钱进培训是哈法地区资深工程师组成的培训机构，通过各位老师的现身说法，帮助各位学员迅速掌握实战知识，为求职打下坚实的基础。电子邮件：jin.qian.canada@gmail.com钱老师报名、答疑微信号：qianjincanada，或扫描以下二维码添加：

钱老师

加拿大

扫一扫上面的二维码图案，加我微信

Java高级速成班

本课程针对有一定编成基础，希望在短时间内对Java企业级开发有所了解的同学。

通过本课程的学习，学员能够顺利掌握J2EE的体系结构，了解常见的Java框架并熟练使用，具体内容包括:

1. B/s体系结构，HTTP协议栈相关内容（HTML，CSS，JS）
2. 数据库访问的不同方法（Hibernate使用）
3. Spring IOC在企业开发中的应用
4. Restful API/SOAP开发及应用
5. SVN/GIT/MAVEN的应用

第五讲：工程相关

软件工程

scrum

Junit

Log4j

Restful API

SOAP

软件工程术语 Unit testing 中的 Unit（单元）有多种含义，差不多就是程序模块（Module）的意思。

Java的程序主要是由一个个的 Class 组成的，一个类或一个对象当然也是一个单元，而比类更小的单元是类的方法。如果类中的基本单元——如某些方法不能正常工作，在某些输入条件下会得出错误的执行结果，那么如何保证你的类/对象乃至整个应用软件或系统作为一个整体能正常工作呢？所以，简单说，单元测试（优先）的目的就是首先保证一个系统的基本组成单元、模块（如对象以及对象中的方法）能正常工作，这是一种分而治之中的 bottom-up 思想。

所以，单元就是相对独立的功能模块。一个完整的、模块化的程序，都是由许多单元构成，单元完成自己的任务、然后与其它单元进行交互，最终协同完成整个程序功能。

单元测试就是对构成程序的每个单元进行测试。工程上的一个共识是，如果程序的每个模块都是正确的、模块与模块的连接是正确的、那么程序基本上就会是正确的。所以单元测试就是这么一个概念，一种努力保证构成程序的每个模块的正确性，从而保证整个程序的正确性的方法论。

另外一个方面，代码的终极目标有两个，第一个是实现需求，第二个是提高代码质量和可维护性。

单元测试是为了提高代码质量和可维护性，是实现代码的第二个目标的一种方法。

（注：代码的可维护性是指增加一个新功能，或改变现有功能的成本，成本越低，可维护性即越高。）

在2001 年前，开发社区很少听说单元测试，或者有人在积极地做自动单元测试。主要原因还是因为当时缺乏好的工具支持，而且大家对自动单元测试的重要性、必要性与价值也缺乏足够深入的认识。

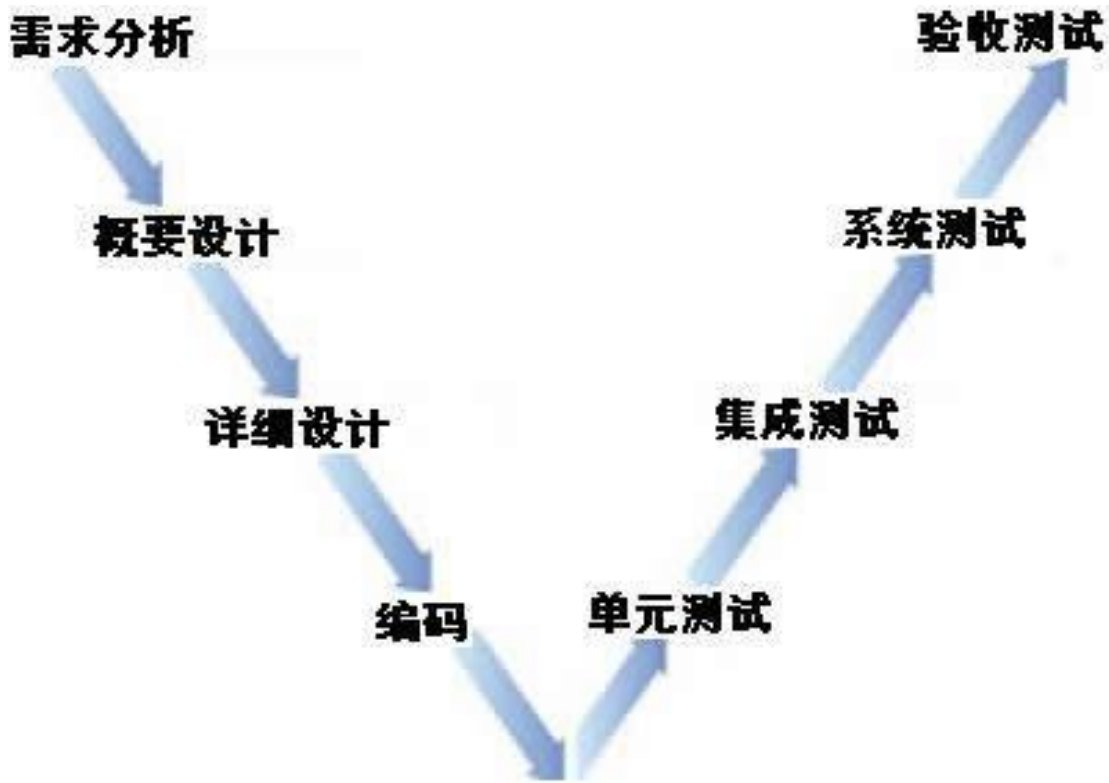
最近这 15 年，随着敏捷运动以及 Java 等新一代 OO 语言、技术和开发工具的兴起，自动单元测试在实践中得到了大面积普及，这主要还是得益于 Kent Beck、Erich Gamma 两位大师联合为 Eclipse IDE 开发的自动单元测试工具 JUnit 的一举成名，以及后来喷涌而出的庞大 xUnit 家族（其中就包括 cppUnit）。

Junit测试是程序员测试，即所谓白盒测试，因为程序员知道被测试的软件如何（How）完成功能和完成什么样（What）的功能。

Junit本质上是一套框架，即开发者制定了一套条条框框，遵循这此条条框框要求编写测试代码，如继承某个类，实现某个接口，就可以用Junit进行自动测试了。

由于Junit相对独立于所编写的代码，可以测试代码的编写可以先于实现代码的编写，XP 中推崇的 test first design的实现有了现成的手段：用Junit写测试代码，写实现代码，运行测试，测试失败，修改实现代码，再运行测试，直到测试成功。以后对代码的修改和优化，运行测试成功，则修改成功。

