

高级软件工程复习题

2021 版

罗铁坚

2021/12/10

目录

一、 概念解释	3
■ 三类问题	3
■ 建模和模型	5
■ 设计与重构	7
■ 代码和界面	10
■ 测试	13
■ 两种基本途径	14
二、 选择题	15
■ 软件需求和应用场景	15
三、 判断题	18
■ 三类问题	18
■ 需求	18
■ 建模和模型	18
■ 设计与重构	18
■ 代码和界面	18
■ 测试	19
■ 两种基本途径	19
四、 设计题	19
■ 需求分析	19
■ 设计模式	20
■ 程序与界面设计	25
■ 测试	32
五、 简答题	33
■ 软件理论	33
■ 需求	35
■ 建模与模型	35
■ 设计与重构	37
■ 代码和界面	38
■ 框架与开发	45
■ 测试	51
■ 两种途径	55

一、 概念解释

■ 三类问题

1. Answer:

软件是利用计算机拓展人类理解事物、执行任务和解决问题的智力系统。

软件工程的使命和任务是探索和实践高效研发面向解决实际问题的优质智力系统或产品的理论方法和实现技术。

2. Answer:

Description, 有描述清楚一件事物的能力;

Prediction, 该理论有预测该事物行为或发展的能力;

Control, 按照其理论的描述和预测做到精准调控。

3. Answer:

软件第一定律: (3 分)

1) 软件必须具备拓展人类解决问题的能力, 并且具有衡量软件外部可观察行为能力的评价指标。

2) 软件系统能力由其外部可观察及可测试验证来界定, 并且软件内部实现必须支持软件系统能力的升级变化要求。

3) 软件系统能力的价值与某些具体的业务问题密切相关。

软件第二定律: (1 分)

任何实现用户价值的功能一定通过与运行系统交互完成, 而每个具体交互有一个明确的目的和操作。

软件第三定律: (2 分)

所有的软件技术进步都要经受不断设计出更合理的软件内部架构及其支撑软件的持续升级的检验。

4. Answer:

RUP(Rational Unified Process)即统一软件过程, RUP 是一种用例驱动的, 以架构为中心的, 采用迭代增量方式开发的软件工程过程。UML(Unified Modeling Language)即统一建模语言, 是用图形方式描述一个系统的静态结构和动态行为的一种可视化的面向对象建模语言。RUP 利用 UML 进行可视化建模, RUP 更倾向于为面向对象的软件系统开发提供了一种方法论的指导。

5. Answer:

TDD 是从内到外的研发方式，从代码层面入手，要求先有单元测试，然后针对测试实现函数内部功能，通过所有测试。并且对现有代码出现的 bug 也应当归纳到新的测试中，并在后续开发中通过所有测试。

BDD 是从外到内的研发方式，从需求和领域场景入手，先分析软件面对的领域真实场景，提出软件应该满足的 feature，将每个 feature 分解为可能的应用场景，再对每一个应用场景进行软件内部逻辑的设计。实际的需求分析和设计方案生成过程中，领域知识相关的专家也可以参与进来，为后续项目成功验收提供保障。

两者的区别主要体现在研发步骤，TDD 借助单元测试来保证现有代码和新加入的代码尽可能满足设计所描述的行为，但软件可能满足了测试却没有满足实际应用场景的需要。BDD 从软件的领域相关表现着手，软件对外应该表现的行为出发，设计软件的内层逻辑，力求使软件实现从一开始就面向较为真实的场景

6. Answer:

软件生命周期分为三个时期八个阶段：

软件定义：问题定义、可行性研究；

软件开发：需求分析、概要设计、详细设计、编码、测试；

软件运行：软件维护

7. Answer:

产品代码托管—使用版本管理，一般是 git+gitlab

运维配置托管—同样使用 git+gitlab 进行变更跟踪和管理

自动构建—所有新增产品代码自动地进行构建（Build）

自动回归测试—所有新增代码在构建成功后执行自动测试

自动打包—所有代码在完成回归测试之后生成内部发布软件包

自动部署—在完成自动打包可以自动调用运维脚本完成部署

模糊测试（可选）—通过自动化生成测试用例，寻找代码缺陷

8. Answer:

高内聚是指设计某个模块时，模块内部的一系列相关功能的相关程度要高。

低耦合是指模块之间相互联系的紧密程度，模块之间联系越紧密，则耦合性越高，模块的独立性就越差。

9. Answer:

三大根本问题:

- (1) 构思创造发明有价值的软件系统或产品,找到检验其质量的理论和方法。
- (2) 探寻应对领域问题和软件需求变化的可动态升级软件模块和架构的理论和方法。
- (3) 探讨测算软件设计实现和部署运营的效益模型的理论和方法。

九项重要任务:

- (1) 软件需求和应用场景 — 定义
- (2) 业务建模和数据模型 —
- (3) 软件架构和应用框架 —
- (4) 设计模式和代码重构 — 开发
- (5) 接口设计和前端编程 —
- (6) 软件测试与自动测试 —
- (7) 安全问题和防御设计 —
- (8) 性能评价和设计优化 —
- (9) 集成部署和系统运营 — 运营

需求
分析
设计
开发
测试
部署
运营
维护
升级
迁移

两种研发方式:

- (1) 基于人类学习的软件设计
- (2) 基于机器学习的软件设计

10. Answer:

a) User Story 用来定义功能需求,用来帮助定义用户用例,用来定义交替使用的用例

b) 判断一个 User Story 好坏的标准:

- i. 确定性 (Specific) S
- ii. 可评估 (Measurable) M
- iii. 可实现 (Achievable) A
- iv. 相关性 (Relevant) R
- v. 时间限制 (Time boxed) T

c) 五个为什么是一种帮助你探索发现 User Story 的真正商业相关性的方法

■ 建模和模型

11. Answer:

- 1) 确定业务边界;
- 2) 找到业务主角;
- 3) 获取业务用例。

12. Answer:

UML 建模过程就是一个迭代过程，每次迭代都建立相应的模型，分为以下几个阶段：

- 1、**分析阶段**。建模的目的就是捕捉系统的功能需求，分析、提取所开发系统的“客观世界”领域的类以及描述它们的合作关系。常以用例图为首选模型。
- 2、**设计阶段**。建模的目的就是通过考虑实现环境，将分析阶段的模型拓展与转化为可行的技术实现方案。常建立以类图为主的静态模型，及包括状态模型、顺序模型、活动模型与合作模型等动态模型。
- 3、**实现阶段**。主要工作就是进行编码，同时对已构造的模型作相应的修正。
- 4、**配置阶段**。通过模型描述所开发系统的软硬件配置情况建立配置模型。
- 测试阶段**。使用前几个阶段所构造的模型来指导与协助测试工作。

13. Answer:

敏捷开发，以用户的需求进化为核心，采用迭代、循序渐进的方法进行软件开发。在敏捷开发中，软件项目在构建初期被切分成多个子项目，各个子项目的成果都经过测试，具备可视、可集成和可运行使用的特征。换言之，就是把一个大项目分为多个相互联系，但也可独立运行的小项目，并分别完成，在此过程中软件一直处于可使用状态。

Scrum 方法：是迭代式增量软件开发过程，通常用于敏捷软件开发。Scrum 包括了一系列实践和预定义角色的过程骨架。Scrum 中的主要角色包括同项目经理类似的 **Scrum 主管**角色负责维护过程 and 任务，产品负责人代表利益所有者，开发团队包括了所有开发人员

14. Answer:

计划-文档开发的核心思想是按工序将问题化简，便于分工协作。它将软件生命周期划分为需求分析和规范、架构设计、实现和综合、验证、操作和维护，并且规定了它们自上而下、相互衔接的固定次序，所以它也被称为瀑布软件开发过程。

15. Answer:

持续集成 (CI) 指的是频繁地（一天多次）将代码集成到主干。它有两个好处：快速发现错误和防止分支大幅偏离主干。它的目的是让产品可以快速迭代，同时还能保持高质量。它的核心措施是代码集成到主干之前，必须通过自动化测试。只要有一个测试用例失败，就不能集成。(from: 博客)

持续集成 (CI) 是一种软件开发实践，即团队开发成员经常集成他们的工作，通常每个成员每天至少集成一次，也就意味着每天可能会发生多次集成。每次集成都通过自动化的构建（包括编译，发布，自动化测试）来验证，从而尽快地发现集成错误。(from: 大师 Martin Fowler 对持续集成的定义)

持续集成 (CI)：是对部署的代码提高保障的一个关键技术，其中的每一个改变都

会推送到代码库中，引发一个集成测试集合来确保任何事不会垮掉。(from: SaaS 课本)

16. Answer:

MVC 是模型(model)-视图(view)-控制器(controller)的缩写，一种软件设计典范，用一种业务逻辑、数据、界面显示分离的方法组织代码，控制器接收用户的请求，并决定应该调用哪个模型(model)来进行处理。然后模型根据用户请求进行相应的业务逻辑处理，并返回数据。最后控制器调用相应的视图(view)格式化模型返回的数据，并通过视图(view)呈现给用户。

