**中软国际考题-笔试**

1. 单选题

1、下列是Java中正确的单行注释写法的是（A）

A. //注释内容 B.#注释内容 C. /\*注释内容\*/ D. /\*\*注释内容\*\*/

2、下列关于Java代码描述中正确的是（B）

A. /\*\* \*\*/表示单行注释B. /\* \*/表示多行注释

C. //表示文档注释D. <!-- -->表示多行注释

3、下列关于注释描述错误的是（B）

1. 单行注释只能注释某一行的代码
2. 文档注释与多行注释本质上并无差别
3. 多行注释可以用来注释多行代码
4. 双斜杠表示单纯的路径分割符。

4、java中注释符描述正确的是（D）

A．在注释区域中代码会依次执行B. Java中没有标识的概念，有标识符的概念

C. Java注释种类与javaScript完全一致D. 在注释区域可以添加中文说明信息

5、下列是Java中正确的单行注释写法的是（A）

A. //注释内容 B.#注释内容 C. /\*注释内容\*/ D. /\*\*注释内容\*\*/

6、下列哪项是单行注释（B）

A./\* B.// C./\*\* D./\*/

7、下面哪个选项正确声明了一个注解类型？（C）

A、public interface TestAnnotation{} B、public Annotation TestAnnotation{}

C、public @interface TestAnnotation{} D、public @Annotation TestAnnotation{}

8、哪个内置的注解类型可以用来定义注解的修饰目标？（A）

A、@Target B、@Scope C、@Object D、@Direct

9、注解的修饰目标值定义在哪个枚举类型中？（B）

A、Element B、ElementType C、Target D、TargetType

10、下面哪个不是@Target中的修饰目标值？（D）

A、TYPE B、METHOD C、CONSTRUCTOR D、CLASS

11、下面哪个值能够指定注解的修饰目标是类？（A）

A、TYPE B、FIELD C、METHOD D、CLASS

12、注解的生命周期定义在哪个枚举类型中？（A）

A、RetentionPolicy B、Retention

C、RetentionType D、RetentionElement

13、哪个不是注解的生命周期常量？（D）

A、SOURCE B、CLASS C、RUNTIME D、COMPILE

14、下面哪个类型可以作为注解的属性类型？（A）

A、int[] B、Object C、File D、Integer

10、下面哪个类型不可以作为注解的属性类型？（D）

A、int[] B、boolean C、String[] D、Double

11、定义一个注解使用（ B ）

A. interface B. anno

C. @interface D. @anno

12， 如果需要在运行时获取注解信息， 需要设置RetentionPolicy的值为（ D ）。

A. SOURCE B. CLASS

C. Retention D. RUNTIME

13， 注解可以应用到类中的任何元素， @Target的值要设置为（ A ）

A. TYPE B. PACKAGE

C. FIELD D. METHOD

14、、下列声明一个名字为TestAnnotation注解的定义正确的是（ A ）

A.public @interface TestAnnotation {

}

B.public interface @ TestAnnotation {

}

C. public @ TestAnnotation interface {

}

D. public interface TestAnnotation @ {

}

15、下列语句序列执行后，x 的值是(A)

int a=3, b=4, x=5; if( ++a<b ) x=x+1;