在开发的各个阶段，包括需求分析、概要设计、详细设计、编码过程中都应该考虑相对应的测试工作，完成相关的测试用例的设计、测试方案、测试计划的编写。这里提到的开发阶段只是举例，根据实际的开发活动进行调整。相关的测试文档也不一定是非常详细复杂的文档，或者什么形式，但应该养成测试驱动的习惯。



第一个写软件的人是Ada（Augusta Ada Lovelace）,在1860年代他尝试为Babbage（Charles Babbage）的机械式计算机写软件。尽管他们的努力失败了，但他们的名字永远载入了计算机发展的史册。

在1950年代，软件伴随着第一台电子计算机的问世诞生了。以写软件为职业的人也开始出现，他们多是经过训练的数学家和电子工程师。1960年代美国大学里开始出现授予计算机专业的学位，教人们写软件。

本世纪中叶软件产业从零开始起步，在短短的50年的时间里迅速发展成为推动人类社会发展的龙头产业，并造就了一批百万、亿万富翁。随着信息产业的发展，软件对人类社会性越来越重要。

在计算机系统发展的初期，硬件通常用来执行一个单一的程序，而这个程序又是为一个特定的目的而编制的。早期当通用硬件成为平常事情的时候，软件的通用性却是很有限的。大多数软件是由使用该软件的个人或机构研制的，软件往往带有强烈的个人色彩。早期的软件开发也没有什么系统的方法可以遵循，软件设计是在某个人的头脑中完成的一个隐藏的过程。而且，除了源代码往往没有软件说明书等文档。

从60年代中期到70年代中期是计算机系统发展的第二个时期，在这一时期软件开始作为一种产品被广泛使用，出现了“软件作坊”专职应别人的需求写软件。这一软件开发的方法基本上仍然沿用早期的个体化软件开发方式，但软件的数量急剧膨胀，软件需求日趋复杂，维护的难度越来越大，开发成本令人吃惊地高，而失败的软件开发项目却屡见不鲜。“软件危机”就这样开始了！

“软件危机”使得人们开始对软件及其特性进行更深一步的研究，人们改变了早期对软件的不正确看法。早期那些被认为是优秀的程序常常很难被别人看懂，通篇充满了程序技巧。现在人们普遍认为优秀的程序除了功能正确，性能优良之外，还应该容易看懂、容易使用、容易修改和扩充。

软件同传统的工业产品相比，有其独特的特性：

1）软件是一种逻辑实体，具有抽象性。这个特点使它与其它工程对象有着明显的差异。人们可以把它记录在纸上、内存、和磁盘、光盘上，但却无法看到软件本身的形态，必须通过观察、分析、思考、判断，才能了解它的功能、性能等特性。

2）软件没有明显的制造过程。一旦研制开发成功，就可以大量拷贝同一内容的副本。所以对软件的质量控制，必须着重在软件开发方面下工夫。

3）软件在使用过程中，没有磨损、老化的问题。软件在生存周期后期不会因为磨损而老化，但会为了适应硬件、环境以及需求的变化而进行修改，而这些修改有不可避免的引入错误，导致软件失效率升高，从而似的软件退化。当修改的成本变得难以接受时，软件就被抛弃。

4）软件对硬件和环境有着不同程度的依赖性。这导致了软件移植的问题。

5）软件的开发至今尚未完全摆脱手工作坊式的开发方式，生产效率低。

6）软件是复杂的，而且以后会更加复杂。软件是人类有史以来生产的复杂度最高的工业产品。软件涉及人类社会的各行各业、方方面面，软件开发常常涉及其它领域的专门知识，这对软件工程师提出了很高的要求。

7）软件的成本相当昂贵。软件开发需要投入大量、高强度的脑力劳动，成本非常高，风险也大。现在软件的开销已大大超过了硬件的开销。

8）软件工作牵涉到很多社会因素。许多软件的开发和运行涉及机构、体制和管理方式等问题，还会设计到人们的观念和心理。这些人的因素，常常成为软件开发的困难所在，直接影响到项目的成败。

1968年北大西洋公约组织的计算机科学家在联邦德国召开的国际学术会议上第一次提出了“软件危机”(software crisis)这个名词。

概括来说，软件危机包含两方面问题：一、如何开发软件，以满足不断增长，日趋复杂的需求；二、如何维护数量不断膨胀的软件产品。

具体地说，软件危机主要有以下表现：

（1）对软件开发成本和进度的估计常常不准确。开发成本超出预算，实际进度比预定计划一再拖延的现象并不罕见。

（2）用户对“已完成”系统不满意的现象经常发生。

（3）软件产品的质量往往靠不住。Bug一大堆，Patch一个接一个。

（4）软件的可维护程度非常之低。

（5）软件通常没有适当的文档资料。

（6）软件的成本不断提高。

（7）软件开发生产率的提高赶不上硬件的发展和人们需求的增长。

软件危机的原因，一方面是与软件本身的特点有关；另一方面是由软件开发和维护的方法不正确有关。

1968年秋季，NATO（北约）的科技委员会召集了近50名一流的编程人员、计算机科学家和工业界巨头，讨论和制定摆脱“软件危机”的对策。在那次会议上第一次提出了软件工程（software engineering）这个概念。

软件工程是一门研究如何用系统化、规范化、数量化等工程原则和方法去进行软件的开发和维护的学科。

软件工程包括两方面内容：软件开发技术和软件项目管理。

软件开发技术包括软件开发方法学、软件工具和软件工程环境。

软件项目管理包括软件度量、项目估算、进度控制、人员组织、配置管理、项目计划等。

瀑布模型是由W.W.Royce在1970年最初提出的软件开发模型。

瀑布模型式是最典型的预见性的方法，严格遵循预先计划的需求分析、设计、编码、集成、测试、维护的步骤顺序进行。步骤成果作为衡量进度的方法，例如需求规格，设计文档，测试计划和代码审阅等等。

瀑布式的主要的问题是它的严格分级导致的自由度降低，项目早期即作出承诺导致对后期需求的变化难以调整，代价高昂。瀑布式方法在需求不明并且在项目进行过程中可能变化的情况下基本是不可行的。

迭代式开发也被称作迭代增量式开发或迭代进化式开发，是一种与传统的瀑布式开发相反的软件开发过程，它弥补了传统开发方式中的一些弱点，具有更高的成功率和生产率。

什么是迭代式开发？

每次只设计和实现这个产品的一部分,

逐步逐步完成的方法叫迭代开发,

每次设计和实现一个阶段叫做一个迭代.