17. Answer:

螺旋模型是一种演化软件开发过程模型，它兼顾了快速原型的迭代的特征以及瀑布模型的系统化与严格监控 (3分，关键词是结合瀑布模型和快速原型)

螺旋模型最大的特点在于引入了其他模型不具备的风险分析，使软件在无法排除重大风险时有机会停止，以减小损失。(2分，关键词是风险分析)

■ 设计与重构

18. Answer:

结构化分析：简称 SA，面向数据流进行数据分析的方法。采用自顶向下逐层分解的分析策略。顶层抽象地描述整个系统，底层具体地画出系统工程的每个细节。中间层则是从抽象到具体的过渡。使用数据流图，数据字典，作为描述工具，使用结构化语言，判定表，判定树描述加工逻辑。

19. Answer:

结构化程序设计的思路就是、自顶向下、逐步求精；其程序结构就是按照功能划分为若干份基本模块；各模块之间的关系尽可能简单，在功能上相对独立，每一模块内部均是由顺序、选择与循环三种基本结构组成；其模块化实现的具体方法就是使用子程序，它的优点是 将一个较复杂的程序系统设计任务分解成许多易于控制与处理的子任务，便于开发与维护。缺点是当数据结构改变时，所有相关的处理过程都要进行相应的修改，带来额外的开销，程序的可用性差。

20. Answer:

工厂模式、抽象工厂模式、单例模式、建造者模式、原型模式、适配器模式、桥接模式、过滤器模式、组合模式、装饰器模式、外观模式、享元模式、代理模式、责任链模式(只要答出其中五种即可)。

21. Answer:

- a) 测试驱动开发 (TDD) 循环又被成为红色-绿色-重构
- b) TDD 的红色-绿色-重构周期从写一个测试开始, 因为它测试的主体代码还不存在, 所以会失败(红色)。然后, 添加最少的必要代码来使它通过这个测试例子(绿色)
- c) 一个红色-绿色-重构的例子
 - i. 在你写任何新的代码前, 写一个这段代码行为应当由的在某个方面的测试。由于被测试的代码还不存在, 写下测试迫使你思考想要代码如何表现, 以及与可能存在的协作者如何交互。我们把这种方法叫做“检测你想要的代码”
 - ii. 红色步骤: 运行测试, 并确认它失败了, 因为你还没有实现能让它通过测试的代码。
 - iii. 绿色步骤: 写出能使这个测试通过但不损坏其它已有的测试的尽可能简单的代码。
 - iv. 重构步骤: 寻找重构代码或测试的机会——改变代码的结构来消除冗余、重复或其他可能由添加新代码而引起的丑陋现象。测试能保证重构不会产生错误。
 - v. 重复以上步骤指导所有必须通过情景步骤的行为都完成。

22. Answer:

任何一部分的知识在系统内必须有一个单一的、明确的、权威的描述, 这条指导原则被称为无重复原则。该原则产生的原因是解决在开发过程中, 复制粘贴相似代码后, 导致的在修复或添加一个功能时, 开发者未修改所有副本这一问题 (课本 P15)

23. Answer:

- (1) 软件开发中的重构是什么: (1.5 分)

在不影响软件外部表现的前提下, 对其内部实现进行的调整和修改。

- (2) 为什么要重构: (2 分)

- 1) 重构的代码更加清晰, 更易于理解
- 2) 提高应用程序的性能
- 3) 方便对旧版的库或者框架进行更新
- 4) 重构可以改进系统设计

- (3) 有哪些基本原则: (2.5 分)

- 1) 以一致性为目标
- 2) 避免深度嵌套
- 3) 采取单独关注 SRP 原则

4) 避免重复

5) 高内聚、低耦合

24. Answer:

代码味道，即不易度量描述的源代码的结构特征。和真实气味一样，代码味道能将我们的注意力吸引到可能有问题的地方。

SOFA 代表四种代码味道，代表了一个编写良好的方法应当有的特征：

- (1) 保持简短 (short)：以便能迅速获取主要目的。
- (2) 只做一件 (one) 事情：以便测试能集中于彻底检查这一件事。
- (3) 输入少量 (few) 的参数：以便非常重要的参数值组合都能被测试到。
- (4) 维持一个固定级别的抽象 (Abstraction)：以便它不会在如何做和怎么做之间来回跳转。

25. Answer:

原则	意义	违反该原则的代码的不好特征	重构和修复的方法
单一责任原则	一个类有且仅有一个修改的理由	庞大的类，低的 LCOM 得分，数据泥团	提取类，移动方法
开闭原则	类应该对扩展保持开发，而对修改保持关闭	条件复杂度，基于 case 的分发器	使用策略或者模板方法，可能结合其他抽象工厂模式；使用装饰器来避免子类数量爆炸
里氏替换原则	替换一个类的子类应该保持正确的程序行为	拒绝继承：子类破坏性地覆盖一个继承的方法	用委托来替代继承
依赖注入原则	那些它的实现在运行时可能会改变的类应该依赖一个中间状态的“注入的”依赖	单元测试需要特别设计的桩来创建嫁接；构造器硬编码到另一个构造器的调用，而不是允许运行时决定使用哪个其他类	注入一个依赖到一个共享的接口，以分离类；根据需要使用适配器模式，外观模式或者代理器模式来使跨变量的接口统一
迪米特法则	仅和你的朋友交谈，把你的朋友的朋友当做陌生人	不合适的亲密，特性依恋，连环模拟	委托行为和调用委托方法

■ 代码和界面

26. Answer:

文档对象模型 (DOM) 是一个包含了层次结构的元素的 HTML、XML 或者 XHTML 文档的一个与语言无关的标准表示 (例子见数 P179 图 6.7)。支持 JavaScript 的浏览器提供 JavaScript 的语言绑定, 用以访问和遍历 DOM。这组功能和 JavaScript 访问浏览器的其他功能, 统称 JSAPI (应用程序编程接口)

jQuery 库为浏览器中不同的 JSAPI 提供统一的适配器并添加了基于 CSS 的 DOM 遍历, 动画化及许多其他特殊效果的增强性功能

27. Answer:

- 1) 通过字面量创建
- 2) 赋值给变量或属性
- 3) 作为参数传递, 作为函数结果返回
- 4) 拥有属性和方法
- 5) 浏览器将函数作为各类事件处理程序进行调用
- 6) 函数的形参和实参列表长度可以不同 (少: undefined, 多不绑定)
- 7) 每个函数传两个隐式参数 (arguments: 参数集合, this: 对象引用)
- 8) 调用方式: 普通函数 (全局对象 window)、作为方法 (拥有该方法对象)、作为构造器调用 (新分配对象)、apply() 或 call() 调用 (上下文可以设置成任意值)

28. Answer:

HTML 是用来描述网页的一种语言。

HTML 指的是超文本标记语言: HyperText Markup Language

HTML 不是一种编程语言, 而是一种标记语言, 标记语言是一套标记标签 (markup tag)

HTML 使用标记标签来描述网页

HTML 文档包含了 HTML 标签及文本内容

HTML 文档也叫做 web 页面

29. Answer:

Ruby 是纯面向对象的语言, Ruby 中的一切都是以对象的形式出现。Ruby 中的每个值都是一个对象, 即使是最原始的东西: 字符串、数字, 甚至连 true 和 false 都是对象。类本身也是一个对象, 是 Class 类的一个实例。

类用于指定对象的形式, 它结合了数据表示法和方法, 把数据整理成一个整齐的包。类

中的数据和方法被称为类的成员。

30. Answer:

Public: Public 方法可被任意对象调用。默认情况下，方法都是 public 的，除了 initialize 方法总是 private 的。

Private: Private 方法不能从类外部访问或查看。只有类方法可以访问私有成员。

Protected: Protected 方法只能被类及其子类的对象调用。访问也只能在类及其子类内部进行。

31. Answer:

(1) Ruby 是一种纯粹的面向对象编程语言，是脚本语言。

(2) 特性:

完全面向对象

在 Ruby 语言中，任何东西都是对象，包括其他语言中的基本数据类型，比如整数变量没有类型

Ruby 的变量可以保存任何类型的数据。

任何东西都有值

不管是数学或者逻辑表达式还是一个语句，都会有值。

ruby 语言很优雅，可以做到不需要注释就可以读懂。

(3) 优点: (答出 3 点即可)

语法简单

普通的面向对象功能 (类, 方法调用等)

特殊的面向对象功能 (Mixin, 特殊方法等)

操作符重载

错误处理功能

迭代器和闭包

垃圾回收

动态载入 (取决于系统架构)

可移植性高. 不仅可以运行在多数 UNIX 上, 还可以运行在 DOS, Windows, Mac, BeOS 等平台上
适合于快速开发, 一般开发效率是 JAVA 的 5 倍