A.5 B.3 C.4 D.6

16、下面哪个选项正确声明了一个注解类型？（C）

A. public interface TestAnnotation{}

B. public Annotation TestAnnotation{}

C. public @interface TestAnnotation{}

D. public @Annotation TestAnnotation{}

1. 多选题
2. JavaDoc工具能够识别以下哪些标签？（A、B、C）

A．@author B. @deprecated C. @param D. @Target

2、以下关于匿名类说法错误的是？（A、C）

A. 文档注释只能用来注释class类 B. 多行注释从“/\*”开始 “\*/”结束

C. /\* \*/注释也可以编写为文档注释D. 多行注释可以覆盖单行注释

3、JavaDoc工具能够识别以下哪些标签？（A、B、C）

A．@author B. @deprecated C. @param D. @Target

4、Java注释方式（A、B、C）

A. 单行注释 B. 多行注释 C. 文档注释 D. 内容注释

5、以下哪些注解是内置注解？（BCD）

A、@MyAnnotation B、@Override

C、@Deprecated D、@SuppressWarning

6、下面对注解的描述正确的是？（A、C、D）

A.元注解是可以注解到注解上的注解，或者说元注解是一种基本注解

B.Target 是目标的意思，表示定义一个注解。

C.元注解是可以注解到注解上的注解，或者说元注解是一种基本注解

D. @Repeatable 是 Java 1.8 才加进来的，所以算是一个新的特性。

7、下面说法正确的是（BCD）

A:环境变量可在编译source code（源编译）时指定。

B:在编译程序时，所能指定的环境变量不包括classpath。

C:javac一次可同时编译数个Java源文件。

D:javac.exe能指定编译结果要置于哪个directory（目录）

8、下面说法正确的是正确答案：BCD

A：环境变量不能再源编译的时候指定的，它的配置一般都是在系统环境变量中。

B：错误，可以设置classpath。

C：这个不知道怎么讲，我只知道可以。

D：可以，在CDM中就能进行相应的操作

9、下面哪些是@Target中的修饰目标值？（A、B、C）

A. TYPE B. METHOD C. CONSTRUCTOR

D. CLASS E.SOURCE

1. 填空题（每空1分，共10分）
2. Java中的注释有三种类型，分别是：\_\_\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_。(单行注释、多行注释、文档注释)

2、\_\_\_\_\_\_\_\_\_是单行注释，\_\_\_\_\_\_\_\_\_是多行注释，\_\_\_\_\_\_\_\_\_是文档注释 。(//, /\* \*/,/\*\* \*\*/)

3、Java中的注释有三种类型，分别是：\_\_\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_。(单行注释、多行注释、文档注释)

4、在javadoc注释中，常用\_\_\_\_\_\_\_\_\_\_来表示一个javadoc标记。(@)

5、@Override注解的作用是什么？：\_\_\_\_\_\_\_。(限定重写父类方法，该注解只能用于方法)

6、声明注解的关键字是什么？：\_\_\_\_\_\_\_。(@interface)

7、@Overrride的含义是 。 当前的方法定义将覆盖超类中的方法

8、@Deprecated的含义是 。不赞成使用的代码，被弃用的代码

9、@SuppressWarnings的含义是 。关闭不当编译器警告信息

10、 @Target的含义是 。该注解可以用于什么地方

11、@Retention的含义是 。需要在什么级别保存该注解信息

12、 @Document的含义是 。将注解包含在Javadoc中

13、 @Inherited的含义是 。 允许子类继承父类中的注解

14、 一个注解可以应用到其他注解是上需要设置 。@Target的值为ANNOTATION\_TYPE

15、 @Component的作用是 。定义一个Spring Bean

16、@Autowired的作用是 。指使Spring容器依据类型执行一个自动组装

17、JDK5时支持的元注解有\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_、 @Target、 @Inherited。(@Documented @Retention)

18、Java具有简单、面向对象、稳定、与平台无关、解释型、多线程、动态等特点

19、Java中子类通过关键字 super 来实现对父类成员的访问.

20、JDK中已经定义了注解\_\_\_\_\_\_\_\_\_\_，用来定义注解的声明周期。 (@Retention)

1. 判断题
2. Java中的注释可以设置其生命周期，分别是编译时舍弃、保留在字节码中以及保留到运行时通过反射读取（ F ）
3. 在文档注释中，@throws表示抛出的异常，和@exception同义（T）
4. 除了包注释外，还有一种类型的文档无法从Java源文件中提取，就是对所有类文件提供概要说明的文件。同样的，也可以为这类注释单独新建一个HTML文件，这个文件的名字为“overview.html”，它的<BODY>和</BODY>标记之间的内容都会被提取。（T）
5. 文档注释会被生成为HTML，因此它应符合HTML的语言规范。（ Y ）
6. Java中的注释可以设置其生命周期，分别是编译时舍弃、保留在字节码中以及保留到运行时通过反射读取（ F ）
7. 在文档注释中，@throws表示抛出的异常，和@exception同义（T）