在迭代式开发方法中，整个开发工作被组织为一系列的短小的、

固定长度（如3周）的小项目，被称为一系列的迭代。

每一次迭代都包括了需求分析、设计、实现与测试。

采用这种方法，开发工作可以在需求被完整地确定之前启动，

并在一次迭代中完成系统的一部分功能或业务逻辑的开发工作。

再通过客户的反馈来细化需求，并开始新一轮的迭代。

敏捷软件开发又称敏捷开发，是一种从1990年代开始逐渐引起广泛关注的一些新型软件开发方法，是一种应对快速变化的需求的一种软件开发能力。它们的具体名称、理念、过程、术语都不尽相同，相对于“非敏捷”，更强调程序员团队与业务专家之间的紧密协作、面对面的沟通（认为比书面的文档更有效）、频繁交付新的软件版本、紧凑而自我组织型的团队、能够很好地适应需求变化的代码编写和团队组织方法，也更注重软件开发中人的作用。

001年2月11日到13日，17位软件开发领域的领军人物聚集在美国犹他州的滑雪胜地雪鸟（Snowbird）雪场。经过两天的讨论，“敏捷”（Agile）这个词为全体聚会者所接受，用以概括一套全新的软件开发价值观。这套价值观，通过一份简明扼要的《敏捷宣言》，传递给世界，宣告了敏捷开发运动的开始。

《敏捷宣言》

我们通过身体力行和帮助他人来揭示更好的软件开发方式。经由这项工作，我们形成了如下价值观：

个体与交互 重于 过程和工具

可用的软件 重于 完备的文档

客户协作　重于 合同谈判

响应变化　重于 遵循计划

在每对比对中，后者并非全无价值，但我们更看重前者。

合同谈判

项目开发一般都是跟随着合同开始的。由客户经理同客户交谈，在合同中确定一些法律条文以及交付产品包含的功能，然后客户经理拿着合同回到公司由开发小组经过长时间开发后再交付给客户。在开发期间，如果需要变更合同，则需要经过一系列流程。开发过程中，有些客户可能只是简单的询问一下进度，有的甚至是到最后交付日了直接来问你要东西，当产品不能满足合同规定时就需要和客户谈判进行解决。仅仅通过合同谈判来开发产品，客户在开发过程中就不会进行实质性的反馈，导致最终的产品不满意也就很正常了。

产品开发在早期市场不成熟的时候一般为公司领导（产品经理、LPDT）驱动，后期转变为用户驱动、销售驱动、服务驱动，在矩阵式管理模式下，产品事业部和开发管理部作为两个部门时，在做产品开发的时候就会类似在进行合同谈判，从一开始就会在两个部门之间产生争执而不是合作，这势必会影响产品的开发。

遵循计划

当拿到合同时，公司就开始组建项目组进行开发。此时需要项目经理制定出明确的计划，必须详细列出需求、设计、开发、测试、部署的各项任务。当项目组提交看似完整齐全的计划后，公司就视这个不变的计划作为项目组对公司的承诺，没有特殊情况时必须遵循制定的计划，如果想要变化，则需要经过公司评审。

软件复杂度有三个主要因素：业务、技术和人员。

技术和业务最终都需要人来执行，而每个人拥有不同的技能、经验和观点，当这些人在一起合作时又会使得开发过程变得复杂。

这些复杂性将导致开发过程中存在很多不确定性，所以项目初期制定的计划其实基本上不能真正的坚持下来。而当项目开发遇到困难时，项目组可能会为了表明自己做计划的能力，或者不想经历复杂的变更过程，而继续努力的坚持这个已经错误的计划。范围、时间和成本，这个金三角几乎没有领导不知道，而项目组为了保证”遵循计划“，对外宣称项目组运行状况良好，保证当前人员在预计时间肯定能保质保量的完成开发。而代码质量只有开发人员知道，领导们容易忽略和难以控制这个环节，所以最后一味的遵循计划就势必导致提供给客户的是一个不满意的产品。

过程与工具

计划制定后，项目组需要在类似ISO9000、CMMI、NPD等一些常用的项目管理方法下进行开发管理。工欲善其事，必先利其器，可以找到很多工具来支持开发，需求阶段有原型工具、需求管理跟踪工具，设计阶段有Rose、PD，开发阶段有各种IDE和辅助插件，测试阶段有TD等。

合适的工具能很好的帮助开发，但当在开发人员面前出现大量庞大笨重甚至不好用的工具和开发环境时，就会像缺少工具一样，都是不好的。在开发过程中，可能会出现夸大了工具的作用,当反应说开发工具对开发起起决定性的影响时，这很有可能是在计划阶段就开始错了。

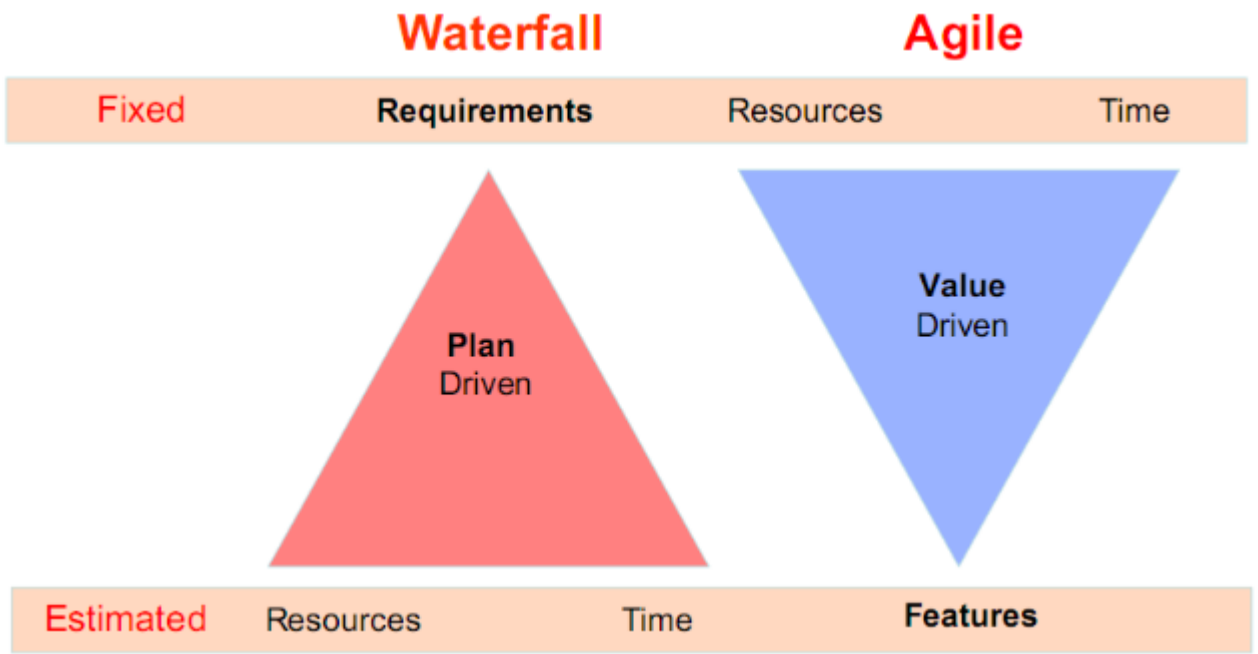
面面俱到的文档

瀑布式开发下，文档承载着各阶段之间的信息传递。需求文档中定义详细用例，每个细节，原型中定义界面表现，甚至每个控件的具体位置，设计中的UML图，数据库设计图，测试用例文档等等，如果没有文档，开发将不能在过程中顺利依次展开。

