

补充知识

- 线程回顾
- JavaFX
- 注释(Annotation)



The image shows a Java Swing application window titled "HelloJava1". The window contains a single line of text: "Hello, Java!". A red arrow points from the code editor on the left to the application window on the right, indicating the execution result of the code.

```
1 2  public class HelloJava1 extends javax.swing.JComponent{  
3  
4@  public static void main(String[] args) {  
5      javax.swing.JFrame f = new javax.swing.JFrame("HelloJava1");  
6      f.setSize(300, 300);  
7      f.getContentPane().add(new HelloJava1());  
8      f.setVisible(true);  
9  }  
10  
11@  public void paintComponent(java.awt.Graphics g){  
12      g.drawString("Hello, Java!", 125, 95);  
13  }  
14 }  
15
```

```

2 import java.awt.*;
3 import java.awt.event.*;
4 import javax.swing.*;
5
6 public class HelloJava2 {
7     extends JPanel implements MouseMotionListener {
8
9     // Coordinates for the message
10    int messageX = 125, messageY = 95;
11    String theMessage;
12
13    public HelloJava2(String message) {
14        theMessage = message;
15        addMouseMotionListener(this);
16    }
17
18    public void paintComponent(Graphics g) {
19        g.drawString(theMessage, messageX, messageY);
20    }
21
22    public void mouseDragged(MouseEvent e) {
23        // Save the mouse coordinates and paint the message.
24        messageX = e.getX();
25        messageY = e.getY();
26        repaint();
27    }
28
29    public void mouseMoved(MouseEvent e) {}
30
31    public static void main(String[] args) {
32        JFrame f = new JFrame("HelloJava2");
33        // Make the application exit when the window is closed.
34        f.addWindowListener(new WindowAdapter() {
35            public void windowClosing(WindowEvent we) { System.exit(0); }
36        });
37        f.setSize(300, 300);
38        f.getContentPane().add(new HelloJava2("Hello, Java!"));
39        f.setVisible(true);
40    }
41 }

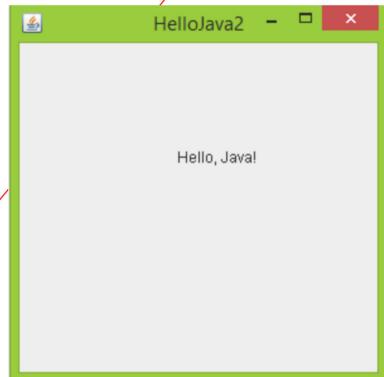
```

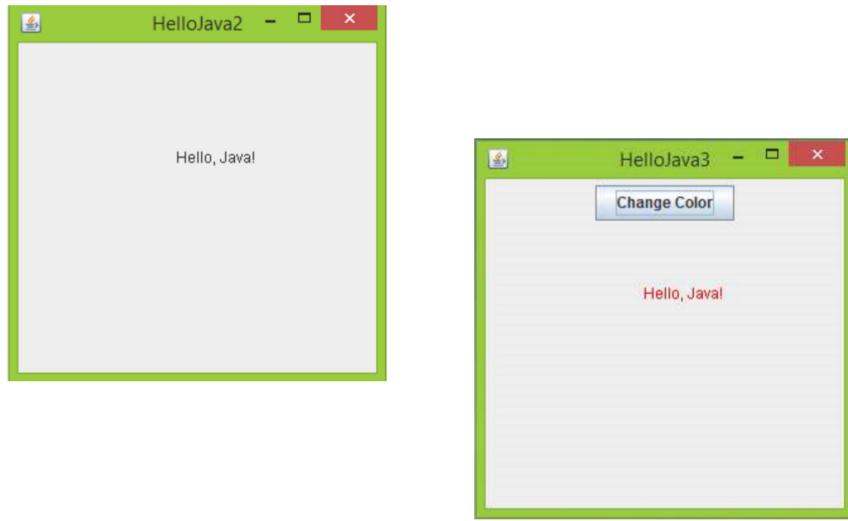
The code above shows two Java classes: HelloJava1 and HelloJava2. HelloJava1 is a simple JFrame with a central panel that displays the string "Hello, Java!". HelloJava2 is a JPanel that implements the MouseMotionListener interface. It has a constructor that takes a string message and adds itself as a mouse motion listener. The paintComponent method draws the message at coordinates (125, 95). The mouseDragged method updates the message coordinates to the current mouse position and calls repaint. The main method creates a JFrame, adds a HelloJava2 panel to it, and sets the frame visible.

```

2 public class HelloJava1 extends javax.swing.JComponent {
3
4    public static void main(String[] args) {
5        javax.swing.JFrame f = new javax.swing.JFrame("HelloJava1");
6        f.setSize(300, 300);
7        f.getContentPane().add(new HelloJava1());
8        f.setVisible(true);
9    }
10
11    public void paintComponent(java.awt.Graphics g) {
12        g.drawString("Hello, Java!", 125, 95);
13    }
14 }

