以下肯定是不完整的列表，欢迎补充】  
  
【好像还缺什么：缓存技术。欢迎补充】  
  
Java是一个通用的编程语言，其实可以干很多事，怎么学Java就看怎么用了。  
  
但有一些一般的步骤：  
  
1. 熟悉一种文本编辑器，比如Vim, Emacs, Notepad++, TextMate等。知道哪些是开源的，哪些是闭源的，哪些要收费。养成不用盗版软件的习惯。  
2. 安装JDK（建议用你的Linux发行版自带的软件包管理器安装openjdk，过程中可能需要读发行版特定的文档）  
3. 写一个Java的Hello world程序，并用命令行工具javac编译，再用java命令运行这个程序。过程中熟悉源代码、字节码、虚拟机这些东西，以及Java的包（package）对.class文件所在的路径的影响。  
   3.5. 如果这两个命令行工具使用熟练了，可以开始选一个喜欢的集成开发环境，比如Eclipse。当然，养成不用盗版软件的习惯。熟悉一下如何建立“工程”，以及快捷键的使用。  
4. 学习Java的面向过程编程，包括基本数据结构、表达式、语句、控制流、函数调用。  
5. 学习Java的面向对象编程，包括类、引用类型和值类型的区别、成员、方法、访问控制、继承、多态、接口、接口实现。顺便学习一下面向对象的基本思想，即对象、消息、封装、继承、多态等，这些通用的内容不是Java特有的。  
   5.5 这时候应该已经涉及了Java的垃圾回收。要留意即使有垃圾回收的情况下也会发生的内存泄露（如自己设计数组容器，元素是引用，逻辑上删除了元素，但并没有清成null）。注意垃圾回收只能回收**内存**中的对象，除了内存以外，**其它资源不能依靠垃圾回收来关闭**。比如，文件、管道、Socket、数据库连接等，垃圾回收是不会帮你关闭的。  
6. 学习Java的异常处理，但更重要的是学习什么时候用特殊返回值而不使用异常，什么时候**应该抛出异常而不处理异常**，知道什么是pokemon catch及其危害，了解为什么Java的checked exception是一个糟糕的特性。如果愿意，同时学习一下Java1.7的try-with-resource语句和AutoCloseable接口。  
7. 熟悉Java常用的数据结构，如基本的数组类型，以及泛型容器（java.util.\*），尤其是java.util.List<T>接口和java.util.ArrayList<T>实现；以及java.util.Map<T,U>接口和java.util.HashMap<T,U>实现。（java1.5以前的没有泛型参数的就不用碰了）同时留意一下基本类型int, double等和装箱类型Integer和Double的区别，以及它们是如何自动转换的。  
8. 熟悉Java标准库里的各种工具，包括日期时间、字符串格式化、IO等。**知道文件要自己在finally子句中close()，或者用Java1.7的try-with-resource，不要妄想垃圾回收器会帮你关掉文件。**  
9. 学习一下Java的命名习惯，以及JavaBeans的常规，知道为什么getter/setter比直接操作成员变量好。按这种方式给Java的变量、方法命名。同时看看你的IDE能不能自动帮你生成getter和setter。  
10. 使用一个第三方的库（比如Apache Commons Lang通用工具库），让你的程序依赖于它的二进制jar包（而不是直接拷贝源代码），用命令行编译、运行（注意classpath等）；也熟悉一下如何用你的集成开发环境添加第三方依赖。感受一下手动管理依赖关系的麻烦。  
11. 学习Maven的使用，试着让Maven帮你解决依赖关系，再试着让Maven帮你创建一个Eclipse工程。再试试用Maven打包发布。  
12. 学习软件测试，以及JUnit的使用，以及怎么在IDE中使用JUnit。有空看一下coverage工具。  
13. 读读四人帮写的《设计模式》（这本书是用C++和Smalltalk语言为例子的，但仍然适合Java）。具体的是这本书， <http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612> 图书馆应该能还借到英文原版，因为我借到过。   
  
