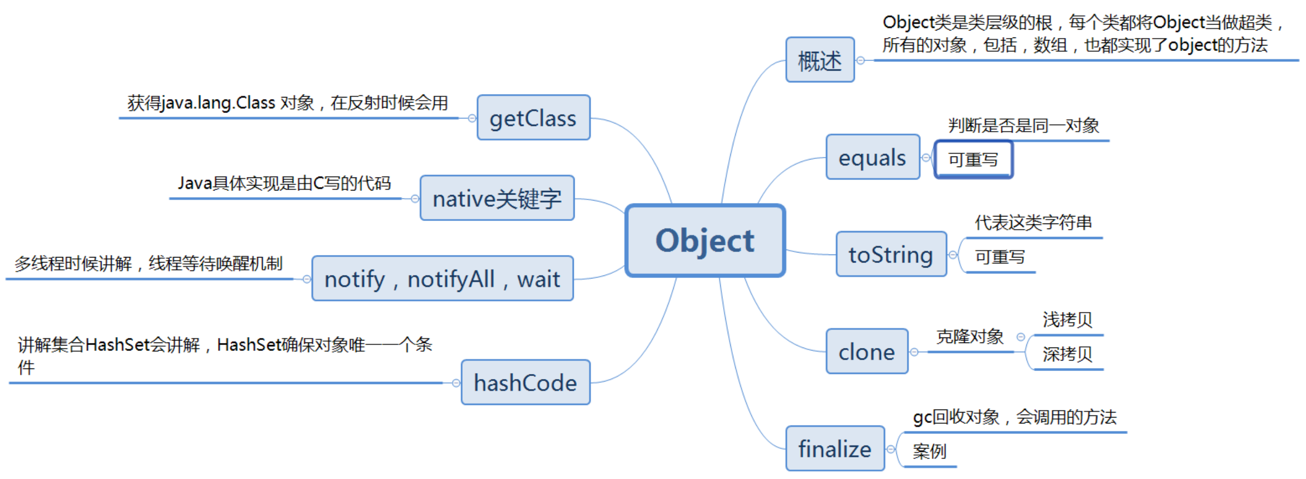
* 今天学习目标
  * 能够掌握Object类常见的方法
    * 所有父类的超类（父类，基类）
    * equals,toString
    * 常用...
    * clone:浅拷贝，深拷贝
    * native,finalize
  * 能够掌握DecimalFormat对数字的格式化
    * double d=0.1 --->10%
  * 能够在科学计算/财务计算时使用BigInteger/BigDecimal
    * 精度不够精确，范围不够大（long）
    * double d1=2； double d2=1.1； --->0.8999999999
  * 能够掌握日期相关的类
    * Date， Calendar，SImpleDateFormat，LocalDate,LocalTime,LocalDateTime

----

* 能够掌握Object类常见的方法


Object
概述
Object类是类层级的根，每个类都将Object当做超类，所有的对象，包括，数组，也都实现了object的方法
getClass
获得java.lang.Class 对象，在反射时候会用
native关键字
Java具体实现是由C写的代码
notify，notifyAll，wait
多线程时候讲解，线程等待唤醒机制
hashCode
讲解集合HashSet会讲解，HashSet确保对象唯一一个条件
equals
判断是否是同一对象
可重写
toString
代表这类字符串
可重写
clone
克隆对象
浅拷贝
深拷贝
finalize
gc回收对象，会调用的方法
案例