```





```

3 import java.awt.event.*;
4 import javax.swing.*;
5
6 public class HelloJava3
7     extends JComponent
8     implements MouseMotionListener, ActionListener {
9
10    // Coordinates for the message
11    int messageX = 125, messageY = 95;
12    String theMessage;
13
14    JButton theButton;
15
16    // Current index into someColors
17    int colorIndex;
18    static Color[] someColors = { Color.black, Color.red,
19        Color.green, Color.blue, Color.magenta };
20
21    public HelloJava3(String message) {
22        theMessage = message;
23        theButton = new JButton("Change Color");
24        setLayout(new FlowLayout());
25        add(theButton);
26        theButton.addActionListener(this);
27        addMouseMotionListener(this);
28    }
29
30    public void paintComponent(Graphics g) {
31        g.drawString(theMessage, messageX, messageY);
32    }
33
34    public void mouseDragged(MouseEvent e) {
35        // Save the mouse coordinates and paint the message.
36        messageX = e.getX();
37        messageY = e.getY();
38        repaint();
39    }
40
41    public void mouseMoved(MouseEvent e) {}
42
43    public void actionPerformed(ActionEvent e) {
44        // Did somebody push our button?
45        if (e.getSource() == theButton)
46            changeColor();
47    }
48
49    synchronized private void changeColor() {
50        // Change the index to the next color.
51        if (++colorIndex == someColors.length)
52            colorIndex = 0;
53        setForeground(currentColor());
54        repaint(); // Paint again so we can see the change.
55    }
56
57    synchronized private Color currentColor() {
58        return someColors[colorIndex];
59    }
60
61    public static void main(String[] args) {
62        JFrame f = new JFrame("HelloJava3");
63        // Make the application exit when the window is closed.
64        f.addWindowListener(new WindowAdapter() {
65            public void windowClosing(WindowEvent we) { System.exit(0); }
66        });
67        f.setSize(300, 300);
68        f.getContentPane().add(new HelloJava3("Hello, Java!"));
69        f.setVisible(true);