7、除了包注释外，还有一种类型的文档无法从Java源文件中提取，就是对所有类文件提供概要说明的文件。同样的，也可以为这类注释单独新建一个HTML文件，这个文件的名字为“overview.html”，它的<BODY>和</BODY>标记之间的内容都会被提取。（T）

1. 多行注释中可以嵌套多行注释。（ F ）
2. 注释是不参与运行的（ T ）
3. 注释可以放在任意位置（F）
4. @SuppressWarning注解可以规定类的声明周期（ F ）
5. @Target（Element.Type）注解只能修饰接口（ F ）
6. 当一个注解被 @Target 注解时，这个注解就被限定了运用的场景。（T）
7. @Inherited 注解过的注解进行注解的话，那么如果它的子类没有被任何注解应用的话，那么这个子类就继承了超类的注解。 （T）
8. 注解只能用于类型上，不能用于属性和方法上。（ F ）
9. @Retention使用枚举RetentionPolicy定义生命周期，共有三种情况，SOURCE, CLASS, RUNTIME。（ T ）
10. public String urlpattern(); 这行代码为注解声明了一个方法。（ F ）

4、Class类提供了返回注解对象的方法，可以通过注解对象获得注解属性。（ T ）

五、简答题

1、什么是Java的文档注释

答案：在开始的 /\*\* 之后，第一行或几行是关于类、变量和方法的主要描述。

之后，你可以包含一个或多个何种各样的 @ 标签。每一个 @ 标签必须在一个新行的开始或者在一行的开始紧跟星号(\*).

多个相同类型的标签应该放成一组。例如，如果你有三个 @see 标签，可以将它们一个接一个的放在一起。

下面是一个类的说明注释的实例：

/\*\*\* 这个类绘制一个条形图

\* @author runoob

\* @version 1.2

\*/

javadoc 工具将你 Java 程序的源代码作为输入，输出一些包含你程序注释的HTML文件。

每一个类的信息将在独自的HTML文件里。javadoc 也可以输出继承的树形结构和索引。

由于 javadoc 的实现不同，工作也可能不同，你需要检查你的 Java 开发系统的版本等细节，选择合适的 Javadoc 版本。

2、请列举重要的文档注释标签

答案：

@author：作者。

@version：版本。

@docroot：表示产生文档的根路径。

@deprecated：不推荐使用的方法。

@param：方法的参数类型。

@return：方法的返回类型。

@see：用于指定参考的内容。

@exception：抛出的异常。

@throws：抛出的异常，和exception同义

3、请简述关于Java注释符的总结（5分）

答案：Java 支持三种注释方式。前两种分别是 // 和 /\* \*/，第三种被称作说明注释，它以 /\*\* 开始，以 \*/结束。

说明注释允许你在程序中嵌入关于程序的信息。你可以使用 javadoc 工具软件来生成信息，并输出到HTML文件中。

说明注释，使你更加方便的记录你的程序信息。

4、什么是Java的文档注释

答案：在开始的 /\*\* 之后，第一行或几行是关于类、变量和方法的主要描述。

之后，你可以包含一个或多个何种各样的 @ 标签。每一个 @ 标签必须在一个新行的开始或者在一行的开始紧跟星号(\*).