* 查看Object的源码

* Object的概述

* Object类是类层级的根，每个类都将Object当做超类，所有的对象，包括，数组，也都实现了Object的方法

* equals

* 默认比较当前对象引用

* 回顾String重写equals方法，比较是内容

* 定义JavaBean重写equals方法

```
* 为什么要重写equals
public class User {
    private int id;
    private String name;

    public User() {
        super();
    }
    public User(int id, String name) {
        super();
        this.id = id;
        this.name = name;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

}
public static void main(String[] args) {
        User u1=new User(1001,"xiaohei");
        User u2=new User(1001,"xiaohei");
        System.out.println(u1.equals(u2));
}
结果: false
  * 当User的id和名字一样，在现实中应该是同一用户，但是程序判断为false,因为它默认只对比

* 编写思路
1 判断是否同一对象，假如是返回true
2 判断传进来对象是否为空，假如为null，返回false
3 判断传进来的对象是否与当前的对象Class一样，假如不一样，返回false
4 强制转换类型
5 判断所选属性是否一样（属性对象要做非空判断），假如不一样返回false
6 全部通过返回true
@Override
    public boolean equals(Object obj) {
//      1 判断是否同一对象，假如是返回true
        if(obj==this) {
            return true;
        }
//      2 判断传进来对象是否为空，假如为null，返回false
        if(obj==null) {
            return false;
        }
//      3 判断传进来的对象是否与当前的对象Class一样，假如不一样，返回false
        if(!this.getClass().equals(obj.getClass())) {
            return false;
        }
//      4 强制转换类型
        User other=(User)obj;
//      5 判断所选属性是否一样（属性对象要做非空判断），假如不一样返回false
        if(id !=other.id) {
            return false;
        }
        if(name==null) {
            if(other.name!=null) {
                return false;
            }
        }else if(!name.equals(other.name)) {
            return false;
        }
//      6 全部通过返回true
        return true;
    }
 public static void main(String[] args) {
        User u1=new User(1001,"xiaohei");
        User u2=new User(1001,"xiaohei");
        System.out.println(u1.equals(u2));
    }
 结果为：true
* 可以自动生成
```

* toString

  * 默认是：全类名+'@'+hashcode十六进制值

    * getClass().getName() + "@" + Integer.toHexString(hashCode())

  * 例子

```
    public static void main(String[] args) {
        User u1=new User(1001,"xiaohei");
        System.out.println(u1.toString());
    }
  结果：com.lg.test1.User@84a40cc5
```

  * 重写

```
@Override
public String toString() {
    return "用户id: "+id+";用户名: "+name;
}
 结果：用户id：1001;用户名：xiaohei

可自动生成
```

* clone

    * 浅拷贝

        * 基本数据类型直接copy值

        * 引用copy引用地址值

```java
* 需要拷贝对象，需要
    * 实现Cloneable接口
    * 重写clone方法
public class Address {
    private String province;// 省份
    private String city;// 城市
    private String area;// 区
    private String street;// 街道
    // 构造器,set/get,toString
}


public class Person implements Cloneable {
    private int age;// 年龄
    private String name;// 姓名
    private Address address;// 地址
    // 构造器,set/get,toString
    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

}

public static void main(String[] args) throws CloneNotSupportedException {
        // 浅拷贝
        Person p1=new Person();
        p1.setAge(20);
        p1.setName("xiaohei");
        Address address=new Address("广东省","广州市","天河区","黄村街道3号");
        p1.setAddress(address);
        Person p2=(Person)p1.clone();
        System.out.println("当前是否是同一对象:"+(p1==p2));
        System.out.println("属性引用是否同一对象"+(p1.getAddress()==p2.getAddress
        System.out.println(p1);
        System.out.println(p2);
        System.out.println("---------------------------");
        p2.setAge(30);
        p2.setName("xiaobai");
        Address address2=p2.getAddress();
        address2.setStreet("车陂街道3号");
        p2.setAddress(address2);
        System.out.println("当前是否是同一对象:"+(p1==p2));
        System.out.println("属性引用是否同一对象"+(p1.getAddress()==p2.getAddress
        System.out.println(p1);
        System.out.println(p2);
    }

结果:
当前是否是同一对象:false
属性引用是否同一对象true
Person [age=20, name=xiaohei, address=Address [province=广东省, city=广州市, are
Person [age=20, name=xiaohei, address=Address [province=广东省, city=广州市, are
---------------------------
当前是否是同一对象:false
属性引用是否同一对象true
Person [age=20, name=xiaohei, address=Address [province=广东省, city=广州市, are
Person [age=30, name=xiaobai, address=Address [province=广东省, city=广州市, are

