
黑马程序员™
www.itheima.com
传智播客旗下
高端IT教育品牌
继承
GO

1. 继承

1.1 继承概述

```
public class Student {
    private String name;
    private int age;

    public void study(){
        System.out.println("努力学习");
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

相同的属性

相同特征

相同的方法

```
public class Teacher {
    private String name;
    private int age;

    public void teach(){
        System.out.println("教书育人");
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

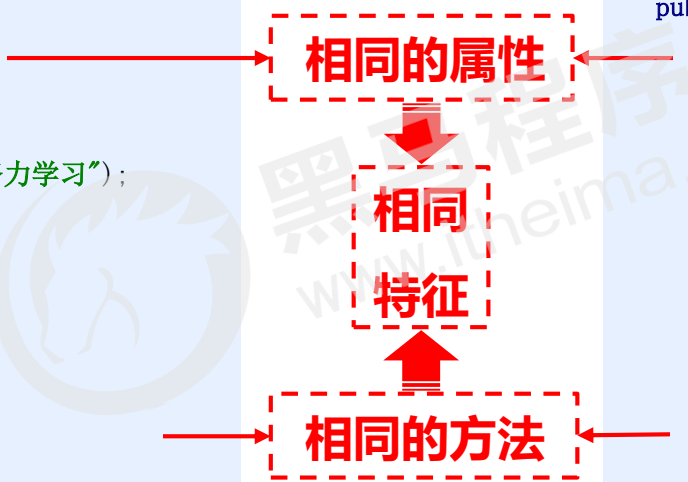
1. 继承

1.1 继承概述

```
public class Student {


    public void study(){
        System.out.println("努力学习");
    }


}
```




```
public class Teacher {


    public void teach(){
        System.out.println("教书育人");
    }


}
```

1. 继承

1.1 继承概述

```
public class Student {

    public void study(){
        System.out.println("努力学习");
    }
}
```

```
public class ?????? {
    private String name;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

```
ass Teacher {

    c void teach(){
    ystem.out.println("教书育人");
}
```

？？关系

？？关系

```
public class Student {
    public void study(){
        System.out.println("努力学习");
    }
}
```