接下来就看具体要做哪方面的应用了，看需求。比如（下面的没有顺序）  
  
【....】如果学Java学得不舒服了，学Python。  
  \* 如果对面向对象编程的概念有点不习惯，学Smalltalk。（Ruby也行，但不如Smalltalk经典。Ruby的文档是一大硬伤。）  
  \* 如果嫌Java太啰嗦，学Python  
  \* 如果嫌Java太啰嗦，又想用JVM，自己又有精力，学Scala  
  \* 如果对对象之间的关系有点晕，学一学UML，以及它的一些图，可以对程序和运行进行直观的建模。你的IDE也许有插件可以生成UML图。但是不要太沉迷于这些方法论。  
  
【调试和辅助工具】学习一下你的集成开发环境提供的调试工具，加一些断点试试  
  \* 试试用jconsole或者VisualVM监控另一个jvm的状态。  
  \* 用profiling工具寻找程序中慢的地方。Eclipse有profiling工具。VisualVM也有这样的功能。（如果不介意使用闭源软件的话，也试试JProfiler和YourKit）  
  \* 有的JVM允许在运行时更新代码。Eclipse可以和某些JVM集成。这样你可以频繁修改代码而不用频繁重启JVM。对于某些“重型”工程很有用。（如果不介意使用闭源软件的话，也试试jRebel）  
  
【多线程】先读[Oracle的Java Tutorial里的这一章](https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html" \t "_blank)，请自始至终关注“happens-before“这个词，学完以后应该可以写出正确的并发程序了。  
  \* 抽空顺便了解一下多核处理器、缓存、内存的关系，以及CPU内部的内存读写指令的重排序等细节，体会一下为什么多线程编程这么难。  
  \* 学习Java的多线程编程接口，主要是lock、condition的用法  
    \* Java里每个对象都可以用synchronized语句同步，有wait, notify方法，但这套机制有缺陷。  
    \* 看看java.util.concurrent.lock里的Lock和Condition接口，看它们是怎么解决这些缺陷的。  
    \* 学完之后，学学用Future和Promise来同步，而不是手动用lock和condition。  
    \* 还有semaphore、cyclic barrier、count-down latch、phaser等高级同步工具，或许可以少重新发明一些轮子。  
    \* 还有BlockingQueue  
  \* 学习一下如何让线程停下来，以及为什么要频繁确认isInterrupted()而不要用Thread.stop()（看Java API里Thread.stop()的文档）。  
  \* 如果还舒服，学习一下用Runnable来封装“任务”而不是“线程”，并用Java自带的ThreadPoolExecuter、ForkJoinPool等工具帮你管理线程。  
  \* 应该已经留意到java.util里面的很多容器不是线程安全的，但是java.util.Collections可以帮你创建一些安全的版本。另外关注一下java.util.concurrent里面有ConcurrentMap等容器可供使用。  
  \* 如果有空的话，看看memory model（内存一致性模型）和无锁同步（见java memory model和java.util.concurrent.atomic）。  
  \* 如果还有空，再了解一下除了“共享内存多线程编程”以外有没有别的模型（多进程multi-processing、消息传递message passing等）。或许这时候你已经爱上BlockingQueue了。  
  
【反射、元编程】学习Java的反射机制，以及Annotation的用法。  
  \* 如果还舒服，试试java.lang.reflect.Proxy的用法。  
  \* 如果仍然还舒服，玩一玩CGLib（一个第三方的库）。  
  
【网络编程】学习一下IP, TCP协议（计算机专业的应该学过，复习一下），学习Socket编程（**注意垃圾回收器不会帮你关掉Socket**）。  
  1. 如果不是很关心HTTP，看看java.nio，学习单线程轮询式IO复用（Selector）。  
     \* 如果有点不明白nio的意图的话，了解一下c10k问题。 <http://www.kegel.com/c10k.html>  
     \* 如果身体没有异样的话，大概了解一下操作系统（包括C语言）提供的select, poll, epoll, kqueue等接口。  
     \* 如果身体仍然没有异样的话，试着用java.nio写一个文件服务器。  
     \* 如果还有精力的话，上网扒一扒有没有其他的通信库，如netty等。  
  2. 如果关心Web还有HTTP，就学习一下HTTP协议，以及用Java进行HTTP的客户端编程。  
     \* 如果还舒服，学学HTML，写写HTML的静态网页（不需要Java）  
     \* 如果还舒服，用Java写一个基于DOM、XPath或者CSS Selector的网页解析器（爬网页）。  
     \* 如果还舒服，学学Java的Servlet接口（先别学jsp）进行Web服务器端编程。学学标准的Servlet容器怎么用，包括web.xml的用法以及listener、filter等概念。以及某个Servlet容器（如Jetty或者Tomcat）的具体用法。  
     \* 如果仍然还舒服，试着学一种模板语言（如haml, velocity, freemarker，【还有其他更好的框架吗？我不知道】, String.format，如果真的想学JSP的话JSP倒是也行，但不推荐）。  
     \* 如果仍然觉得舒服，学学Spring框架中的Web框架，或者Struts，看你的口味。  
     \* 如果还舒服，看看Spring Bean Container以及里面各种乱七八糟的工具。  
     \* 如果还舒服，或者有需求，了解一下什么是RESTful Web Service，复习一下HTTP，找找适合的Java工具。  
       \* 你可能会觉得Jackson是一个解析JSON用的好用的东西。  
  