```

```

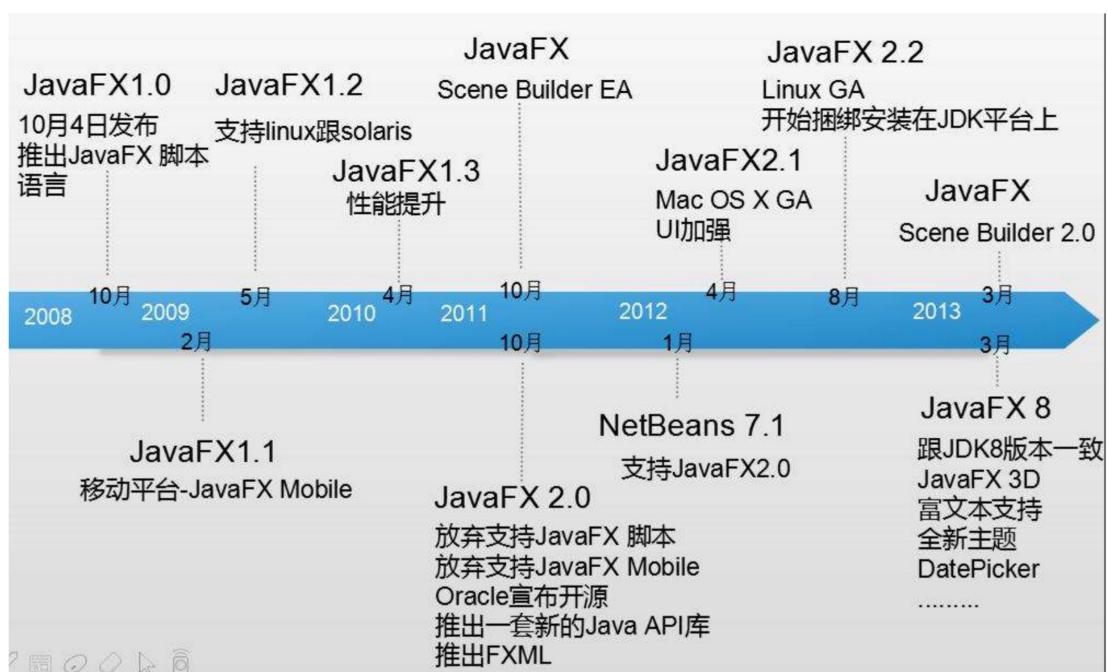
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class HelloJava4
6     extends JPanel
7     implements MouseMotionListener, ActionListener, Runnable {
8
9     // Coordinates for the message
10    int messageX = 125, messageY = 95;
11    String theMessage;
12
13    JButton theButton;
14
15    int colorIndex; // Current index into someColors.
16    static Color[] someColors = { Color.black, Color.red,
17        Color.green, Color.blue, Color.magenta };
18
19    boolean blinkState;
20
21    public HelloJava4(String message) {
22        theMessage = message;
23        theButton = new JButton("Change Color");
24        setLayout(new FlowLayout());
25        add(theButton);
26        theButton.addActionListener(this);
27        addMouseMotionListener(this);
28        Thread t = new Thread(this);
29        t.start();
30    }
31
32    public void paintComponent(Graphics g) {
33        g.setColor(blinkState ? getBackground() : currentColor());
34        g.drawString(theMessage, messageX, messageY);
35    }
36    public void mouseDragged(MouseEvent e) {
37        messageX = e.getX();
38        messageY = e.getY();
39        repaint();
40    }
41
42    public void mouseMoved(MouseEvent e) {}
43
44    public void actionPerformed(ActionEvent e) {
45        // Did somebody push our button?
46        if (e.getSource() == theButton)
47            changeColor();
48    }
49    synchronized private void changeColor() {
50        // Change the index to the next color.
51        if (++colorIndex == someColors.length)
52            colorIndex = 0;
53        setForeground(currentColor());
54        repaint(); // Paint again so we can see the change.
55    }
56    synchronized private Color currentColor() {
57        return someColors[colorIndex];
58    }
59
60    public void run() {
61        try {
62            while(true) {
63                blinkState = !blinkState; // Toggle blinkState.
64                repaint(); // Show the change.
65                Thread.sleep(500);
66            }
67        } catch (InterruptedException ie) {}
68    }
69
70    public static void main(String[] args) {
71        JFrame f = new JFrame("HelloJava4");
72        // Make the application exit when the window is closed.
73        f.addWindowListener(new WindowAdapter() {
74            public void windowClosing(WindowEvent we) { System.exit(0); }
75        });
76        f.setSize(300, 300);
77        f.getContentPane().add(new HelloJava4("Hello, Java!"));
78        f.setVisible(true);
79    }
80