32. Answer:

CSS 规则由两个主要的部分构成: 选择器, 以及一条或多条声明:



选择器通常是您需要改变样式的 HTML 元素。

每条声明由一个属性和一个值组成。

属性 (property) 是您希望设置的样式属性 (style attribute)。每个属性有一个值。属性和值被冒号分开。

33. Answer:

数组字面量通过[]中以逗号分隔定义，且支持 range 定义。

(1) 数组通过[]索引访问

(2) 通过赋值操作插入、删除、替换元素

(3) 通过+、-号进行合并和删除元素，且集合做为新集合出现

(4) 通过<<号向原数据追加元素

(5) 通过*号重复数组元素

(6) 通过|和&符号做并集和交集操作 (注意顺序)

34. Answer:

A = Hash.new { |h,k| h[k] = k } *xx 是默认值*

哈希 (Hash) 是类似 "key" => "value" 这样的键值对集合。哈希类似于一个数组，只不过它的索引不局限于使用数字。

Hash 的索引 (或者叫“键”) 几乎可以是任何对象。

Hash 虽然和数组类似，但却有一个很重要的区别：Hash 的元素没有特定的顺序。如果顺序很重要的话就要使用数组了。

35. Answer:

简单来说：迭代 (iterate)指的是重复做相同的事，所以迭代器 (iterator) 就是用来重复多次相同的事。

迭代器是集合支持的方法。存储一组数据成员的对象称为集合。在 Ruby 中，数组 (Array) 和哈希 (Hash) 可以称之为集合。

*arg.each do |i|
 puts i
end*

*b = arg.collect { |x| x*2 }
puts b*

迭代器返回集合的所有元素，一个接着一个。在这里我们将讨论两种迭代器，each 和 collect。

each 迭代器返回数组或哈希的所有元素。

collect 迭代器返回集合的所有元素

36. Answer:

迭代器是一个简单的方法，这个方法遍历某个数据结构，并且用 yield 每次传一个元素给迭代器的接受者。

■ 测试

```
def test
  yield
end
```

\$> test {puts "0"}

输出: 0.

37. Answer:

验收测试、集成测试、功能测试、单元测试、黑盒/白盒测试、测试覆盖、回归测试、持续集成 (CI) 测试

38. Answer:

A/B 测试是一种数据进行产品决策的方法，它用于比较网页或应用程序的两个或多个版本，以确定哪一个版本更好，A/B 测试本质上是一个实验机制，其向用户随机显示页面的某一个版本，并通过统计分析确定哪个版本对于设定的转化目标有更好的表现。

测试目的:

(1)消除 UI 设计的冲突纷争，确定最好的方案;(2)对比试验，找出软件中出现的问题以及原因;(3)降低新产品发布风险，为创新提供保障。

测试步骤:

(1)现状分析;(2)假设建立;(3)设定目标;(4)界面设计;(5)技术实现

39. Answer:

TDD(Test-driven development)测试驱动开发，是敏捷开发中的一项核心实践和技术，也是一种设计方法论。TDD 的原理是在开发功能代码之前，先编写单元测试用例代码，由测试代码来确定需要编写什么产品代码。

40. Answer:

创建一个好的测试有以下五个原则，首字母缩写为 FIRST:

(1)Fast(快速): 它应该能简单和快速地运行与当前的编码任务相关的测试用例，来避免妨碍你连贯的思路。

(2)Independent(独立): 任何测试都不应该依赖于由别的测试引出的先决条件，这样我们才能优先运行覆盖最近改动代码的测试子集。

(3) Repeatable(可重复): 测试行为应当不依赖于外部因素, 例如改变日期或改变了它们的值会破坏测试的特殊常数。

(4) Self-checking(自校验): 每一个测试都应该是能够自己决定测试是通过还是失败, 不依赖于人工来检查它的输出。

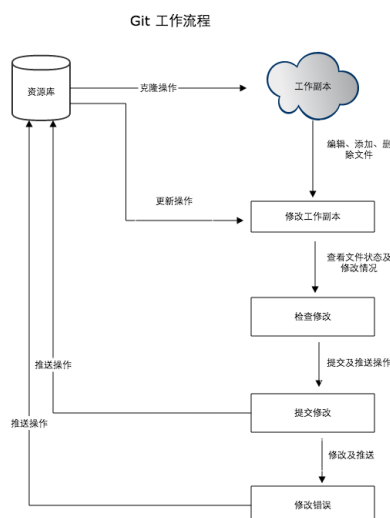
(5) Timely(及时): 测试应该在代码被测试的时候创建或更新。在测试驱动开发(TDD)中, 测试是在编码的前一刻才完成的。

41. Answer:

一般工作流程如下:

- 克隆 Git 资源作为工作目录。
- 在克隆的资源上添加或修改文件。
- 如果其他人修改了, 你可以更新资源。
- 在提交前查看修改。
- 提交修改。
- 在修改完成后, 如果发现错误, 可以撤回提交并再次修改并提交。

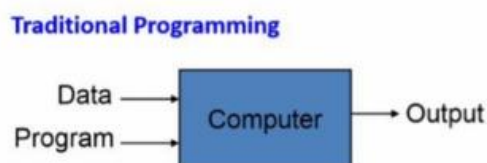
下图展示了 Git 的工作流程:



■ 两种基本途径

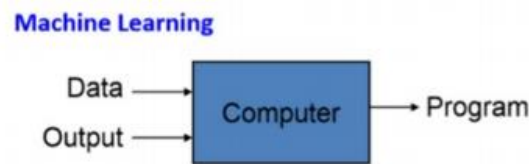
42. Answer:

(1) 基于人类学习的软件设计



第一种途径是:人类发现问题, 提出解决方案(构思模型), 然后设计代码(程序)来, 让计算机执行来解决问题。

(2) 基于机器学习的软件设计



第二种途径是, 人类发现问题, 但没法构思出有效的解决方案(模型), 人类设计一些学习程序,

让计算机执行学习程序, 根据给出的样本数据, 计算机训练出解决方案(模型), 然后集成到计算机中(人编制的代码和机器产生的代码结合在一起)解决问题。

二、 选择题

■ 软件需求和应用场景

1-5. BBCCC

6-10. ABBAC

11-15. DDDBB

16-20. BBBCC

21-25. CBBBD

26-30. CCBA A

31-35. CDADD

36-40. CBCCD

41-45. BCACB

46-50. DDAAD

51-55. CBDCC

56. A

■ 业务建模和数据模型

57-60. DBBD

61-65. CDDCB

66-70. CADCC

71-75. BDCCB

76-80. DCADD

81-83. CBA

■ 软件架构和应用框架

84-85. AC

86-90. DBBBB

91. B

■ 设计模式和代码重构

92-95. DACA

96-100. CADDB

101-105. ABADC

106-110. DADBE

111-115. AADDB

116. D

117. ABCD

118-120. CCD

121-125. DBDEC

126-130. DCAAD

131-135. CACDC

136-140. BDABA

141-145. AACBA

146-150. D ACAA

151-155. DBABC

156-160. CD

158. ABCDEF

159. ABCDE

160. A

161-165. CBBDA

166-170. AAABC

171-175. BCBBC

176-180. BBCDB

181-185. DACDC

186-190. BBCBC

191-195. BDAA

196-200. DCCBB

201-205. DADBC

206-210. BXXDA

211-215. AAAAB

216-220. CABDA

221-225. CBABD

226-230. DAACC

231-235. DDBBB

236-240. CAABB

241-245. CBCDC
246-250. DDBDA
251-255. BDDDA
256-257. BC

■ 接口设计和前端编程

258-260. DBA
261-265. ACCAA
266-270. BBACB
271. D

■ 软件验证与自动测试

272-275. CDDA
276-280. AABBD
281-285. ABABA
286-290. BBADA
291-295. CDCBB
296-300. ABCAC
301-302. CB

■ 性能评价和设计优化

303-304. CD

■ 集成部署和系统运营

305. D
306-310. AABCD
311-315. BDDAD
316-320. ACCAD

■ 三大问题

321-325. BCDBB
326-330. D

三、 判断题

■ 三类问题

1-5 对 对 对 错 错
6-10. 对 对 对 对 对
11-15 对 错 错 错 对
16-20. 错 错 对 错 对
21-25. 对 对 错 错 错
26-30. 对 对 对 对 错
31-35 错 错 对 错 对
36-40. 对 对 错 错 对
41-45. 错 错 对 错 错
46-47. 对 对

■ 需求

48-52 对 对 对 错 对
53-57 对 对 对 对 错
58-62 错 错 错 错 错
63-67 错 错 对 对 对
68-69 错

■ 建模和模型

70-74 对 错 错 错 对
75-76 对 错

■ 设计与重构

77-81 错 错 对 对 错
82-86 对 错 对 错 错
87-91 对 对 对 对 对
92-96 错 错 错 对 错
97-101 对 对 错 对 错
102 错

