# 一、资源

构建

这里搜集了用来构建应用程序的工具。

* [Apache Maven](http://maven.apache.org/)：Maven使用声明进行构建并进行依赖管理，偏向于使用约定而不是配置进行构建。Maven优于Apache Ant。后者采用了一种过程化的方式进行配置，所以维护起来相当困难。
* [Gradle](http://www.gradle.org/)：Gradle采用增量构建。Gradle通过Groovy编程而不是传统的XML声明进行配置。Gradle可以很好地配合Maven进行依赖管理，并且把Ant脚本当作头等公民。

字节码操作

编程操作Java字节码的函数库。

* [ASM](http://asm.ow2.org/)：通用底层字节码操作及分析。
* [Javassist](http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/)：尝试简化字节码编辑。
* [Byte Buddy](http://bytebuddy.net/)：使用“流式API”进一步简化字节码生成。

代码分析

软件度量和质量评估工具。

* [Checkstyle](http://checkstyle.sourceforge.net/)：对编程规范和标准进行静态分析。
* [FindBugs](http://findbugs.sourceforge.net/)：通过字节码静态分析找出潜在Bug。
* [PMD](http://pmd.sourceforge.net/)：对源代码中不良编程习惯进行分析。
* [SonarQube](http://www.sonarqube.org/)：通过插件集成其它分析组件，提供评估最终结果报告。

编译器

创建分析器、解释器和编译器的框架。

* [ANTLR](http://www.antlr.org/)：功能完备的自顶向下分析复杂框架。
* [JavaCC](https://javacc.java.net/)：相对ANTLR更具体，上手略为简单。支持语法语法超前预测（syntactic lookahead）。

持续集成

支持持续集成、测试和应用发布的工具。

* [Bamboo](https://www.atlassian.com/software/bamboo)：Atlassian的持续集成（CI）解决方案，包含很多其它产品。
* [CircleCI](https://circleci.com/)：提供托管服务，可免费试用。
* [Codeship](https://www.codeship.io/features)：提供托管服务，提供有限免费计划。
* [Go](http://www.thoughtworks.com/products/go-continuous-delivery)：ThoughtWork开源持续集成解决方案。
* [Jenkins](http://jenkins-ci.org/)：提供基于服务器的部署服务。
* [TeamCity](http://www.jetbrains.com/teamcity/)：JetBrain持续集成方案，提供免费版。
* [Travis](https://travis-ci.org/)：提供托管服务，常用于开源项目。

数据库

简化数据库交互的工具、库。

* [Flyway](http://flywaydb.org/)：使用Java API轻松完成数据库迁移。
* [H2](http://h2database.com/)：小型SQL数据库，以内存操作著称。
* [JDBI](http://jdbi.org/)：便捷的JDBC抽象。
* [jOOQ](http://www.jooq.org/)：基于SQL schema生成类型安全代码。
* [Presto](https://github.com/facebook/presto)：针对大数据的分布式SQL查询引擎。
* [Querydsl](http://www.querydsl.com/)：针对Java的类型安全统一查询。

日期和时间

处理日期和时间的函数库。

* [Joda-Time](http://www.joda.org/joda-time/)：Java 8出现之前，它是日期、时间处理的标准函数库。
* [Time4J](https://github.com/MenoData/Time4J)：Java高级日期、时间函数库。

依赖注入

帮助代码实现[控制反转](http://en.wikipedia.org/wiki/Inversion_of_control)模式的函数库。

* [Dagger](http://square.github.io/dagger/) ：编译期的注入框架，没有使用反射，主要用于Android开发。
* [Guice](https://github.com/google/guice)：轻量级注入框架，功能强大可与Dagger媲美。

开发库

从基础层次上改进开发流程。

* [AspectJ](https://eclipse.org/aspectj/)：面向切面编程扩展，与程序无缝连接。
* [Auto](https://github.com/google/auto)：源代码生成器集合。
* [DCEVM](http://ssw.jku.at/dcevm/)：通过修改JVM，在运行时可无限次重定义已加载的类。OpenJDK 7、8已提供支持，详情可查看[这个分支（fork）](http://dcevm.github.io/)。
* [JRebel](http://zeroturnaround.com/software/jrebel/)：商用软件，无需重新部署可即时重新加载代码及配置。
* [Lombok](http://projectlombok.org/)：代码生成器，旨在减少Java冗余代码。
* [RxJava](https://github.com/Netflix/RxJava)：使用JVM中可观察序列，创建异步、基于事件应用程序的函数库。
* [Spring Loaded](https://github.com/spring-projects/spring-loaded)：另一个JVM类重载代理。
* [vert.x](http://vertx.io/)：JVM多语言事件驱动应用框架。

分布式应用

用来开发分布式、具有容错性应用程序的函数库和框架。

* [Akka](http://akka.io/)：构建并发、分布式和具有容错功能的事件驱动应用程序所需的工具包和运行时。
* [Apache Storm](http://storm.incubator.apache.org/)：分布式实时计算系统。
* [Apache ZooKeeper](http://zookeeper.apache.org/)：为大型分布式系统，使用分布式配置、同步和命名注册提供协调服务。
* [Hazelcast](http://hazelcast.org/)：分布式、高可扩展性内存网格。
* [Hystrix](https://github.com/Netflix/Hystrix)：为分布式系统提供延迟和容错处理。
* [JGroups](http://www.jgroups.org/)：一组提供可靠消息传输的工具包，可用来创建集群。集群中的节点可互相发送消息。
* [Quasar](http://www.paralleluniverse.co/quasar/)：为JVM提供轻量级线程和Actor。

发布

使用本机格式分发Java应用程序的工具。

* [Bintray](https://bintray.com/)：对二进制发布进行版本控制，可与Maven或Gradle配合使用。
* [IzPack](http://izpack.org/)：为跨平台部署建立授权工具。
* [Launch4j](http://launch4j.sourceforge.net/)：将JAR包装为小巧的Windows可执行文件。
* [packr](https://github.com/libgdx/packr#packr)：将程序JAR、资源和JVM打包成Windows、Linux和Mac OS X的本机文件。

文档处理

用来处理Office格式文档的函数库。

* [Apache POI](http://poi.apache.org/)：支持OOXML （XLSX、DOCX、PPTX）以及 OLE2 （XLS, DOC or PPT）格式的文档。
* [jOpenDocument](http://www.jopendocument.org/)：处理OpenDocument格式文档。

游戏开发

游戏开发框架。

* [jMonkeyEngine](http://jmonkeyengine.org/)：支持现代3D开发的游戏引擎。
* [libGDX](http://libgdx.badlogicgames.com/)：全面的跨平台高级开发框架。
* [LWJGL](http://lwjgl.org/)：抽象了OpenGL、CL、AL等函数库的健壮框架。

GUI

用来创建现代图形用户界面的函数库。

* [JavaFX](http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html)：Swing的继承者。
* [Scene Builder](http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html)：JavaFX虚拟布局工具。

高性能

与高性能计算有关的资源，包括集合以及很多具体功能的函数库。

* [Disruptor](http://lmax-exchange.github.io/disruptor/)：线程间消息函数库。
* [fastutil](http://fastutil.di.unimi.it/)：快速紧凑的Java类型安全集合。
* [GS Collections](https://github.com/goldmansachs/gs-collections)：受Smalltalk启发的集合框架。
* [hftc](https://github.com/OpenHFT/hftc)：Hash set和hash map。
* [HPPC](http://labs.carrotsearch.com/hppc.html)：基本类型集合。
* [Javolution](http://javolution.org/)：针对实时嵌入式系统的函数库。
* [Trove](http://trove.starlight-systems.com/)：基本类型集合。

IDE

视图简化开发的集成开发环境。

* [Eclipse](http://www.eclipse.org/)：后台做了很多工作，以其丰富插件著称。
* [IntelliJ IDEA](http://www.jetbrains.com/idea/)：支持很多JVM语言，为Android开发提供了很多不错的选项。其商业版本主要面向企业用户。
* [NetBeans](https://netbeans.org/)：集成了很多Java SE和Java EE特性，包括数据库访问、服务器、HTML5以及AngularJS。

图像处理

用来帮助创建、评估或操作图形的函数库。

* [Picasso](http://square.github.io/picasso/)：Android下载图像和图像缓存函数库。
* [ZXing](https://github.com/zxing/zxing)：多种格式的一维、二维条形码处理函数库。

JSON

简化JSON处理的函数库。

* [Gson](https://code.google.com/p/google-gson/)：将Java对象序列化为JSON及反向操作。使用时提供了很好的性能。
* [Jackson](http://wiki.fasterxml.com/JacksonHome)：与GSON类似，但如果需要频繁初始化Jackson库会带来性能问题。

JVM和JDK

目前的JVM、JDK实现。

* [JDK 9](https://jdk9.java.net/)：JDK 9早期访问版本。
* [OpenJDK](http://openjdk.java.net/)：开源实现。

日志

记录应用程序的日志函数库。

* [Apache Log4j 2](http://logging.apache.org/log4j/)：对之前版本进行了完全重写。现在的版本具备一个强大的插件和配置架构。
* [kibana](http://www.elasticsearch.org/overview/kibana/)：对日志进行分析并进行可视化。
* [Logback](http://logback.qos.ch/)：Log4j原班人马作品。被证明是一个强健的日志函数库，通过Groovy提供了很多有意思的配置选项。
* [logstash](http://logstash.net/)：日志文件管理工具。
* [SLF4J](http://www.slf4j.org/)：日志抽象层，需要与某个具体日志框架配合使用。

机器学习

提供具体统计算法的工具。其算法可从数据中学习。

* [Apache Hadoop](http://hadoop.apache.org/)：对商用硬件集群上大规模数据存储和处理的开源软件框架。
* [Apache Mahout](https://mahout.apache.org/)：专注协同过滤、聚类和分类的可扩展算法。
* [Apache Spark](http://spark.apache.org/)：开源数据分析集群计算框架。
* [h2o](http://0xdata.github.io/h2o/)：用作大数据统计的分析引擎。
* [Weka](http://www.cs.waikato.ac.nz/ml/weka/)：用作数据挖掘的算法集合，包括从预处理到可视化的各个层次。

消息

在客户端之间进行消息传递，确保协议独立性的工具。

* [Apache ActiveMQ](http://activemq.apache.org/)：实现JMS的开源消息代理（broker），可将同步通讯转为异步通讯。
* [Apache Kafka](http://kafka.apache.org/)：高吞吐量分布式消息系统。
* [JBoss HornetQ](http://hornetq.jboss.org/)：清晰、准确、模块化且方便嵌入的消息工具。
* [JeroMQ](https://github.com/zeromq/jeromq)：ZeroMQ的纯Java实现。

其它

其它资源。

* [Design Patterns](https://github.com/iluwatar/java-design-patterns)：实现并解释了最常见的[设计模式](http://www.amazon.cn/gp/product/B001130JN8/ref=as_li_qf_sp_asin_il_tl?ie=UTF8&tag=importnew-23&linkCode=as2&camp=536&creative=3200&creativeASIN=B001130JN8)。
* [Jimfs](https://github.com/google/jimfs)：内存文件系统。
* [Lanterna](https://code.google.com/p/lanterna/)：类似curses的简单console文本GUI函数库。
* [LightAdmin](http://lightadmin.org/)：可插入式CRUD UI函数库，可用于快速应用开发。
* [Metrics](http://metrics.codahale.com/)：创建自己的软件度量或者为支持框架添加度量信息，通过JMX或HTTP进行发布或者发送到数据库。
* [OpenRefine](http://openrefine.org/)：用来处理混乱数据的工具，包括清理、转换、使用Web Service进行扩展并将其关联到数据库。
* [RoboVM](http://www.robovm.org/)：Java编写原生iOS应用。

自然语言处理

用来专门处理文本的函数库。

* [Apache OpenNL](https://opennlp.apache.org/)：处理类似分词等常见任务的工具。
* [CoreNLP](http://nlp.stanford.edu/software/corenlp.shtml)：斯坦佛的CoreNLP提供了一组基础工具，可以处理类似标签、实体名识别和情感分析这样的任务。
* [LingPipe](http://alias-i.com/lingpipe/)：一组可以处理各种任务的工具集，支持POS标签、情感分析等。
* [Mallet](http://mallet.cs.umass.edu/)：统计学自然语言处理、文档分类、聚类、主题建模等。

网络

网络编程函数库。

* [Netty](http://netty.io/)：构建高性能网络应用程序开发框架。
* [OkHttp](http://square.github.io/okhttp/) ：一个Android和Java应用的HTTP+SPDY客户端。

ORM

处理对象持久化的API。

* [EclipseLink](https://www.eclipse.org/eclipselink/)：支持许多持久化标准，JPA、JAXB、JCA和SDO。
* [Hibernate](http://hibernate.org/orm/)：广泛使用、强健的持久化框架。Hibernate的技术社区非常活跃。
* [Ebean](http://ebean-orm.github.io/)：支持快速数据访问和编码的ORM框架。

PDF

用来帮助创建PDF文件的资源。

* [Apache FOP](http://xmlgraphics.apache.org/fop/)：从XSL-FO创建PDF。
* [Apache PDFBox](http://pdfbox.apache.org/)：用来创建和操作PDF的工具集。
* [DynamicReports](http://dynamicreports.org/)：JasperReports的精简版。
* [iText](http://itextpdf.com/)：一个易于使用的PDF函数库，用来编程创建PDF文件。注意，用于商业用途时需要许可证。
* [JasperReports](http://community.jaspersoft.com/project/jasperreports-library)：一个复杂的报表引擎。

REST框架

用来创建RESTful 服务的框架。

* [Dropwizard](https://dropwizard.github.io/dropwizard/)：偏向于自己使用的Web框架。用来构建Web应用程序，使用了Jetty、Jackson、Jersey和Metrics。
* [Jersey](https://jersey.java.net/)：JAX-RS参考实现。
* [RESTEasy](http://resteasy.jboss.org/)：经过JAX-RS规范完全认证的可移植实现。
* [Retrofit](http://square.github.io/retrofit/)：一个Java类型安全的REST客户端。
* [Spark](http://www.sparkjava.com/)：受到Sinatra启发的Java REST框架。
* [Swagger](https://helloreverb.com/developers/swagger)：Swagger是一个规范且完整的框架，提供描述、生产、消费和可视化RESTful Web Service。