```

JavaFX

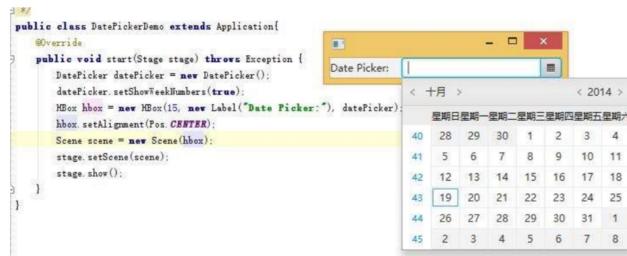
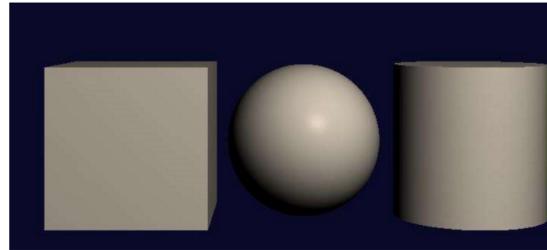
- JavaFX主要致力于富客户端开发，以弥补swing的缺陷，主要提供图形库与media库，支持audio,video,graphics,animation,3D等，同时采用现代化的css方式支持界面设计。同时又采用XUI方式以XML方式设计UI界面，达到显示与逻辑的分离。与android这方面确实有点相似性。

JavaFX的历史





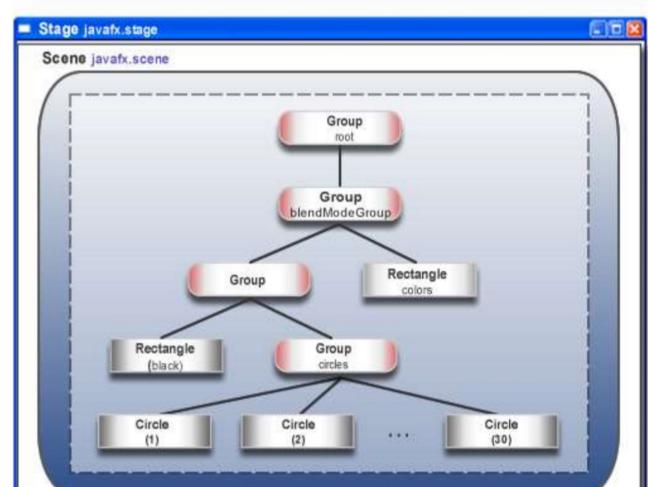
JavaFX新特性



JavaFX



- 舞台（Stage）是应用的顶层容器，场景则是负责绘制应用的外观。场景中的内容是以一棵树来组织结构的。
- ColorfulCircles应用的场景结构。枝干上的节点（Node）都是Group类的实例，而叶子节点则是Rectangle和Circle类的实例。



ColorfulCirclesDemo.java

```
1 public class ColorfulCircles extends Application {  
2     public static void main(String[] args) {  
3         launch(args);  
4     }  
5     @Override  
6     public void start(Stage primaryStage) {  
7         primaryStage.show();  
8     }  
9 }  
10 }  
11 }  
12 }
```

ColorfulCircles类扩展了Application类并包含两个方法。第一个方法是主方法main()，在主方法中调用了launch()方法。作为JavaFX的最佳实践，在main()方法中最好只调用launch()。然后，ColorfulCircles类覆盖了抽象方法start()。start()方法的传入参数是应用最初的stage。最后一行使stage显示出来。

添加场景

```
1  @Override  
2  public void start(Stage primaryStage) {  
3      Group root = new Group();  
4      Scene scene = new Scene(root, 800, 600, Color.BLACK);  
5      primaryStage.setScene(scene);  
6  
7      primaryStage.show();  
8  }
```