■ 代码和界面

103-107 错 对 对 错 对
108-112 错 对 错 对 对
113-117 对 对 对 对 错
118-122 对 错 对 对 对
123-127 错 对 对 错 对

128-132 对对对对对
133-137 对对对对对
138-142 对错对对对
143-147 对对对对对
148-152 对对对对对
153 对

■ 测试

154-158 错对对对对
159-163 错对对对对
164-168 对对对对对
169-173 对对对对对
174-178 对对对对对
179-183 错对对对对
184-188 对对对对对
189-193 对对对对对
194-197 对对对对对

■ 两种基本途径

198-202 对对对对对
203-206 对对对对对

四、 设计题

■ 需求分析

1. Answer(合理即可)

2. Answer: 设计需求: 用户身份唯一识别、14 天内行程记录、14 天内健康记录、判断是否需要隔离、给出出行的红码绿码等(言之有理即可, 3 分)

设计方法: 建立数据库, 利用身份证唯一认证来识别身份; 通过 GPS 定位获得地理位置, 记录十四天内的位置信息, 获取车票机票等跨地域信息; 通过自我登记, 或手表等外设传感获得体温情况; 设计一个状态判定, 统计全国各个地区的风险程度, 根据十四天内的行程是否经过中高风险区给出状态, 根据近期健康情况给出是否需要隔离。(6 分, 言之有理即可)
改进之处: 可以增设每日提醒闹钟, 可以根据地理位置自动填写是否在校、14 天内是否离京等信息。(1 分, 言之有理即可)

3. Answer:

用户故事: 我是 xx 中学的老师, 目前主要教授高三一班的语文课程, 由于处于关键时期,

学校每一星期都会对学生进行测试，我想使用软件将每次测试的学生成绩进行排序，以便我能了解班里同学的成绩变动，从而改进下一步教学计划。场景 1：用户点击学生成绩分析，系统将统计语文最高分、最低分、平均分，以及全班成绩分布图，包含上一次考试的成绩，并在网页上显示出来。

场景 2：用户点击全班成绩分布图，系统将本次考试成绩和上一次考试成绩采用不同的颜色标注在分布图上，同时注明进步及退步学生，并在网页上显示出来。

■ 设计模式

4. Answer:

上面的代码显然是没有循序 Single Responsibility Principle 单一职责原则的，原因就是 `Student` 类拥有计算每个学期平均分的逻辑，但是 `Student` 类是用来封装关于学生信息的而不是用来计算分数的，计算分数的逻辑应当放在 `Grade` 类中才对。

```
class Student
  def initialize
    @terms = [ Grade.new(:first), Grade.new(:second) ]
  end

  def first_term_grade
    term(:first).grade
  end

  def
    second_term_grade term(:second).grade
  end private

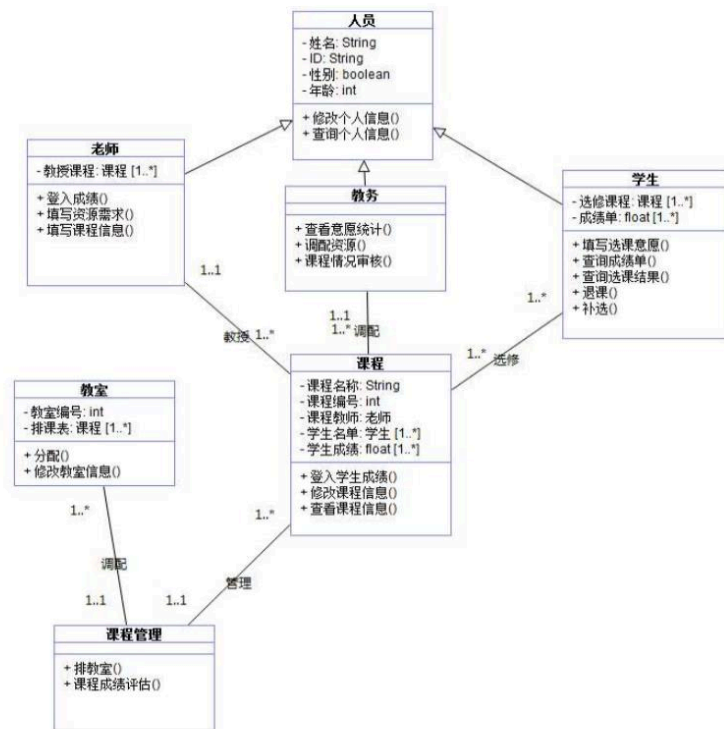
  def
    term reference @terms.find {|term| term.name == reference}
  end
end

class Grade
  attr_reader :name, :home_work, :test, :paper
  def initialize(name)
    @name = name @home_work = 0 @test = 0 @paper = 0
  end

  def grade
    (home_work + test + paper) / 3
  end
end
```

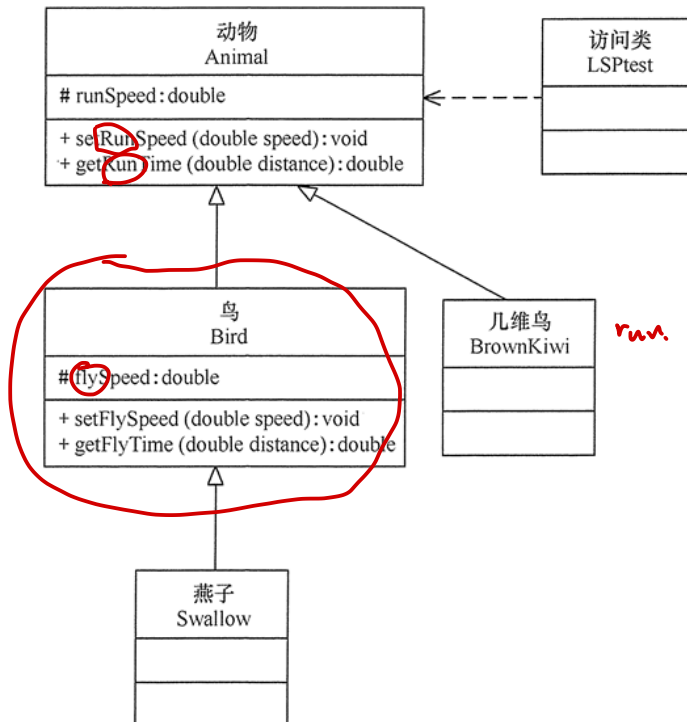
5. Answer:

答案如图：（OOD 规则版）



（类的规划设计 3 分，类的内部数据和方法设计 3 分，类之间的关系 2 分，表述规范和整洁性 2 分）

6. Answer:



7. Answer:

用户场景:

两类人，一类是普通的想喝咖啡的人，一类是员工。

前者会使用咖啡机进行咖啡冲泡，整个过程分四步完成，分别是选择咖啡类型、选择份数、选择加多少糖、点击开始冲泡咖啡。后者需要对咖啡机进行维护，分别有清洗咖啡机、填充咖啡豆、加水、加糖。

不符合接口隔离原则。修改如下:

```
class CoffeeMachineInterface
  def select_drink_type
    puts "select drink type"
  end

  def select_portion
    puts "select portion"
  end

  def select_suger_amount
    puts "select suger"
  end

  def brew_coffee
    puts "brew coffee"
  end
end

class CoffeeMachineServiceInterface
  def clean_coffee_machine
    puts "clean coffee machine"
  end

  def fill_coffee_beans
    puts "fill coffee beans"
  end

  def fill_water_supply
    puts "fill water"
  end

  def fill_sugar_supply
    puts "fill sugar"
  end
end

class Person
  def initialize
    @coffee_machine = CoffeeMachineInterface.new
```

```

end

def make_coffee
  @coffee_machine.select_drink_type
  @coffee_machine.select_portion
  @coffee_machine.select_suger_amount
  @coffee_machine.brew_coffee
end

end

class staff
  def initialize
    @coffee_machine = CoffeeMachineServiceInterface.new
  end

  def serv
    @coffee_machine.clean_coffee_machine
    @coffee_machine.fill_coffee_beans
    @coffee_machine.fill_water_supply
    @coffee_machine.fill_sugar_supply
  end
end
End

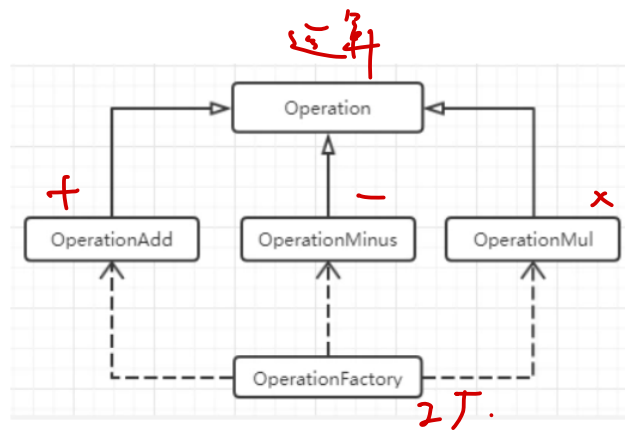
```