科学

用于科学计算和分析的函数库。

* [SCaVis](http://jwork.org/scavis/)：用于科学计算、数据分析和数据可视化环境。

搜索

文档索引引擎，用于搜索和分析。

* [Apache Solr](http://lucene.apache.org/solr/) ：一个完全的企业搜索引擎。为高吞吐量通信进行了优化。
* [Elasticsearch](http://www.elasticsearch.org/)：一个分布式、支持多租户（multitenant）全文本搜索引擎。提供了RESTful Web接口和无schema的JSON文档。

安全

用于处理安全、认证、授权或会话管理的函数库。

* [Apache Shiro](http://shiro.apache.org/)：执行认证、授权、加密和会话管理。
* [Cryptomator](http://cryptomator.org/)：在云上进行客户端跨平台透明加密。
* [Keycloak](http://keycloak.jboss.org/)：为浏览器应用和RESTful Web Service集成SSO和IDM。目前还处于beta版本，但是看起来非常有前途。
* [PicketLink](http://picketlink.org/)：PicketLink是一个针对Java应用进行安全和身份认证管理的大型项目（Umbrella Project）。
* [Spring Security](http://projects.spring.io/spring-security/)：专注认证、授权和多维度攻击防护框架。

序列化

用来高效处理序列化的函数库。

* [FlatBuffers](https://github.com/google/flatbuffers)：序列化函数库，高效利用内存，无需解包和解析即可高效访问序列化数据。
* [Kryo](https://github.com/EsotericSoftware/kryo)：快速和高效的对象图形序列化框架。
* [MessagePack](https://github.com/msgpack/msgpack-java)：一种高效的二进制序列化格式。

服务器

用来部署应用程序的服务器。

* [Apache Tomcat](http://tomcat.apache.org/)：针对Servlet和JSP的应用服务器，健壮性好且适用性强。
* [Apache TomEE](http://tomee.apache.org/)：Tomcat加Java EE。
* [GlassFish](https://glassfish.java.net/)：Java EE开源参考实现，由Oracle资助开发。
* [Jetty](http://www.eclipse.org/jetty/)：轻量级、小巧的应用服务器，通常会嵌入到项目中。
* [WildFly](http://www.wildfly.org/)：之前被称作JBoss，由Red Hat开发。支持很多Java EE功能。

模版引擎

对模板中表达式进行替换的工具。

* [Apache Velocity](http://velocity.apache.org/)：提供HTML页面模板、email模板和通用开源代码生成器模板。
* [FreeMarker](http://freemarker.org/)：通用模板引擎，不需要任何重量级或自己使用的依赖关系。
* [Handlebars.java](http://jknack.github.io/handlebars.java/)：使用Java编写的模板引擎，逻辑简单，支持语义扩展（semantic Mustache）。
* [JavaServer Pages](https://jsp.java.net/)：通用网站模板，支持自定义标签库。
* [Thymeleaf](http://www.thymeleaf.org/)：旨在替换JSP，支持XML文件。

测试

测试内容从对象到接口，涵盖性能测试和基准测试工具。

* [Apache JMeter](http://jmeter.apache.org/)：功能性测试和性能评测。
* [Arquillian](http://arquillian.org/)：集成测试和功能行测试平台，集成Java EE容器。
* [AssertJ](http://joel-costigliola.github.io/assertj/)：支持流式断言提高测试的可读性。
* [JMH](http://openjdk.java.net/projects/code-tools/jmh/)：JVM微基准测试工具。
* [JUnit](http://junit.org/)：通用测试框架。
* [Mockito](http://code.google.com/p/mockito/)：在自动化单元测试中创建测试对象，为TDD或BDD提供支持。
* [Selenium](http://docs.seleniumhq.org/)：为Web应用程序提供可移植软件测试框架。
* [Selenide](http://selenide.org/)：为Selenium提供精准的周边API，用来编写稳定且可读的UI测试。
* [TestNG](http://testng.org/) ：测试框架。
* [VisualVM](http://visualvm.java.net/)：提供可视化方式查看运行中的应用程序信息。

工具类

通用工具类函数库。

* [Apache Commons](http://commons.apache.org/)：提供各种用途的函数，比如配置、验证、集合、文件上传或XML处理等。
* [Guava](http://code.google.com/p/guava-libraries/)：集合、缓存、支持基本类型、并发函数库、通用注解、字符串处理、I/O等。
* [javatuples](http://www.javatuples.org/)：正如名字表示的那样，提供tuple支持。尽管目前tuple的概念还有留有争议。

网络爬虫

用于分析网站内容的函数库。

* [Apache Nutch](http://nutch.apache.org/) ：可用于生产环境的高度可扩展、可伸缩的网络爬虫。
* [Crawler4j](https://code.google.com/p/crawler4j/)：简单的轻量级爬虫。
* [JSoup](http://jsoup.org/) ：刮取、解析、操作和清理HTML。

Web框架

用于处理Web应用程序不同层次间通讯的框架。

* [Apache Tapestry](http://tapestry.apache.org/)：基于组件的框架，使用Java创建动态、强健的、高度可扩展的Web应用程序。
* [Apache Wicket](http://wicket.apache.org/)：基于组件的Web应用框架，与Tapestry类似带有状态显示GUI。
* [Google Web Toolkit](http://www.gwtproject.org/)：一组Web开发工具集，包含在客户端将Java代码转为JavaScript的编译器、XML解析器、RCP API、JUnit集成、国际化支持和GUI控件。
* [Grails](https://grails.org/)：Groovy框架，旨在提供一个高效开发环境，使用约定而非配置、没有XML并支持混入（mixin）。
* [Play](http://www.playframework.com/)： 使用约定而非配置，支持代码热加载并在浏览器中显示错误。
* [PrimeFaces](http://primefaces.org/)：JSF框架，提供免费版和带技术支持的商业版。包含一些前端组件。
* [Spring Boot](http://projects.spring.io/spring-boot/)：微框架，简化了Spring新程序的开发过程。
* [Spring](http://projects.spring.io/spring-framework/)：旨在简化Java EE的开发过程，提供依赖注入相关组件并支持面向切面编程。
* [Vaadin](https://vaadin.com/)：基于GWT构建的事件驱动框架。使用服务端架构，客户端使用Ajax。
* [Ninja](http://www.ninjaframework.org/)：Java全栈Web开发框架。非常稳固、快速和高效。
* [Ratpack](http://www.ratpack.io/)：一组Java开发函数库，用于构建快速、高效、可扩展且测试完备的HTTP应用程序。

二、资源

社区

活跃的讨论区。

* [r/java](http://www.reddit.com/r/java)：Java社区的Subreddit。
* [stackoverflow](http://stackoverflow.com/questions/tagged/java)：问答平台。

有影响的书籍

具有广泛影响且值得阅读的Java经典书籍。

* [Effective Java (2nd Edition)](http://www.amazon.cn/gp/product/0321356683/ref=as_li_qf_sp_asin_il_tl?ie=UTF8&camp=536&creative=3200&creativeASIN=0321356683&linkCode=as2&tag=vastwork-23)
* [Java Concurrency in Practice](http://www.amazon.com/Java-Concurrency-Practice-Brian-Goetz/dp/0321349601) | [Java并发编程实战](http://www.amazon.cn/gp/product/B0077K9XHW/ref=as_li_qf_sp_asin_il_tl?ie=UTF8&camp=536&creative=3200&creativeASIN=B0077K9XHW&linkCode=as2&tag=vastwork-23)
* [Thinking in Java](http://www.amazon.cn/gp/product/B0011C3CVA/ref=as_li_qf_sp_asin_il_tl?ie=UTF8&camp=536&creative=3200&creativeASIN=B0011C3CVA&linkCode=as2&tag=vastwork-23) | [中文版](http://www.amazon.cn/gp/product/B0011F7WU4/ref=as_li_qf_sp_asin_il_tl?ie=UTF8&camp=536&creative=3200&creativeASIN=B0011F7WU4&linkCode=as2&tag=vastwork-23)

播客

可以一边编程一边听的东西。

* [The Java Posse](http://www.javaposse.com/)

Twitter

值得关注的帐号。

* [Adam Bien](https://twitter.com/AdamBien/)：自由职业者、作家、JavaONE明星演讲者、顾问、Java Champion。
* [Antonio Goncalves](https://twitter.com/agoncal/)：Java Champion、JUG Leader、Devoxx France、Java EE 6/7、JCP、作家。
* [Arun Gupta](https://twitter.com/arungupta/)：Java Champion、JavaONE明星演讲者、JUG Leader、Devoxx4Kids成员、Red Hatter。
* [Bruno Borges](https://twitter.com/brunoborges)：Oracle产品经理、Java Jock。
* [Ed Burns](https://twitter.com/edburns)：Oracle技术团队顾问。
* [Eugen Paraschiv](https://twitter.com/baeldung)：Spring安全课程作者。
* [James Weaver](https://twitter.com/JavaFXpert)：Java、JavaFX、IoT开发者、作者和演讲者。
* [Java EE](https://twitter.com/Java_EE/)：Java EE Twitter官方账号。
* [Java Magazine](https://twitter.com/Oraclejavamag)：Java杂志官方账号。
* [Java.net](https://twitter.com/javanetbuzz/)：Java.net官方账号。
* [Java](https://twitter.com/java/)：Java Twitter官方账号。
* [Javin Paul](https://twitter.com/javinpaul)：知名Java博客作者。
* [Lukas Eder](https://twitter.com/lukaseder)：Data Geekery（jOOQ）创始人兼CEO。
* [Mario Fusco](https://twitter.com/mariofusco)：RedHatter、JUG协调、活跃讲师和作者。
* [Mark Reinhold](https://twitter.com/mreinhold)：Oracle首席架构师、Java平台开发组。
* [Martijn Verburg](https://twitter.com/karianna)：London JUG co-leader、演讲者、作家、Java Champion等。
* [OpenJDK](https://twitter.com/OpenJDK)：OpenJDK官方账号。
* [Reza Rahman](https://twitter.com/reza_rahman)：Java EE、GlassFish、WebLogic传道者、作家、演讲者、开源黑客。
* [Simon Maple](https://twitter.com/sjmaple)：Java Champion、virtualJUG创始人、LJC leader、RebelLabs作者。
* [Stephen Colebourne](https://twitter.com/jodastephen)： Java Champion、演讲者。
* [Tim Boudreau](https://twitter.com/kablosna)：作家、NetBeans大牛。
* [Trisha Gee](https://twitter.com/trisha_gee)：Java Champion、演讲者。

网站

值得阅读的网站。

* [Google Java Style](http://google-styleguide.googlecode.com/svn/trunk/javaguide.html)
* [InfoQ](http://www.infoq.com/)
* [Java Code Geeks](http://www.javacodegeeks.com/)
* [Java.net](http://java.net/)
* [Javalobby](http://java.dzone.com/)
* [JavaWorld](http://www.javaworld.com/)
* [RebelLabs](http://zeroturnaround.com/rebellabs/)
* [The Java Specialist’ Newsletter](http://www.javaspecialists.eu/archive/archive.jsp)
* [TheServerSide.com](http://www.theserverside.com/)
* [Thoughts On Java](http://www.thoughts-on-java.org/)
* [ImportNew](http://www.importnew.com/)（ImportNew 专注 Java 技术）

参与贡献

热烈欢迎参与贡献此列表！

请参阅[CONTRIBUTING](https://github.com/akullpp/awesome-java/blob/master/CONTRIBUTING.md)加入贡献。

原文链接： [github akullpp](https://github.com/akullpp/awesome-java) 翻译： [ImportNew.com](http://www.importnew.com/)- [唐尤华](http://www.importnew.com/author/tangyouhua)  
译文链接： <http://www.importnew.com/14429.html>

# 三、更好的java

Java是最流行的语言之一，但是似乎没人喜欢使用它。好吧，Java仅仅是一种“还好”的编程语言。自从Java 8的面世，我决定编辑一个关于Java的列表，包括库、最佳实践以及工具让我们能更好的使用Java。 

这篇文章在Github上，你可以自由的添加你所使用到的一些Java工具及最佳实践。

* [风格](http://www.importnew.com/16160.html#style)
  + [结构](http://www.importnew.com/16160.html#struct)
    - [构建者模式](http://www.importnew.com/16160.html#builder)
  + [Dependency injection](http://www.importnew.com/16160.html#di)
  + [Avoid Nulls](http://www.importnew.com/16160.html#an)
  + [默认为不可变](http://www.importnew.com/16160.html#ibd)
  + [避免太多的工具类](http://www.importnew.com/16160.html#au)
  + [格式化](http://www.importnew.com/16160.html#format)
    - [Javadoc](http://www.importnew.com/16160.html#javadoc)
  + [Streams](http://www.importnew.com/16160.html#stream)
* [部署](http://www.importnew.com/16160.html#deploy)
  + [框架](http://www.importnew.com/16160.html#fk)
  + [Maven](http://www.importnew.com/16160.html#maven)
    - [重复依赖检测](http://www.importnew.com/16160.html#dc)
  + [持续集成](http://www.importnew.com/16160.html#ci)
  + [Maven仓库](http://www.importnew.com/16160.html#mr)
  + [配置管理](http://www.importnew.com/16160.html#cm)
* [库](http://www.importnew.com/16160.html#lib)
  + [缺失的特性](http://www.importnew.com/16160.html#mf)
    - [Apache Commons](http://www.importnew.com/16160.html#apache-c)
    - [Guava](http://www.importnew.com/16160.html#guava)
    - [Gson](http://www.importnew.com/16160.html#gson)
    - [Java Tuples](http://www.importnew.com/16160.html#jt)
    - [Joda-Time](http://www.importnew.com/16160.html#joda)
    - [Lombok](http://www.importnew.com/16160.html#lombok)
    - [Play框架](http://www.importnew.com/16160.html#play)
    - [SLF4J](http://www.importnew.com/16160.html#slf4j)
    - [jOOQ](http://www.importnew.com/16160.html#jooq)
  + [测试](http://www.importnew.com/16160.html#test)
    - [jUnit 4](http://www.importnew.com/16160.html#junit)
    - [jMock](http://www.importnew.com/16160.html#jmock)
    - [AssertJ](http://www.importnew.com/16160.html#assertj)
* [工具](http://www.importnew.com/16160.html#tools)
  + [IntelliJ IDEA](http://www.importnew.com/16160.html#idea)
    - [Chronon](http://www.importnew.com/16160.html#chronon)
  + [JRebel](http://www.importnew.com/16160.html#jrebel)
  + [Checker框架](http://www.importnew.com/16160.html#checker)
  + [Eclipse Memory Analyzer](http://www.importnew.com/16160.html#e-m-a)
* [资源](http://www.importnew.com/16160.html#resource)
  + [书籍](http://www.importnew.com/16160.html#books)
  + [播客](http://www.importnew.com/16160.html#procast)