添加图形

```
1  Group circles = new Group();  
2  
3  for (int i = 0; i < 30; i++) {  
4      Circle circle = new Circle(150, Color.web("white", 0.05));  
5  
6      circle.setStrokeType(StrokeType.OUTSIDE);  
7      circle.setStroke(Color.web("white", 0.16));  
8      circle.setStrokeWidth(4);  
9      circles.getChildren().add(circle);  
10 }  
11  
12 root.getChildren().add(circles);
```

以上的代码建立了一个名叫circles的组（Group），然后使用for循环向组中添加了30个圆形。每个圆形的半径为150，使用白色填充，透明度是5%，这意味着他们都是几乎透明的。为了给圆形加上边框，这里代码加入了StrokeType类。笔刷：OUTSIDE意思是对于圆形的外边加上StrokeWidth为4的边框。笔刷的颜色是white，透明度16%，使得他看起来比圆更加亮一点。图4展示了现在应用的情况。因为代码并没有指定每个圆的位置，所以所有的圆形都重叠在左上角。重叠后透明的效果使得圆看起来像是灰色的。



添加视觉效果

```
circles.setEffect(new BoxBlur(10, 10, 3));
```

添加了一个长宽都是10，模糊迭代是3的模糊，效果很接近高斯模糊。这种模糊技术使圆的边界有了平滑的效果，



创建渐变背景

```
Rectangle colors = new Rectangle(scene.getWidth(), scene.getHeight(),
    new LinearGradient(0f, 1f, 1f, 0f, true, CycleMethod.NO_CYCLE, new
    Stop[]{(
        new Stop(0, Color.web("#f8bd55")),
        new Stop(0.14, Color.web("#c0fe56")),
        new Stop(0.28, Color.web("#5dfbc1")),
        new Stop(0.43, Color.web("#64c2f8")),
        new Stop(0.57, Color.web("#be4af7")),
        new Stop(0.71, Color.web("#ed5fc2")),
        new Stop(0.85, Color.web("#ef504c")),
        new Stop(1, Color.web("#f2660f")))}));
root.getChildren().add(colors);
```

创建了一个叫做**colors**的矩形。这个矩形与**scene**的长宽相同，并使用从左下角(0,0)到右上角(1,1)的线性填充。第五个参数为**true**意味着填充是与矩形成正比的，第六个参数**NO_CYCLE**标志着颜色循环不会重复。**Stop**数据指出了在某些特定位置填充的颜色。最后一行则是将**colors**矩形加入**root**节点。

原来灰色的圆形现在会变得像彩虹一般，



应用混合模式

```
Group blendModeGroup =
    new Group(new Group(new Rectangle(scene.getWidth(), scene.getHeight(),
        Color.BLACK), circles), colors);
colors.setBlendMode(BlendMode.OVERLAY);
root.getChildren().add(blendModeGroup);
```

blendModeGroup组建立了重叠的结构。他的两个孩子一个是包含黑色矩形和**circles**组的匿名组，另一个则是**colors**矩形。

setBlendMode()方法为**colors**矩形设置了重叠混合。最后一行使将这个组作为孩子节点加入**root**节点。

重叠混合是一个在图形应用程序中非常常见的效果。可以根据混合的颜色使图像变暗或变亮。在这里，线性填充矩形被作为底层，黑色的矩形使背景保持黑色，而透明的圆形将线性填充矩形的颜色显现出来，



加入动画

```
Timeline timeline = new Timeline();
for (Node circle: circles.getChildren()) {
    timeline.getKeyFrames().addAll(
        new KeyFrame(Duration.ZERO, // set start position at 0
            new KeyValue(circle.translateXProperty(), random() * 800),
            new KeyValue(circle.translateYProperty(), random() * 600)
        ),
        new KeyFrame(new Duration(40000), // set end position at 40s
            new KeyValue(circle.translateXProperty(), random() * 800),
            new KeyValue(circle.translateYProperty(), random() * 600)
        )
    );
}
// play 40s of animation
timeline.play();
```