```
public class Teacher {
    public void teach(){
        System.out.println("教书育人");
    }
}
```

1. 继承

1.1 继承概述

继承是面向对象三大特征之一。可以使得子类具有父类的属性和方法，还可以在子类中重新定义，追加属性和方法

1. 继承

1.1 继承概述

继承的格式

- 格式：public class 子类名 **extends** 父类名 { }
- 范例：public class Zi **extends** Fu { }
- Fu：是父类，也被称为基类、超类
- Zi：是子类，也被称为派生类

继承中子类的特点：

- 子类可以有父类的内容
- 子类还可以有自己特有的内容

1.2 继承的好处和弊端

继承好处

- 提高了代码的**复用性**(多个类相同的成员可以放到同一个类中)
- 提高了代码的**维护性**(如果方法的代码需要修改，修改一处即可)

继承弊端

- 继承让类与类之间产生了关系，类的耦合性增强了，当父类发生变化时子类实现也不得不跟着变化，削弱了子类的独立性

什么时候使用继承?

- 继承体现的关系：**is a**
- 假设法：我有两个类A和B，如果他们满足A是B的一种，或者B是A的一种，就说明他们存在继承关系，这个时候就可以考虑使用继承来体现，否则就不能滥用继承
- 举例：苹果和水果，猫和动物，猫和狗

1.3 继承中变量的访问特点

在子类方法中访问一个变量

- 子类局部范围找
- 子类成员范围找
- 父类成员范围找
- 如果都没有就报错(不考虑父亲的父亲...)

1.4 super

super 关键字的用法和 **this** 关键字的用法相似

- **this**：代表本类对象的引用
- **super**：代表父类存储空间的标识(可以理解为父类对象引用)

关键字	访问成员变量	访问构造方法	访问成员方法
this	this.成员变量 访问本类成员变量	this(...) 访问本类构造方法	this.成员方法(...) 访问本类成员方法
super	super.成员变量 访问父类成员变量	super(...) 访问父类构造方法	super.成员方法(...) 访问父类成员方法

1.5 继承中构造方法的访问特点

子类中所有的构造方法默认都会访问父类中无参的构造方法

为什么呢?

- 因为子类会继承父类中的数据，可能还会使用父类的数据。所以，子类初始化之前，一定要先完成父类数据的初始化
- 每一个子类构造方法的第一条语句默认都是：**super()**

如果父类中没有无参构造方法，只有带参构造方法，该怎么办呢?

- 通过使用super关键字去显示的调用父类的带参构造方法
- 在父类中自己提供一个无参构造方法

推荐：自己给出无参构造方法

1.6 继承中成员方法的访问特点

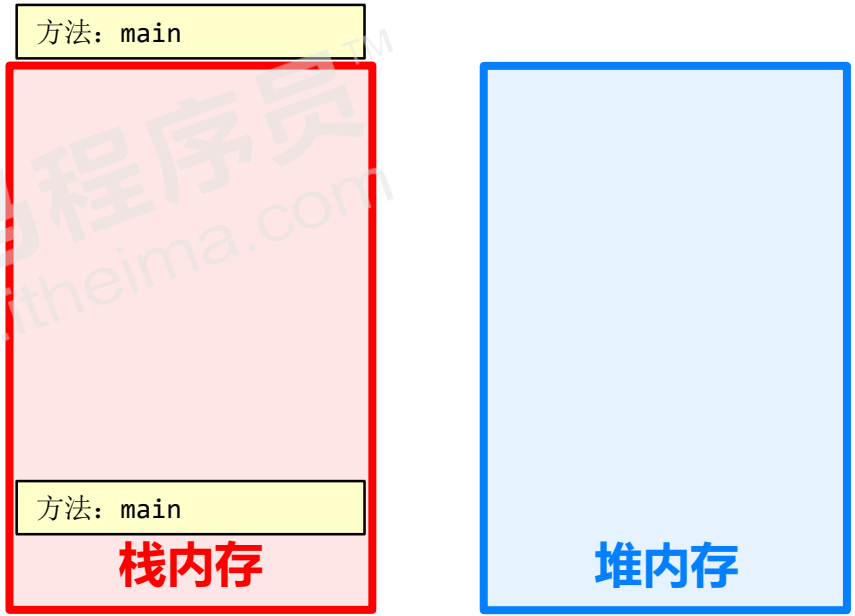
通过子类对象访问一个方法

- 子类成员范围找
- 父类成员范围找
- 如果都没有就报错(不考虑父亲的父亲...)

1. 继承

1.7 super内存图

```
public class Demo {
    public static void main(String[] args) {
        Zi z = new Zi();
        z.show();
        z.method();
    }
}
```



1. 继承

1.7 super内存图

```
public class Demo {
    public static void main(String[] args) {
        Zi z = new Zi();
        z.show();
```

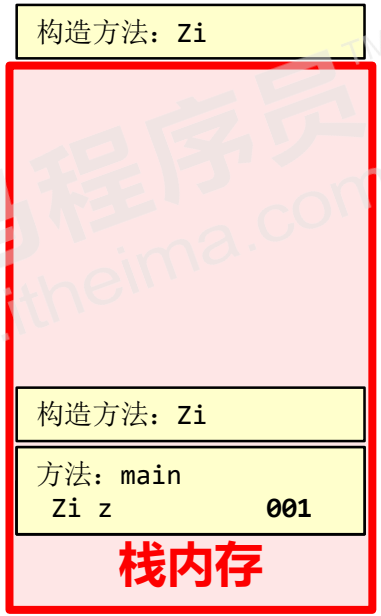
```
public class Zi extends Fu {
    public int age = 20;
```

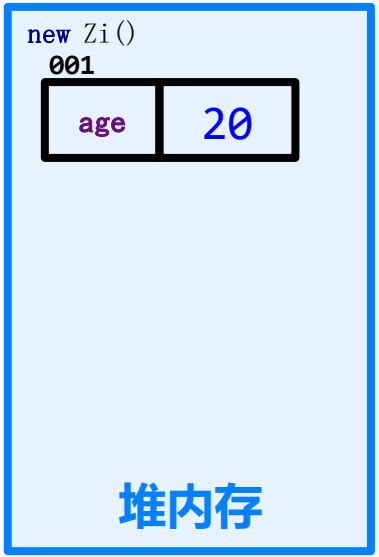
```
public class Fu {
    public int age = 40;