编写和维护一份详尽的需求文档总是一个好主意，然而就像前面所说业务复杂性带来的不确定性，除非给我们充足的时间，否则我们不可能一开始就想清楚所有细节。另一方面，编写文档需要花费大量的时间，如果考虑和代码的同步时，工作量更是急速上升，如果不考虑同步时，过多的文档反而比过少的文档还糟。当我们花大部分时间浪费的文档，仍旧只能以降低质量来遵循计划的执行。

瀑布式开发是计划驱动的，合同谈判后项目组制定计划并且遵循计划，在过程与工具支持下通过面面俱到的文档来定义不变的需求和其他文档，在时间不够时可以通过增加人员来缓解压力。

而敏捷开发是价值驱动，通过自组织团队在短期迭代过程中不断的交付对用后有用的功能来进行产品开发。



敏捷方法强调以人为本，专注于交付对客户有价值的软件。在高度协作的开环境中，使用迭代式的方式进行增量开发，经常使用反馈进行思考、反省和总结，不停的进行自我调整和完善。

敏捷是一种指导思想或开发方式，但是它没有明确告诉我们到底采用什么样的流程进行开发，而Scrum和XP就是敏捷开发的具体方式了，你可以采用Scrum方式也可以采用XP方式。

敏捷开发方法是很多轻量级方法的总称，它旗下有很多具体的开发过程和方法。主要的有：极限编程（XP）、特征驱动软件开发（FDD）、SCRUM开发方法 等等。SCRUM开发方法是由Jeff Sutherland在1993年创立，Jeff也是制定敏捷宣言的17位专家之一。SCRUM借用了橄榄球运动中的术语——一个团队拿球向前冲。

Scrum的英文意思是橄榄球运动的一个专业术语，表示“争球”的动作；把一个开发流程的名字取名为Scrum，意味着开发团队在开发一个项目时，大家像打橄榄球一样迅速、富有战斗激情、人人你争我抢地完成它。

严格的说，SCRUM是遵循敏捷方法的一个软件开发框架。在SCRUM框架中，融入敏捷开发的精神和思想，就被称作SCRUM开发方法。SCRUM是一个 什么样的开发框架呢？简单说，它由三个角色（Role），三种会议（Ceremonie），三项工件（Artifact）组成。

角色（Role）：产品主管（Procuct Owner），他负责项目的商业价值；SCRUM师傅（ScrumMaster），他负责团队的运转和生产；以及自组织的团队。

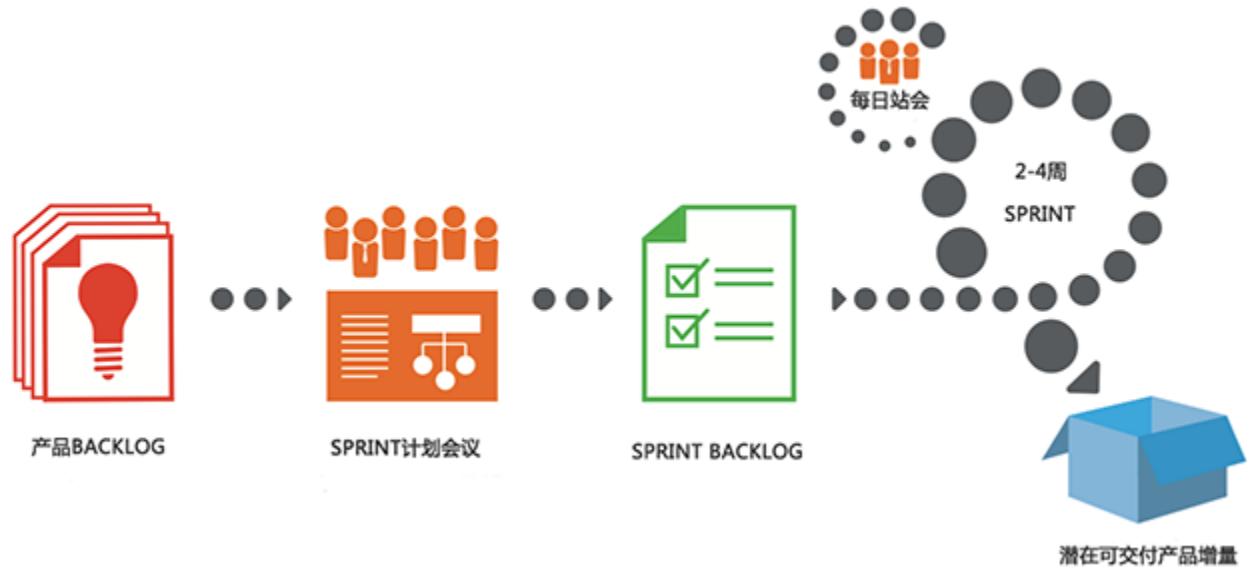
会议（Ceremonie）：迭代计划会议，每日晨会（daily scrum meetings），迭代回顾会议。

工件（Artifact）：用来排列任务的优先级和跟踪任务。待开发任务列表（product backlog），迭代任务列表（the sprint backlog），进度图（burndown chart）

以每日晨会为例，在SCRUM中，明显的提到，在会议中每个人只可以说三件事情：

1. 我昨天做了什么
2. 我今天准备做什么
3. 我在工作中遇到了什么障碍。

Scrum 是一个用于开发和维持复杂产品的框架，是一个增量的、迭代的开发过程。在这个框架中，整个开发过程由若干个短的迭代周期组成，一个短的迭代周期称为一个Sprint，每个Sprint的建议长度是2到4周(互联网产品研发可以使用1周的Sprint)。在Scrum中，使用产品Backlog来管理产品的需求，产品backlog是一个按照商业价值排序的需求列表，列表条目的体现形式通常为用户故事。Scrum团队总是先开发对客户具有较高价值的需求。在Sprint中，Scrum团队从产品Backlog中挑选最高优先级的需求进行开发。挑选的需求在Sprint计划会议上经过讨论、分析和估算得到相应的任务列表，我们称它为Sprint backlog。在每个迭代结束时，Scrum团队将递交潜在可交付的产品增量。 Scrum起源于软件开发项目，但它适用于任何复杂的或是创新性的项目。