8. Answer:

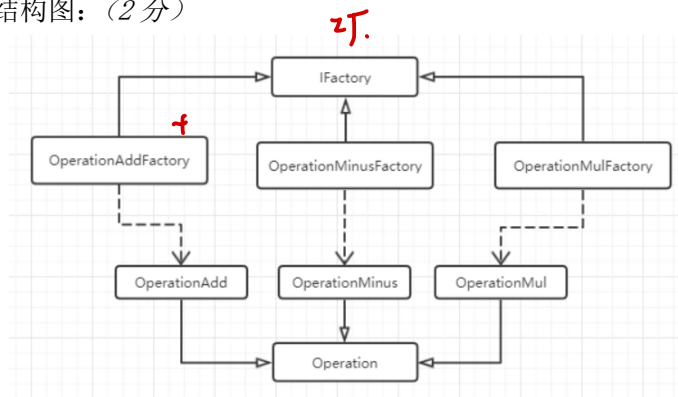
需求分析：该系统需要使用校园卡作为用户身份的唯一认证。创建一个 user 类，类中包含该 user 的 ID 号、用户状态（正常、异常，异常比如休学、毕业、生病等）、出校许可，公开方法预约班车 appointment()。Student 类和 Teacher 类继承 user 类，student 类里面加入私有变量比如 Teacher 变量数组 supervisor[]，私有方法比如申请出校 apply()。Teacher 类加入私有变量管理的学生 supervised[]，私有方法比如同意出校 approve() 和拒绝申请 reject()。在出校门的时候对每个 user 的出校许可进行检查，如果为 true 则能够进出校园并且使用预约班车功能（里氏替换原则）。如果学校或老师有通知，可以通过 update() 方法通知 supervised[] 列表下的每一个学生（observer 模式）。

9. Answer:

(1) 简单工厂模式的结构图：(2 分)



工厂方法模式的结构图：(2分)



(2) Answer:

简单工厂模式和工厂方法模式显然都满足单一职责原则，每个类的职责是单一的，只实现单个操作的功能 (1分)

简单工厂模式不符合开放封闭原则，在 OperationFactory 类中包含了所有 Operation 派生类的创建，如果需要增加新的派生类，就要不断修改 OperationFactory，违反 Open-Close Principle，而工厂方法模式符合这一原则，只要新建新的工厂类就可以 (2分)

里氏替换原则主要在实现中体现，要让 Operation 的派生类遵循 LSP 原则，才能让工厂中创建的派生类都能够用多态替换之后对 Operation 的引用 (1分)

在工厂类中实现的接口必须符合接口分离原则，即只包含必要的部分 (1分)

工厂模式返回抽象类，使得调用工厂的高层模块依赖 Operation 这个抽象类而不是某个具体的派生类，主要满足依赖倒置原则 (1分)

10. Answer: (a) 该接口类违背了单一职责的原则。Reader 类应该是用来封装读者的信息，它应该包含读者的全部职责，而对于属于其他类的职责不应包含。

(b) 该图中的前两个方法借书和还书属于读者的职责，而后两个类是否超期和超期罚款的计算应该属于 book 类的职责。Book 类中维护书的信息，里面包含书的借阅时间和返还时间，可以方便计算该书是否超期以及是否应该缴纳罚款。

11 Answer:

1)


```
@users.sort!{|a,b| a.name <=> b.name}
@users = @users.order(:name)
```

2)

```
class User < ActiveRecord::Base
  has_and_belongs_to_many :courses
end
```

```
class Course < ActiveRecord::Base
  has_and_belongs_to_many :users
end
...
```

或者新增一个中间表grades:

```
...
class User < ActiveRecord::Base
  has_many :grades
  has_many :courses, through: :grades
end
```

```
class Course < ActiveRecord::Base
  has_many :grades
  has_many :users, through: :grades
end
```

3)

```
user = User.create(name: "David", age: 20, email:'David@test.com',
  password:123456)
...
或者:
...
user = User.new(name: "David", age: 20, email:'David@test.com',
  password:123456)
user.save
```

4)

```
user = User.find_by(name: 'David')
...
或者
...
user = User.where(name: 'David')
user.password= 666666
...
或者
...
user.update_attribute(:password,666666)
user.save
```

5)

```
user = User.last
if !user.nil?
  user.destroy
end
```

■ 程序与界面设计

12. Answer:

```

<table width=300 border="1" align="center">
  <tr>
    <td rowspan="2" align="center">
      <b>A</b>
      <td colspan="2" height="100" align="center">
        <b>B</b>
    <tr align="center">
      <td height="100">
        <b>E</b>
      <td rowspan="2">
        <b>C</b>
    <tr align="center">
      <td colspan="2" height="100">
        <b>D</b>
  </table>

```

13. Answer:

```

<!DOCTYPE html>
<html>
<body>
  <select onchange="_sel(this)">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="opel">Opel</option>
    <option value="audi">Audi</option>
  </select>
</body>
</html>

<script>
  function _sel(obj){
    alert("显示文本: " + obj.options[obj.selectedIndex].text +
      "\n 值: " + obj.options[obj.selectedIndex].value);
  }
</script>

```

14. Answer:

示例程序:

C:

```

void fibo(int n)
{
    int x1,x2,x; x1=1; x2=1;
    printf("%d %d ",x1,x2); //前两个数为 1 并输出
    for(int i=1;i<=(n-2);i++) //循环计算，一边算一边输出
    {
        x=x1+x2;
        printf("%d ",x);
        x1=x2;
        x2=x;
    }
    printf("\n");
}

void main()
{
    fibo(100);
}

```

Ruby:

```

def calculateFibonacci(num)
    if num==0 || num==1          // 前两项为 1
        1
    else
        calculateFibonacci(num-2)+calculateFibonacci(num-1) //递归计算
    end
end

result = []
100.times do |i|                //计算前 100 个数
    result << calculateFibonacci(i)
end

p result

```

15. **Answer:** 写出将格式类传递到 printer 类这一设计思路即可得大部分分数，有语法错误的酌情扣 1-2 分。

16. **Answer:** `validates :password, presence: true, length: { minimum: 6 }`

17.

(1) (该部分代码已经过自身测试没有问题)

Customer name: John

Customer is vip: Yes

goods_total_price: 10

Total number of customers: 1

Customer name: Poul

Customer is vip: No

goods_total_price: 26

Total number of customers: 2

(2)

该函数实现的功能为：给出一个目标数 `target` 以及一个数组，寻找数组中是否存在两个数相加的值等于 `target`，若存在，则将这两个数在数组中的位置打印出来（存在多组则依次打印）

例如：

给定 `nums = [2, 7, 11, 15]`, `target = 9`

因为 `nums[0] + nums[1] = 2 + 7 = 9`

所以输出对应数的位置 [1, 2]

调用 `cust1.func(5)` 函数会对应输出 [2, 3] 和 [1, 4] (两个输出的先后顺序可以忽略, 测试时输出的顺序如上)

该函数的缺陷为: 要求 `nums` 数组中的值都是整数, 否则无法正确计算。

(3)

```
class Vip_Customer < Customer
def initialize(is_vip, name, price_list)
  @cust_is_vip=is_vip
  @cust_name=name
  @goods_price_list = price_list.map{|item| item*0.9}
end
end
```

18. Answer(合理即可)

19. Answer

输出结果:

Subject: Attached an observer.

Subject: Attached an observer.

Subject: I'm doing something important.

Subject: My state has just changed to: 2

Subject: Notifying observers...

ConcreteObserverA: Reacted to the event

ConcreteObserverB: Reacted to the event

Subject: I'm doing something important.

Subject: My state has just changed to: 10

Subject: Notifying observers...

ConcreteObserverB: Reacted to the event

Subject: I'm doing something important.

Subject: My state has just changed to: 2

Subject: Notifying observers...