### 风格

传统上，Java的编程风格是一种非常冗长的企业级JavaBean风格。但新的风格相对更清晰、更正确、也更容易读懂。

#### 结构

对程序员来说，一件最简单的事件就是数据的传递。传统的方法是定义一个像这样的JavaBean：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | public class DataHolder {      private String data;        public DataHolder() {      }        public void setData(String data) {          this.data = data;      }        public String getData() {          return this.data;      }  } |

这段代码非常冗长、浪费。即使IDE可以自动生成这些代码，但是它还是一种浪费，所以[不要这么做](http://www.javapractices.com/topic/TopicAction.do?Id=84)。

相反，我更喜欢C结构风格的类，该类只保存数据：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public class DataHolder {      public final String data;        public DataHolder(String data) {          this.data = data;      }  } |

这段代码的行数减少了一半。更重要的是，该类是不可以变的，除非你继承该类。所以我们更容易使用它，因为我们知道它不能改变。

你应该使用 ImmutableMap 和 ImmutableList 来替代将对象存储在一个能轻易改变 Map 或 List 中，这些内容在关于不可变性的部分中讨论。

**建造者模式**

如果你有一个相当复杂的对象，你想使用某种结构来创建这种对象的时候，可以考虑使用建造者模式。

你可以在构造你的对象的类中创建一个子类，它的状态是可变的，但是一旦你调用 build 方法，你就能创建一个不可变的对象。

想象一下我们有一个很复杂的 DataHolder，它的建造者类似如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | public class ComplicatedDataHolder {      public final String data;      public final int num;      // lots more fields and a constructor        public static class Builder {          private String data;          private int num;            public Builder data(String data) {              this.data = data;              return this;          }            public Builder num(int num) {              this.num = num;              return this;          }            public ComplicatedDataHolder build() {              return new ComplicatedDataHolder(data, num); // etc          }      }  } |

然后这么使用：

|  |  |
| --- | --- |
| 1  2  3  4 | final ComplicatedDataHolder cdh = new ComplicatedDataHolder.Builder()      .data("set this")      .num(523)      .build(); |

可能会有[其它更好的建造者模式](http://jlordiales.wordpress.com/2012/12/13/the-builder-pattern-in-practice/)例子，这个例子只是让你了解一下建造者模式是什么东西？最后，这里我们试着避开很多公式化的例子，创建了一个不可变的对象和一个很流畅的接口。

#### 依赖注入

这个更多的是软件工程的内容，而不是Java的内容。但是，编写可测试软件的最好方式就是使用[依赖注入（DI）](http://en.wikipedia.org/wiki/Dependency_injection)。因为Java强力推荐面向对象设计，为了创建可测试软件，你需要使用DI。

在Java里面，最典型的例子是[Spring框架](http://projects.spring.io/spring-framework/)。它可以使用基于注解的注入或基于XML配置的注入。如果你想使用XML配置，很重要的事情是[不要过度使用Spring](http://programmers.stackexchange.com/questions/92393/what-does-the-spring-framework-do-should-i-use-it-why-or-why-not)的这种基于XML的配置形式。在这个XML中绝不应该有逻辑和控制结构，它仅仅是依赖注入。

使用的比较好的Spring例子是Google和Square的[Dagger库](http://square.github.io/dagger/)以及Google的[Guice](https://code.google.com/p/google-guice/)。它们都没有使用Spring的XML配置文件，而是把注入的逻辑写在了注解和代码里面。

#### 避免空值

尽可能的避免使用null。不要返回集合时使用null，而应当返回一个空的集合。如果你将要使用null，可以考虑使用 [@Nullable](http://code.google.com/p/google-guice/wiki/UseNullable) 的注解。[IntelliJ IDEA](http://www.jetbrains.com/idea/)内嵌了对 @Nullable 注解的支持。

如果你使用的是[Java 8](http://www.java8.org/)，你可以选择更好的 [Optional](http://www.oracle.com/technetwork/articles/java/java8-optional-2175753.html)类。如果一个值可能存在也可能不存在，可以把它包装为一个 Optional 类，类似于：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | public class FooWidget {      private final String data;      private final Optional<Bar> bar;        public FooWidget(String data) {          this(data, Optional.empty());      }        public FooWidget(String data, Optional<Bar> bar) {          this.data = data;          this.bar = bar;      }        public Optional<Bar> getBar() {          return bar;      }  } |

所以，现在非常清晰，data 从来不会为 null，但是 bar 可能存在，也可能不存在。Optional 有一些方法，如 isPresent，该方法感觉跟判空没什么不同，但是它允许你像下面这样写代码：

|  |  |
| --- | --- |
| 1  2  3  4 | final Optional<FooWidget> fooWidget = maybeGetFooWidget();  final Baz baz = fooWidget.flatMap(FooWidget::getBar)                           .flatMap(BarWidget::getBaz)                           .orElse(defaultBaz); |

这样比连续的进行 null 检查好多了。唯一的缺点是标准库不能很好的支持 Optional，所以对null的处理在某些地方还是必须。

#### 默认为不可变

除非你又充分的理由，否则变量、类和集合都应该是不可变的。

引用变量类型可以通过 final 关键字来指定为不可变：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | final FooWidget fooWidget;  if (condition()) {      fooWidget = getWidget();  } else {      try {          fooWidget = cachedFooWidget.get();      } catch (CachingException e) {          log.error("Couldn't get cached value", e);          throw e;      }  }  // fooWidget is guaranteed to be set here |

现在，你可以保证 fooWidget 不会有被重新负值的风险了。final 关键字可以和 if/else 及 try/catch 块一起工作。当然，如果 fooWidget 自身不是不可变的，你还是可以很轻易的改变它。

集合类型，只要有可能，请使用Guava的 [ImmutableMap](http://docs.guava-libraries.googlecode.com/git/javadoc/com/google/common/collect/ImmutableMap.html)、[ImmutableList](http://docs.guava-libraries.googlecode.com/git/javadoc/com/google/common/collect/ImmutableList.html)或 [ImmutableSet](http://docs.guava-libraries.googlecode.com/git/javadoc/com/google/common/collect/ImmutableSet.html)类。这些已存在的建造者可以让你通过调用它们的 build 方法动态的建造不可变的集合。

你应该通过声明不可变的成员变量（通过 final）和使用不可变的集合类型来创建不可变的类。还有一种选择，你可以申明类自身为 final，这样的话该类就是不能被继承和修改的。

#### 避免使用过多的工具类

如果你发现加了很多的方法在 Util 类中，你就要小心了：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public class MiscUtil {      public static String frobnicateString(String base, int times) {          // ... etc      }        public static void throwIfCondition(boolean condition, String msg) {          // ... etc      }  } |

首先，这些类看起来都非常有吸引力，因为这些方法不属于任何地方，所以你可以在任何地方重用这些代码。

比疾病更糟糕的是治疗。这些类就应该放到属于它们的地方，否则你必须使用这样的通用方法，考虑一下[Java 8](http://www.java8.org/) 的接口的默认方法。然后你组合通用的方法在一个接口里。因为它们是接口，所以你能够用多种方式实现它们。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public interface Thrower {      default void throwIfCondition(boolean condition, String msg) {          // ...      }        default void throwAorB(Throwable a, Throwable b, boolean throwA) {          // ...      }  } |

任何需要它的类都可以很简单地实现这个接口。

#### 格式化

格式对编码的人来说没那么重要。但是它是不是能帮助你持续的关注你的原稿？是不是能帮助别人阅读你的代码？很明显是的。但是请不要浪费一天去对 if 块增加空格来保证它是“匹配”的。

如果你特别想要一份代码格式化指南，我强烈推荐[Google的Java编程风格指南](http://google-styleguide.googlecode.com/svn/trunk/javaguide.html)。该指南最好的部分是[编程实践](http://google-styleguide.googlecode.com/svn/trunk/javaguide.html)，绝对值得一读。

**Javadoc**

文档对使用你代码的人来说是很重要的。这包括[使用的实例](http://docs.guava-libraries.googlecode.com/git-history/release/javadoc/com/google/common/collect/ImmutableMap.Builder.html)及对变量、方法和类的有意义的描述等。

这个推论是如果不需要文档说明的时候，不要写文档。如果对参数没有什么需要说明，或者说明是模糊的，那就不要写说明文档。无意义的文档比完全没有文档还要糟糕，因为它会误导用户以为这就是说明文档。

#### Stream

[Java 8](http://www.java8.org/)拥有非常好的[流](http://blog.hartveld.com/2013/03/jdk-8-33-stream-api.html)和 lambda 表达式语法，你可以这样编写代码：

|  |  |
| --- | --- |
| 1  2  3 | final List<String> filtered = list.stream()      .filter(s -> s.startsWith("s"))      .map(s -> s.toUpperCase()); |

来代替这样的写法：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | final List<String> filtered = Lists.newArrayList();  for (String str : list) {      if (str.startsWith("s") {          filtered.add(str.toUpperCase());      }  } |

这样可以让你写出更流畅的代码，可读写也更强。

### 部署

部署Java程序可能有一点复杂。现在主要的两种部署方式是：使用一个框架或者使用更灵活的原生方法。

#### 框架

因为部署Java比较困难，框架可以帮助解决这个问题。两个最好框架是[Dropwizard](https://dropwizard.github.io/dropwizard/)和[Spring Boot](http://projects.spring.io/spring-boot/)。[Play框架](http://www.playframework.com/)也是部署框架中的一种。

所有的这些框架都是为了降低代码部署的门槛。它们尤其对Java新手或者需要快速完成某件事情有很好的帮助。单个jar包部署比复杂的WAR和EAR部署要更简单。

但是，框架也从某种意义上降低了灵活性，不能自由发挥。所以，如果你的项目不适合于你选择的框架，你将不得不迁移到一个更多手工操作的配置上去。

#### Maven

**其它好的选择：**[Gradle](http://www.gradle.org/)

Maven仍然是标准的构建、打包和运行测试工具。有其它可选的工具，例如Gradle，但是他们与maven采用的方式不同。如果你刚开始使用Maven，你应该从[Maven实例](http://books.sonatype.com/mvnex-book/reference/index.html)开始。

我喜欢有一个根POM来包含所有的使用到的外部依赖。它看起来[像这样](https://gist.github.com/cxxr/10787344)，只有这个根POM有外部依赖，但是你的产品非常大，可以有很多个模块。你的根POM文件应该归属于这样一个项目：有版本控制和发布，类似于其他的Java项目。

所有的Maven项目应该包含你的根POM及所有的版本信息。用这种方式，你能使你的公司使用的每个外部依赖和maven插件都是一致的。如果你需要加一个外部依赖，只要这样就可以了：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <dependencies>      <dependency>          <groupId>org.third.party</groupId>          <artifactId>some-artifact</artifactId>      </dependency>  </dependencies> |

如果你想使用内部依赖，你需要独立的管理项目的各个部分。否则，这将比通过POM来管理稳定的版本更加困难。

**重复依赖检测**

关于Java最好的一部分是存在大量的第三方库可以做任何事情。本质上讲每个API或者工具都是建立在Java SDK上的，所有很容易通过Maven获取下来。

这些库本身又会依赖指定版本的其它类库。如果你下载下来够多的库，你就会遇到版本冲突，类似这样：

|  |  |
| --- | --- |
| 1  2 | Foo库依赖Bar库的1.0版  Widget库依赖Bar库的0.9版本 |

那到底哪个版本会拉到你的项目里面呢？

使用[Maven依赖收敛插件](https://maven.apache.org/enforcer/enforcer-rules/dependencyConvergence.html)，如果你的依赖使用到不同的版本，将会出现错误。你可以有两种选择来解决冲突：

1. 显示的在依赖管理部分选择一个Bar版本。
2. 从Foo或Widget中排除Bar。

至于选择哪种方式就要根据你的具体情况了：如果你想监测一个项目的版本，排除是有意义的。另一方面，如果你想显示的知道是哪个版本，就应该自己选择一个版本，虽然这样当其他依赖更新的时候同时需要更新该依赖。

#### 持续集成

很明显，你需要一种持续集成的服务器，这样能持续的为你构建SNAPSHOT版本和基于git标签的标签构建。

[Jenkins](http://jenkins-ci.org/)和[Travis-CI](https://travis-ci.org/)是很自然的选择。

代码覆盖率是很有用的，[Cobertura](http://cobertura.github.io/cobertura/)有一个[很好的Maven的插件](http://mojo.codehaus.org/cobertura-maven-plugin/usage.html)，以及对CI的支持。还有一些其它Java检查代码覆盖率的工具，但是我只用过Cobertura。

