目录

[计算机语言 1](#_Toc501035250)

[软件开发流程 1](#_Toc501035251)

[UML 1](#_Toc501035252)

[字符集和字符编码 2](#_Toc501035253)

[获取路径 2](#_Toc501035254)

[高内聚、低耦合 3](#_Toc501035255)

[深，浅clone() 3](#_Toc501035256)

[assert 3](#_Toc501035257)

[POJO和JavaBean 3](#_Toc501035258)

[反射 3](#_Toc501035259)

# 计算机语言

|  |  |  |  |
| --- | --- | --- | --- |
| 机器语言 | 用[二进制](https://baike.baidu.com/item/%E4%BA%8C%E8%BF%9B%E5%88%B6" \t "_blank)代码表示的指令集，称[机器码](https://baike.baidu.com/item/%E6%9C%BA%E5%99%A8%E7%A0%81" \t "_blank)（machine code），是电脑的CPU可直接解读的数据。 | | |
| 汇编语言 | 不同的设备中，汇编语言对应着不同的机器语言[指令集](https://baike.baidu.com/item/%E6%8C%87%E4%BB%A4%E9%9B%86" \t "_blank)，通过汇编过程转换成机器指令，不同平台之间不可直接移植。它通常被应用在底层，硬件操作和高要求的程序优化的场合。驱动程序、嵌入式操作系统和实时运行程序都需要汇编语言。 | | |
| 高级语言 | 编译型语言 | | 编译性语言写的程序在被执行之前，需要一个专门的编译过程，把程序编译成为机器语言的文件，比如exe文件，以后要运行的话就不用重新翻译了，直接使用exe文件，因为翻译只做了一次，运行时不需要翻译，所以编译型语言的程序执行效率高。 |
| 解析型语言 | 解析型 | 不需要编译，在运行程序的时候才翻译，每个语句都是执行的时候才翻译。这样解释性语言每执行一次就需要逐行翻译一次，效率比较低。现代解释性语言通常把源程序编译成中间代码，然后用解释器把中间代码一条条翻译成目标机器代码，一条条执行。 |
| 脚本语言 | 脚本语言是一种解释性的语言，脚本语言是为了缩短传统的编写-编译-链接-运行（edit-compile-link-run）过程而创建的计算机编程语言。脚本语言一般都 有相应的脚本引擎来解释执行。 他们一般需要解释器才能运行。一般都是以文本形式存在,类似于一种命令。 |

# 软件开发流程

# UML

|  |  |
| --- | --- |
|  | |
| 单向依赖 |  |
| 单向关联 |  |
| 聚合 |  |
| 组合 |  |

# 字符集和字符编码

|  |  |  |  |
| --- | --- | --- | --- |
| 字符集合（形状）  character set | 编码字符集（数字）  coded character set | 字符编码方案（字节）  character-encoding schema | 字符集  =编码字符集+字符编码方案  charset |
|  | Unicode  USC-2: 1个字符2字节  USC-4: 1个字符4字节 | UTF-8: 变长  UTF-16: 变长  UTF-32: 1个字符4字节 | GBK: 1个汉字2字节  1个英文或数字1字节  GB18030: 兼容GBK  iso-88591-1: 兼容ASCII |

|  |  |
| --- | --- |
| 编码 | byte[] bytes = str.getBytes(charset) |
| 解码 | String str = new String(bytes, charset) |

# 获取路径

|  |  |
| --- | --- |
| System.getProperty("user.dir") | E:\tts9\workspace\L\testMaven |
| File directory = new File("./abc");   1. directory.getPath() 2. directory.getCanonicalPath() 3. directory.getAbsolutePath() | 注：如输入的是绝对路径，以下都会是2）的结果   1. .\abc 2. E:\tts9\workspace\L\testMaven\abc 3. E:\tts9\workspace\L\testMaven\.\abc |