ConcreteObserverB: Reacted to the event

20. Answer:

```
<form action="/index/" method="post">
```

```
<P> <label>欢迎登录! </label> </P>
```

```
<P>
```

```
<label >用户名:</label>
```

```
<input type="text" name =' name' />
```

```
</P>
```

```
<P>
```

```
<label>密码: </label>
```

```
<input type="text" name=' password' />
```

```
</P>
<p><input type="submit" value="确认"/></p>
</form>
```

本题为作业原题，主要用于复习简单的 html 基础。

21. **Answer:**

```
class Customer
  @@no_of_customers=0
  def initialize(id, name, addr)
    @cust_id=id
    @cust_name=name
    @cust_addr=addr
  end
  # 访问器方法
  def printId
    @cust_id
  end
end

end
def display_details()
  puts "Customer id #@cust_id"
  puts "Customer name #@cust_name"
  puts "Customer address #@cust_addr"
end
def total_no_of_customers()
  @@no_of_customers += 1
  puts "Total number of customers: #@no_of_customers"
end
end
```

22. **Answer:** 完整写出可运行成绩得 10 分，其中 max min 功能点占 6 分，sum 功能点占 4 分，能够产生正确输出为功能点实现。Rand 功能点占 2 分，循环的功能点占 4 分，代码正确为功能点实现。

23. **Answer:**



```
require 'observer' # 1 分

class TV
  include Observable # 1 分
  def turn_on #1 分
    puts "TV is on!"
    changed #TV state changed #1 分
    notify_observers(self) #1 分
  end
end

class Boy
  def update(tv) #1 分
    puts "I will go to watching TV!"
  end
end

tv = TV.new #1 分
boy = Boy.new #1 分
tv.add_observer(boy) #1 分
tv.turn_on #1 分
```

24. Answer: (合理即可)

25. Answer: (合理即可)

26. Answer:

1)

```
a = 7 unless a == 4
```

含义同: If $a \neq 4$ then $a=7$

2)

```
user = User.create(name: "David", occupation: "Code Artist")
```

3)

```
user = User.new
user.name = "David"
user.occupation = "Code Artist"
user.save
```

4)

Answer:

```
<ul>
  <% @articles.each do |article| %>
    <li>
      <%= link_to article.title, article %>
    </li>
  <% end %>
</ul>
```

27. Answer:

(1) .localhost 指主机名, 8000 指端口, name,password 均指从网页传过来的参数。

(2) . myweb/urls.py 里 urlpatterns = [path('myapp' ,include('myapp.urls'))]

Myapp/urls.py 里 urlpatterns = [path('index' , views.index)]

(3) . myapp/views.py

28. Answer:

```
<!DOCTYPE html>
<html lang="en">
<body>
<button type="button" onclick="timelnit()">showtimes</button>
<h3 id="time"></h3>
</body>
<script type="text/javascript" >
function timelnit() {
    var time = new Date().toString();
    var h =document.getElementById("time");
    h.innerHTML = time;
    id = setTimeout(timelnit,1000);
}
</script>
```

29. Answer:

(1).

```
class Box
  def initialize(w,h)
    @width, @height = w, h
  end
end
```

(2).

```
class Box
  attr_accessor :width
  attr_accessor :height
  def initialize(w,h)
    @width, @height = w, h
  end
end
```

(3).

```

class Box
  attr_accessor :width
  attr_accessor :height
  @@count = 0
  def initialize(w,h)
    @width, @height = w, h
    @@count += 1
  end
end

```

(4).

```

class Box
  attr_accessor :width
  attr_accessor :height
  @@count = 0
  def initialize(w,h)
    @width, @height = w, h
    @@count += 1
  end

  def self.printCount()
    puts "Box count is : #@@count"
  end
end

```

(5).

#两个 printCount 输出一样，都是
Box count is: 2

■ 测试

30. Answer:


```

describe User do
  describe "#send_password_reset" do
    let(:user) { Factory(:user) }

    it "generates a unique password_reset_token each time" do
      user.send_password_reset
      last_token = user.password_reset_token
      user.send_password_reset
      user.password_reset_token.should not eq(last_token)
    end

    it "saves the time the password reset was sent" do
      user.send_password_reset
      user.reload.password_reset_sent_at.should be_present
    end

    it "delivers email to user" do
      user.send_password_reset
      last_email.to.should include(user.email)
    end
  end
end
end

```

31. Answer:

(1) 测试自动化程度: 可以评估测试代码和工具是否可以自动执行 (2 分) 每次系统在更新后是否会自动对更新部分运行测试 (2 分)

(2) 测试效果: 测试覆盖率达到多少个百分点 (2 分) 运用了自动化测试工具后, 软件产品上线后的 Bug 数量和严重程度与之前没用相比有了怎样的变化 (2 分) 使用该自动化测试工具后, 软件迭代更新周期时间缩短了多少 (2 分)

(3) 运行性能: 运行完整的自动化测试需要多长时间 (2 分) 每次局部更新后, 是否可以只自动运行受影响部分的测试 (2 分)

(4) 可演化和可扩展性: 软件系统规模和复杂度增大后, 自动化测试代码和工具是否依然可以有效地发现缺陷 (2 分) 当软件系统内部结构发生了迭代调整, 自动化测试代码是否可以复用未被更改的部分, 是否需要全部调整更新 (2 分)

五、 简答题

■ 软件理论

1. Answer:

1). 开发模式与交互模式不同: 传统软件依靠存储介质拷贝传播, 需要安装使用时占用一定磁盘空间。而 SaaS 则是部署在服务器上, 更新迭代较为方便, SaaS 可以在手机、电脑等多平台接入系统进行操作。

2.盈利模式不同：购买传统软件一次性投入数额高昂的版权及维护费用等；而 SaaS 则是租赁制，按照服务的使用时长计费。

3.SaaS 部署时间相对于传统软件更方便。

4.SaaS 比传统软件适用的时间与空间更广。

5.数据安全性：传统的软件是安装在用户自己的服务器下的，相关的数据可控，而 SaaS 的数据是存放在厂商服务器上的，数据的隐私性以及安全性都将受到较大的考验。

2)

(1)对数据安全性上要求较高的。传统软件安装在用户自己的服务器下，相关数据可控，而 SaaS，数据是存放在厂商服务器上的，数据的隐私性及安全性受到考验。

(2)对性能以及实时性响应要求较高的。例如大数据流量的应用程序，SaaS 应用远程的特性使得网络瓶颈对此类应用程序的影响特别严重。性能也会根据用户的行为、供应商网络以及根据快速增长或者临时峰值的动态调节能力而有很大的不同。

2. Answer:

理论上，寻找更好的方法来推断或预测拟开发的软件系统有价值并满足质量要求。

实践中，可根据拟解决的领域问题归纳出有价值的使用场景或找到优化目标函数和有效的机器学习训练数据集，判断需求是否合理、准确或是否有智力成份，尽量找到最小的测试用例使之能覆盖和捕捉到软件运行时所有可能条件分支或异常情况。

3. Answer: 建立软件生产的激励机制可以根据软件设计人员的贡献给予相应奖励，同时在设计团队之间引入竞争机制；建立软件的产业生态环境，需要与政府合作，通过促进政策、教育和维权等方式，建立更加健康的软件产业生态环境。积极应对各种新挑战、资产管理促进技术创新、打击盗版多管齐下，营造一个良好的知识产权环境，为软件产业健康发展提供更好的保障。

4. Answer: 复杂网络中的边来表示。这样软件系统可以构建成一个复杂网络，然后就可以用复杂网络中的方法来对软件系统进行分析，如基于复杂网络的软件复杂性度量研究，基于加权软件网络缺陷传播分析的面向对象软件结构质量度量，通过研究网络节点度可以发现软件的依赖程度。

5. Answer:

情况一不属于“构思创造发明有价值的软件系统或产品”。能完成特定的目的不代表有价值，在情况一中，该学生的行为其实是一种作弊行为，有违公平性。(3分)

情况二的人群覆盖达到了上千万的量级，但是肯定也有很多人没有使用该软件，总体上这也是一种损害公平性的行为，不属于“构思创造发明有价值的软件系统或产品”。但是因为覆盖比例和软件开放的原因，该软件尽力将不公平性降低。(4分)

其他思考：针对抢票软件，铁路部门也间接给出了自己的意见：在官方 12306 软件中改变机

制，将以前的抢票变成了候补机制，一定程度上降低了抢票软件的功能。间接说明了类似抢票软件虽然有一定的覆盖面，但是总体上不属于“构思创造发明有价值的软件系统或产品”。

(2分)在设计实现软件系统时，尤其是这种涉及到国计民生的系统时，技术绝对不是第一要素，这种系统的面世相当于一项“政策”的颁布，一定要优先考虑社会因素。(1分)

6. Answer:

(1) 3个过程是：软件定义过程、软件开发过程、软件使用与维护过程；

(2) 9个阶段有：可行性研究、需求分析、概要设计、详细设计、实现、组装测试、验收测试、使用与维护、退役。

7. Answer:

1) 软件需求和应用场景，首先要理解待开发软件所在领域的业务问题，业务需求以及软件的应用场景。2) 业务建模和数据模型，要从众多的业务应用场景中提炼归纳出一般性的业务模型和相应的数据模型。3) 软件架构和应用架，根据业务需求选择合适的软件架构。4) 设计模式和代码重构，为提高开发效率和应对软件的变更应在开发前选择合适的设计模式。5) 接口设计和前端编程，详细地定义软件系统如何实现、测试和升级。6) 软件验证与自动测试，测试并验证软件地正确性实现预期地价值。7) 安全问题和防御设计，应用场景中主要地安全威胁和风险，如何应对。8) 性能评价和设计优化，软件系统的性能评价指标和如何优化设计。9) 集成部署和系统运营，让开发的软件真正实现其价值。