```

* 深拷贝

```
 * 在浅拷贝基础修改
 * Address 实现 Cloneable
 * Address 重写 clone方法
 * Person 修改clone方法
   @Override
   protected Object clone() throws CloneNotSupportedException {
       // 调用父类clone方法
       Person person=(Person) super.clone();
       address=(Address) person.address.clone();
       return person;
```

```
    }

* 测试结果
当前是否是同一对象:false
属性引用是否同一对象false
Person [age=20, name=xiaohei, address=Address [province=广东省, city=广州市, are
Person [age=20, name=xiaohei, address=Address [province=广东省, city=广州市, are
---------------------------
当前是否是同一对象:false
属性引用是否同一对象false
Person [age=20, name=xiaohei, address=Address [province=广东省, city=广州市, are
Person [age=30, name=xiaobai, address=Address [province=广东省, city=广州市, are

```

* finalize

* Java垃圾回收器在某个时机（例如内存不足的时候）回收空引用的时候，会触发此方法

```
public class Person {
    String name;

    @Override
    protected void finalize() throws Throwable {
        super.finalize();
        System.out.println("垃圾回收器回收Person对象...");
    }
}

public static void main(String[] args) throws InterruptedException {
        Person p=new Person();
        p.name="xiaohei";
        p=null;
        String str="";
        int i=0;
        while(true) {
            str+=i;
            i++;
        }
//      Runtime.getRuntime().gc();
    }
结果：
垃圾回收器回收Person对象...
```

* 能够掌握DecimalFormat对数字的格式化

* setMaximumFractionDigits，setMaximumIntegerDigits，setMinimumFractionDigits setMinimumIntegerDigits

```
public static void main(String[] args) {
        DecimalFormat df=new DecimalFormat();
        double d=888.66666666;
        // 默认显示3位小数
        System.out.println(df.format(d));
        System.out.println("---------------------------");
        // 设置小数点后最大位数为6
        df.setMaximumFractionDigits(6);
        // 设置正数最大多少位
        df.setMaximumIntegerDigits(15);
        System.out.println(df.format(d));
        System.out.println("---------------------------");
        // 设置小数点后最小位数为10（不够会自动后面补0）
        df.setMinimumFractionDigits(10);
        // 设置正数最小位数为10（不够会自动补前面0）
        df.setMinimumIntegerDigits(10);
        System.out.println(df.format(d));

    }
结果：
888.667
---------------------------
888.666667
---------------------------
0,000,000,888.6666666600
```

* DecimalFormatSymbols， setGroupingSize

  * setGroupingSeparator，setDecimalSeparator

```
public static void main(String[] args) {
        double d=88888.66666666;
        DecimalFormat df=new DecimalFormat();
        df.setMaximumFractionDigits(8);
        df.setGroupingSize(2);
        DecimalFormatSymbols sfs = df.getDecimalFormatSymbols();
        sfs.setGroupingSeparator(';'); //设置分组分隔符(默认是，)
        sfs.setDecimalSeparator('p'); //设置小数点分隔符（默认是.）
        df.setDecimalFormatSymbols(sfs);
        System.out.println(df.format(d));
        System.out.println("------------------------");
        //取消分组
        df.setGroupingUsed(false);//
        System.out.println(df.format(d));
    }
结果:
8;88;88p66666666
------------------------
88888p66666666
```

 * pattern：##%##,00%00

```
public static void main(String[] args) {
        double a=1.220;
        double b=11.33;
        double c=0.26666;
        DecimalFormat df=new DecimalFormat();
        df.applyPattern("##.##%");
//      df.applyPattern("000.00%");
        System.out.println(df.format(a));
        System.out.println(df.format(b));
        System.out.println(df.format(c));
    }
结果:
```

```
122%
1133%
26.67%
```