多个相同类型的标签应该放成一组。例如，如果你有三个 @see 标签，可以将它们一个接一个的放在一起。

下面是一个类的说明注释的实例：

/\*\* 这个类绘制一个条形图

\* @author runoob

\* @version 1.2

\*/

javadoc 工具将你 Java 程序的源代码作为输入，输出一些包含你程序注释的HTML文件。

每一个类的信息将在独自的HTML文件里。javadoc 也可以输出继承的树形结构和索引。

由于 javadoc 的实现不同，工作也可能不同，你需要检查你的 Java 开发系统的版本等细节，选择合适的 Javadoc 版本。

5、请列举重要的文档注释标签

答案：

@author：作者。

@version：版本。

@docroot：表示产生文档的根路径。

@deprecated：不推荐使用的方法。

@param：方法的参数类型。

@return：方法的返回类型。

@see：用于指定参考的内容。

@exception：抛出的异常。

@throws：抛出的异常，和exception同义

6、Java注释规则有哪些？

答案：

1）.注释应该使代码更加清晰易懂

2）.注释要简洁明了，只要提供能够明确理解程序必要的信息就可以了。如果注释太复杂会影响程序整洁度和阅读感。

3）.注释不仅描述程序作了什么，还要描述为什么这样做以及约束。

4）.对于一般的getter和setter方法不用注释。

5）.类、接口、构造函数、方法、全局变量必须添加注释。字段属性可以选择添加简单注释。

6）.简单注释一般不超过10个字。

7）.特殊地方必须要添加注释。比如一下几个地方：典型算法，代码不明晰处，在代码修改处，在循环和逻辑分支组成代码处，为他人提供的接口。

7、注解的作用是什么?

答案：注解其实就是代码里的特殊标记，这些特殊标记可以在编译，类加载，运行时被读取，并执行相应的处理，通过使用注解，程序员可以在不改变原有逻辑的情况下，在源文件嵌入一些补充信息，修饰包，类，构造器，方法，成员变量，参数，局部变量的声明

8、Java为什么要引入注解机制？

答：

注解是在 J2SE 1.5版本中引入的。 引入的原因是提供了一个种机制， 让程序员可以在代码中直接编写元数据。在引入注解以前， 程序员描述代码的没有标准的方式。

9， 什么是注解， 如何使用注解， 如何获取注解信息？

答：

我们使用元数据来解释什么是注解： 所谓的元数据就是包含了描述自身信息的数据。

注解是代码元数据； 它们包含了关于代码自身的信息。

注解可以应用在包， 类， 方法， 变量和参数上。

注解被编译到类文件中， 在运行时可以通过反射机制获取注解信息。

10，描述注解@Target？

答：

表示该注解可以用于什么地方，可能的ElementType参数有：

CONSTRUCTOR：构造器的声明

FIELD：域声明（包括enum实例）

LOCAL\_VARIABLE：局部变量声明

METHOD：方法声明

PACKAGE：包声明

PARAMETER：参数声明

TYPE：类、接口（包括注解类型）或enum声明

11，描述注解@Retention。

答：

表示需要在什么级别保存该注解信息。可选的RetentionPolicy参数包括：

SOURCE：注解将被编译器丢弃

CLASS：注解在class文件中可用，但会被VM丢弃

RUNTIME：VM将在运行期间保留注解，因此可以通过反射机制读取注解的信息。

12、请简述注解的用处。

答案：

提供信息给编译器： 编译器可以利用注解来探测错误和警告信息   
编译阶段时的处理： 软件工具可以用来利用注解信息来生成代码、Html文档或者做其它相应处理。   
运行时的处理： 某些注解可以在程序运行的时候接受代码的提取

13、short s1 = 1; s1 = s1 + 1（）

short s1 = 1; s1 += 1（）是否有问题

　答：对于short s1 = 1; s1 = s1 + 1; 由于s1+1运算时会自动提升表达式的类型，所以结果是int型，再赋值给short类型s1时，编译器将报告需要强制转换类型的错误。

对于short s1 = 1; s1 += 1;由于 += 是java语言规定的运算符，java编译器会对它进行特殊处理，因此可以正确编译。

14、final修饰符的作用？

final修饰的类不能被继承

final修饰的方法不可重写

final修饰的变量最多只能赋值一次，值不可变

15、怎样开启注解装配？

注解装配在默认情况下是不开启的，为了使用注解装配，我们必须在Spring配置文件中配置 <context:annotation-config/>元素。

16.下面的代码会编译吗？

@Target({ ElementType.FIELD, ElementType.TYPE, ElementType.FIELD })

public @interface TestAnnotation {

int[] value() default {};