#### Maven仓库

你需要一个地方来存放你打包的JAR、WAR及EAR，所以你需要一个仓库：

通常的选择是[Artifactory](http://www.jfrog.com/)和[Nexus](http://www.sonatype.com/nexus)。两个都还可以，各自有它们的[优缺点](http://stackoverflow.com/questions/364775/should-we-use-nexus-or-artifactory-for-a-maven-repo)。

你应该自己安装的 Artifactory/Nexus 服务器，并在上面[建立依赖包的镜像](http://www.jfrog.com/confluence/display/RTF/Configuring+Artifacts+Resolution)。这样避免在构建你的项目时由于要从外部仓库下载jar而被中断。

#### 配置管理

现在你的代码已经编译了，你的仓库也已经建立了，这个时候你需要将你的代码从开发环境推到生产环境。不要节省这个过程，因为这个过程的自动化将会获得长久的好处。

[Chef](http://www.getchef.com/chef/)、[Puppet](http://puppetlabs.com/)和[Ansible](http://www.ansible.com/home)都是典型的选择。也可以选择我曾写过的[Squadron](http://www.gosquadron.com/)。当然，我想你应该测试一下，因为正确了解这些东西比选择更容易。

尽管有很多工具可以选择，但是不要忘记使你的部署自动化。

### 库

Java可能最好的特性是存在许多可扩展的库。对大部分人的应用来说，用到的库只是整个库中的很小一部分。

#### 缺失的特性

Java的标准库，每向前一步都非常令人惊讶，不过就现在来看，还是缺乏几个关键的特性。

**Apache Commons**

[Apache Commons项目](http://commons.apache.org/)提供了许多有用的库。

* **Commons Codec**提供了许多关于Base64和16进制字符串的编码和解码方法。不要浪费你的时间去重写这些方法。
* **Commons Lang**提供了String类的创建、操作及字符集等一系列五花八门的工具方法。
* **Commons IO**拥有所有的你能想到的文件相关的方法。它有[FileUtils.copyDirectory](http://commons.apache.org/proper/commons-io/javadocs/api-release/org/apache/commons/io/FileUtils.html#copyDirectory(java.io.File,%20java.io.File))、[FileUtils.writeStringToFile](http://commons.apache.org/proper/commons-io/javadocs/api-release/org/apache/commons/io/FileUtils.html#writeStringToFile(java.io.File,%20java.lang.String))、[IOUtils.readLines](http://commons.apache.org/proper/commons-io/javadocs/api-release/org/apache/commons/io/IOUtils.html#readLines(java.io.InputStream))等方法。

**Guava**  
[Guava](http://code.google.com/p/guava-libraries/)是Google旗下一个优秀库，提供了Java现在缺乏的特性。很难提炼关于这个库的所有东西，但是我尽量试试。

**Cache**提供一个简单的方式来建立内存级别的缓存，这可以用来缓存网络访问、磁盘访问以及内存函数或其它的东西。只需要实现[CacheBuilder](http://docs.guava-libraries.googlecode.com/git-history/release/javadoc/com/google/common/cache/CacheBuilder.html)来告诉Guava怎样创建你缓存，这就是你所有的设置。

**Immutable**集合，这里面有许多这种集合：[ImmutableMap](http://docs.guava-libraries.googlecode.com/git/javadoc/com/google/common/collect/ImmutableMap.html)、[ImmutableList](http://docs.guava-libraries.googlecode.com/git/javadoc/com/google/common/collect/ImmutableList.html)甚至[ImmutableSortedMultiSet](http://docs.guava-libraries.googlecode.com/git-history/release/javadoc/com/google/common/collect/ImmutableSortedMultiset.html)。

我也喜欢使用Guava的方式来编写不可变集合：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | // Instead of  final Map<String, Widget> map = new HashMap<String, Widget>();    // You can use  final Map<String, Widget> map = Maps.newHashMap();  There are static classes for Lists, Maps, Sets and more. They're cleaner and easier to read. |

这个库为[Lists](http://docs.guava-libraries.googlecode.com/git-history/release/javadoc/com/google/common/collect/Lists.html)、[Maps](http://docs.guava-libraries.googlecode.com/git-history/release/javadoc/com/google/common/collect/Maps.html)、[Sets](http://docs.guava-libraries.googlecode.com/git-history/release/javadoc/com/google/common/collect/Sets.html)等创建了静态的类。它们读起来更清晰、简单。

如果你使用的是Java 6或7，你可以使用[Collections2](http://docs.guava-libraries.googlecode.com/git-history/release/javadoc/com/google/common/collect/Collections2.html)类，该类提供类似 filter 和 transform 的方法。这些方法允许你在没有[Java 8](http://www.java8.org/)的stream的支持下写出流式的代码。

Guava也做一些简单的事情，如**Joiner**，就是通过分隔符连接字符串以及[中断处理的类](http://docs.guava-libraries.googlecode.com/git-history/release/javadoc/com/google/common/util/concurrent/Uninterruptibles.html)。

**Gson**

Google的[Gson](https://code.google.com/p/google-gson/)库是一个简单快速处理JSON的库，它类似如下工作：

|  |  |
| --- | --- |
| 1  2 | final Gson gson = new Gson();  final String json = gson.toJson(fooWidget); |

它使用起来确实很简单令人愉悦。[Gson用户指南上](https://sites.google.com/site/gson/gson-user-guide)有更多的例子。

**Java Tuples**

一件让我很烦恼的事情是Java没有内嵌建立元组的标准库。幸运的是，[Java tuples项目](http://www.javatuples.org/)解决了这个问题。它使用简单，而且效果很好：

|  |  |
| --- | --- |
| 1  2  3  4 | Pair<String, Integer> func(String input) {      // something...      return Pair.with(stringResult, intResult);  } |

**Joda-Time**

[Joda-Time](http://www.joda.org/joda-time/)是我使用过的最简单的时间库。简单、直接且易于测试。你还能有什么要求？

你唯一的要求如果没使用Java 8，这样的话你就不能使用最新的日期时间库。

**Lombok**

[Lombok](http://projectlombok.org/)是一个非常有趣的库。通过注解，可以让你减少模式化的代码，这种模式化代码让Java看起很不友好。

想要为你变量生成 setters 和 getters 方法吗？非常简单：

|  |  |
| --- | --- |
| 1  2  3 | public class Foo {      @Getter @Setter private int var;  } |

现在，你可以这样做了：

final Foo foo = new Foo();

foo.setVar(5);

这个库还有[更多的功能](http://jnb.ociweb.com/jnb/jnbJan2010.html)，虽然我还没有在生产环境中使用Lombok，但是我已经等不及要使用了。

**Play框架**

**其它好的选择：**[Jersey](https://jersey.java.net/)或[Spark](http://www.sparkjava.com/)

Java中关于RESTful的Web Service存在两个主要的阵营：[JAX-RS](http://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services)和其它。

JAX-RS是传统的方式。通过联合注解和接口的实现的形式来实现Web Service，[Jersey](https://jersey.java.net/)就是这样实现的。这样做的好处是可以很容易的建立客户端，只需要一个实现接口的类就行了。

[Play框架](http://www.playframework.com/)使用非常不同的方式在JVM上建立Web Service：你有一个远程文件，然后你将一个类的引用写入到这个远程文件中。它的本质是一个[完整的MVC框架](http://www.playframework.com/documentation/2.3.x/Anatomy)，但是可以很简单使用它来做Rest Web Service。它对Java和Scala都是有效的。它刚开始是用于Scala，但是在Java中同样很好用。

**SLF4J**

存在有很多Java日志解决方案。我最喜欢的是[SLF4J](http://www.slf4j.org/)，因为它是一个可插入的且能同时联合许多不同的日志框架。你是否有一个奇怪的项目使用了java.util.logging、JCL以及log4j？如果是的话，那SLF4J非常适合你。

[两页的手册](http://www.slf4j.org/manual.html)完全能够满足入门要求。

**jOOQ**

我不喜欢重量级的ORM框架，因为我喜欢SQL。所以我写了许多[JDBC模板](http://docs.spring.io/spring/docs/4.0.3.RELEASE/javadoc-api/org/springframework/jdbc/core/JdbcTemplate.html)，但是很难维护。[jOOQ](http://www.jooq.org/)是一个更好的解决方案。

它使得你在Java里面编写SQL，而同时又能保证类型安全：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | // Typesafely execute the SQL statement directly with jOOQ  Result<Record3<String, String, String>> result =  create.select(BOOK.TITLE, AUTHOR.FIRST\_NAME, AUTHOR.LAST\_NAME)      .from(BOOK)      .join(AUTHOR)      .on(BOOK.AUTHOR\_ID.equal(AUTHOR.ID))      .where(BOOK.PUBLISHED\_IN.equal(1948))      .fetch(); |

使用这种DAO模式，可以通过类的方式来进行数据库访问了。

#### 测试

测试对于你的软件来说是至关重要。这些包可以使你的测试更加简单。

**jUnit 4**

[jUnit](http://junit.org/)应该不需要介绍。在Java里，jUnit是标准的单元测试工具。

但是你可能没有用到jUnit的全部功能。jUnit支持[参数化的测试](https://github.com/junit-team/junit/wiki/Parameterized-tests)，支持一些[规则](https://github.com/junit-team/junit/wiki/Rules)可以让你不需要写那么多的模式化代码，支持随机测试指定代码的[思想](https://github.com/junit-team/junit/wiki/Theories)，支持[假设](https://github.com/junit-team/junit/wiki/Assumptions-with-assume)。

**jMock**

如果你已经使用了依赖注入，这是你付出代价的地方：模拟出能够产生副作用（类似于一个REST的服务器）的代码，并声明该代码调用时的行为。

在Java里，[jMock](http://jmock.org/)是一个标准的模拟工具，它类似于这样：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | public class FooWidgetTest {      private Mockery context = new Mockery();        @Test      public void basicTest() {          final FooWidgetDependency dep = context.mock(FooWidgetDependency.class);            context.checking(new Expectations() {{              oneOf(dep).call(with(any(String.class)));              atLeast(0).of(dep).optionalCall();          }});            final FooWidget foo = new FooWidget(dep);            Assert.assertTrue(foo.doThing());          context.assertIsSatisfied();      }  } |

这段代码通过jMock建立一个 FooWidgetDependency 对象，然后增加期望值。我们期望dep的 call 方法一旦被同一个String调用，则dep的optionalCall 将被调用0次或多次。

如果你重复设置相同的依赖，可能需要将它们放到[测试夹具（test fixture）](https://github.com/junit-team/junit/wiki/Test-fixtures)中，并在@After夹具后添加assertIsSatisfied 。

**AssertJ**

你是不是从来没有用过jUnit这个功能？

|  |  |
| --- | --- |
| 1  2  3  4  5 | final List<String> result = some.testMethod();  assertEquals(4, result.size());  assertTrue(result.contains("some result"));  assertTrue(result.contains("some other result"));  assertFalse(result.contains("shouldn't be here")); |

这确实是很烦人的模式。AssertJ解决了这个问题，你可以将相同的代码做这样的转换：

|  |  |
| --- | --- |
| 1  2  3 | assertThat(some.testMethod()).hasSize(4)                               .contains(&quot;some result&quot;, &quot;some other result&quot;)                               .doesNotContain(&quot;shouldn&#039;t be here&quot;); |

这个流式的接口使得你的测试可读性更强。还有什么是你需要的？

### 工具

**IntelliJ IDEA**

**其它好的选择：**[Eclipse](https://www.eclipse.org/)和[Netbeans](https://netbeans.org/)

最好的Java IDE是[IntelliJ IDEA](http://www.jetbrains.com/idea/)。它有很多的非常好的特性，确确实实的使得冗长的Java变得干脆利落。自动补全相当好用，检测功能也极为强大以及[重构](http://www.amazon.cn/gp/product/B003BY6PLK/ref=as_li_qf_sp_asin_il_tl?ie=UTF8&tag=importnew-23&linkCode=as2&camp=536&creative=3200&creativeASIN=B003BY6PLK)工具也非常的有帮助。

免费的版本对我来说已经足够，但是完整的版本存在更多优秀的特性，例如数据库工具、Spring框架的支持以及Chronon。

**Chronon**

我最喜欢的GDB7功能是在调试时可以即时返回（travel back）。如果你使用的是“旗舰版”，可以通过[Chronon IntelliJ插件](http://blog.jetbrains.com/idea/2014/03/try-chronon-debugger-with-intellij-idea-13-1-eap/)使用这个功能。

你可以获取历史变量、向后的步骤、历史方法等等。第一次使用的时候可能会觉得有点奇怪，但是它能帮助你调试出很多错综复杂的bug，Heisenbugs等工具与之类似的。

**JRebel**

持续集成通常的目标是软件即服务的产品。但是，如果你想不需要等到产品构建完成的时候就能看到代码的变化是什么？

这就是[JRebel](http://zeroturnaround.com/software/jrebel/)做的事情。一旦你将你的服务器挂上一个JRebel客户端，你可以实时的观察到服务器的变化。当你想快速试验的时候这会给你节省大量的时间。

**Checker框架**

Java的类型系统是非常脆弱的。它区分不是字符串，而是正则表达式，也不做[污点检测（taint checking）](http://en.wikipedia.org/wiki/Taint_checking)。但是，[Checker框架](http://types.cs.washington.edu/checker-framework/)会这样做，并且还有其他。它使用像 @Nullable 这样的注解来做类型检查。你甚至能[自定义注解](http://types.cs.washington.edu/checker-framework/tutorial/webpages/encryption-checker-cmd.html)，使得静态分析更加强大。

**Eclipse Memory Analyzer**

即使在Java中，内存泄露也是会发生的。幸运的是，有工具来检测内存泄露。我用过的最好的工具是[Eclipse Memory Analyzer](http://www.eclipse.org/mat/)，它能从heap dump文件帮助你发现问题。

有多种方式来获取JVN进程的heap dump文件，我使用的是[jmap](http://docs.oracle.com/javase/7/docs/technotes/tools/share/jmap.html)：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | $ jmap -dump:live,format=b,file=heapdump.hprof -F 8152  Attaching to process ID 8152, please wait...  Debugger attached successfully.  Server compiler detected.  JVM version is 23.25-b01  Dumping heap to heapdump.hprof ...  ... snip ...  Heap dump file created |

