1、系统预定义的三个最基本的注解
@Override：用在重写的方法上
@SuppressWarnings：抑制警告
@Deprecated：标记xx已过时

2、请列出常见的几个文档注解
@param：表示方法有形参
@return：表示方法的返回值类型不是void
@throws：表示方法会抛出异常
@author：表示作者
@version：表示版本
@since：表示起始版本
@see：表示另请参阅
...

3、请解释4个元注解
@Target：标记该注解可以用在xx上面，具体的位置由ElementType枚举的常量对象决定，例如：METHOD,TYPE,....
@Retention：标记该注解可以滞留到xx时候，具体的生命周期由RetentionPolicy枚举的常量对象决定，例如：SOURCE,
@Documented：标记该注解是否可以被javadoc.exe等文档工具读取
@Inherited：标记该注解是否可以被子类继承

4、声明枚举的语法格式
【修饰符】 enum 枚举类名 【implements 接口们】{
    常量对象列表;
    其他成员列表
}

5、声明注解的语法格式
@元注解
【修饰符】 @interface 注解名{
    配置参数列表
}

1. 枚举类的理解：

当某个类型的对象是有限的几个。

补充说明：实际开发中一般定义为枚举类型的，那么这个类型不会很复杂（属性和方法特别多），而且这些个对象的属性一般不会经常变。

2. JDK1.5之前如何自定义枚举类？步骤：

（1）构造器私有化
（2）在枚举类中创建好几个对象，用类变量（静态变量）存储起来

```
public class Season{
    public static final Season SPRING = new Season();
    public static final Season SUMMER = new Season();
    public static final Season AUTUMN = new Season();
    public static final Season WINTER = new Season();
    private Season(){}
}
```

3. jdk 5.0之后可以使用enum定义枚举类：

```
【修饰符】 enum 枚举类名 【implements 接口们】{
    常量对象列表;
    其他成员列表
}

public enum Week{
    MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY,SUNDAY;
    //...
}
```

4．使用enum定义枚举类之后，枚举类常用方法：

从java.lang.Enum类继承：

（1）name()：返回常量对象的名称

（2）ordinal()：返回常量对象的序号，从0开始

（3）compareTo()：默认按照常量对象的顺序比较大小

（4）toString()：返回常量对象的名称，我们可以重写

还有两个API中没有的方法：

（5）枚举类型[] values()：返回所有的常量对象

（6）枚举类型 valueOf(常量名)：返回指定常量对象

1．注解的理解：

注解也是一种注释，是用代码去注释代码。

2．JDK提供的三个常用注解

@Override：用在重写的方法上

@SuppressWarnings：抑制警告

@Deprecated：标记xx已过时

3．如何自定义注解

```
@元注解
【修饰符】 @interface 注解名{
    配置参数列表
}
```

配置参数的语法格式：

数据类型　参数名()　【default　默认值】；

如果一个注解有配置参数，使用时，要为该配置参数赋值，除非它有默认值。

配置参数的类型：8种基本数据类型，String，枚举，Class，注解类型，以及它们组成的数组。

4．元注解 ：对现有的注解进行解释说明的注解。

@Target：标记该注解可以用在xx上面，具体的位置由ElementType枚举的常量对象决定，例如：METHOD,TYPE,....
@Retention：标记该注解可以滞留到xx时候，具体的生命周期由RetentionPolicy枚举的常量对象决定，例如：SOURCE,CLASS,RUNTIME
@Documented：标记该注解是否可以被javadoc.exe等文档工具读取
@Inherited：标记该注解是否可以被子类继承