1、我们首先需要确定一个Product Backlog（按优先顺序排列的一个产品需求列表），这个是由Product Owner 负责的；

2、Scrum Team根据Product Backlog列表，做工作量的预估和安排；

3、有了Product Backlog列表，我们需要通过 Sprint Planning Meeting（Sprint计划会议） 来从中挑选出一个Story作为本次迭代完成的目标，这个目标的时间周期是1~4个星期，然后把这个Story进行细化，形成一个Sprint Backlog；

4、Sprint Backlog是由Scrum Team去完成的，每个成员根据Sprint Backlog再细化成更小的任务（细到每个任务的工作量在2天内能完成）；

产品需求

产品 BACKLOG（示例）					
ID	Name	Imp	Est	How to demo	Notes
1	存款	30	5	登录，打开存款界面，存入 10 欧元，转到我的账户余额界面，检查我的余额增加了 10 欧元。	需要 UML 顺序图。目前不需要考虑加密的问题。
2	查看自己的交易明细	10	8	登录，点击“交易”，存入一笔款项。返回交易页面，看到新的存款显示在页面上。	使用分页技术避免大规模的数据库查询。和查看用户列表的设计相似。

5、在Scrum Team完成计划会议上选出的Sprint Backlog过程中，需要进行 Daily Scrum Meeting（每日站立会议），每次会议控制在15分钟左右，每个人都必须发言，并且要向所有成员当面汇报你昨天完成了什么，并且向所有成员承诺你今天要完成什么，同时遇到不能解决的问题也可以提出，每个人回答完成后，要走到黑板前更新自己的 Sprint burn down（Sprint燃尽图）；

6、做到每日集成，也就是每天都要有一个可以成功编译、并且可以演示的版本；很多人可能还没有用过自动化的每日集成，其实TFS就有这个功能，它可以支持每次有成员进行签入操作的时候，在服务器上自动获取最新版本，然后在服务器中编译，如果通过则马上再执行单元测试代码，如果也全部通过，则将该版本发布，这时一次正式的签入操作才保存到TFS中，中间有任何失败，都会用邮件通知项目管理人员；

7、当一个Story完成，也就是Sprint Backlog被完成，也就表示一次Sprint完成，这时，我们要进行 Srpint Review Meeting（演示会议），也称为评审会议，产品负责人和客户都要参加（最好本公司老板也参加），每一个Scrum Team的成员都要向他们演示自己完成的软件产品（这个会议非常重要，一定不能取消）；

8、最后就是 Sprint Retrospective Meeting（回顾会议），也称为总结会议，以轮流发言方式进行，每个人都要发言，总结并讨论改进的地方，放入下一轮Sprint的产品需求中；

每日站立会议

任务看板

```
<dependency>
                    <groupId>junit</groupId>
                    <artifactId>junit</artifactId>
                    <version>4.12</version>
                    <scope>test</scope>
                </dependency>
```

```
public class RegularExpressionTest extends TestCase {

 private String zipRegEx = "^\\d{5}([\\-]\\d{4})?$";
 private Pattern pattern;

 protected void setUp() throws Exception {
  this.pattern = Pattern.compile(this.zipRegEx);
 }

 public void testZipCode() throws Exception{
  Matcher mtcher = this.pattern.matcher("22101");
  boolean isValid = mtcher.matches();
  assertTrue("Pattern did not validate zip code", isValid);
 }
}
```

```
public class RegularExpressionTest {
 private static String zipRegEx = "^\\d{5}([\\-]\\d{4})?$";
 private static Pattern pattern;

 @BeforeClass
 public static void setUpBeforeClass() throws Exception {
  pattern = Pattern.compile(zipRegEx);
 }

 @Test
 public void verifyGoodZipCode() throws Exception{
  Matcher mtcher = this.pattern.matcher("22101");
  boolean isValid = mtcher.matches();
  assertTrue("Pattern did not validate zip code", isValid);
 }
}
```

@Test：把一个方法标记为测试方法

@Before：每一个测试方法执行前自动调用一次

@After：每一个测试方法执行完自动调用一次

@BeforeClass：所有测试方法执行前执行一次，在测试类还没有实例化就已经被加载，所以用static修饰

@AfterClass：所有测试方法执行完执行一次，在测试类还没有实例化就已经被加载，所以用static修饰

@Ignore：暂不执行该测试方法

在test方法内除了使用Assert的assertEquals()方法外，还能使用assertFalse()、assertTrue()、assertNull()、assertNotNull()、assertSame()、assertNotSame()等断言函数。而且如果使用的是Junit4，结合Hamcrest，使用

assertThat([value], [matcher statement])方法可以实现更灵活的断言判断（前提是引入hamcrest的jar包）。

assertEquals(true,

jsonObj.getJSONObject("response").getBoolean("success"));

Logback 与 Log4J实际上，这两个日志框架都出自同一个开发者之手，Logback 相对于 Log4J 有更多的优点：

同样的代码路径，Logback 执行更快

更充分的测试

原生实现了 SLF4J API（Log4J 还需要有一个中间转换层）

内容更丰富的文档

支持 XML 或者 Groovy 方式配置

配置文件自动热加载

从 IO 错误中优雅恢复

自动删除日志归档

自动压缩日志成为归档文件

支持 Prudent 模式，使多个 JVM 进程能记录同一个日志文件

支持配置文件中加入条件判断来适应不同的环境

更强大的过滤器

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <!-- encoders are assigned the type ch.qos.logback.classic.encoder.PatternLayoutEncoder             by default -->
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n            </pattern>
    </encoder>
  </appender>
  <root level="debug">
    <appender-ref ref="STDOUT"/>
  </root>
</configuration>
```