然后你就可以通过Memory Analyzer来打开heapdump.hprof文件，很快的发现发生了什么：

### 资源

资源能帮助你成为Java大师。

#### 书籍

* [Effective Java](http://www.amazon.com/Effective-Java-Edition-Joshua-Bloch/dp/0321356683)
* [Java Concurrency in Practice](http://www.amazon.com/Java-Concurrency-Practice-Brian-Goetz/dp/0321349601)

#### 播客

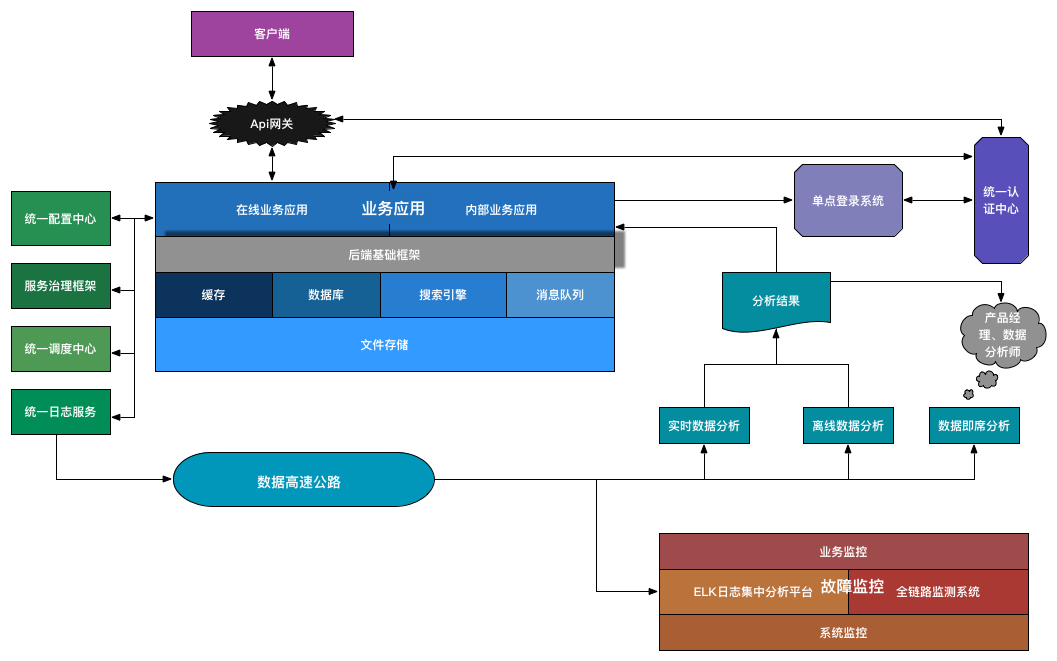
* [The Java Posse](http://www.javaposse.com/)

原文链接： [Better Java](http://www.seancassidy.me/better-java.html) 翻译： [ImportNew.com](http://www.importnew.com/)- [paddx](http://www.importnew.com/author/paddx)  
译文链接： <http://www.importnew.com/16160.html>  
[ **转载请保留原文出处、译者和译文链接。**]

# 四、谈后端

本文更新于2016.12.06,加入了Netflix组件部分

对于一个互联网企业，后端服务是必不可少的一个组成部分。抛开业务应用来说，往下的基础服务设施做到哪些才能够保证业务的稳定可靠、易维护、高可用呢？纵观整个互联网技术体系再结合公司的目前状况，个人认为必不可少或者非常关键的后端基础技术/设施如下图所示：

[](http://www.rowkey.me/images/blog_images/server_basic_stack/server-basic-tech-stack.png)

这里的后端基础设施主要指的是应用在线上稳定运行需要依赖的关键组件/服务等。开发或者搭建好以上的后端基础设施，一般情况下是能够支撑很长一段时间内的业务的。此外，对于一个完整的架构来说，还有很多应用感知不到的系统基础服务，如负载均衡、自动化部署、系统安全等，并没有包含在本文的描述范围内。

## Api网关

在移动app的开发过程中，通常后端提供的接口需要以下功能的支持：

* 负载均衡
* api访问权限控制
* 用户鉴权

一般的做法，使用nginx做负载均衡，然后在每个业务应用里做api接口的访问权限控制和用户鉴权，更优化一点的方式则是把后两者做成公共类库供所有业务调用。但从总体上来看，这三种特性都属于业务的公共需求，更可取的方式则是集成到一起作为一个服务，既可以动态地修改权限控制和鉴权机制，也可以减少每个业务集成这些机制的成本。这种服务就是Api网关(<http://blog.csdn.net/pzxwhc/article/details/49873623>)，可以选择自己实现，也可以使用开源软件实现，如[Kong](https://getkong.org/)。如下图所示：

但是以上方案的一个问题是由于所有api请求都要经过网关，它很容易成为系统的性能瓶颈。因此，可以采取的方案是：去掉api网关，让业务应用直接对接统一认证中心，在基础框架层面保证每个api调用都需要先通过统一认证中心的认证，这里可以采取缓存认证结果的方式避免对统一认证中心产生过大的请求压力。

## 业务应用和后端基础框架

业务应用分为：在线业务应用和内部业务应用。

* 在线业务应用：直接面向互联网用户的应用、接口等，典型的特点就是：请求量大、高并发、高可用、对故障的容忍度低。
* 内部业务应用：这个是面向公司内部的应用。比如，内部数据管理平台、广告投放平台等。相比起在线业务应用，其特点: 数据保密性高、压力小、并发量小、允许故障的发生。

业务应用基于后端的基础框架开发，针对Java后端来说，应该有的几个框架如下：

* MVC框架：从十年前流行的Struts1、2到现在最为推崇的SpringMVC、Jersey以及国人开发的JFinal、阿里的WebX等等，这些框架尤其是后面流行的这些都是各有千秋的。选型的主要因素是看你的团队是否有一个对某框架能够做二次开发、定制的人在。很多时候，针对这些通用的框架，你是需要做一些特定的开发才能满足特定的需求的。比如，很多团队传递参数使用的都是UnderScore的命名法(下划线连接单词)，但是Java中确是使用LowCamel命名的。对于SpringMVC，可以通过注解的alias来指定，但这样需要对每一个参数都要指定alias有点效率太低，此外ModelAttribute也不支持别名，更好的方式是在框架层面统一对参数做Camel命名的转换达到目的。
* IOC框架：ioc带来的好处无须多言。目前Java中最为流行的Spring自诞生就天然支持IOC。
* ORM框架：MyBatis是目前最为流行的orm框架。此外，Spring ORM中提供的JdbcTemplate也很不错。当然，对于分库分表、主从分离这些需求，一般就需要实现自己的ORM框架来支持了，像阿里的tddl、当当的[sharding-jdbc](https://github.com/dangdangdotcom/sharding-jdbc)(从datasource层面解决了分库分表、读写分离的问题，对应用透明、零侵入)。此外，为了在服务层面统一解决分库分表、主从分离、主备切换、缓存、故障恢复等问题，很多公司都是有自己的数据库中间件的，比如阿里的Cobar、360的Atlas、网易的DDB，还有官方提供的[MySQL Proxy](http://downloads.mysql.com/archives/proxy/)以及开源的[MyCat](https://github.com/MyCATApache/Mycat-Server)、[kingshard](https://github.com/flike/kingshard)和收费的[oneproxy](http://www.onexsoft.com/?page_id=3391)。目前，线上有一定规模使用的应该是kingshard，当然如果不缺钱也可以上oneproxy。
* 缓存框架：缓存框架主要指的是对redis、memcached这些缓存服务器的操作统一封装，一般使用Spring的RedisTemplate即可，也可以使用jedis做自己的封装，支持客户端分布式方案、主从等。
* JavaEE应用性能检测框架：对于线上的JavaEE应用，需要有一个统一的框架集成到每一个业务中检测每一个请求、方法调用、jdbc连接、redis连接等的耗时、状态等。[jwebap](http://www.oschina.net/p/jwebap)是一个可以使用的性能检测工具，但由于其已经很多年没有更新，有可能的话建议基于此项目做二次开发。

一般来说，以上几个框架即可以完成一个后端应用的雏形。

对于这些框架来说，最为关键的是根据团队技术构成选择最合适的，有能力开发自己的框架则更好。此外，这里需要提供一个后端应用的模板或生成工具(如maven的archetype)给团队成员使用，可以让大家在开发新的应用的时候，迅速的生成雏形应用，而无需再做一些框架搭建的重复性劳动。

## 缓存、数据库、搜索引擎、消息队列

缓存、数据库、搜索引擎、消息队列这四者都是应用依赖的后端基础服务，他们的性能直接影响到了应用的整体性能，有时候你代码写的再好也许就是因为这些服务导致应用性能无法提升上去。

### 缓存

如缓存五分钟法则所讲：如果一个数据频繁被访问，那么就应该放内存中。这里的缓存就是一种读写效率都非常高的存储方案，能够应对高并发的访问请求，通常情况下也不需要持久化的保证。但相对其他存储来说，缓存一般是基于内存的，成本比较昂贵，因此不能滥用。

缓存可以分为：本地缓存和分布式缓存。

* 本地缓存：主要指的是内存中的缓存机制。在Java中，Google Guava中就提供了本地缓存的实现机制。当然使用java的ConncurrentHashMap你也可以实现自己的本地缓存方案。
* 分布式缓存：指的单独的缓存服务。几年前比较流行的是memcached，但其只是一个KV的存储，支持的数据结构太少。现在最为流行的就是Redis，能够支持丰富的数据结构，基于事件驱动的单线程非阻塞IO也能够应对高并发的场景。集群方案除了官方的redis cluster, 目前比较流行的还有豌豆荚的[codis](https://github.com/wandoulabs/codis)、twitter的[twemproxy](https://github.com/twitter/twemproxy)。

对于缓存的使用，需要注意以下几点：

* 缓存的失效机制：当给某一个key设置了有效期，那么缓存何时对此key进行删除呢？一般来说会有以下几种方式：
  + 守护进程定时去扫描key，找到已经失效的key，然后删除
  + 读取key的时候先去判断key是否失效，如果失效则删除并返回空。
* 缓存的淘汰机制：是当缓存内存达到上限时如何删除缓存中的key。Redis提供了以下数据淘汰策略：
  + volatile-lru：从已设置过期时间的数据集中挑选最近最少使用的数据淘汰
  + volatile-ttl：从已设置过期时间的数据集中挑选将要过期的数据淘汰
  + volatile-random：从已设置过期时间的数据集中任意选择数据淘汰
  + allkeys-lru：从数据集中挑选最近最少使用的数据淘汰
  + allkeys-random：从数据集中任意选择数据淘汰
  + no-enviction（驱逐）：禁止驱逐数据

对于其具体的实现机制，可以参考[《Redis设计与实现》](http://redisbook.com/)一书

* 缓存的更新机制: 通常来说有四种方式：Cache aside, Read through, Write through, Write behind caching，具体的可见陈皓大神的这篇总结：[缓存更新的套路](http://coolshell.cn/articles/17416.html)。
* 缓存的服务过载保护：缓存的服务过载指的是由于缓存失效，而引起后端服务的压力骤增，进一步产生雪崩效应。这个现象和缓存更新是相关的，采取何种策略在缓存失效的时候去更新缓存直接决定了服务过载的保护机制。通常的分为客户端和服务端的应对方案。前者的方案有：基于超时的简单模式、基于超时的常规模式、基于刷新的简单模式、基于刷新的常规模式、基于刷新的续费模式。后者的方案则是很常见的流量控制和服务降级。具体的可以看美团技术团队总结的这篇文章：[Cache应用中的服务过载案例研究](http://mp.weixin.qq.com/s?__biz=MjM5NjQ5MTI5OA==&mid=2651745239&idx=1&sn=60490558770ade79fd9f1e88f9c7c0ac&scene=1&srcid=0617o5PapWXlKUP4OxSzA7KE#rd)。

### 数据库

数据库是后端开发中非常常见的一个服务组件。对于数据库的选型，要根据业务的特点和数据结构的特点来决定。

从存储介质上，数据库可以分为：

* 内存数据库： 数据主要存储在内存中，同时也可以采取措施对数据进行持久化到硬盘中。如Redis、H2DB的内存模式。对于这种数据库，由于内存成本昂贵，因此一定要做好存储的量化分析、容量预估，防止内存不足造成服务不可用。
* 硬盘数据库：数据存储在硬盘上的这种数据库是最为常见的。MySQL、Oracle、Postgresql、HBASE、H2DB、SqlLite等等都是硬盘数据库。此外，[SSDB](https://github.com/ideawu/ssdb)是基于SSD硬盘的KV数据库，支持的数据接口很丰富，是Redis的另外一个选择。

从存储数据类型、数据模式上，数据库可以分为：

* 关系型数据库：MySQL、Oracle、Postgresql都是关系型数据库的，是采用关系模型(关系模型指的就是二维表格模型，而一个关系型数据库就是由二维表及其之间的联系所组成的一个数据组织)来组织数据的数据库。
* 非关系型数据库：非关系型数据库是相对关系型数据库来讲的。以键值对存储，且结构不固定，每一个元组可以有不一样的字段，每个元组可以根据需要增加一些自己的键值对，这样就不会局限于固定的结构，可以减少一些时间和空间的开销。但是，其没有关系型数据库那种严格的数据模式，并不适合复杂的查询以及需要强事务管理的业务。非关系型数据库又可以分为：
  + KV数据库：主要以(key,value)键值对存储数据的数据库。以Redis、RocksDB(levelDB)、SSDB为代表。
  + 文档数据库：总体形式上也是键值对的形式，但是值里面又可以有各种数据结构：数组、键值对、字符串等等。以mongodb、couchdb为代表。
  + 列数据库：也叫作稀疏大数据库，一般是用来存储海量数据的。相对于行数据库，这种数据库是以列为单位存储数据在介质上的。以Hbase、Cassendra为代表。