    public Fu() {
        System.out.println("Fu类无参构造方法被调用");
    }

    public void method() {
        System.out.println("Fu类method方法被调用");
    }
}
```

```
}
```





1. 继承

1.7 super内存图

```
public class Demo {
    public static void main(String[] args) {
        Zi z = new Zi();
        z.show();
```

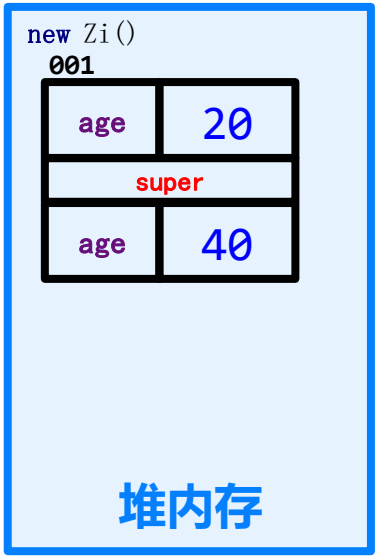
```
public class Zi extends Fu {
    public int age = 20;
```

```
public class Fu {
    public int age = 40;

    public Fu() {
        System.out.println("Fu类无参构造方法被调用");
    }

    public void method() {
        System.out.println("Fu类method方法被调用");
    }
}
```

```
}
```



输出： Fu类无参构造方法被调用

1. 继承

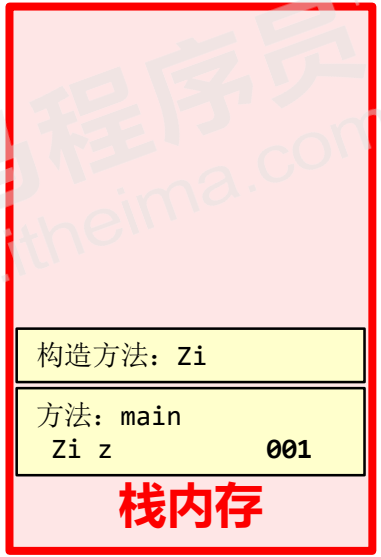
1.7 super内存图

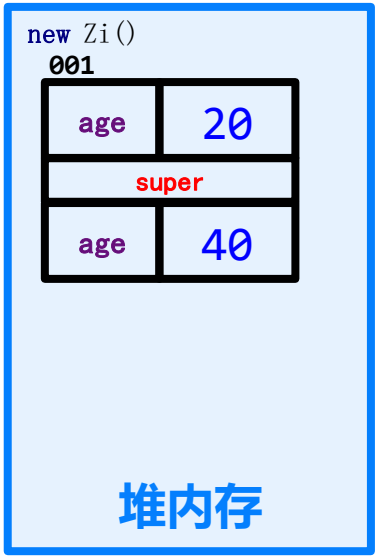
```
public class Demo {
    public static void main(String[] args) {
        Zi z = new Zi();
        z.show();
```

```
public class Zi extends Fu {
    public int age = 20;

    public Zi() {
        super();
        System.out.println("Zi类无参构造方法被调用");
    }

    public void show() {
        int age = 30;
        System.out.println(age);
        System.out.println(this.age);
        System.out.println(super.age);
    }
}
```





输出：Fu类无参构造方法被调用
输出：Zi类无参构造方法被调用

1. 继承

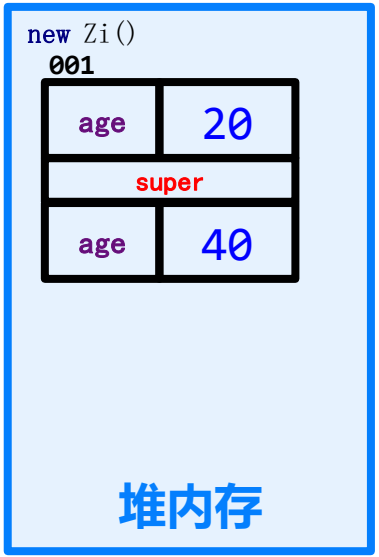
1.7 super内存图

