注解也叫元数据，例如我们常见的@Override，注解是JDK1.5版本引入的，可以对包，类，接口，字段，方法参数，局部变量等进行注解。

主要作用分为以下四方面：

* 生成文档，通过代码里标识的元数据生成javadoc文档
* 编译检查，通过代码里标识的元数据让编译器在编译期间进行检查验证
* 编译时动态处理，编译时通过代码里标识的元数据动态处理，例如动态生成代码
* 运行时动态处理，运行时通过代码里标识的元数据动态处理，例如使用反射注入实例

一般注解可以分为三类：

* java自带的标注准注解，包括@Override等用于标记重写某个方法，用这些标明后编译器就会进行检查
* 元注解，元注解是用于定义注解的注解，包括

@Target : 用于标记注解使用范围

@Retention： 用于注解被保留的阶段

@Inherited： 用于表明是否可继承

@Documented： 用于表明是否生成javadoc文档

* 自定义注解，自己根据需求进行注解的编写

自定义注解：

①创建Test注解，声明作用于类并保留到运行时，默认值为default。

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface Test {
    String value() default "default";
}
```

②创建TestMethod注解，声明作用于方法并保留到运行时。

```
@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface TestMethod {
    String value();
}
```

③测试类，运行后输出default和tomcat-method两个字符串，因为@Test没有传入值，所以输出了默认值，而@TestMethod则输出了注入的字符串。

```
@Test()
public class AnnotationTest {
    @TestMethod("tomcat-method")
    public void test(){
    }
    public static void main(String[] args){
        Test t = AnnotationTest.class.getAnnotation(Test.class);
        System.out.println(t.value());
        TestMethod tm = null;
        try {
            tm = AnnotationTest.class.getDeclaredMethod("test",null).getAnnotation(TestMethod.class);
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println(tm.value());
    }
}
```

注解的原理：

对于注解Test， 如果对AnnotationTest(类) 进行注解，则运行时可以通过AnnotationTest.class.getAnnotation(Test.class) 获取注解声明的值，从上面的句子就可以看出来，它是从class结构中获取出Test注解的，所以肯定是在某个时候被加入到class结构中去了。

```
1  @Test("test")
2  public class AnnotationTest {
3      public void test(){
4      }
5  }
```

从java源码到class字节码是由编译器完成的，编译器会对java源码进行解析并生成class文件， 而注解也是在编译时由编译器进行处理，编译器会对注解符号处理并附加到class结构中，根据jvm规范， class文件结构是严格有序的格式，唯一可以附加信息到class结构中的方式就是保存到class结构的attributes属性中。我们知道对于类 、 字段 、 方法 、 在class结构中都有自己特定的表结构，而且各自都有自己的属性， 而对于注解， 作用的范围也可以不同，可以作用在类上， 也可以作用于字段 、 方法上，这时编译器会对应将注解信息存放到类 、 字段 、方法自己的属性上。

在我们的AnnotationTest类被编译后，在对应的AnnotationTest.class文件中就会包含一个RuntimeVisibleAnnotation属性，由于这个注解是作用在类上，所以此属性被添加到类的属性集上。即Test注解的键值对value = test 会被记录起来。而当jvm加载AnnotationTest.class文件字节码时，就会将RuntimeVisibleAnnotation属性值保存到AnnotationTest的Class对象中， 于是就可以通过AnnotationTest.class.getAnnotation(Test.class) 获取到Test注解对象，进而再通过Test注解对象获取到Test里面的属性值。

这可能会问，Test注解的对象是什么？ 其实注解被编译后的本质就是一个继承Annotation接口的接口， 所以@Test其实就是“ public interface Test extends Annotation ”，当我们通过AnnotationTest.class.getAnnotation(Test.class)调用时，JDK会通过动态代理生成一个实现了Test接口的对象，并把将RuntimeVisibleAnnotation属性值设置进此对象中，此对象即为Test注解对象，通过它的value()方法就可以获取到注解值。