通过时间线（`timeline`）驱动的，这段代码创建了一个时间线，并使用`for`循环为每个圆形都创建两个关键帧，第一关键帧在0秒使用`translateXProperty`和`translateYProperty`创建一个在窗口内部的随机位置。第二关键帧在40秒后做同样的事情。这样，当时间线开始运行时，动画在40秒内就显示所有的圆形都从一个随机位置移动到另一个随机位置。



JavaFX 回顾及动画演示

- | 主类应继承Application类
- | 主类应覆盖start()方法
- | start()方法的参数是应用的原始舞台（Stage）
- | main()方法应该调用launch()方法，并且最好只调用launch()方法
- | 场景要添加一个root节点，可以是本教程中的Group节点，也可是其他祖先节点如layout pane。
- | 当创建场景时最好为其设定长宽。否则场景默认为能够显示其内容的最小尺寸
- | 为root节点添加孩子结点：root.getChildren().add(circles);
- | show()方法使舞台显示出来，最好将其放在最后。

Annotation

- 面向人的注释，→ // or /* */ or /** */
- 面向机器的注释， → Annotation
 - 一种编译器可以识别的语法结构
 - 可以在编译时被读取并进行相应的处理
 - 一种机制，将程序的元素与元数据联系起来，Annotation被添加到代码中，编译后保留在字节码文件中，运行时可以通过反射机制来获取元数据。
 - 从Java 5以后才增加

反射

内置Annotation

- 系统内建
 - @Override,@Deprecated,@SupressWarnings
- **@Override**
 - 强制一个子类重写父类的方法
 - 只能用于方法
 - 如果子类不重写这个方法，编译无法通过

```
public class OverrideTest {  
    @Override  
    public String tostring(){ //找不到父类的中tostring()方法  
        return "test to String";  
    }  
}
```

内置Annotation

```
D:\ch13>javac OverrideTest.java
OverrideTest.java:2: 错误：方法不会覆盖或实现超类型的方法
    @Override
    ^
1 个错误

hadron@Linux ~/ch13 $ javac OverrideTest.java
OverrideTest.java:2: 错误：方法不会覆盖或实现超类型的方法
    @Override
    ^
1 个错误
```

内置Annotation

- **@Deprecated**
 - 表示不建议使用的操作，即过时的
 - 编译时会对这些方法产生警告

```
import java.util.*;
public class DeprecatedTest{
    public static void main(String[] args){
        // 调用已经过时的方法
        new Date().getYear();
    }
}
```

内置Annotation

```
D:\ch13>javac DeprecatedTest.java  
注: DeprecatedTest.java 使用或覆盖了已过时的 API。  
注: 有关详细信息, 请使用 -Xlint:deprecation 重新编译。
```

```
hadron@Linux ~ /ch13 $ javac DeprecatedTest.java  
注: DeprecatedTest.java 使用或覆盖了已过时的 API。  
注: 有关详细信息, 请使用 -Xlint:deprecation 重新编译。
```

内置Annotation

- **@SupressWarnings**
 - 表示压制警告信息，编译时不显示
 - 需要添加内建的参数才能使用，如`deprecation`或`unchecked`

```
import java.util.*;  
public class SupressWarningsTest{  
    @SupressWarnings("deprecation")  
    public static void main(String[] args) {  
        new Date().getYear(); //过时方法  
    }  
}
```

自定义Annotation

- Annotation类:
 - `@interface` 关键字
 - `public @interface name { fields; }`
 - `fields`: 以方法的形式定义，可以有`default`设置
 - `public @interface myAnnotation {`
 - `String name() default "HelloWorld!";`
 - `}`

自定义Annotation

- public class myTest {
- @myAnnotation(name = “测试用例”)
- public static void main(String[] args){
- // method
- }
- }
- 一般Annotation与反射机制结合使用