```
public class Demo {
    public static void main(String[] args) {
        Zi z = new Zi();
        z.show();
```

```
public class Zi extends Fu {
    public int age = 20;

    public Zi() {
        super();
        System.out.println("Zi类无参构造方法被调用");
    }

    public void show() {
        int age = 30;
        System.out.println(age);
        System.out.println(this.age);
        System.out.println(super.age);
    }
}
```



输出： Fu类无参构造方法被调用
输出： Zi类无参构造方法被调用

1. 继承

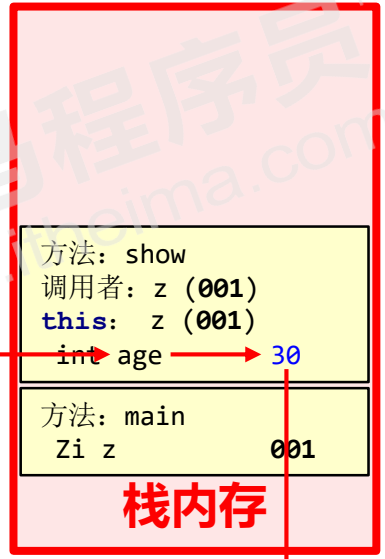
1.7 super内存图

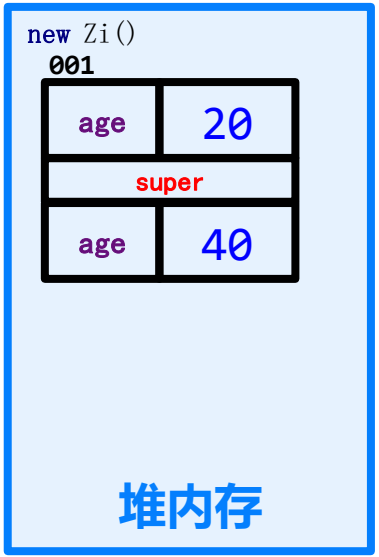
```
public class Demo {
    public static void main(String[] args) {
        Zi z = new Zi();
        z.show();
```

```
public class Zi extends Fu {
    public int age = 20;

    public Zi() {
        super();
        System.out.println("Zi类无参构造方法被调用");
    }

    public void show() {
        int age = 30;
        System.out.println(age);
        System.out.println(this.age);
        System.out.println(super.age);
    }
}
```





输出： Fu类无参构造方法被调用
输出： Zi类无参构造方法被调用
输出： 30

1. 继承

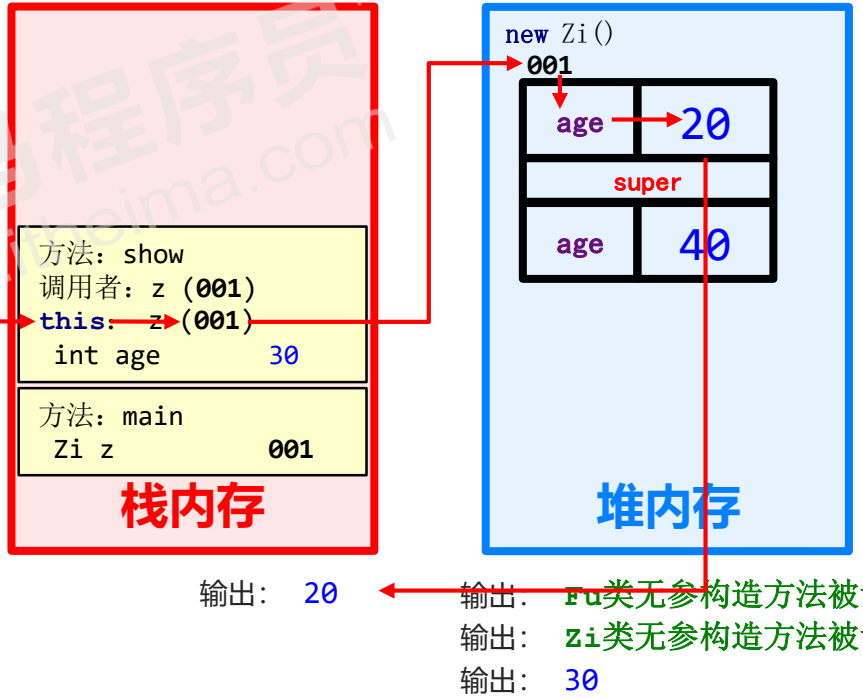
1.7 super内存图