5．如何获取注解信息：

前提：该注解的生命周期是RUNTIME

反射：？

```
/*
 * 异常：
 *   程序正常情况下是可以运行的，只是偶然因为其他的原因，导致“异常”情况。
 *   程序的运行过程中，可能遇到了“不正常”的情况，导致程序暂停或崩溃了。
 *
 * 不是异常的情况：
 * （1）语法错误，编译都不通过
 */
```

```
/*
 * 异常：
 *   程序正常情况下是可以运行的，只是偶然因为其他的原因，导致“异常”情况。
 *         例如：用户输入参数的问题，用户文件被删除，用户的磁盘空间已满，网络中断.....
 *   程序的运行过程中，可能遇到了“不正常”的情况，导致程序暂停或崩溃了。
 *
 * 不是异常的情况：
 * （1）语法错误，编译都不通过
 * （2）逻辑错误
 */
```

```
* Java如何处理异常？或者Java的异常处理机制是什么样？
* Java程序运行过程中，如果某句代码发生异常，Java会在这句代码处停下来，
* Java会创建一个“异常的对象”并且“抛”出来。Java会检测在这句代码的外围是否有"try..catch"可以捕获它
*
*/
```

```
* Java如何处理异常？或者Java的异常处理机制是什么样？
* Java程序运行过程中，如果某句代码发生异常，Java会在这句代码处停下来，
* Java会创建一个“异常的对象”并且“抛”出来。
* Java会检测在这句代码的外围是否有"try..catch"可以“捕获”它，如果可以“捕获”它，那么程序从这个try..catch后面继续往下运行。
* 如果外围没有"try..catch"可以“捕获”它，那么程序就会先抛给“上级（调用者）”，上级也会检测是否有"try..catch"可以“捕获”它，如果可以“捕获”它，那么程
* 如果一路上都没有"tyr..catch"可以“捕获”它，那么程序就“挂了”。
*
* 简单说：Java的异常处理机制用“异常对象”来表示异常情况，如果有try..catch可以捕获，就继续，否则就挂了。
```

```
/*
* 异常的父类：java.lang.Throwable
*  Throwable 类是 Java 语言中所有错误或异常的超类。
*
* Throwable：
* （1）Error：Error 是 Throwable 的子类，用于指示合理的应用程序不应该试图捕获的严重问题。
*       例如：VirtualMachineError
* （2）Exception
*
*
* 面试题：编写代码，使得发生VirtualMachineError （OutOfMemoryError, StackOverflowError）
* StackOverflowError错误的示例：
*/
```

```
* Throwable：
* （1）Error：Error 是 Throwable 的子类，用于指示合理的应用程序不应该试图捕获的严重问题。
*       例如：VirtualMachineError
* （2）Exception：
*  又分为两大类：
* A：RuntimeException运行时异常：只有RuntimeException类型或它的子类是属于运行时异常
*       凡是运行时异常，编译器不会要求你必须加"try...catch"或"throws"
* B：编译时异常：除了运行时异常，剩下的全部是编译时异常
*       凡是编译时异常，编译器强制要求你必须加"try...catch"或"throws"之一，否则编译不通过
```

常见的运行时异常：ArrayIndexOutOfBoundsException（数组下标越界）
NullPoniterException（空指针异常）
ClassCastException（类型转换异常）
ArithmeticException（算术异常）

常见的编译时异常：
InterruptedException：线程中断打扰的异常
FileNotFoundException：文件找不到
....

```
/*
 * 异常的处理：try..catch
 * 1、语法结构
 * try{
 *       可能发生异常的代码
 * }catch(异常的类型1   异常名称){
 *       捕获到该异常，要如何处理的代码：（1）什么也不写（2）打印异常（3）其他的处理方式
 * }catch(异常的类型2   异常名称){
 *       捕获到该异常，要如何处理的代码
 * }
 * catch(异常的类型3   异常名称){
 *       捕获到该异常，要如何处理的代码
 * }
 * ....
 *
 */
```

```
* 命令行参数：是指给main的形参赋值的实参
* 如何传：（1）java TestParam chai lin yan
*       （2）Run 菜单-->Run Configurations
*
* Integer.parseInt(xx)：把xx字符串转成一个整数
```