* 能够在科学计算/财务计算时使用BigInteger/BigDecimal

* 概述：JAVA中有两个用于表示大数值的类BigInteger和BigDecimal，可以表示任意长度、任意精度。当整数跟浮点数的取值范围或精度不能满足要求时，就需要用更大或者精度更高的类型BigInteger和BigDecimal了。

```
* BigDecimal
public static void main(String[] args) {
        double d1=2;
        double d2=1.2;
        double d3=1.1;
        System.out.println("double 类型运算结果: "+(d1-d2));
        System.out.println("double 类型运算结果: "+(d1-d3));

        BigDecimal bd1=BigDecimal.valueOf(2);
        BigDecimal bd2=BigDecimal.valueOf(1.2);
        BigDecimal bd3=BigDecimal.valueOf(1.1);

        System.out.println("BigDecimal 类型运算结果: "+(bd1.subtract(bd2).double
        System.out.println("BigDecimal 类型运算结果: "+(bd1.subtract(bd3).double
    }

* 结果
double 类型运算结果: 0.8
double 类型运算结果: 0.8999999999999999
BigDecimal 类型运算结果: 0.8
BigDecimal 类型运算结果: 0.9

* BigInteger
public static void main(String[] args) {
        //正常情况下一个整数最多只能放在long类型之中,但是假如超过long的最大值了, 可以
//      int i=2147483648; // 最大值: 2147483647
//      long l=9223372036854774808; // 最大值: 9223372036854774807
        BigInteger bi=new BigInteger("9223372036854774808");
        System.out.println(bi.subtract(new BigInteger("1")));
    }

结果:
9223372036854774807

```

* 能够掌握日期相关的类

  * Date， Calendar，SImpleDateFormat，LocalDate,LocalTime,LocalDateTime

 * Date

```

public static void main(String[] args) {
        Date d=new Date();
        System.out.println(d.toString());
        System.out.println(d.getTime());// 1970-01-01 00:00:00 开始计算的
    }
* 结果
Mon Aug 12 22:13:13 CST 2019
1565619193827

* 断点提示属性
* 此类的很多方法都过时了，建议是用Calendar
```

 * Calendar

```

public static void main(String[] args) {
        Calendar calendar = Calendar.getInstance();
        int year = calendar.get(Calendar.YEAR);
        int month=calendar.get(Calendar.MONTH)+1;
        int day=calendar.get(Calendar.DAY_OF_MONTH);
        System.out.println("Calendar类获得的时间"+year+":"+month+":"+day);
    }
```

```
结果:
Calendar类获得的时间2019:8:12

```

* 日期的转换

```

public static void main(String[] args) throws ParseException {
        // 时间毫秒值转换成日期，再格式化成字符串
        long now=System.currentTimeMillis();
        Date d=new Date(now);
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        String dateStr = sdf.format(d);
        System.out.println(dateStr);

        // 日期字符串转换成日期(str-->date-->calendar)
        Date date = sdf.parse(dateStr);
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(date);
        System.out.println(calendar.get(Calendar.YEAR));
    }

* 结果
    2019-08-12 22:46:45
    2019

```

* 能够掌握日期相关的类

 * Date

```

public static void main(String[] args) {
        Date d=new Date();
        System.out.println(d.toString());
        System.out.println(d.getTime());// 1970-01-01 00:00:00 开始计算的
    }
* 结果
Mon Aug 12 22:13:13 CST 2019
1565619193827
```

```

* 断点提示属性
* 此类的很多方法都过时了，建议是用Calendar
```

* Calendar

```
public static void main(String[] args) {
        Calendar calendar = Calendar.getInstance();
        int year = calendar.get(Calendar.YEAR);
        int month=calendar.get(Calendar.MONTH)+1;
        int day=calendar.get(Calendar.DAY_OF_MONTH);
        System.out.println("Calendar类获得的时间"+year+":"+month+":"+day);
    }
结果:
Calendar类获得的时间2019:8:12
```

* 日期的转换

```
public static void main(String[] args) throws ParseException {
        // 时间毫秒值转换成日期，再格式化成字符串
        long now=System.currentTimeMillis();
        Date d=new Date(now);
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        String dateStr = sdf.format(d);
        System.out.println(dateStr);

        // 日期字符串转换成日期(str-->date-->calendar)
        Date date = sdf.parse(dateStr);
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(date);
        System.out.println(calendar.get(Calendar.YEAR));
    }

* 结果
    2019-08-12 22:46:45
    2019
```