```
public class Demo {
    public static void main(String[] args) {
        Zi z = new Zi();
        z.show();
```

```
public class Zi extends Fu {
    public int age = 20;

    public Zi() {
        super();
        System.out.println("Zi类无参构造方法被调用");
    }

    public void show() {
        int age = 30;
        System.out.println(age);
        System.out.println(this.age);
        System.out.println(super.age);
    }
}
```



1. 继承

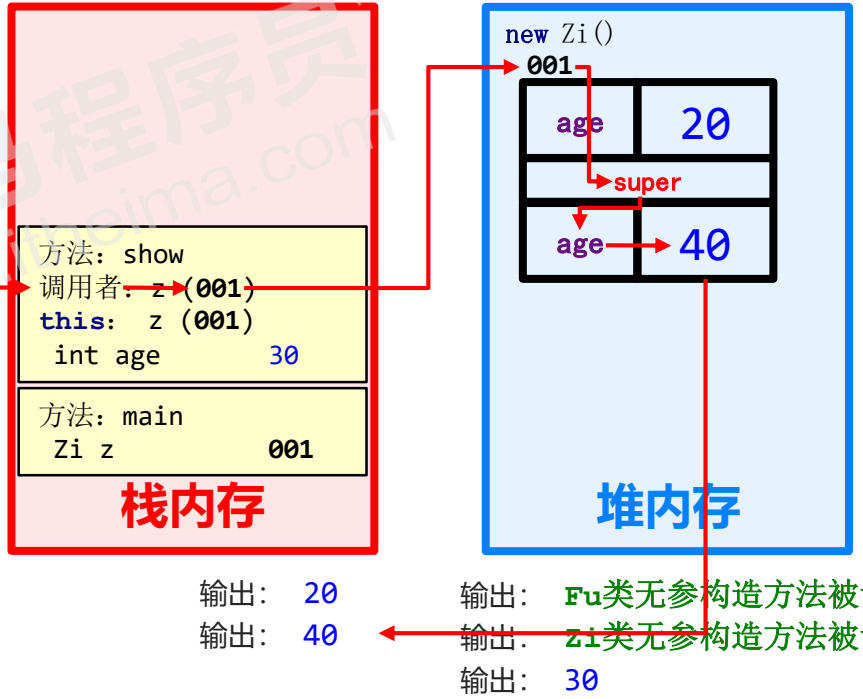
1.7 super内存图

```
public class Demo {
    public static void main(String[] args) {
        Zi z = new Zi();
        z.show();
```

```
public class Zi extends Fu {
    public int age = 20;