```
package chapters.configuration;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class MyApp1 {
  final static Logger logger = LoggerFactory.getLogger(MyApp1.class);

  public static void main(String[] args) {
    logger.info("Entering application.");
  }
}
```

LoggerFactory 的 getLogger()方法接收一个参数，以这个参数决定logger的名字，这里传入了 MyApp1这个类的 Class 实例，那么 logger 的名字便是 MyApp1 这个类的全限定类名：chapters.configuration.MyApp1。

ALL: 是最低等级的，用于打开所有日志记录。

DEBUG: 指出细粒度信息事件对调试应用程序是非常有帮助的。

INFO: 表明消息在粗粒度级别上突出强调应用程序的运行过程。

WARN: 表明会出现潜在错误的情形。

ERROR: 指出虽然发生错误事件，但仍然不影响系统的继续运行。

FATAL: 指出每个严重的错误事件将会导致应用程序的退出。

OFF: 是最高等级的，用于关闭所有日志记录。

优先级从低到高分别是 ALL、DEBUG、INFO、WARN、ERROR、FATAL、OFF

通过定义级别，您可以控制到应用程序中相应级别的日志信息的开关。比如在这里定义了 INFO 级别，则应用程序中所有 DEBUG 级别的日志信息将不被打印出来。程序会打印高于或等于所设置级别的日志，设置的日志等级越高，打印出来的日志就越少。如果设置级别为 INFO，则优先级高于等于 INFO 级别（如：INFO、 WARN、ERROR, FATAL）的日志信息将可以被输出，小于该级别的如 DEBUG 将不会被输出。

在后端logger系统中，有三个最基础的概念需要先熟悉：

Logger 日志记录器 - 日志记录器就是一个普通的Java类而已。

Appender 输出源 - 输出源是日志最终输出的地方，比如控制台或者文件

Layout(Encoder) 布局 - 布局决定了日志的打印格式，比如使用 %r [%t] %-5p %c %x - %m%n 可以打印出 467 [main] INFO org.apache.log4j.examples.Sort - Exiting main method.这样的日志。

```
<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
      <!-- encoders are assigned the type ch.qos.logback.classic.encoder.PatternLayoutEncoder
         by default -->
      <encoder>
         <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n
         </pattern>
      </encoder>
   </appender>

<appender name="FILE" class="ch.qos.logback.core.FileAppender">
   <file>myApp.log</file>
   <encoder>
     <pattern>%date %level [%thread] %logger{10} [%file:%line] %msg%n</pattern>
   </encoder>
</appender>
```

Layout主要用来把log事件转换成String。一般布局都是配置在 Appender 里面的：

```
<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
   <encoder>
    <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
   </encoder>
</appender>
```

%d{HH: mm:ss.SSS}——日志输出时间

%thread——输出日志的进程名字，这在Web应用以及异步任务处理中很有用

%-5level——日志级别，并且使用5个字符靠左对齐

%logger{36}——日志输出者的名字

%msg——日志消息

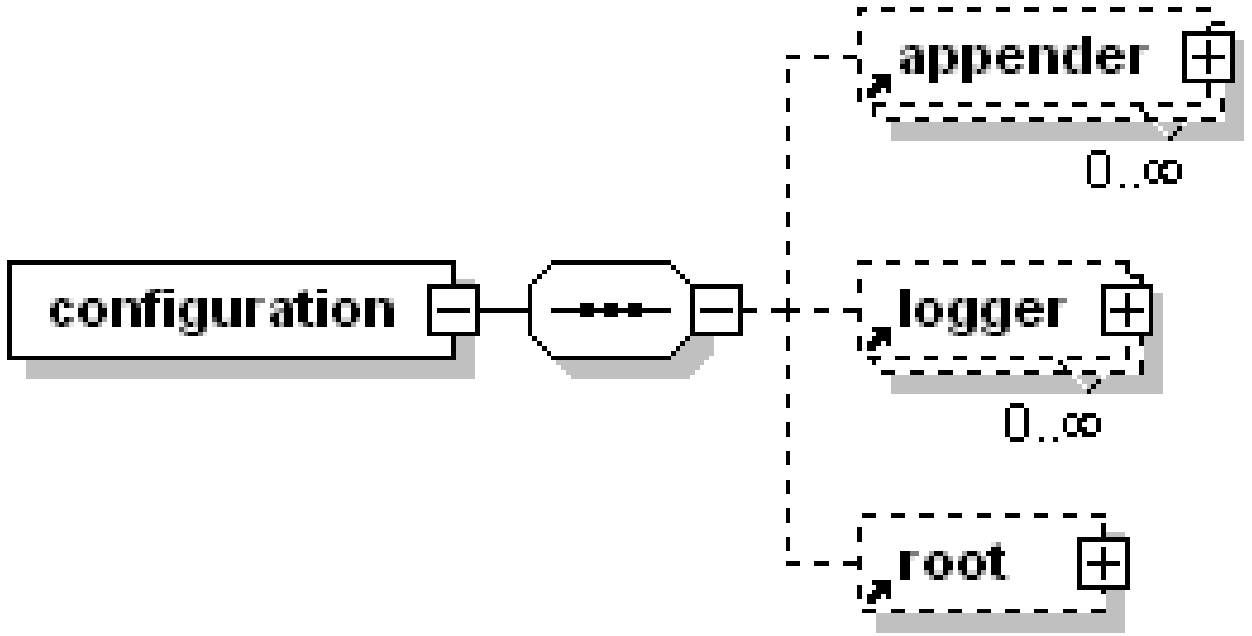
%n——平台的换行符

1、根节点<configuration>，包含下面三个属性：

scan: 当此属性设置为true时，配置文件如果发生改变，将会被重新加载，默认值为true。

scanPeriod: 设置监测配置文件是否有修改的时间间隔，如果没有给出时间单位，默认单位是毫秒。当scan为true时，此属性生效。默认的时间间隔为1分钟。

debug: 当此属性设置为true时，将打印出logback内部日志信息，实时查看logback运行状态。默认值为false。



6、子节点<loger>：用来设置某一个包或具体的某一个类的日志打印级别、以及指定<appender>。<loger>仅有一个name属性，一个可选的level和一个可选的addtivity属性。

可以包含零个或多个<appender-ref>元素，标识这个appender将会添加到这个loger

name: 用来指定受此loger约束的某一个包或者具体的某一个类。

level: 用来设置打印级别，大小写无关：TRACE, DEBUG, INFO, WARN, ERROR, ALL和OFF，还有一个特俗值INHERITED或者同义词NULL，代表强制执行上级的级别。 如果未设置此属性，那么当前loger将会继承上级的级别。

addtivity: 是否向上级loger传递打印信息。默认是true。同<loger>一样，可以包含零个或多个<appender-ref>元素，标识这个appender将会添加到这个loger。

7、子节点<root>:它也是<loger>元素，但是它是根loger,是所有<loger>的上级。只有一个level属性，因为name已经被命名为"root",且已经是最上级了。

level: 用来设置打印级别，大小写无关：TRACE, DEBUG, INFO, WARN, ERROR, ALL和OFF，不能设置为INHERITED或者同义词NULL。 默认是DEBUG。

```
<logger name="org.hibernate.type.descriptor.sql.BasicBinder" level="TRACE" />
<logger name="org.hibernate.type.descriptor.sql.BasicExtractor" level="DEBUG" />
<logger name="org.hibernate.SQL" level="DEBUG" />
<logger name="org.hibernate.engine.QueryParameters" level="DEBUG" />
<logger name="org.hibernate.engine.query.HQLQueryPlan" level="DEBUG" />
```