和数据库相关的一个很重要的就是数据库的索引。有一种说法是：“掌握了索引就等于掌握了数据库”。暂且不去评判此说法是否真的准确，但索引的确关系着数据库的读写性能。需要对数据库的索引原理做到足够的了解才能更好的使用各种数据库。通常来说，Mysql、Oracle、Mongodb这些都是使用的B树作为索引，是考虑到传统硬盘的特点后兼顾了读写性能以及范围查找需求的选择，而Hbase用得LSM则是为了提高写性能对读性能做了牺牲。

### 搜索引擎

搜索引擎也是后端应用中一个很关键的组件，尤其是对内容类、电商类的应用，通过关键词、关键字搜索内容、商品是一个很常见的用户场景。比较成熟的开源搜索引擎有Solr和Elasticsearch，很多中小型互联网公司搜索引擎都是基于这两个开源系统搭建的。它们都是基于Lucence来实现的，不同之处主要在于termIndex的存储、分布式架构的支持等等。

对于搜索引擎的使用，从系统熟悉、服务搭建、功能定制，需要花费较长时间。在这个过程中，需要注意以下问题：

* 搜索引擎与公司现有数据系统的集成。现有的持久化、供搜索的数据的载体是什么, 如何让搜索引擎在全量和增量建索引过程中无缝集成原来的数据载体，才能发挥搜索引擎自身的实时性, 水平扩展性(性能与容量和机器数量成正比)等优势。
* 和数据库一样，对搜索引擎的索引机制也需要做到深入的了解。

更为详细的对于搜索引擎的工程化实践可以参考有赞工程师的这篇文章：[有赞搜索引擎实践(工程篇)](http://www.cnblogs.com/hsydj/p/5303050.html)

另外，搜索引擎还可以用在数据的多维分析上，就是[GrowingIO](https://www.growingio.com/)、[MixPanel](https://mixpanel.com/)中的可以任意维度查询数据报表的功能。当然，[druid](http://druid.io/)也许是一个更好的实现多维分析的方案，官方也有其与es的比较：<http://druid.io/docs/latest/comparisons/druid-vs-elasticsearch.html>。

### 消息队列

软件的组织结构，从开始的面向组件到SOA、SAAS是一个逐渐演变的过程。而到了今天微服务盛行的时代，你都不好意思说自己的系统只是单一的一个系统而没有解耦成一个个service。当然，小的系统的确没有拆分的必要性，但一个复杂的系统拆成一个个service做微服务架构确实是不得不做的事情。

那么问题就来了，service之间的通信如何来做呢？使用什么协议？通过什么方式调用？都是需要考虑的问题。

先抛开协议不谈，service之间的调用方式可以分为同步调用以及异步调用。同步调用的方式无需多说，那么异步调用是怎么进行的呢？一种很常见的方式就是使用消息队列，调用方把请求放到队列中即可返回，然后等待服务提供方去队列中去获取请求进行处理，然后把结果返回给调用方即可（可以通过回调）。

异步调用就是消息中间件一个非常常见的应用场景。此外，消息队列的应用场景还有以下：

* 解耦：一个事务，只关心核心的流程，需要依赖其他系统但不那么重要的事情，有通知即可，无须等待结果。
* 最终一致性：指的是两个系统的状态保持一致，要么都成功，要么都失败，可以有一定的延迟，只要最终达到一致性即可。
* 广播：这是消息队列最基本的功能。生产者只需要发布消息，无须关心有哪些订阅者来消费消息。
* 错峰与流控：当上下游系统处理能力不同的时候就需要类似消息队列的方式做为缓冲区来隔开两个系统。

目前主流的消息队列软件，主要有以下几种：

* ActiveMQ：Java中最为简单的消息队列，是对JMS的实现，没有规定消息的顺序、安全、重发等特性。
* RabbitMQ：是对AMQP协议的实现，对于消息的顺序性、安全、重发等都做了很好的支持。比较适合不允许数据丢失、有事务需求的业务场景下的消息传输。
* Kafka：是基于Log的消息队列，底层依赖于文件的顺序读取，是append-only的。适合对数据丢失不敏感、强调性能的一些海量日志传输场景中。是最近几年大数据领域很火的一个技术。
* ZeroMQ：是一个网络编程的Pattern库，将常见的网络请求形式（分组管理，链接管理，发布订阅等）模式化、组件化，简而言之socket之上、MQ之下。对于MQ来说，网络传输只是它的一部分，更多需要处理的是消息存储、路由、Broker服务发现和查找、事务、消费模式（ack、重投等）、集群服务等。

## 文件存储

不管是业务应用、依赖的后端服务还是其他的各种服务，最终还是要依赖于底层文件存储的。通常来说，文件存储需要满足的特性有：可靠性、容灾性、稳定性，即要保证存储的数据不会轻易丢失，即使发生故障也能够有回滚方案，也要保证高可用率。在底层可以采用传统的RAID作为解决方案，再上一层，目前hadoop的hdfs则是最为普遍的分布式文件存储方案，当然还有NFS、Samba这种共享文件系统也提供了简单的分布式存储的特性。

此外，如果文件存储确实成为了应用的瓶颈或者必须提高文件存储的性能从而提升整个系统的性能时，那么最为直接和简单的做法就是抛弃传统机械硬盘，用SSD硬盘替代。像现在很多公司在解决业务性能问题的时候，最终的关键点往往就是SSD。这也是用钱换取时间和人力成本最直接和最有效的方式。在数据库部分描述的SSDB就是对LevelDB封装之后，利用SSDB的特性的一种高性能KV数据库。

至于HDFS，如果要使用上面的数据，是需要通过hadoop的。类似xx on yarn的一些技术就是将非hadoop技术跑在hdfs上的解决方案(当然也是为了使用MR)。

## 统一认证中心

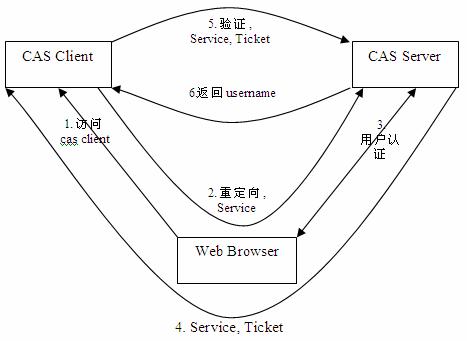
统一认证中心，主要是对app用户、内部用户、app等的认证服务，包括

* 用户的注册、登录验证、token鉴权
* 内部信息系统用户的管理和登录鉴权
* App的管理，包括app的secret生成，app信息的验证(如验证接口签名)等。

之所以需要统一认证中心，就是为了能够集中对这些所有app都会用到的信息进行管理，也给所有应用提供统一的认证服务。尤其是在有很多业务需要共享用户数据的时候，构建一个统一认证中心是非常必要的。此外，通过统一认证中心构建移动app的单点登录也是水到渠成的事情(模仿web的机制，将认证后的信息加密存储到本地磁盘中供多个app使用)。

## 单点登录系统

目前很多大的在线web网站都是有单点登录系统的，通俗的来说就是只需要一次用户登录，就能够进入多个业务应用(权限可以不相同)，非常方便用户的操作。而在移动互联网公司中，内部的各种管理、信息系统同样也需要单点登录系统。目前，比较成熟的、用的最多的单点登录系统应该是耶鲁大学开源的[CAS](https://github.com/Jasig/cas), 可以基于<https://github.com/apereo/cas/tree/master/cas-server-webapp>来定制开发的。此外，国人开源的[kisso](http://git.oschina.net/juapk/kisso)的这个也不错。基本上，单点登录的原理都类似下图所示：



## 统一配置中心

在Java后端应用中，一种读写配置比较通用的方式就是将配置文件写在propeties、yaml、HCON文件中，修改的时候只需要更新文件重新部署即可，可以做到不牵扯代码层面改动的目的。统一配置中心，则是基于这种方式之上的统一对所有业务或者基础后端服务的相关配置文件进行管理的统一服务, 具有以下特性：

* 能够在线动态修改配置文件并生效
* 配置文件可以区分环境(开发、测试、生产等)
* 使用方便: 在java中可以通过注解、xml配置的方式引入相关配置

[disconf](https://github.com/knightliao/disconf)是可以在生产环境使用的一个方案，也可能根据自己的需求开发自己的配置中心(可以选择zookeeper作为配置存储)。

## 服务治理框架

对于外部API调用或者客户端对后端api的访问，可以使用http协议或者说restful(当然也可以直接通过最原始的socket来调用)。但对于内部服务间的调用，一般都是通过RPC机制来调用的。目前主流的RPC协议有：

* RMI
* Hessian
* Thrift
* Dubbo

这些RPC协议各有优劣点，需要针对业务需求做出相应的最好的选择。

这样，当你的系统服务在逐渐增多，RPC调用链越来越复杂，很多情况下，需要不停的更新文档来维护这些调用关系。一个对这些服务进行管理的框架可以大大节省因此带来的繁琐的人力工作。

传统的ESB(企业服务总线)本质就是一个服务治理方案，但esb作为一种proxy的角色存在于client和server之间，所有请求都需要经过esb，使得esb很容易成为性能瓶颈。因此，基于传统的esb，更好的一种设计如下图所示：

如图，以配置中心为枢纽，调用关系只存在于client和提供服务的server之间，就避免了传统esb的性能瓶颈问题。对于这种设计，esb应该支持的特性如下：

* 服务提供方的注册、管理
* 服务消费者的注册、管理
* 服务的版本管理、负载均衡、流量控制、服务降级等
* 服务的容错、熔断等

阿里开源的[dubbo](https://github.com/alibaba/dubbo)则对以上做了很好的实现，也是目前很多公司都在使用的方案。但由于某些原因，dubbo现已不再维护，推荐大家使用当当后来维护的[dubbox](https://github.com/dangdangdotcom/dubbox)。

## 统一调度中心

在很多业务中，定时调度是一个非常普遍的场景，比如定时去抓取数据、定时刷新订单的状态等。通常的做法就是针对各自的业务依赖Linux的cron机制或者java中的quartz。统一调度中心则是对所有的调度任务进行管理，这样能够统一对调度集群进行调优、扩展、任务管理等。[azkaban](https://github.com/azkaban/azkaban)和[oozie](https://github.com/yahoo/oozie)是hadoop的流式工作管理引擎，也可以作为统一调度中心来使用。当然，你也可以使用cron或者quartz来实现自己的统一调度中心。

* 根据cron表达式调度任务
* 动态修改、停止、删除任务
* 支持任务工作流：比如一个任务完成之后再执行下一个任务
* 任务支持脚本、代码、url等多种形式
* 任务执行的日志记录、故障报警

对于Java的quartz这里需要说明一下：这个quartz需要和spring quartz区分，后者是spring对quartz框架的简单实现也是目前使用的最多的一种调度方式。但其并没有做高可用集群的支持。而quartz虽然有集群的支持，但是配置起来非常复杂。现在很多方案都是使用zookeeper来实现spring quartz集群的。这里有一个国人开源的[uncode-shcedule](http://git.oschina.net/uncode/uncode-schedule)对此实现的还不错，可以根据自己的业务需求做二次开发。此外，当当开源的[elastic-job](https://github.com/dangdangdotcom/elastic-job)则在此之上又加入了弹性资源利用等更为强大的功能。

## 统一日志服务

日志是开发过程必不可少的东西。有时候，打印日志的时机、技巧是很能体现出工程师编码水平的。毕竟，日志是线上服务能够定位、排查异常最为直接的信息。

通常的，将日志分散在各个业务中非常不方便对问题的管理和排查。统一日志服务则使用单独的日志服务器记录日志，各个业务通过统一的日志框架将日志输出到日志服务器上。

可以通过实现log4j后者logback的appender来实现统一日志框架，然后通过RPC调用将日志打印到日志服务器上。

## 数据基础设施

数据是最近几年非常火的一个领域。从《精益数据分析》到《增长黑客》，都是在强调数据的非凡作用。很多公司也都在通过数据推动产品设计、市场运营、研发等。详细的可见之前的一篇[《数据杂谈》](http://www.rowkey.me/blog/2016/02/23/data-talk/)，对数据相关的东西做过一些总结。这里需要说明的一点是，只有当你的数据规模真的到了单机无法处理的规模才应该上大数据相关技术，千万不要为了大数据而大数据。很多情况下使用单机程序+mysql就能解决的问题非得上hadoop即浪费时间又浪费人力。

这里需要补充一点的是，对于很多公司，尤其是离线业务并没有那么密集的公司，在很多情况下大数据集群的资源是被浪费的。因此诞生了xx on yarn一系列技术让非hadoop系的技术可以利用大数据集群的资源，能够大大提高资源的利用率，如Docker on yarn(Hulu的VoidBox)。

### 数据高速公路

接着上面讲的统一日志服务，其输出的日志最终是变成数据到数据高速公路上供后续的数据处理程序消费的。这中间的过程包括日志的收集、传输。

* 收集：统一日志服务将日志打印在日志服务上之后，需要日志收集机制将其集中起来。目前，常见的日志收集方案有：scribe、Chukwa、Kakfa和Flume。对比如下图所示：
* 传输：通过消息队列将数据传输到数据处理服务中。对于日志来说，通常选择kafka这种消息队列即可。

此外，这里还有一个关键的技术就是数据库和数据仓库间的数据同步问题，即将需要分析的数据从数据库中同步到诸如hive这种数据仓库时使用的方案。比较简单的、用的也比较多的可以使用[sqoop](http://hortonworks.com/apache/sqoop/)进行基于时间戳的数据同步，此外，阿里开源的[canal](https://github.com/alibaba/canal)实现了基于binlog增量同步，更加适合通用的同步场景，但是基于canal你还是需要做不少的业务开发工作的。推荐另一款国人开源的[MySQL-Binlog](http://git.oschina.net/qiangzigege/MySQL-Binlog)，原理和canal类似，默认提供了任务的后台管理功能，只需要实现接收到binlog后的处理逻辑即可。