* jdk1.8日期类：LocalDate，LocalTime，LocalDateTime

```
* LocalDate
public static void main(String[] args) {
        LocalDate today = LocalDate.now();
        System.out.println("今天的日期是: "+today);
        int year = today.getYear();
        int monthValue = today.getMonthValue();
        int dayOfMonth = today.getDayOfMonth();
        System.out.println("今天的日期是: "+year+"年"+monthValue+"月"+dayOfMonth
        LocalDate years = today.plusYears(1);
        System.out.println("明年的今天是: "+years);
        LocalDate months = today.plusMonths(2);
        System.out.println("两个月后是: "+months);
        LocalDate minusMonths = today.minusMonths(2);
        System.out.println("两个月前是: "+minusMonths);
        LocalDate localDate = LocalDate.of(2019, 8, 25);//月份从1开始，与之前版本
        System.out.println("指定的日期是: "+localDate);
}
结果:
今天的日期是: 2019-08-13
今天的日期是: 2019年8月13日
明年的今天是: 2020-08-13
两个月后是: 2019-10-13
两个月前是: 2019-06-13
指定的日期是: 2019-08-25

public static void main(String[] args) {
        LocalDate today = LocalDate.now();
        System.out.println("今天是: " + today);
        boolean leapYear = today.isLeapYear();
        System.out.println("今年是闰年么: " + leapYear);
        LocalDate localDate = LocalDate.of(2019, 8, 25);
        System.out.println("今天是2019-08-25么: " + localDate.equals(today));
        boolean before = today.isBefore(localDate);
        System.out.println("今天在2019-08-25之前么: " + before);
        Period between = Period.between(today, localDate);
        int days = between.getDays();
        System.out.println("今天与2019-08-25相差几天: " + days);
}
结果:
今天是: 2019-08-13
今年是闰年么: false
今天是2019-08-25么: false
今天在2019-08-25之前么: true
今天与2019-08-25相差几天: 12

public static void main(String[] args) {
         //LocalDate today = LocalDate.now();
        LocalDate dayOfBirth = LocalDate.of(1998, 8, 13);
        MonthDay birthDay = MonthDay.of(dayOfBirth.getMonth(), dayOfBirth.getDa
        MonthDay now = MonthDay.now();
        if(now.equals(birthDay)){
            System.out.println("今天是你的生日");
        }else {
            System.out.println("今天不是你的生日");
        }
    }
结果:
今天是你的生日

* LocalTime
public static void main(String[] args) {
        LocalTime now = LocalTime.now();
        System.out.println("当前时间是: "+now);
        LocalTime plusHours = now.plusHours(2);
        System.out.println("两小时后是: "+plusHours);
        LocalTime minusHours = now.minusHours(2);
        System.out.println("两个小时前是: "+minusHours);
        LocalTime plusMinutes = now.plusMinutes(40);
        System.out.println("40分钟后是: "+plusMinutes);
}
结果:
当前时间是: 09:14:23.960
两小时后是: 11:14:23.960
两个小时前是: 07:14:23.960
40分钟后是: 09:54:23.960
```

```

* LocalDateTime
public static void main(String[] args) throws InterruptedException {
        LocalDateTime now = LocalDateTime.now();
        System.out.println("now:"+now);
        LocalDate localDate = now.toLocalDate();
        LocalTime localTime = now.toLocalTime();
        String format = now.format(DateTimeFormatter.ISO_LOCAL_DATE);
        System.out.println("ISO_DATE_TIME:"+format);
        format = now.format(DateTimeFormatter.ISO_DATE);
        System.out.println("ISO_DATE:"+format);
        format = now.format(DateTimeFormatter.ISO_TIME);
        System.out.println("ISO_TIME:"+format);

        String dateTimeString = "2019-09-24 16:54:32";
        DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyy
        LocalDateTime localDateTime = LocalDateTime.parse(dateTimeString, dateT
        System.out.println("localDateTime: "+localDateTime);
        format = now.format(dateTimeFormatter);
        System.out.println("format:"+format);

    }

结果:
now:2019-08-13T09:17:40.744
ISO_DATE_TIME:2019-08-13
ISO_DATE:2019-08-13
ISO_TIME:09:17:40.744
localDateTime: 2019-09-24T16:54:32
format:2019-08-13 09:17:40


```