```
<properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <logback.version>1.1.7</logback.version>
        <slf4j.version>1.7.21</slf4j.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>${slf4j.version}</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-core</artifactId>
            <version>${logback.version}</version>
        </dependency>
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>${logback.version}</version>
            </dependency>
    </dependencies>
```

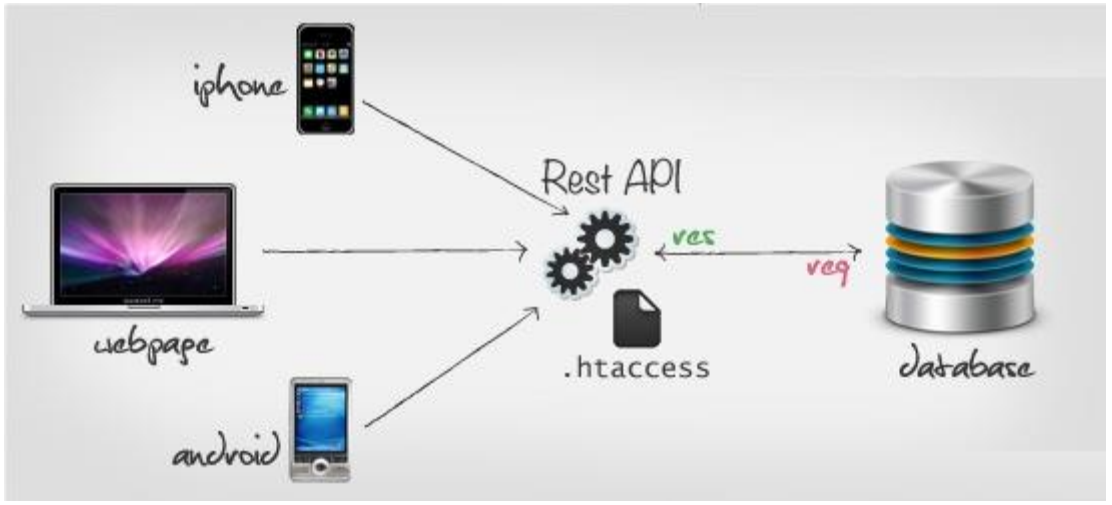
前端后端融在一起的，比如之前的PHP，JSP等。

REST这个词，是Roy Thomas Fielding在他2000年的博士论文中提出的。

Fielding是一个非常重要的人，他是HTTP协议（1.0版和1.1版）的主要设计者、Apache服务器软件的作者之一、Apache基金会的第一任主席。所以，他的这篇论文一经发表，就引起了关注，并且立即对互联网开发产生了深远的影响。

Fielding将他对互联网软件的架构原则，定名为REST，即Representational State Transfer的缩写。这个词组的翻译是"表现层状态转化"。

如果一个架构符合REST原则，就称它为RESTful架构。



"本文研究计算机科学两大前沿----软件和网络----的交叉点。长期以来，软件研究主要关注软件设计的分类、设计方法的演化，很少客观地评估不同的设计选择对系统行为的影响。而相反地，网络研究主要关注系统之间通信行为的细节、如何改进特定通信机制的表现，常常忽视了一个事实，那就是改变应用程序的互动风格比改变互动协议，对整体表现有更大的影响。我这篇文章的写作目的，就是想在符合架构原理的前提下，理解和评估以网络为基础的应用软件的架构设计，得到一个功能强、性能好、适宜通信的架构。“

越来越多的人开始意识到，网站即软件，而且是一种新型的软件。

这种"互联网软件"采用客户端/服务器模式，建立在分布式体系上，通过互联网通信，具有高延时（high latency）、高并发等特点。

网站开发，完全可以采用软件开发的模式。但是传统上，软件和网络是两个不同的领域，很少有交集；软件开发主要针对单机环境，网络则主要研究系统之间的通信。互联网的兴起，使得这两个领域开始融合，现在我们必须考虑，如何开发在互联网环境中使用的软件。

资源（Resources）

REST的名称"表现层状态转化"中，省略了主语。"表现层"其实指的是"资源"（Resources）的"表现层"。

所谓"资源"，就是网络上的一个实体，或者说是网络上的一个具体信息。它可以是一段文本、一张图片、一首歌曲、一种服务，总之就是一个具体的实在。你可以用一个URI（统一资源定位符）指向它，每种资源对应一个特定的URI。要获取这个资源，访问它的URI就可以，因此URI就成了每一个资源的地址或独一无二的识别符。

所谓"上网"，就是与互联网上一系列的"资源"互动，调用它的URI。

表现层（Representation）

"资源"是一种信息实体，它可以有多种外在表现形式。我们把"资源"具体呈现出来的形式，叫做它的"表现层"（Representation）。

比如，文本可以用txt格式表现，也可以用HTML格式、XML格式、JSON格式表现，甚至可以采用二进制格式；图片可以用JPG格式表现，也可以用PNG格式表现。

URI只代表资源的实体，不代表它的形式。严格地说，有些网址最后的".html"后缀名是不必要的，因为这个后缀名表示格式，属于"表现层"范畴，而URI应该只代表"资源"的位置。它的具体表现形式，应该在HTTP请求的头信息中用Accept和Content-Type字段指定，这两个字段才是对"表现层"的描述。

状态转化（State Transfer）

访问一个网站，就代表了客户端和服务器的一个互动过程。在这个过程中，势必涉及到数据和状态的变化。

互联网通信协议HTTP协议，是一个无状态协议。这意味着，所有的状态都保存在服务器端。因此，如果客户端想要操作服务器，必须通过某种手段，让服务器端发生"状态转化"（State Transfer）。而这种转化是建立在表现层之上的，所以就是"表现层状态转化"。

客户端用到的手段，只能是HTTP协议。具体来说，就是HTTP协议里面，四个表示操作方式的动词：GET、POST、PUT、DELETE。它们分别对应四种基本操作：GET用来获取资源，POST用来新建资源（也可以用于更新资源），PUT用来更新资源，DELETE用来删除资源。

RESTful架构：

（1）每一个URI代表一种资源；

（2）客户端和服务器之间，传递这种资源的某种表现层；

（3）客户端通过四个HTTP动词，对服务器端资源进行操作，实现"表现层状态转化"。

RESTful架构有一些典型的设计误区。

最常见的一种设计错误，就是URI包含动词。因为"资源"表示一种实体，所以应该是名词，URI不应该有动词，动词应该放在HTTP协议中。

举例来说，某个URI是/posts/show/1，其中show是动词，这个URI就设计错了，正确的写法应该是/posts/1，然后用GET方法表示show。

当GET, PUT和PATCH请求成功时，要返回对应的数据，及状态码200，即SUCCESS

当POST创建数据成功时，要返回创建的数据，及状态码201，即CREATED

当DELETE删除数据成功时，不返回数据，状态码要返回204，即NO CONTENT

当GET 不到数据时，状态码要返回404，即NOT FOUND

任何时候，如果请求有问题，如校验请求数据时发现错误，要返回状态码 400，即BAD REQUEST

当API 请求需要用户认证时，如果request中的认证信息不正确，要返回状态码 401，即NOT AUTHORIZED

当API 请求需要验证用户权限时，如果当前用户无相应权限，要返回状态码 403，即 FORBIDDEN

Serialization 和 Deserialization即序列化和反序列化。RESTful API以规范统一的格式作为数据的载体，常用的格式为json或xml，以json格式为例，当客户端向服务器发请求时，或者服务器相应客户端的请求，向客户端返回数据时，都是传输json格式的文本，而在服务器内部，数据处理时基本不用json格式的字符串，而是native类型的数据，最典型的如类的实例，即对象（object），json仅为服务器和客户端通信时，在网络上传输的数据的格式，服务器和客户端内部，均存在将json转为native类型数据和将native类型数据转为json的需求，其中，将native类型数据转为json即为序列化，将json转为native类型数据即为反序列化。因此，确保序列化和反序列化方法的实现，是开发RESTful API最重要的一步准备工作

Java EE 6 引入了对 JSR-311 的支持。JSR-311（JAX-RS：Java API for RESTful Web Services）旨在定义一个统一的规范，使得 Java 程序员可以使用一套固定的接口来开发 REST 应用，避免了依赖于第三方框架。同时，JAX-RS 使用 POJO 编程模型和基于标注的配置，并集成了 JAXB，从而可以有效缩短 REST 应用的开发周期。

JAX-RS 的具体实现由第三方提供，例如 Sun 的参考实现 Jersey、Apache 的 CXF 以及 JBoss 的 RESTEasy。

