz.getMethod("printInfo", parameterTypes);  
        m.invoke(test, "Hello");  
        loader.close();  
    } catch(Exception e) {  
        System.out.println(e);  
    }  
}
```

有什么用?

动态加载

反射例子4

□ 可用于实现插件

- 插件实现规定的Interface
- 插件编译成jar包
- 运行时根据信息（如配置文件）获得插件类名
- 创建对象
- cast成规定的Interface，并调用interface规定的接口

□ 插件的作用

- 方便（第三方）扩展程序
- 升级
- 针对需求（数据格式、用户需求等）改变程序行为
- 选择性加载（避免主程序过慢）

思考

- ❑ 思考反射的优缺点？思考它对面向对象原则的影响？
- ❑ 思考反射的应用场景

复旦大学计算机科学技术学院



编程方法与技术

B.5. isinstance关键字

周扬帆

2021-2022第一学期

instanceof关键字

- ❑ 布尔计算(boolean)
- ❑ 用法object instanceof Class
- ❑ a instanceof B
 - 判断a是不是B这个类或者接口的实例
 - 返回true
 - 如果B是a对应的类 a = new B();
 - 或者, B是a对应的类的父类
 - 或者, B是a对应的类实现的接口

instanceof关键字

- 常用于判断某个定义为父类的对象是不是事实上是一个子类的对象
 - 回忆里式替换法则

```
class Dog extends Animal ...
```

```
Animal animal = new Dog();  
callSomething(animal); ...
```

```
...callsomething(Animal animal) {  
    if (animal instanceof Dog) {  
        ((Dog)animal).bark(); //安全调用Dog的方法  
    }  
}
```

```
v.s.      try {  
            Dog dog = (Dog)animal;  
            dog.bark();  
        } catch (ClassCastException e) {  
        }  
}
```

instanceof关键字

- instanceof用于判断: true包括接口, 父类
 - 有什么办法判断exactly就是某个类

```
class Dog extends Animal ...
```

```
Animal animal = new Dog();
```

```
-- animal instanceof Animal  
-- animal instanceof Dog  
-- animal.getClass() == Animal.class  
-- animal.getClass() == Dog.class
```

复旦大学计算机科学技术学院



编程方法与技术

B.5. 其他

周扬帆

2021-2022第一学期

关于能发布的程序

□ Java编译工具集

■ javac编译

■ jar打包

→ jar包是一种zip格式的文件

→ Manifest文件指定

■ 去哪找入口 **Main-Class:**

■ 去哪找其他jar依赖 **Class-Path:**

■ java运行

→ 可以指定虚拟机堆空间大小

■ Young generations

□ Eden/From Survivor/To Survivor

■ Old generation

→ 指定Permanent generation大小

关于能发布的程序

□ Java判断当前jvm版本等系统信息

```
Properties prop = public static Properties getProperties();  
prop.getProperties("os.version");
```

java.version	Java 运行时环境版本	java.io.tmpdir	默认的临时文件路径
java.vendor	Java 运行时环境供应商	java.compiler	要使用的 JIT 编译器的名称
java.vendor.url	Java 供应商的 URL	java.ext.dirs	一个或多个扩展目录的路径
java.home	Java 安装目录	os.name	操作系统的名称
java.vm.specification.version	Java 虚拟机规范版本	os.arch	操作系统的架构
java.vm.specification.vendor	Java 虚拟机规范供应商	os.version	操作系统的版本
java.vm.specification.name	Java 虚拟机规范名称	file.separator	文件分隔符 (在 UNIX 系统中是"/")
java.vm.version	Java 虚拟机实现版本	path.separator	路径分隔符 (在 UNIX 系统中是".")
java.vm.vendor	Java 虚拟机实现供应商	line.separator	行分隔符 (在 UNIX 系统中是"/n")
java.vm.name	Java 虚拟机实现名称	user.name	用户的账户名称
java.specification.version	Java 运行时环境规范版本	user.home	用户的主目录
java.specification.vendor	Java 运行时环境规范供应商	user.dir	用户的当前工作目录
java.specification.name	Java 运行时环境规范名称		
java.class.version	Java 类格式版本号		
java.class.path	Java 类路径		
java.library.path	加载库时搜索的路径列表		

关于能发布的程序

□ 使用Properties类读取配置

```
FileReader fr;  
try {  
    String fileName = "test.conf";  
    fr = new FileReader(fileName);  
    Properties props = new Properties();  
    props.load(fr);  
    if (props.containsKey("name")) {  
        System.out.println("name = " + props.getProperty("name"));  
    }  
    else {  
        ...  
    }  
} catch (IOException e) {  
    ...  
}
```

关于能发布的程序

- 可以应用反射: 动态加载插件

复旦大学计算机科学技术学院



编程方法与技术

B.6. 课堂练习

周扬帆

2021-2022第一学期

编译-打包、插件

- ❑ **主程序：实现一个排序算法的接口(interface)**
 - 接口方法： `int[] sort(int [] input);`
 - 接口和主程序包名： `package edu.fudan`
- ❑ **两个插件：根据接口实现两个排序算法**
 - 排序方法前面println算法名字（两个不一样）
- ❑ **主程序读配置文件./sort.conf决定load那个插件**
 - 调用插件排序（反射）

编译-打包、插件

❑ 排序算法

- 简单就可以，两个可以一样
- 两个算法包名必须不一样
 - package edu.wjc
 - package edu.jw

❑ 全部编译成jar

- 三个jar

❑ 打包成一个zip

- 根目录有三个jar，主程序叫sort.jar
- 根目录有配置文件sort.conf
- 运行jar -jar sort.jar, 根据sort.conf的配置加载相应的排序插件
- 根目录有程序代码src目录（注意目录结构）

❑ DDL: 23:59, DEC-8 (下周三)