|  |  |
| --- | --- |
| http://localhost:8081/jsd1703\_servlet-day06/web/Test | |
| request.getServletPath() | /web/Test |
| request.getContextPath() | /jsd1703\_servlet-day06 |
| request.getRequestURI() | /jsd1703\_servlet-day06/web/Test |
| request.getRequestURL() | http://localhost:8081/jsd1703\_servlet-day06/ web/Test |
| request.getServletContext().getRealPath("") | E:\tts9\apache-tomcat-7.0.67\wtpwebapps\ jsd1703\_servlet-day06 |
| request.getServletContext().getRealPath("abc") | E:\tts9\apache-tomcat-7.0.67\wtpwebapps\ jsd1703\_servlet-day06\abc |

|  |  |
| --- | --- |
| this.getClass().getClassLoader(). getResource("").getPath() | /E:/tts9/apache-tomcat-7.0.67/wtpwebapps/jsd1703\_servlet-day06/WEB-INF/classes/ |
| this.getClass().getClassLoader(). getResource("") | file:/E:/tts9/apache-tomcat-7.0.67/wtpwebapps/jsd1703\_servlet-day06/WEB-INF/classes/ |
| this.getClass().getResource("") | file:/E:/tts9/apache-tomcat-7.0.67/wtpwebapps/jsd1703\_servlet-day06/WEB-INF/classes/web/ |
| this.getClass().getResource("/") | file:/E:/tts9/apache-tomcat-7.0.67/wtpwebapps/jsd1703\_servlet-day06/WEB-INF/classes/ |
| Thread.currentThread().getContextClassLoader().getResource("") | file:/E:/tts9/apache-tomcat-7.0.67/wtpwebapps/jsd1703\_servlet-day06/WEB-INF/classes/ |

# 高内聚、低耦合

|  |  |
| --- | --- |
| 高内聚 | 一个软件模块是由相关性很强的代码组成，只负责一项任务，  也就是常说的单一责任原则（SRP-Single Responsibility Principle） |
| 低耦合 | 让每个模块，尽可能的独立完成某个特定的子功能。模块与模块之间的接口，尽量的少而简单 |

getRealPath("/") getContextPath() getResourceAsStream()

# 深，浅clone()

# assert

|  |
| --- |
| assertion(断言)在软件开发中是一种常用的调试方式。assertion在程序中的就是一条语句，它对一个boolean表达式进行检查。如果该值为false，说明程序已经处于不正确的状态下，assert将给出警告或退出。一般来说，assertion用于保证程序最基本、关键的正确性。assertion检查通常在开发和测试时开启。为了提高性能，在软件发布后，assertion检查通常是关闭的。 |
| 设置：Installed JREs→选择版本→edit→Default VM arguments里输入-ea |

# POJO和JavaBean

|  |  |
| --- | --- |
| POJO | 有一些private的参数作为对象的属性，然后针对每一个参数定义get和set方法访问的接口。  没有继承任何类，没有实现任何接口，没有被其它框架侵入的java对象。 |
| JabaBean | 类必须是公共的；属性必须是私有的，并提供get和set方法来访问；必须有一个公共的无参构造器；实现序列化接口Serializable（分布式应用）。 |

# 反射

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 定义 | 在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意方法和属性；这种动态获取信息以及动态调用对象方法的功能成为java语言的反射机制 | | | | |
| 实现步骤 | 获取类 | | 1. Class<?> clazz = Class.forName("java.util.Random"); （动态） 2. Class<?> clazz = Random.class; 3. Class<?> clazz = random.getClass(); | | |
| 动态创建对象 | | | Object obj = clazz.newInstance(); | |
| API | 动态获取类中声明的方法 | | | Method[] methods = cls.getDeclaredMethods() |
| 动态查找一个方法 | | | Method method =  cls.getDeclaredMethod(methodName, Class<?>... parameterTypes) |
| 改变访问限制 | | | method.setAccessible(true) |
| 动态调用方法 | | | method.invoke(Object obj, Object... args) |
| 动态获取类中声明的属性 | | | Field[] fields = cls.getDeclaredFields() |
| 动态查找一个属性 | | | Field field = cls.getDeclaredField(className) |