@Target

- 在定义Annotation时，可以设置适用的程序元素，通过@Target设置元素级别
 - 类型，属性，方法，类，包等
- public @interface Target {
- ElementType[] value() ;
- }
- 枚举类型的常量
- 系统提供了一个枚举类型来控制每个注释的使用范围，例如，有些只能用于方法，而不能用于构造器。

@Target

```
package java.lang.annotation;
public enum ElementType {
    TYPE, // 类、接口、枚举等
    FIELD, // 属性和枚举常量
    METHOD, // 方法，不含构造器
    PARAMETER, // 方法参数
    CONSTRUCTOR, // 构造器
    LOCAL_VARIABLE, // 局部变量
    ANNOTATION_TYPE, // annotation类型
    PACKAGE // Java包
}
```

@Retention

- 设置注释的有效范围
- @Retention(RetentionPolicy.RUNTIME)
- @Target(ElementType.Annotation_TYPE)
- public @interface Retention {
- RetentionPolicy value();
- }
- @Retention的值是枚举型RetentionPolicy的常量值

@Retention

```
public enum RetentionPolicy {  
    //只保留在程序源码里，不会保留在编译好的.class文件中  
    SOURCE,  
    //可以编译到.class文件中，但是执行时不加载  
    CLASS,  
    //编译后保留在.class文件中，在执行的时加载到JVM中  
    RUNTIME  
}
```

自定义Annotation

- 例子：应用@Target和@Retention自定义一个Annotation类 Describe.java
 - @Target({ElementType.FIELD, ElementType.METHOD})指定@Describe注释可以作用于属性和方法
 - @Retention(RetentionPolicy.RUNTIME)设定@Describe注释的有效范围是RUNTIME，也就是在程序运行时是有效的。

自定义Annotation

```
1 import java.lang.annotation.*;
2
3 @Target({ElementType.FIELD, ElementType.METHOD})
4 @Retention(RetentionPolicy.RUNTIME)
5
6 public @interface Describe {
7     public String describe();
8     @SuppressWarnings("rawtypes")
9     public Class type() default void.class;
10 }
11
12 public class AnnotationDemo {
13     @Describe(describe="ID", type=int.class)
14     int pid;
15     @Describe(describe="Name", type=String.class)
16     String name;
17
18     @Override
19     @Describe(describe="String to String", type=String.class)
20
21     public String toString(){
22         return "(pid = " + pid + "," + name + ")";
23     }
24 }
```

```
1 import java.lang.reflect.Field;
2 import java.lang.reflect.Method;
3 import java.lang.annotation.*;
4
5 @Target({ElementType.FIELD, ElementType.METHOD})
6 @Retention(RetentionPolicy.RUNTIME)
7
8 @interface Describe {
9     public String describe();
10    @SuppressWarnings("rawtypes")
11    public Class type() default void.class;
12 }
13
14 class test {
15    @Describe(describe="ID", type=int.class)
16    int pid;
17    @Describe(describe="Name", type=String.class)
18    String name;
19
20    @Override
21    @Describe(describe="String to String", type=String.class)
22
23    public String toString(){
24        return "(pid = " + pid + "," + name + ")";
25    }
26}
27
28 public class AnnotationDemo {
29    public static void main(String[] args) throws Exception {
30        Class <?> c = Class.forName("test");
31        Field[] f = c.getDeclaredFields();
32        for(int i = 0; i<f.length; i++){
33            Field ff = f[i];
34            if(ff.isAnnotationPresent(Describe.class)){
35                Describe da = ff.getAnnotation(Describe.class);
36                System.out.print(da.describe() + ":");
37                System.out.println(da.type());
38            }
39        }
40        Method m = c.getMethod("toString");
41        if(m.isAnnotationPresent(Describe.class)){
42            Describe da = m.getAnnotation(Describe.class);
43            System.out.print(da.describe() + ":");
44            System.out.println(da.type());
45        }
46    }
47}
```