说明：多个catch有要求，必须是小的类型在上面，大的类型在下面，如果没有大小关系，就随意。这里的小和大是继承关系，子类小

2、执行的特点
（1）如果try中的代码没有发生异常，只执行try中，不会执行catch
（2）如果try中的代码发生异常
A：有catch可以捕获它，那么哪个可以捕获就进哪个，按顺序找catch。只会执行其中一个catch。
　　如果catch住了，那么会从try..catch继续运行
B：所有的catch都无法捕获它，那么自动往“上”抛
　　如果没catch住，当前方法就结束了，带着“异常”回到上一级调用的位置。

```
/*
 * 练习：
1、从命令行接收一个整数，作为int[]数组的长度
2、再从键盘输入几个整数，为数组的元素赋值
3、找出最大值
4、给代码加上try...catch，看看可能发生哪些异常
 */
```

```
* 异常的处理：try..catch...finally
* 1、语法结构
* try{
*       可能发生异常的代码
* }catch(异常的类型1   异常名称){
*       捕获到该异常，要如何处理的代码：（1）什么也不写（2）打印异常（3）其他的处理方式
* }catch(异常的类型2   异常名称){
*       捕获到该异常，要如何处理的代码
* }catch(异常的类型3   异常名称){
*       捕获到该异常，要如何处理的代码
* }
* ....
* finally{
*    无论try中是否发生异常，也不管catch是否可以捕获异常，都要执行的代码块。
* }
```

面试题：

final,finalize,finally的区别？

final是修饰符，可以修饰类（不能被继承）、方法（不能被重写）、变量（不能修改值）

finalize：是一个Object中声明的方法，表示有GC调用，在对象被回收之前调用。

finally：是try..catch结构的一部分，

2、try...catch的形式

（1）try...catch

（2）try...catch...finally

（3）try...finally

```
* 当finally和return 一起出现时的情况：
* （1）情况一：finally里面有return，结果就是返回finally中
* （2）情况二：finally里面没有return，结果就是返回try或catch中的
*/
```

throws：

用在声明一个方法时，明确声明该方法可能抛出xx异常。说明这些异常在该方法中没有try..catch，由调用者处理。

语法格式：

【修饰符】 返回值类型 方法名(【形参列表】)【throws 异常列表】{}

说明：throws后面可以接多个异常，顺序无所谓，一般如果有父类的异常，子类异常就不写

```
* throws与方法重写：
*
*
* 方法的重写（Override）的要求：
* （1）方法名：必须相同
* （2）形参列表：必须相同
* （3）返回值类型：
*       基本数据类型和void：必须相同
*       引用数据类型：<=
* （4）权限修饰符：>=
* （5）抛出的异常列表的类型：
```

```
* 异常对象：
* （1）JVM自动抛出
* （2）程序员手动抛出
*
* 无论是JVM自动抛出还是throw手动抛出，最终都用try..catch处理或者通过throws抛给上级。
*
* throw关键字是用于主动抛出异常对象。
* 语法格式：
*   语句：throw 异常对象;
```

```
* throw关键字是用于主动抛出异常对象。
* 语法格式：
*   语句：throw 异常对象;
*
* 说明：如果没有try...catch，它可以代替return语句，结束当前方法。
```

```
* Java中都讲究“见名知意”
*
* 自定义异常的要求
* 1、
* Throwable 类是 Java 语言中所有错误或异常的超类。
* 只有当对象是此类（或其子类之一）的实例时，才能通过 Java 虚拟机或者 Java throw 语句抛出。
* 类似地，只有此类或其子类之一才可以是 catch 子句中的参数类型。
*
* 结论：要想一个类称为“异常类型”，必须继承Throwable或它的子类。
*
* 2、自定义异常的对象，只能由throw语句手动抛出。
*
* 3、建议自定义异常，增加两个构造器
* （1）无参构造尽量保留
* （2）有参构造：异常类型(String message)，可以为message属性赋值
```

```
*  Exception：
*  （1）构造器：可以给message的属性赋值
*  new Exception("校验码输入错误")
*  （2）String getMessage()
*  （3）printStackTrace()：打印异常的堆栈信息
*  专门打印错误信息的：System.err
```