    public Zi() {
        super();
        System.out.println("Zi类无参构造方法被调用");
    }

    public void show() {
        int age = 30;
        System.out.println(age);
        System.out.println(this.age);
        System.out.println(super.age);
    }
}
```



1. 继承

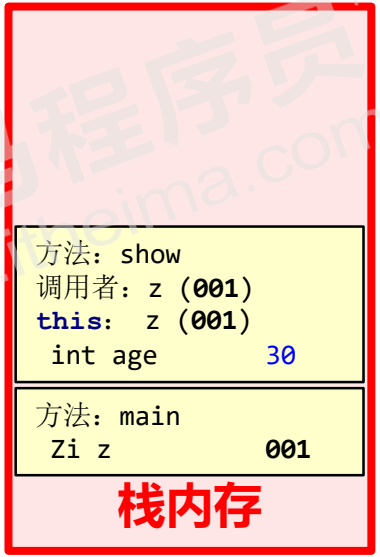
1.7 super内存图

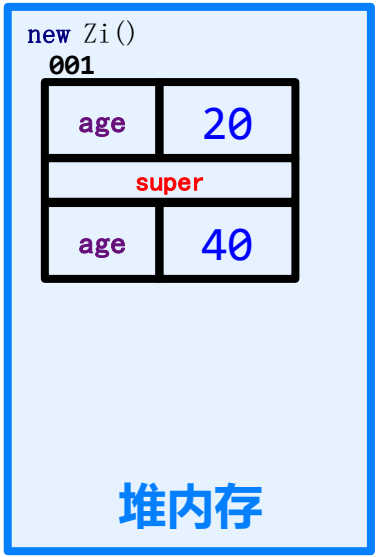
```
public class Demo {
    public static void main(String[] args) {
        Zi z = new Zi();
        z.show();
```

```
public class Zi extends Fu {
    public int age = 20;

    public Zi() {
        super();
        System.out.println("Zi类无参构造方法被调用");
    }

    public void show() {
        int age = 30;
        System.out.println(age);
        System.out.println(this.age);
        System.out.println(super.age);
    }
}
```





输出： 20
输出： 40

输出： Fu类无参构造方法被调用
输出： Zi类无参构造方法被调用
输出： 30

1. 继承

1.7 super内存图

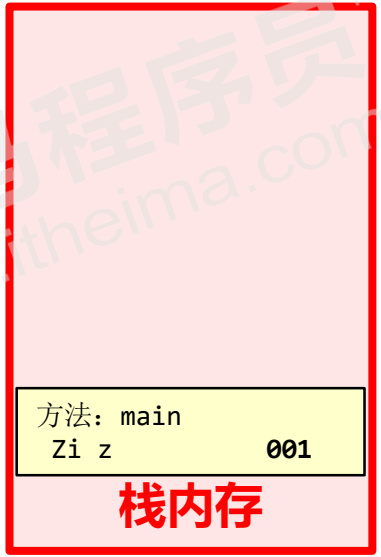
```
public class Demo {
    public static void main(String[] args) {
        Zi z = new Zi();
        z.show();
        z.method();
```

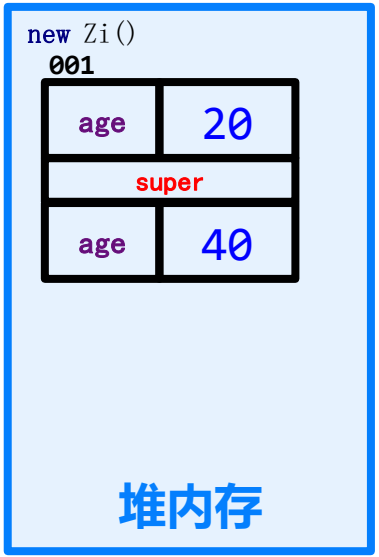
```
blic class Zi extends Fu {
    ublic int age = 20;
```

```
public class Fu {
    public int age = 40;