}

不。如果在@Target注释中多次出现相同的枚举常量，那么这是一个编译时错误。

删除重复常量将使代码成功编译：

@Target({ ElementType.FIELD, ElementType.TYPE})

16、@Target是什么？可以取哪些值？（5分）

答案：@Target说明了Annotation所修饰的对象范围（即：被描述的注解可以用在什么地方）

取值(ElementType)有：

　　　　1.CONSTRUCTOR:用于描述构造器

　　　　2.FIELD:用于描述域

　　　　3.LOCAL\_VARIABLE:用于描述局部变量

　　　　4.METHOD:用于描述方法

　　　　5.PACKAGE:用于描述包

　　　　6.PARAMETER:用于描述参数

　　　　7.TYPE:用于描述类、接口(包括注解类型) 或enum声明

17、@Retention的作用？可以取哪些值？

答案：　作用：表示需要在什么级别保存该注释信息，用于描述注解的生命周期（即：被描述的注解在什么范围内有效）

取值（RetentionPoicy）有：

　　　　1.SOURCE:在源文件中有效（即源文件保留）   
　　　　2.CLASS:在class文件中有效（即class保留）   
　　　　3.RUNTIME:在运行时有效（即运行时保留）

六、程序实现题

1、完成一个文档注释实例，为SquareNum类的各元素选择恰当的标签提供文档说明，并提供利用该类生成JavaDoc的命令

|  |
| --- |
| **package** com.chinasofti.etc.test;  **import** java.io.\*;  /\*\*  \* 这个类演示了文档注释  \* **@author** SE  \* **@version** 1.0  \*/  **public** **class** SquareNum {  /\*\*  \* This method returns the square of num.  \* This is a multiline description. You can use  \* as many lines as you like.  \* **@param** num The value to be squared.  \* **@return** num squared.  \*/  **public** **double** square(**double** num) {  **return** num \* num;  }  /\*\*  \* This method inputs a number from the user.  \* **@return** The value input as a double.  \* **@exception** IOException On input error.  \* **@see** IOException  \*/  **public** **double** getNumber() **throws** IOException {  InputStreamReader isr = **new** InputStreamReader(System.*in*);  BufferedReader inData = **new** BufferedReader(isr);  String str;  str = inData.readLine();  **return** (**new** Double(str)).doubleValue();  }  /\*\*  \* This method demonstrates square().  \* **@param** args Unused.  \* **@return** Nothing.  \* **@exception** IOException On input error.  \* **@see** IOException  \*/  **public** **static** **void** main(String args[]) **throws** IOException  {  SquareNum ob = **new** SquareNum();  **double** val;  System.*out*.println("Enter value to be squared: ");  val = ob.getNumber();  val = ob.square(val);  System.*out*.println("Squared value is " + val);  }  }  JavaDoc处理命令：javadoc SquareNum.java |

2、文档注释： 实现java代码中文档注释规范（10分）

|  |
| --- |
| /\*\*  \* @author  \* @deprecated  \* {@docRoot}  \* {@inheritDoc}  \* {@link}  \*{@linkplain}  \* @param  \*@return  \*@see  \*@serial  \* @serialData  \* @serialField  \*@since  \*@throws  \*{@value}  \*@version  \*/ |