【数据库】学习一下关系数据库（计算机专业的应该学过，复习一下），包括SQL。选一个数据库管理系统熟悉一下（比如MariaDB，或者（如果你不讨厌Oracle的话）用被Oracle收购了的MySQL。先脱离Java单独学学）。然后看它们的官方文档教你怎么用Java连接这种数据库。这中间会涉及到JDBC接口。同时**一定要知道SQL注入安全漏洞，以及掌握如何用PreparedStatement防止注入！！**。建议看 <http://bobby-tables.com/>  
  \* 可能中间会涉及“事务”问题，让你不知不觉地开始去了解java transaction api(JTA)。  
  \* 如果还舒服，学一学对象关系转换（如Hibernate）。  
  \* 也可以学学非关系数据库，以及如何用Java访问它们。  
  
【日志记录】学习一下slf4j和logback的用法。  
  \* 如果有精力的话，大概了解一下世界上有多少种Java日志框架，以及slf4j是怎么桥接这些框架的。  
  
【构建(build)系统】学习一下Ant的用法。  
  \* 如果还舒服的话，学习一下用Ivy从Maven的仓库里下载软件包，解决依赖关系。  
  
【版本控制】学习一种分布式版本控制器（如Git、Mercurial、Bzr、Darcs等，推荐Git）的基本用法，以及如何用它管理Java工程。希望你已经开始使用Maven了，并且知道为什么把IDE生成的工程文件（如eclipse的.project，.classpath和.metadata）放入版本控制器不好。然后了解一下如何在IDE中使用版本控制（Eclipse自带Git插件）。  
  \* 如果感觉很舒服的话，为你们实验室搭建一个Linux+SSH+Git服务器，装个GitLab（一种Web界面）。  
  \* 了解“集中式版本控制器”和“分布式版本控制器”的区别，并说服同事们不要再用SVN、CVS或者SourceSafe等老旧的“集中式版本控制器”了。  
  \* 开设一个GitHub账户。如果你不喜欢Git，就用BitBucket等。  
  
【持续集成】自己（或者为你们实验室）搭建一个持续集成（Continuous Integration）服务器，如Jenkins，定期编译你的程序。建议同时使用Git等分布式版本控制器。  
  \* 如果你做开源软件，试试GitHub和Travis。  
  
【零碎工具】淘一淘java.nio.files里面有什么好用的东东，然后再淘一淘Apache Commons Lang和Commons IO里有什么好用的工具。Commons Logging就不要再用了，用SLF4j和Logback。  
  
【XML】学学XML、DOM、XPath。XML这东西到处都可能用到。也可以用它写自己的配置文件。  
  \* 如果觉得不舒服了，就学学JSON和YAML。  
  \* 如果还是不舒服，就学学文本文件解析。  
  
【语法分析和编译器】学学Antlr或者别的Parser Generator的用法  
  \* 如果觉得舒服，自己写一个计算器。  
  \* 如果还觉得舒服，自己写一种Domain-Specific Language (DSL)。  
  
【高效容器】学学FastUtil或者Trove，如果你需要进行大量数值运算的话。  
  
【分布式计算】学学MapReduce的思想以及它的实现。  
  \* 如果还舒服，学学Scala语言以及号称比MapReduce快得多的Apache Spark。  
  