    public Fu() {
        System.out.println("Fu类无参构造方法被调用");
    }

    public void method() {
        System.out.println("Fu类method方法被调用");
    }
}
```





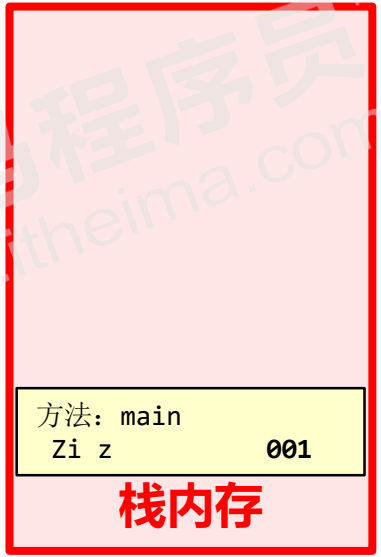
输出： 20
输出： 40
输出： Fu类method方法被调用

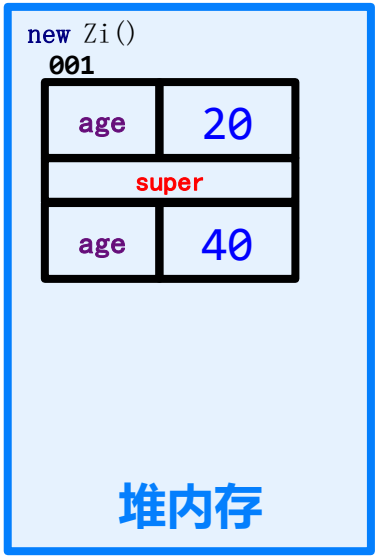
输出： Fu类无参构造方法被调用
输出： Zi类无参构造方法被调用
输出： 30

1. 继承

1.7 super内存图

```
public class Demo {
    public static void main(String[] args) {
        Zi z = new Zi();
        z.show();
        z.method();
    }
}
```





输出： 20
输出： 40
输出： Fu类method方法被调用

输出： Fu类无参构造方法被调用
输出： Zi类无参构造方法被调用
输出： 30

1.8 方法重写

方法重写概述

- 子类中出现了和父类中一模一样的方法声明

方法重写的应用

- 当子类需要父类的功能，而功能主体子类有自己特有内容时，可以重写父类中的方法，这样，即沿袭了父类的功能，又定义了子类特有的内容
- 练习：手机类和新手机类

@Override

- 是一个注解(注解后面会学习到)
- 可以帮助我们检查重写方法的方法声明的正确性

1.9 方法重写注意事项

- 私有方法不能被重写(父类私有成员子类是不能继承的)
- 子类方法访问权限不能更低(public > 默认 > 私有)

1.10 Java中继承的注意事项

- Java中类只支持单继承，不支持多继承
- Java中类支持多层继承

```
public class Granddad {
    public void drink() {
        System.out.println("爷爷爱喝酒");
    }
}
```

```
public class Mother {
    public void dance() {
        System.out.println("妈妈爱跳舞");
    }
}
```

```
public class Father extends Granddad {
    public void smoke() {
        System.out.println("爸爸爱抽烟");
    }
}
```

```
public class Son extends Father, Mother {
}
```

```
public class Son extends Father {
}
```

1. 继承

案例：老师和学生

需求：定义老师类和学生类，然后写代码测试；最后找到老师类和学生类当中的共性内容，抽取出一个父类，用继承的方式改写代码，并进行测试

思路：

① 定义老师类(姓名，年龄，教书())
② 定义学生类(姓名，年龄，学习())
③ 定义测试类，写代码测试
④ 共性抽取父类，定义人类(姓名，年龄)
⑤ 定义老师类，继承人类，并给出自己特有方法：教书()
⑥ 定义学生类，继承人类，并给出自己特有方法：学习()
⑦ 定义测试类，写代码测试

1. 继承

案例：猫和狗

需求：请采用继承的思想实现猫和狗的案例，并在测试类中进行测试

分析：

① 猫：

成员变量：姓名，年龄

构造方法：无参，带参

成员方法：get/set方法，抓老鼠()

② 狗：

成员变量：姓名，年龄

构造方法：无参，带参

成员方法：get/set方法，看门()

③ 共性：

成员变量：姓名，年龄；构造方法：无参，带参；成员方法：get/set方法

1. 继承

案例：猫和狗

需求：请采用继承的思想实现猫和狗的案例，并在测试类中进行测试

思路：

① 定义动物类(Animal)

成员变量：姓名，年龄

构造方法：无参，带参

成员方法：get/set方法

② 定义猫类(Cat)，继承动物类

构造方法：无参，带参

成员方法：抓老鼠()

③ 定义狗类(Dog)，继承动物类

构造方法：无参，带参

成员方法：看门()

④ 定义测试类(AnimalDemo)，写代码测试

传智播客旗下高端IT教育品牌