3、完成一个文档注释实例，为SquareNum类的各元素选择恰当的标签提供文档说明，并提供利用该类生成JavaDoc的命令

|  |
| --- |
| **package** com.chinasofti.etc.test;  **import** java.io.\*;  /\*\*  \* 这个类演示了文档注释  \* **@author** SE  \* **@version** 1.0  \*/  **public** **class** SquareNum {  /\*\*  \* This method returns the square of num.  \* This is a multiline description. You can use  \* as many lines as you like.  \* **@param** num The value to be squared.  \* **@return** num squared.  \*/  **public** **double** square(**double** num) {  **return** num \* num;  }  /\*\*  \* This method inputs a number from the user.  \* **@return** The value input as a double.  \* **@exception** IOException On input error.  \* **@see** IOException  \*/  **public** **double** getNumber() **throws** IOException {  InputStreamReader isr = **new** InputStreamReader(System.*in*);  BufferedReader inData = **new** BufferedReader(isr);  String str;  str = inData.readLine();  **return** (**new** Double(str)).doubleValue();  }  /\*\*  \* This method demonstrates square().  \* **@param** args Unused.  \* **@return** Nothing.  \* **@exception** IOException On input error.  \* **@see** IOException  \*/  **public** **static** **void** main(String args[]) **throws** IOException  {  SquareNum ob = **new** SquareNum();  **double** val;  System.*out*.println("Enter value to be squared: ");  val = ob.getNumber();  val = ob.square(val);  System.*out*.println("Squared value is " + val);  }  }  JavaDoc处理命令：javadoc SquareNum.java |

4、编写一段可运行代码，使用Java三种注释。

（10分）

|  |
| --- |
| package com.test;  /\*\*  \* 文档注释  \* @version1.1  \*/  public class Demo {  /\*  \*多行注释  \*  \*  \*/  public static void main(String[] args) {  //单行注释  //System.out.println("注释示例");  System.out.println("注释使用");  }  } |

5、编写自定义注解

（10分）

|  |
| --- |
| @Retention(RetentionPolicy.RUNTIME)  @Target(value={CONSTRUCTOR, FIELD, LOCAL\_VARIABLE, METHOD, PACKAGE, PARAMETER, TYPE})  public @interface MyAnnotation {  String value() default "hello";  } |

6、定义一个注解， 并使用这个注解， 最后使用反射机制打印出注解的内容。

|  |
| --- |
| @Target(ElementType.METHOD)  @Retention(RetentionPolicy.RUNTIME)  public @interface UseCase {  public String id();  public String description() default "no description";  }  public class PasswordUtils {  @UseCase(id = 47, description = "Passwords must contain at least one numeric")  public boolean validatePassword(String password) {  return (password.matches("\\w\*\\d\\w\*"));  }    @UseCase(id = 48)  public String encryptPassword(String password) {  return new StringBuilder(password).reverse().toString();  }  }  public static void main(String[] args) {  List<Integer> useCases = new ArrayList<Integer>();  Collections.addAll(useCases, 47, 48, 49, 50);  trackUseCases(useCases, PasswordUtils.class);  }    public static void trackUseCases(List<Integer> useCases, Class<?> cl) {  for (Method m : cl.getDeclaredMethods()) {  UseCase uc = m.getAnnotation(UseCase.class);  if (uc != null) {  System.out.println("Found Use Case:" + uc.id() + " "  + uc.description());  useCases.remove(new Integer(uc.id()));  }  }  for (int i : useCases) {  System.out.println("Warning: Missing use case-" + i);  }  }  运行结果：  Found Use Case:47 Passwords must contain at least one numeric  Found Use Case:48 no description  Warning: Missing use case-49  Warning: Missing use case-50 |

7、定义两个自定义注解，用来修饰属性，一个标注名字，一个标注性别。

（10分）

|  |
| --- |
| public class Test {  import java.lang.annotation.ElementType;  import java.lang.annotation.Retention;  import java.lang.annotation.RetentionPolicy;  import java.lang.annotation.Target;    @Retention(RetentionPolicy.RUNTIME)  @Target(ElementType.FIELD)  public @interface Name {//名字自定义注解    public String value() default "";  }  import java.lang.annotation.ElementType;  import java.lang.annotation.Retention;  import java.lang.annotation.RetentionPolicy;  import java.lang.annotation.Target;    @Retention(RetentionPolicy.RUNTIME)  @Target(ElementType.FIELD)  public @interface Sex {//性别自定义注解    public enum GenderType {  Male("男"),  Female("女");  private String genderStr;  private GenderType(String arg0) {  this.genderStr = arg0;  }  @Override  public String toString() {  return genderStr;  }  }  GenderType gender() default GenderType.Male;  }          步骤二：自定义注解标注在属性上  public class Person {    @Name(value = "cool\_summer\_moon")  public String name;  public String age;  @Sex(gender = Sex.GenderType.Male)  public String sex;    public String getName() { return name; }  public void setName(String name) {  this.name = name;  }  public String getAge() {  return age;  }  public void setAge(String age) {  this.age = age;  }  public String getSex() {  return sex;  }  public void setSex(String sex) {  this.sex = sex;  } |