```
@Path ("items")
 @Produces (MediaType.APPLICATION_XML)
 Public class ItemsResource {

    @Context UriInfo uriInfo;

    @GET
    Items listItems() {
       Return Allitems();
    }

    @POST
    @Consumes (MediaType.APPLICATION_XML)
    Public Response create(Item item) throws ItemCreationException {
       Item newItem = createItem(item);
       URI newItemURI = uriInfo.getRequestUriBuilder().path(newItem.getId()).build();
       return Response.created(newItemURI).build();
    }

    ...
 }
```

ItemsResource类是管理一组项目的Web服务，类中导入了JAX-RS 1.1注解，类和接口。

@Path注解指定了资源的相对路径，在这里是“items”，类资源URI是基于应用程序上下文的，因此，如果应用程序上下文在这个例子中是http://example.com，那么类资源的URI就是http://example.com/items，这意味着如果一个客户端直接请求URI http://example.com/items，ItemsResource类将会执行。

@GET注解指定了注解的方法，这里是listItems()方法，它处理HTTP GET请求，当某个客户端直接发起对ItemsResource资源的HTTP GET请求时，JAX-RS运行时调用listItems()方法处理这个GET请求。

@Produces注解，它指定了返回给客户端的MIME媒体类型，在ItemsResource这个例子中，@Produces注解指定了MediaType.APPLICATION_XML，MediaType类是一个抽象的MIME媒体类型，MediaType.APPLICATION_XML是XML内容MIME媒体类型的抽象 — application/xml。

注解如@Produces建议JAX-RS自动转换某些内容类型，例如，listItems()方法返回一个Items类型的Java对象，JAX-RS自动将这个Java类型转换成application/xml MIME类型，使用这个MIME类型响应客户端的HTTP请求。注意仅当返回的类型默认是支持的才会自动转换，例如，如果Items是一个JAXB注解Bean，那么将会自动转换

@POST注解指定了注解的方法，这里是create()方法，它负责响应HTTP POST请求。在这个例子中，这个方法创建了一个新项目，然后返回一个表示它已创建了一个新项目的响应，当客户端直接向ItemsResource资源发起HTTP POST请求时，JAX-RS运行时调用create()方法处理POST请求。

注意@Consumes注解是在create()方法上指定的，注解指定了方法能够接受的来自客户端的MIME媒体类型。如果你在类上指定@Consumes，它适用于类中的所有方法，如果你在方法上指定@Consumes，它会覆盖你在类上指定的@Consumes注解包含的MIME类型。在这个例子中，@Consumes注解指定create()方法可接受XML内容，即MIME类型application/xml，这是从MIME类型转换到Java类型。当某个客户端在POST请求中提交XML内容时，JAX-RS调用create()方法自动将传入的XML内容转换成方法需要的Item Java类型。

JAX-RS提供了一些标注将一个资源类，一个POJOJava类，封装为Web资源。标注包括：

@Path，标注资源类或方法的相对路径

@GET，@PUT，@POST，@DELETE，标注方法是用的HTTP请求的类型

@Produces，标注返回的MIME媒体类型

@Consumes，标注可接受请求的MIME媒体类型

@PathParam，@QueryParam，@HeaderParam，@CookieParam，@MatrixParam，@FormParam,分别标注方法的参数来自于HTTP请求的不同位置，例如@PathParam来自于URL的路径，@QueryParam来自于URL的查询参数，@HeaderParam来自于HTTP请求的头信息，@CookieParam来自于HTTP请求的Cookie。作者：冬瓜baba链接：http://www.jianshu.com/p/bf93c944a4c4來源：简书著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

https://jersey.github.io/documentation/latest/getting-started.html

钱进培训是哈法地区资深工程师组成的培训机构，通过各位老师的现身说法，帮助各位学员迅速掌握实战知识，为求职打下坚实的基础。电子邮件：jin.qian.canada@gmail.com钱老师报名、答疑微信号：qianjincanada，或扫描以下二维码添加：

钱老师

加拿大

扫一扫上面的二维码图案，加我微信

Java高级速成班

本课程针对有一定编成基础，希望在短时间内对Java企业级开发有所了解的同学。

通过本课程的学习，学员能够顺利掌握J2EE的体系结构，了解常见的Java框架并熟练使用，具体内容包括:

1. B/s体系结构，HTTP协议栈相关内容（HTML，CSS，JS）
2. 数据库访问的不同方法（Hibernate使用）
3. Spring IOC在企业开发中的应用
4. Restful API/SOAP开发及应用
5. SVN/GIT/MAVEN的应用