### 离线数据分析

离线数据分析是可以有延迟的，一般针对是非实时需求的数据分析工作，产生的也是T-1的报表。目前最常用的离线数据分析技术除了hadoop还有spark。相比hadoop，spark性能上有很大优势，当然对硬件资源要求也高。

对于hadoop，传统的MR编写很复杂，也不利于维护，可以选择使用hive来用sql替代编写mr，但是前提务必要对hive的原理做到了解。可以参见美团的这篇博文来学习:[Hive SQL的编译过程](http://tech.meituan.com/hive-sql-to-mapreduce.html)。而对于spark，也有类似hive的spark sql。

此外，对于离线数据分析，还有一个很关键的就是数据倾斜问题。所谓数据倾斜指的是region数据分布不均，造成有的结点负载很低，而有些却负载很高，从而影响整体的性能。因此，处理好数据倾斜问题对于数据处理是很关键的。对于hive的数据倾斜，可见:[hive大数据倾斜总结](http://www.cnblogs.com/ggjucheng/archive/2013/01/03/2842860.html)。对于spark的倾斜问题，可见：[Spark性能优化指南——高级篇](http://mp.weixin.qq.com/s?__biz=MjM5NjQ5MTI5OA==&mid=2651745207&idx=1&sn=3d70d59cede236eb1cb4f7374387a235&scene=0#rd)。

### 实时数据分析

相对于离线数据分析，实时数据分析也叫在线数据分析，针对的是对数据有实时要求的业务场景，如广告结算、订单结算等。目前，比较成熟的实时技术有storm和spark streaming。相比起storm，spark streaming其实本质上还是基于批量计算的。如果是对延迟很敏感的场景，还是应该使用storm。

对于实时数据分析，需要注意的就是实时数据处理结果写入存储的时候，要考虑并发的问题，虽然对于storm的bolt程序来说不会有并发的问题，但是写入的存储介质是会面临多任务同时读写的。通常采用的方案就是采用时间窗口的方式对数据做缓冲后批量写入。

此外，实时数据处理一般情况下都是基于增量处理的，相对于离线来说并非可靠的，一旦出现故障(如集群崩溃)或者数据处理失败，是很难对数据恢复或者修复异常数据的。因此结合离线+实时是目前最普遍采用的数据处理方案。[Lambda架构](http://www.csdn.net/article/2014-07-08/2820562-Lambda-Linkedln)就是一个结合离线和实时数据处理的架构方案。

### 数据即席分析

离线和实时数据分析产生的一些报表是给数据分析师、产品经理参考使用的，但是很多情况下，线上的程序并不能满足这些需求方的需求。这时候就需要需求方自己对数据仓库进行查询统计。针对这些需求方，SQL上手容易、易描述等特点决定了其可能是一个最为合适的方式。因此提供一个SQL的即席查询工具能够大大提高数据分析师、产品经理的工作效率。Presto、Impala、Hive都是这种工具。如果想进一步提供给需求方更加直观的ui操作界面，可以搭建内部的[Hue](https://github.com/cloudera/hue)。

## 故障监控

对于面向用户的线上服务，发生故障是一件很严重的事情。因此，做好线上服务的故障检测告警是一件非常重要的事情。可以将故障监控分为以下两个层面的监控：

* 系统监控：主要指的对主机的带宽、cpu、内存、硬盘、io等硬件资源的监控。这可以使用开源的nagios、cacti等开源软件进行监控。目前，市面上也有很多第三方服务能够提供对于主机资源的监控，如监控宝等。对于分布式服务集群(如hadoop、storm、kafka、flume等集群)的监控则可以使用[ganglia](http://ganglia.github.io/)。此外，小米开源的[OpenFalcon](https://github.com/open-falcon)也很不错，涵盖了系统监控、JVM监控等，也支持自定义的监控机制。
* 业务监控：是在主机资源层面以上的监控，比如app的pv、uv数据异常、交易失败等。需要业务中加入相关的监控代码，比如在异常抛出的地方，加一段日志记录。

监控还有一个关键的步骤就是告警。告警的方式有很多种：邮件、im、短信等。考虑到故障的重要性不同、告警的合理性、便于定位问题等因素，有以下建议：

* 告警日志要记录发生故障的机器id，尤其是在集群服务中，如果没有记录机器id，那么对于后续的问题定位会很困难。
* 要对告警做聚合，不要每一个故障都单独进行告警，这样会对工程师造成极大的困扰。
* 要对告警做等级划分，不能对所有告警都做同样的优先级处理。
* 使用微信做为告警软件，能够在节省短信成本的情况下，保证告警的到达率。

故障告警之后，那么最最关键的就是应对了。对于创业公司来说，24小时待命是必备的素质，当遇到告警的时候，需要尽快对故障做出反应，找到问题所在，并能在可控时间内解决问题。对于故障问题的排查，基本上都是依赖于日志的。只要日志打的合理，一般情况下是能够很快定位到问题所在的，但是如果是分布式服务，并且日志数据量特别大的情况下，如何定位日志就成为了难题。这里有几个方案：

* 建立ELK(Elastic+Logstash+Kibana)日志集中分析平台，便于快速搜索、定位日志。对于ELK的介绍，可以见：[使用Elasticsearch + Logstash + Kibana搭建日志集中分析平台实践](https://xiequan.info/%E4%BD%BF%E7%94%A8elasticsearch-logstash-kibana%E6%90%AD%E5%BB%BA%E6%97%A5%E5%BF%97%E9%9B%86%E4%B8%AD%E5%88%86%E6%9E%90%E5%B9%B3%E5%8F%B0%E5%AE%9E%E8%B7%B5/)
* 建立分布式请求追踪系统(也可以叫全链路监测系统)，对于分布式系统尤其是微服务架构，能够极大的方便在海量调用中快速定位并收集单个异常请求信息，也能快速定位一条请求链路的性能瓶颈。Google的[Dapper](http://www.cnblogs.com/LBSer/p/3390852.html?spm=5176.100239.blogcont58408.6.xuC3MP)、唯品会的[Mercury](http://mp.weixin.qq.com/s?__biz=MzAwMDU1MTE1OQ==&mid=2653547643&idx=1&sn=c06dc9b0f59e8ae3d2f9feb734da4459&scene=1&srcid=0808MaLgymxNlsh4Z31oWKUi#rd)、阿里的[鹰眼](https://bigbully.github.io/Dapper-translation/?spm=5176.100239.blogcont58408.7.xuC3MP)、新浪的[WatchMan](http://ishare.iask.sina.com.cn/f/68869649.html)都是类似的思路。此外，[腾讯的染色日志机制](https://www.zhihu.com/question/20292868)本质上也是在链路追踪之上根据响应信息做了染色机制。

## Netflix组件

近几年Netflix开源了其内部很多的服务：<https://github.com/Netflix>，包括大数据、构建交付工具、通用运行时服务类库、数据持久化、安全等。里面有一些对应了上面所说的基础设施：

1. [zuul](https://github.com/Netflix/zuul/wiki)这是Netflix所有后端服务最前端的一道门，也就是我们上面说的Api网关, 主要包含了以下功能：
   * 认证授权和安全：识别合法的外部请求，拒绝非法的。
   * 监控：跟踪记录所有有意义的数据以便于给我们一个精确的产品视图。
   * 动态路由：根据需要动态把请求路由到合适的后端服务上。
   * 压力测试：渐进式的增加对集群的压力直到最大值。
   * 限流：对每一种类型的请求都限定流量，拒绝超出的请求。
   * 静态响应控制：对于某些请求直接在边缘返回而不转发到后端集群。
   * 多区域弹性：在aws的多个region中进行请求路由。
2. [Eureka](https://github.com/Netflix/eureka)是Netflix的服务注册发现服务，类似于dubbo的功能。包括负载均衡和容错。
3. [Hystrix](https://github.com/Netflix/hystrix)hystrix是一个类库。基于命令模式，实现依赖服务的容错、降级、隔离等。在依赖多个第三方服务的时候非常有用。此外，还可以通过自定义实现dubbo的filter来给dubbo添加hystrix的特性支持。

此外，Netflix的这些开源组件统称做Netflix oss，现在很火的Spring cloud很多部分都是在这些组件基础上实现的，提供了一整套分布式系统解决方案，涵盖了做分布式微服务需要的服务发现、服务容错、负载均衡、权限控制等。当然，如果你直接选用docker的话，那么K8s本身也提供了这些东西。

以上是本人实践的一些经验。由于知识有限，难免有纰漏，敬请指出。

# 五、秒杀

### [Web系统大规模并发——电商秒杀与抢购](http://blog.csdn.net/linlzk/article/details/42147903)

分类：**Java**

 （2312）  （5）

电商的秒杀和抢购，对我们来说，都不是一个陌生的东西。然而，从技术的角度来说，这对于Web系统是一个巨大的考验。当一个Web系统，在一秒钟内收到数以万计甚至更多请求时，系统的优化和稳定至关重要。这次我们会关注秒杀和抢购的技术实现和优化，同时，从技术层面揭开，为什么我们总是不容易抢到火车票的原因？

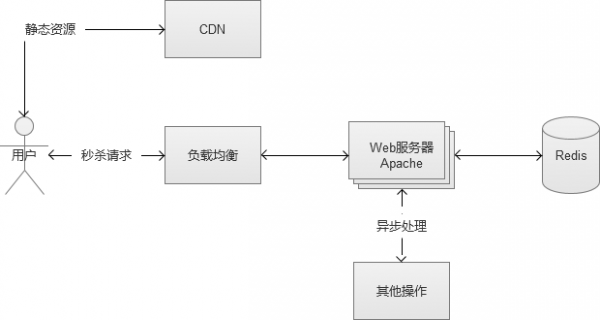
## **一、大规模并发带来的挑战**

在过去的工作中，我曾经面对过5w每秒的高并发秒杀功能，在这个过程中，整个Web系统遇到了很多的问题和挑战。如果Web系统不做针对性的优化，会轻而易举地陷入到异常状态。我们现在一起来讨论下，优化的思路和方法哈。

### **1. 请求接口的合理设计**

一个秒杀或者抢购页面，通常分为2个部分，一个是静态的HTML等内容，另一个就是参与秒杀的Web后台请求接口。

通常静态HTML等内容，是通过CDN的部署，一般压力不大，核心瓶颈实际上在后台请求接口上。这个后端接口，必须能够支持高并发请求，同时，非常重要的一点，必须尽可能“快”，在最短的时间里返回用户的请求结果。为了实现尽可能快这一点，接口的后端存储使用内存级别的操作会更好一点。仍然直接面向MySQL之类的存储是不合适的，如果有这种复杂业务的需求，都建议采用异步写入。



当然，也有一些秒杀和抢购采用“滞后反馈”，就是说秒杀当下不知道结果，一段时间后才可以从页面中看到用户是否秒杀成功。但是，这种属于“偷懒”行为，同时给用户的体验也不好，容易被用户认为是“暗箱操作”。

### **2. 高并发的挑战：一定要“快”**

我们通常衡量一个Web系统的吞吐率的指标是QPS（Query Per Second，每秒处理请求数），解决每秒数万次的高并发场景，这个指标非常关键。举个例子，我们假设处理一个业务请求平均响应时间为100ms，同时，系统内有20台Apache的Web服务器，配置MaxClients为500个（表示Apache的最大连接数目）。

那么，我们的Web系统的理论峰值QPS为（理想化的计算方式）：

20\*500/0.1 = 100000 （10万QPS）

咦？我们的系统似乎很强大，1秒钟可以处理完10万的请求，5w/s的秒杀似乎是“纸老虎”哈。实际情况，当然没有这么理想。在高并发的实际场景下，机器都处于高负载的状态，在这个时候平均响应时间会被大大增加。

就Web服务器而言，Apache打开了越多的连接进程，CPU需要处理的上下文切换也越多，额外增加了CPU的消耗，然后就直接导致平均响应时间增加。因此上述的MaxClient数目，要根据CPU、内存等硬件因素综合考虑，绝对不是越多越好。可以通过Apache自带的abench来测试一下，取一个合适的值。然后，我们选择内存操作级别的存储的Redis，在高并发的状态下，存储的响应时间至关重要。网络带宽虽然也是一个因素，不过，这种请求数据包一般比较小，一般很少成为请求的瓶颈。负载均衡成为系统瓶颈的情况比较少，在这里不做讨论哈。

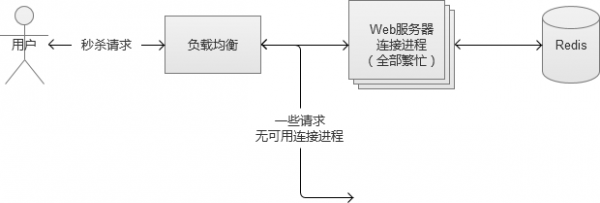
那么问题来了，假设我们的系统，在5w/s的高并发状态下，平均响应时间从100ms变为250ms（实际情况，甚至更多）：

20\*500/0.25 = 40000 （4万QPS）

于是，我们的系统剩下了4w的QPS，面对5w每秒的请求，中间相差了1w。

然后，这才是真正的恶梦开始。举个例子，高速路口，1秒钟来5部车，每秒通过5部车，高速路口运作正常。突然，这个路口1秒钟只能通过4部车，车流量仍然依旧，结果必定出现大塞车。（5条车道忽然变成4条车道的感觉）

同理，某一个秒内，20\*500个可用连接进程都在满负荷工作中，却仍然有1万个新来请求，没有连接进程可用，系统陷入到异常状态也是预期之内。



其实在正常的非高并发的业务场景中，也有类似的情况出现，某个业务请求接口出现问题，响应时间极慢，将整个Web请求响应时间拉得很长，逐渐将Web服务器的可用连接数占满，其他正常的业务请求，无连接进程可用。