8、对于一个有序数组，我们通常采用二分查找的方式来定位某一元素，请编写二分查找的算法，在数组中查找指定元素。   
给定一个整数数组A及它的大小n，同时给定要查找的元素val，请返回它在数组中的位置(从0开始)，若不存在该元素，返回-1。若该元素出现多次，请返回第一次出现的位置。   
分析：重点在返回第一次出现的位置。

（10分）

|  |
| --- |
| public class c {  public static int getPos(int[] A, int n, int val) {  int low = 0, high = n - 1, mid;  if (n == 0 || A == )  return -1;  while (low <= high) {  mid = (low + high) / 2;  //当第一次找出相等的位置后需要继续向前查找，最后返回第一次出现的位置  if (val == A[mid]) {  for(int j = mid; j >= 0; j--) {  if(A[j] != A[mid]) {  mid = j + 1;  break;  }  else if(A[j] == A[mid]) {  mid = j;  continue;  }  }  return mid;  } else if (val < A[mid])  high = mid - 1;  else  low = mid + 1;  }  return -1;  }    public static void main(String[] args) {  int A[] = { 4, 4, 5, 5, 5, 5 };  int val = 4;  int n = A.length;  int result = getPos(A, n, val);  System.out.println(result);  }  } |

9、自定义注解WebInitParam以及WebServlet，其中WebInitParam定义字符串类型属性name及value；WebServlet定义字符串类型属性name以及displayName；int类型属性loadOnStartup ；boolean类型属性asyncSupported；String[]类型 属性urlPatterns；WebInitParam []类型属性initParams。在类LoginServlet中使用注解。（10分）

|  |
| --- |
| @Target(value=ElementType.***TYPE***)  @Retention(value=RetentionPolicy.***RUNTIME***)  **public** **@interface** WebInitParam {  **public** String name();  **public** String value();  }  @Target(value=ElementType.***TYPE***)  @Retention(value=RetentionPolicy.***RUNTIME***)  **public** **@interface** WebServlet {  **public** String name();  **public** String displayName();  **public** **int** loadOnStartup();  **public** **boolean** asyncSupported();  **public** String[] urlPatterns();  **public** WebInitParam[] initParams();  }  @WebServlet(name = "loginServlet",  displayName = "loginServlet",  loadOnStartup = 1,  asyncSupported = **false**,  urlPatterns = { "/login.do" },  initParams = {@WebInitParam(name = "encoding", value = "UTF-8") })  **public** **class** LoginServlet {  **public** **void** doPost(String name, String displayName, **int** loadOnStartup, **boolean** asyncSupported,  String[] urlPatterns, WebInitParam[] initParams ) {  System.***out***.println("name==>"+name);  System.***out***.println("displayName==>"+displayName);  System.***out***.println("loadOnStartup==>"+loadOnStartup);  System.***out***.println("asyncSupported==>"+asyncSupported); System.***out***.println("urlPatterns==>"+Arrays.*toString*(urlPatterns));  System.***out***.println("initParamName==>"+ initParams[0].name());  System.***out***.println("initParamValue==>"+initParams[0].value());  }  }  **public** **class** TestLoginServlet {  **public** **static** **void** main(String[] args) {  WebServlet ws = LoginServlet.**class**.getAnnotation(WebServlet.**class**);  String name = ws.name();  String displayName = ws.displayName();  **int** loadOnStartup = ws.loadOnStartup();  **boolean** asyncSupported = ws.asyncSupported();  String[] urlPatterns = ws.urlPatterns();  WebInitParam[] initParams = ws.initParams();  LoginServlet login = **new** LoginServlet();  login.doPost(name, displayName, loadOnStartup, asyncSupported, urlPatterns, initParams);  }  } |