【进程间通信】看看ActiveMQ、MINA和RMI。  
  
【其他语言（JVM）】学习另一门跑在JVM上的语言或者实现（如Groovy、Scala、Clojure、JRuby、Jython、JavaScript……）  
  \* 如果还舒服，学一学Java Scripting API（注意不是JavaScript。给个链接： <http://docs.oracle.com/javase/6/docs/technotes/guides/scripting/programmer_guide/> ），然后试着在自己的Java程序里嵌入脚本。  
  
【其他语言（非JVM）】学习另一门通用脚本语言（如Python、Ruby，其实perl也行，但不推荐），知道什么时候Java不是最好的选择。  
  
【Java语言和Java虚拟机】通读一遍（一目十行地读，不用细读）Java Language Specification，以及Java Virtual Machine Specification。  
  \* 了解以下解释器（interpreter）、编译器（compiler）、即时编译器（just-in-time compiler）和优化器（optimiser）的概念。  
    \* 如果对编译器的话题不感到畏惧，了解一下method JIT和tracing JIT的概念和区别。  
  
【内存管理】学学垃圾回收的几种基本算法，包括mark-sweep、mark-compact、semi-space、generational、mark-region等，各自的性能，以及为什么朴素的reference counting是不完整的。知道为什么finalizer性能很糟糕，而且标准并不要求finalizer在程序退出前一定会执行。  
  \* 如果还舒服，了解一下如何设置Java虚拟机的堆大小限制（如HotSpot虚拟机的-Xmx选项等）。  
  \* 了解一下Java里的WeakReference以及SoftReference和PhantomReference，以及它们什么时候有用，以及为什么它们实现起来有些困难。  
  \* 如果有精力，了解一下Hotspot虚拟机的内存管理算法是什么样的。  
  
【动态装载】学学Java的动态装载（class loading）  
  \* 如果还舒服的话，学学OSGI以及它的一种实现（如Felix或者Equinox）  
  \* 如果仍然很舒服的话，学学写基于Eclipse平台的程序。不是Eclipse集成开发环境，只是利用他们的图形框架，写自己的应用程序。  
  \* 如果还觉得舒服的话，写Eclipse集成开发环境的插件。  
  
【本地/外语接口】学习一下Java Native Interface（JNI），试着写一个Java和C语言混合编程的程序。  
  \* 如果觉得不舒服了或者觉得欲仙欲死，就学一学Java Native Access（JNA），试一试不用任何胶水代码而从Java直接装载C库，直接调用C函数。  
  \* 如果连JNA也懒得学，就学一学SWIG，自动生成绑定。  
  \* 如果觉得舒服，就学一学Java Virtual Machine Tooling Interface（JVMTI），用C程序来监视JVM的状态。  
  
【密码学】学一学密码学，包括编码、密码分析、攻击、对称密钥、公钥系统、数字签名、哈希算法等，看看Java有没有实现。  
  \* 如果觉得有点不舒服（你应该不会觉得舒服吧，除非你是学密码学的，要不然总会觉得自己写的程序有安全漏洞），就写一个“人品计算器”来放松一下，要求每次输入同一个姓名，得到的人品值是固定的，但又要让人无法用别人的人品值猜自己的人品值。  
  
【移动终端】学一学Android开发。  
  \* 如果有精力的话，看看Dalvik虚拟机是怎么回事。  
  \* 建议买一个iPhone或者iPad，或许你再也不想用Android手机或者平板了。  
  
【历史】如果觉得有些无聊，就去挖一挖历史上比较火的可能和Java相关技术，比如：  
  \* Applet，想想它比起html5+css3+javascript的缺点在哪里。  
  \* AWT、Swing，想想为什么很少有人用Java写图形界面程序。你觉得Swing的程序看上去舒服吗？中国人和残疾人喜欢用Swing程序吗？  
  \* JNDI，想想它比起Spring Bean Container的缺点在哪里。  
  \* JSP，想想它比起MVC结构的缺点在哪里。  
  \* WSDL/SOAP，把它们和XML-RPC、RESTful Web Service比较一下。  
  \* XSLT，以及为什么它是图灵完备的。可是它真的比Java本身更好用吗？  
  \* Log4j、java.util.logging、Apache Commons Logging，各自有什么问题，以及Log4j的作者本人为什么又开发了SLF4j和Logback？  
  \* Java最早是为什么设计的？  
  \* Type erasure是怎么回事？为什么ArrayList<int>不行但ArrayList<Integer>就可以？挖一挖历史。