更可怕的问题是，是用户的行为特点，系统越是不可用，用户的点击越频繁，恶性循环最终导致“雪崩”（其中一台Web机器挂了，导致流量分散到其他正常工作的机器上，再导致正常的机器也挂，然后恶性循环），将整个Web系统拖垮。

### **3. 重启与过载保护**

如果系统发生“雪崩”，贸然重启服务，是无法解决问题的。最常见的现象是，启动起来后，立刻挂掉。这个时候，最好在入口层将流量拒绝，然后再将重启。如果是redis/memcache这种服务也挂了，重启的时候需要注意“预热”，并且很可能需要比较长的时间。

秒杀和抢购的场景，流量往往是超乎我们系统的准备和想象的。这个时候，过载保护是必要的。如果检测到系统满负载状态，拒绝请求也是一种保护措施。在前端设置过滤是最简单的方式，但是，这种做法是被用户“千夫所指”的行为。更合适一点的是，将过载保护设置在CGI入口层，快速将客户的直接请求返回。

## **二、作弊的手段：进攻与防守**

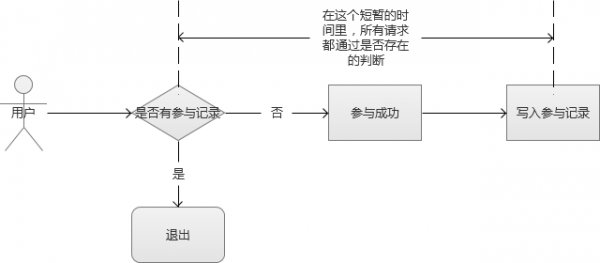
秒杀和抢购收到了“海量”的请求，实际上里面的水分是很大的。不少用户，为了“抢“到商品，会使用“刷票工具”等类型的辅助工具，帮助他们发送尽可能多的请求到服务器。还有一部分高级用户，制作强大的自动请求脚本。这种做法的理由也很简单，就是在参与秒杀和抢购的请求中，自己的请求数目占比越多，成功的概率越高。

这些都是属于“作弊的手段”，不过，有“进攻”就有“防守”，这是一场没有硝烟的战斗哈。

### **1. 同一个账号，一次性发出多个请求**

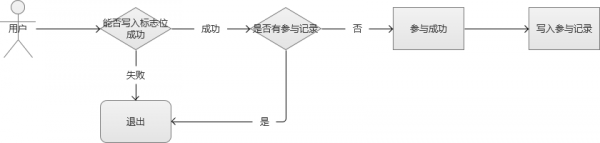
部分用户通过浏览器的插件或者其他工具，在秒杀开始的时间里，以自己的账号，一次发送上百甚至更多的请求。实际上，这样的用户破坏了秒杀和抢购的公平性。

这种请求在某些没有做数据安全处理的系统里，也可能造成另外一种破坏，导致某些判断条件被绕过。例如一个简单的领取逻辑，先判断用户是否有参与记录，如果没有则领取成功，最后写入到参与记录中。这是个非常简单的逻辑，但是，在高并发的场景下，存在深深的漏洞。多个并发请求通过负载均衡服务器，分配到内网的多台Web服务器，它们首先向存储发送查询请求，然后，在某个请求成功写入参与记录的时间差内，其他的请求获查询到的结果都是“没有参与记录”。这里，就存在逻辑判断被绕过的风险。



**应对方案：**

在程序入口处，一个账号只允许接受1个请求，其他请求过滤。不仅解决了同一个账号，发送N个请求的问题，还保证了后续的逻辑流程的安全。实现方案，可以通过Redis这种内存缓存服务，写入一个标志位（只允许1个请求写成功，结合watch的乐观锁的特性），成功写入的则可以继续参加。

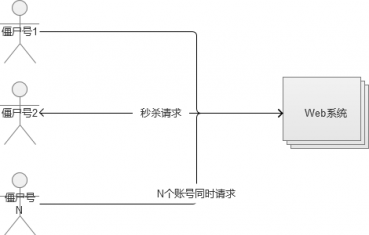


或者，自己实现一个服务，将同一个账号的请求放入一个队列中，处理完一个，再处理下一个。

### **2. 多个账号，一次性发送多个请求**

很多公司的账号注册功能，在发展早期几乎是没有限制的，很容易就可以注册很多个账号。因此，也导致了出现了一些特殊的工作室，通过编写自动注册脚本，积累了一大批“僵尸账号”，数量庞大，几万甚至几十万的账号不等，专门做各种刷的行为（这就是微博中的“僵尸粉“的来源）。举个例子，例如微博中有转发抽奖的活动，如果我们使用几万个“僵尸号”去混进去转发，这样就可以大大提升我们中奖的概率。

这种账号，使用在秒杀和抢购里，也是同一个道理。例如，iPhone官网的抢购，火车票黄牛党。



**应对方案：**

这种场景，可以通过检测指定机器IP请求频率就可以解决，如果发现某个IP请求频率很高，可以给它弹出一个验证码或者直接禁止它的请求：

1. 弹出验证码，最核心的追求，就是分辨出真实用户。因此，大家可能经常发现，网站弹出的验证码，有些是“鬼神乱舞”的样子，有时让我们根本无法看清。他们这样做的原因，其实也是为了让验证码的图片不被轻易识别，因为强大的“自动脚本”可以通过图片识别里面的字符，然后让脚本自动填写验证码。实际上，有一些非常创新的验证码，效果会比较好，例如给你一个简单问题让你回答，或者让你完成某些简单操作（例如百度贴吧的验证码）。
2. 直接禁止IP，实际上是有些粗暴的，因为有些真实用户的网络场景恰好是同一出口IP的，可能会有“误伤“。但是这一个做法简单高效，根据实际场景使用可以获得很好的效果。

### **3. 多个账号，不同IP发送不同请求**

所谓道高一尺，魔高一丈。有进攻，就会有防守，永不休止。这些“工作室”，发现你对单机IP请求频率有控制之后，他们也针对这种场景，想出了他们的“新进攻方案”，就是不断改变IP。



有同学会好奇，这些随机IP服务怎么来的。有一些是某些机构自己占据一批独立IP，然后做成一个随机代理IP的服务，有偿提供给这些“工作室”使用。还有一些更为黑暗一点的，就是通过木马黑掉普通用户的电脑，这个木马也不破坏用户电脑的正常运作，只做一件事情，就是转发IP包，普通用户的电脑被变成了IP代理出口。通过这种做法，黑客就拿到了大量的独立IP，然后搭建为随机IP服务，就是为了挣钱。

**应对方案：**

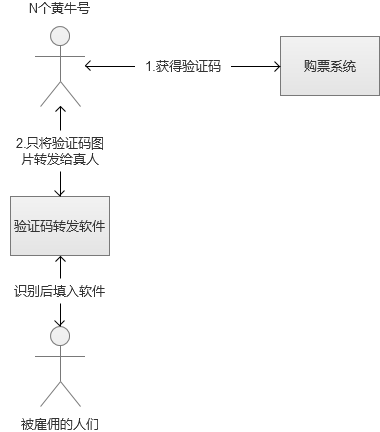
说实话，这种场景下的请求，和真实用户的行为，已经基本相同了，想做分辨很困难。再做进一步的限制很容易“误伤“真实用户，这个时候，通常只能通过设置业务门槛高来限制这种请求了，或者通过账号行为的”数据挖掘“来提前清理掉它们。

僵尸账号也还是有一些共同特征的，例如账号很可能属于同一个号码段甚至是连号的，活跃度不高，等级低，资料不全等等。根据这些特点，适当设置参与门槛，例如限制参与秒杀的账号等级。通过这些业务手段，也是可以过滤掉一些僵尸号。

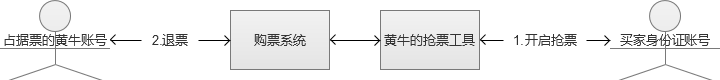
### **4. 火车票的抢购**

看到这里，同学们是否明白你为什么抢不到火车票？如果你只是老老实实地去抢票，真的很难。通过多账号的方式，火车票的黄牛将很多车票的名额占据，部分强大的黄牛，在处理验证码方面，更是“技高一筹“。

高级的黄牛刷票时，在识别验证码的时候使用真实的人，中间搭建一个展示验证码图片的中转软件服务，真人浏览图片并填写下真实验证码，返回给中转软件。对于这种方式，验证码的保护限制作用被废除了，目前也没有很好的解决方案。



因为火车票是根据身份证实名制的，这里还有一个火车票的转让操作方式。大致的操作方式，是先用买家的身份证开启一个抢票工具，持续发送请求，黄牛账号选择退票，然后黄牛买家成功通过自己的身份证购票成功。当一列车厢没有票了的时候，是没有很多人盯着看的，况且黄牛们的抢票工具也很强大，即使让我们看见有退票，我们也不一定能抢得过他们哈。



最终，黄牛顺利将火车票转移到买家的身份证下。

**解决方案：**

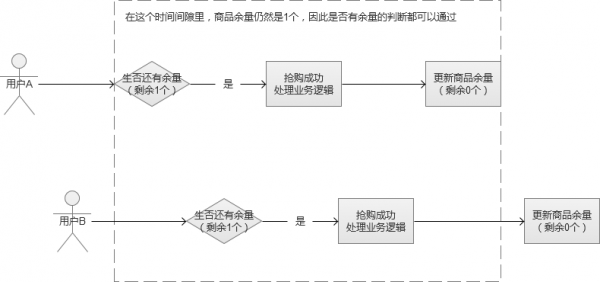
并没有很好的解决方案，唯一可以动心思的也许是对账号数据进行“数据挖掘”，这些黄牛账号也是有一些共同特征的，例如经常抢票和退票，节假日异常活跃等等。将它们分析出来，再做进一步处理和甄别。

## **三、高并发下的数据安全**

我们知道在多线程写入同一个文件的时候，会存现“线程安全”的问题（多个线程同时运行同一段代码，如果每次运行结果和单线程运行的结果是一样的，结果和预期相同，就是线程安全的）。如果是MySQL数据库，可以使用它自带的锁机制很好的解决问题，但是，在大规模并发的场景中，是不推荐使用MySQL的。秒杀和抢购的场景中，还有另外一个问题，就是“超发”，如果在这方面控制不慎，会产生发送过多的情况。我们也曾经听说过，某些电商搞抢购活动，买家成功拍下后，商家却不承认订单有效，拒绝发货。这里的问题，也许并不一定是商家奸诈，而是系统技术层面存在超发风险导致的。

### **1. 超发的原因**

假设某个抢购场景中，我们一共只有100个商品，在最后一刻，我们已经消耗了99个商品，仅剩最后一个。这个时候，系统发来多个并发请求，这批请求读取到的商品余量都是99个，然后都通过了这一个余量判断，最终导致超发。（同文章前面说的场景）

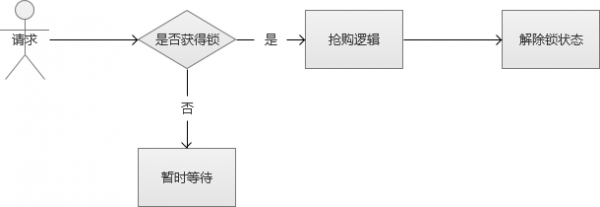


在上面的这个图中，就导致了并发用户B也“抢购成功”，多让一个人获得了商品。这种场景，在高并发的情况下非常容易出现。

### **2. 悲观锁思路**

解决线程安全的思路很多，可以从“悲观锁”的方向开始讨论。

悲观锁，也就是在修改数据的时候，采用锁定状态，排斥外部请求的修改。遇到加锁的状态，就必须等待。



虽然上述的方案的确解决了线程安全的问题，但是，别忘记，我们的场景是“高并发”。也就是说，会很多这样的修改请求，每个请求都需要等待“锁”，某些线程可能永远都没有机会抢到这个“锁”，这种请求就会死在那里。同时，这种请求会很多，瞬间增大系统的平均响应时间，结果是可用连接数被耗尽，系统陷入异常。

### **3. FIFO队列思路**

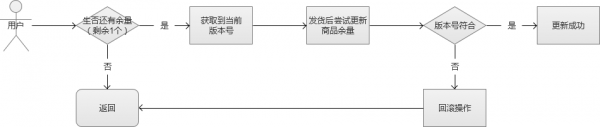
那好，那么我们稍微修改一下上面的场景，我们直接将请求放入队列中的，采用FIFO（First Input First Output，先进先出），这样的话，我们就不会导致某些请求永远获取不到锁。看到这里，是不是有点强行将多线程变成单线程的感觉哈。



然后，我们现在解决了锁的问题，全部请求采用“先进先出”的队列方式来处理。那么新的问题来了，高并发的场景下，因为请求很多，很可能一瞬间将队列内存“撑爆”，然后系统又陷入到了异常状态。或者设计一个极大的内存队列，也是一种方案，但是，系统处理完一个队列内请求的速度根本无法和疯狂涌入队列中的数目相比。也就是说，队列内的请求会越积累越多，最终Web系统平均响应时候还是会大幅下降，系统还是陷入异常。

### **4. 乐观锁思路**

这个时候，我们就可以讨论一下“乐观锁”的思路了。乐观锁，是相对于“悲观锁”采用更为宽松的加锁机制，大都是采用带版本号（Version）更新。实现就是，这个数据所有请求都有资格去修改，但会获得一个该数据的版本号，只有版本号符合的才能更新成功，其他的返回抢购失败。这样的话，我们就不需要考虑队列的问题，不过，它会增大CPU的计算开销。但是，综合来说，这是一个比较好的解决方案。



有很多软件和服务都“乐观锁”功能的支持，例如Redis中的watch就是其中之一。通过这个实现，我们保证了数据的安全。

## **四、小结**

互联网正在高速发展，使用互联网服务的用户越多，高并发的场景也变得越来越多。电商秒杀和抢购，是两个比较典型的互联网高并发场景。虽然我们解决问题的具体技术方案可能千差万别，但是遇到的挑战却是相似的，因此解决问题的思路也异曲同工。