■ 需求

8. Answer:

需求分析阶段分成四个方面：对问题的识别、分析与综合、制定规格说明和评审。

三个基本原则：必须能够表达和理解问题的数据域和功能域；必须按自顶向下、逐步分解的方式对问题进行分解和不断细化；要给出系统的逻辑视图和物理视图。

9. Answer:

(1) 好的架构不依赖于某一个技术的实现。

(2) 好的架构需求变化增加减少，代码不影响其他已经正确与运行的功能。

(3) 不专门为某一类的业务而实现的，应该具有‘普适性’。

■ 建模与模型

10. Answer:

(1) has_many...那行底下添加：

validates :name, :email, presence: true

(2) validates 行改为:

```
validates :content, presence: { message: "微博内容不能为空" }, length: { maximum: 140 }
```

11. Answer:

```
migration CreateUsers name:string email:string age:int
```

12. Answer:

```
add_column :users, :password, :string
```

13. Answer:

直接修改原来的迁移表是无法生效的,除非重建这个数据表,但是这样会将之前的所有记录清空;在软件版本迭代的过程中,我们不希望每次迭代都要将相关的数据清空,这是不现实的,因此需要一个新的数据迁移文件来定义新的修改

14. Answer:

```
validates :password, presence: true, length: { minimum: 6 }
```

或者

```
validates :password, allow_nil: false, length: { minimum: 6 }
```

15. Answer:

(1) 标题中使用了嵌入式Ruby代码的视图

```
<% provide(:title, "The Title") %>
<!DOCTYPE html>
<html>
  <head>
    <title><%= yield(:title) %> | Ruby on Rails Tutorial Sample App</title>
  </head>
  <body>
    Contents
  </body>
</html>
```

<% provide(:title, "The Title") %>, 通过 <% ... %> 调用 Rails 提供的 provide 函数, 把字符串"The Title"赋给 :title。然后在标题中, 使用类似的符号<%= ... %>, 通过 Ruby 的 yield 函数把标题插入模板中。

(2) 使用full_title辅助方法的网站布局 (先定义好full_title辅助方法)

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= full_title(yield(:title)) %></title>
    <%= csrf_meta_tags %>
    <%= stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track': 'reload' %>
    <%= javascript_include_tag 'application', 'data-turbolinks-track': 'reload' %>
```

```

</head>
<body>
  <%= yield %>
</body>
</html>

<!DOCTYPE html>
<html>
<head>
<title><%= yield(:title) %> | Ruby on Rails Tutorial Sample App</title>
<%= csrf_meta_tags %>
<%= stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track': 'reload' %>
<%= javascript_include_tag 'application', 'data-turbolinks-track': 'reload' %>
</head>
<body>
  <%= yield %>
</body>
</html>

```

`<%= yield %>`，把每个页面插入布局中。在布局中使用这行代码后，访问 `/static_pages/home` 时会把 `home.html.erb` 中的内容转换成 HTML，然后插入 `<%= yield %>` 所在的位置。

■ 设计与重构

16. Answer:

C 是面向过程的。C++是面向对象的，但是完全包含了c的内容。java是纯面向对象的。
 ruby是面向对象的脚本语言（前三种是编程语言）。

优点：

- 1) 解释型执行，方便快捷：Ruby是解释型语言，其程序无需编译即可执行。
- 2) 语法简单、优雅：语法比较简单，类似Algol系语法。
- 3) 完全面向对象：Ruby从一开始就被设计成纯粹的面向对象语言，因此所有东西都是对象，例如整数等基本数据类型。
- 4) 内置正则式引擎，适合文本处理：Ruby支持功能强大的字符串操作和正则表达式检索功能，可以方便的对字符串进行处理。
- 5) 自动垃圾收集：具有垃圾回收（Garbage Collect, GC）功能，能自动回收不再使用的对象。不需要用户对内存进行管理。
- 6) 跨平台和高度可移植性：Ruby支持多种平台，在Windows, Unix, Linux, MacOS上都可以运行。Ruby程序的可移植性非常好，绝大多数程序可以不加修改的在各种平台上加以运行。
- 7) 有优雅、完善的异常处理机制：Ruby提供了一整套异常处理机制，可以方便优雅地处理代码处理出错的情况。
- 8) 拥有很多高级特性：Ruby拥有很多高级特性，例如操作符重载、Mix-ins、特殊方法

等等，是用这些特性可以方便地完成各种强大的功能。

缺点：

从技术角度来说，Ruby解析器的性能比较差，远低于Java的JVM。

相对比Java丰富的各种第三方类库来说，Ruby的第三方类库显得非常缺乏，而且不成熟，

特别是针对企业应用的各种第三方类库支持，空白点很多。

Ruby，特别是用 rails 开发的 web 应用在 Windows 操作系统上面得到的支持还很不够，传统上都是部署在 Unix/Linux 操作系统上的。

17. Answer:

可以分为面向对象与面向过程 (2 分)，面向过程语言比较节省内存，而面向对象语言尽管升级、维护、模块化结构、代码重用率都很好 (2 分)，JAVA 为面向对象语言，C 为面向过程语言，C++为面向对象语言，C#为面向对象语言，python 为面向对象语言，ruby 为面向对象语言（每个语言分类正确得 1 分，共 6 分）。

18. Answer:

- (1) 可以使软件结构清晰，不仅容易设计，也容易阅读和理解；
- (2) 容易测试和调试，有助于提高软件的可靠性；
- (3) 可以提高软件的可修改性；
- (4) 有助于开发过程的组织和管理。

19. Answer:

- (1) 软件重用可以显著地改善软件的质量和可靠性；
- (2) 软件重用可以极大地提高软件开发的效率；
- (3) 节省软件开发的成本，避免不必要的重复劳动和人力。

20. Answer:

21. Answer:

■ 代码和界面

22. Answer:

```
$sqrt5 = Math.sqrt 5
$phi = (1 + $sqrt5) / 2
def fib3 n
  (($phi**n - (1-$phi)**n) / $sqrt5).to_int
end

def fib4 n
  (Matrix[[1,1],[1,0]] ** n)[1,0]
end
```

23. Answer:

响应时间：

AJAX 应用可以仅向服务器发送并取回必需的数据，它使用 SOAP 或其它一些基于

XML 的 Web Service 接口，并在客户端采用 JavaScript 处理来自服务器的响应。因为在服务器和浏览器之间交换的数据大量减少，结果我们就能看到响应更快的应用。同时很多的处理工作可以在发出请求的客户端机器上完成，所以 Web 服务器的处理时间也减少了。

利用缓存：

Ajax 能将状态保存在客户端，从而实现常用资源，如重复使用的数据和源程序，可以有效被缓存和再利用。当 Ajax 第一次发送请求后，会把请求的 URL 和返回的响应结果保存在缓存内，当下一次调用 Ajax 发送相同的请求时，注意，这里相同的请求指的是 URL 完全相同，包括参数，浏览器就不会与服务器交互，而是直接从缓存中把数据取出来，这是为了提高页面的响应速度和用户体验。

方案中 Rails 利用页面缓存：这样做的好处是，可以把一些经常访问的页面作为页面缓存。缺点是，这种页面不能有太多用户的个人信息，因为这个页面对所有人访问都是相同的内容。如果必须考虑个人信息，可以改为 js 形式，或者使用方法缓存（Action Cache）。

数据源限制每个 IP 的 API 访问次数：

AJAX 最大优点就是能在不刷新整个页面的前提下与服务器通信维护数据。这使得 Web 应用程序更为迅捷地响应用户交互，并避免了在网络上发送那些没有改变的信息，减少用户等待时间，带来非常好的用户体验。

可扩展性：

Ajax 先加载静态的东西，把动态的东西后加载，这种局部加载，因此 Ajax 具有很好的扩展性。

24. Answer:

Creating a new MyName

0

Y

25. Answer:

Hello there, Dave!

26. Answer:

1

0

27. Answer:

[2]

[2, 3, 4, 5]

28. Answer:

Hello World!

Goodbye World. Come back soon!

Hello Zeke!

Goodbye Zeke. Come back soon!

Hello Albert!

Hello Brenda!

Hello Charles!

Hello Dave!

Hello Engelbert!

Goodbye Albert, Brenda, Charles, Dave, Engelbert. Come back soon!

...

...

29. Answer:

局部

实例

类

全局.

30. Answer:

```
class Person
  @@no_of_person=0
  def initialize(name, age)
    @cust_name=name
    @cust_age=age
    @@no_of_person += 1
  end

  def Person.how_many()
    @@no_of_person
  end
end

p1=Person.new("John",27)
p2=Person.new("Jane",37)

puts Person.how_many
```

31. Answer:

HTTP 方法 路径
GET /photos

控制器#动作
photos#index

GET	/photos/new	photos#new
POST	/photos	photos#create
GET	/photos/:id	photos#show
GET	/photos/:id/edit	photos#edit
PATCH/PUT	/photos/:id	photos#update
DELETE	/photos/:id	photos#destroy

32. Answer:

render: 在创建失败后, 需要重新返回创建页面, 也就是 new 的视图

redirect_to: 创建成功后, 重新导向用户浏览器至 show 控制器动作处理并渲染

33. Answer:

book, edit_book_path(book), :delete, new_book_path

34. Answer:

form_for, f.text_field, :email, f.submit

35. Answer:

assert_template, assert_select

36. "foo".curvy?

37. Answer:

@user.valid?, assert_not @user.valid?

38. Answer:

- (1) A. 按设计意图工作。
B. 不按设计意图工作。
- (2) A. 按设计意图工作。
B. 不按设计意图工作。
- (3) A. 按设计意图工作。
B. 按设计意图工作, 如果把 attr_accessor :title, :release_date 加到模型代码中。
C. 不按设计意图工作, 即使把 attr_accessor 加到模型代码中。

39. Answer:

1) 新建一个名为 blog 的 rails 项目

\$ rails new blog

2) 启动一个 web 服务器

```
$ cd blog
```

```
$ bin/rails server
```

3) 打开浏览器访问 <http://localhost:3000> 你将看到 Rails 默认信息页面:



4) 创建自己的欢迎页面

首先需要创建新的控制器: 运行 “controller” 生成器, 并告诉它您需要一个名为 “Welcome” 的控制器, 并执行一个名为 “index” 的操作

```
$ bin/rails generate controller Welcome index
```

输出如下:

```
create  app/controllers/welcome_controller.rb
route   get 'welcome/index'
invoke  erb
create  app/views/welcome
create  app/views/welcome/index.html.erb
invoke  test_unit
create  test/controllers/welcome_controller_test.rb
invoke  helper
create  app/helpers/welcome_helper.rb
invoke  test_unit
invoke  assets
invoke  coffee
create  app/assets/javascripts/welcome.coffee
invoke  scss
create  app/assets/stylesheets/welcome.scss
```

然后需要在视图/[views/welcome/index.html.erb](#) 中，写入如下代码：

```
<h1>Hello, Rails!</h1>
```

5) 设置主页

打开 config/[routes.rb](#),

```
Rails.application.routes.draw do
  get 'welcome/index'
  _____
end
```

加入 [root 'welcome#index'](#)，从而设置欢迎页面为主页。

6) 创建一个 resource。resource 是指类似物品的集合，如物品、人或动物。您可以为 resource 创建、读取、更新和销毁项，这些操作称为 CRUD 操作。

Rails 提供了一个可以用来声明标准 REST 资源的 resource 方法。将 article 资源添加到 config/routes 中需要在横线处填入 [resources :articles](#)

```
Rails.application.routes.draw do
  get 'welcome/index'
  _____

  root 'welcome#index'
end
```

使加入的 resource 生效: [bin/rails routes](#)

7) 创建 form

New Article

Title

Text

Save Article

在 app/views/articles/new.html.erb

```

<%= form_with scope: :article, local: true do |form| %>
  <p>
    <%= form.label :title %><br>
    <%= form.text_field :title %>
  </p>

  <p>
    <%= form.label :text %><br>
    <%= form.text_area :text %>
  </p>

  <p>
    <%= form.submit %>
  </p>
<% end %>

```

表单需要使用不同的 URL 才能到达其他地方。这可以通过 form_with 的:url 选项来完成。

```

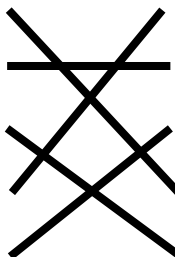
<%= form_with scope: :article, url: articles_path,
local: true do |form| %>

```

使 route 生效: \$ bin/rails routes

40. **Answer:** rails new blog -d mysql

41. **Answer:**

一般小写字母、下划线开头		类变量 (Class variable)
\$开头		全局变量 (Global variable)
@开头		常数 (Constant)
@@开头		变量 (Variable)
大写字母开头		实例变量 (Instance variable)

(每一条线连对给 2 分)

42. **Answer:**

Load 和 require 方法:

共性: 用来加载库, 他们的使用情景是模块和类位于不同的文件时, 用 load 或 require 引入代码。

区别: (1) require 使用是不需要加扩展名, require 'test_library'

Load 需要加扩展名, load 'test_library.rb'

(1) require 加载库只能加载一次, load 可用来多次加载库

Include 和 extend 方法:

共性: 他们的使用吃用情景是模块和库位于同一个文件, 将模块中定义的方法插入类。

区别: include 对实例方法进行引用, extend 直接将模块中的方法作为类方法插入。

43. Answer:

请求行、请求头、请求体

第一行为 http 请求行, 包含方法, URI 和 http 版本

2-7 为请求头, 包含浏览器, 主机, 接受的编码方式和压缩方式

第 8 行表示一个 空行 表示请求头结束 这个空行是必须的

第 9 行是数据体, 比如是需要查询的信息。

44. Answer:

修改内置的类是个很强大的功能, 不过功能强大意味着责任也大, 如果没有很好的理由, 向内置的类中添加方法是不好的习惯。Rails 自然有很好的理由。例如, 在 Web 应用中 我们经常要避免变量的值是空白的 (blank), 像用户名之类的就不应该是 空格或空白, 所以 Rails 为 Ruby 添加了一个 blank? 方法。Rails 控制台会自动加载 Rails 添加的功能, 下面看几个例子 (在 irb 中不可以)

```
>> "".blank?
=> true
>> "   ".empty?
=> false
>> "   ".blank?
=> true
>> nil.blank?
=> true
```

可以看出, 一个包含空格的字符串不是空的 (empty), 却是空白的 (blank)。还要注意, nil 也是空白的。因为 nil 不是字符串, 所以上面的代码说明了 Rails 其实是把 blank? 添加到 String 的基类 Object 中的

■ 框架与开发

45. Answer:

1)

Cloud9 是一种基于云的集成开发环境 (IDE), 在任何地方可以使用已连接 Internet 的计算机完成项目, Cloud9 还可以为开发无服务器应用程序提供无缝体验, 能够轻松定义资源、进行调试, 并在本地和远程执行无服务器应用程序之间来回切换。简而言之, 云计算可以为 SaaS 提供一个可扩展且可靠的硬件计算和存储, 云计算中的集群计算可以通过局域网交换机连接商业服务器, 这些大型集群或仓库规模可以为计算机提供 一定规模的经济效益, 利用 规模经济, 从而让云计算能以 更低 的费用提供更多的计算资源。

2)

云计算采用 MapReduce 编程框架, 而 MapReduce 编程框架是一个 share nothing 的设

计，也就是集群的每个节点都有独立的内存和硬盘。云计算平台具有良好的可扩展性，并不是指可以无限扩展，而受 MapReduce 支持节点数的限制，也并非任何一个应用都可无限横向扩展。“无线扩展”的主要例子是亚马逊网络服务、谷歌的 App Engine 和微软 Azure。

46. Answer:

1)

三层体系结构包括表示层（渲染视图并与用户进行交互）、逻辑层（运行 SaaS 应用程序代码）、持久层（存储应用程序数据）。

三层无共享架构之所以称为无共享架构，是因为层内实体一般不进行内部沟通，它允许在每层中独立地增加计算机以满足需求。

2)

该架构中的负载均衡器可以通过硬件设备或专门配置的 Web 服务器均匀地分配工作量。无状态的 HTTP 可以让无共享组件成为可能，所有请求都是独立的，可以向表示层或逻辑层中的任何服务器分配任何请求。

3)

持久层是难以横向扩展的。上图显示了主从方法，是在数据库中读取比写入更频繁的时候使用的。任何从机都可以执行读取，只有主机才能执行写入，然后主机使用写入的结果尽可能更快地更新从机。这种技术只是延缓了扩展性问题的出现，而无法从根本上解决这个问题。

47. Answer:

- 2. Users index
- 3. index User Userall
- 4. User
- 5. User
- 6. @users index

48. Answer:

单一系统架构

优点：子系统之间可以直接共享数据

缺点：

- (1) 子系统之间耦合性太高，其中一个升级其他都得升级
- (2) 系统扩展性差，开发困难，各个开发团队最后要整合到一起
- (3) 不能灵活进行分布式部署

面向服务设计架构（SOA）

优点：

(1) 把系统拆分, 使用接口通信, 降低子系统之间的耦合度, 增加功能时只需要增加子项目, 调用其他系统的接口就可以了, 功能升级方便

(2) 由于 SOA 的组件性和优良的扩展性, 使其可以根据不同的需求进行重新组合和构造

(3) 可以灵活进行分布式部署

缺点:

(1) 子系统之间交互需要使用远程通信

(2) 接口开发增加工作量

49. Answer: 路由在 url.py 文件中配置。path: 用于普通路径, 不需要自己手动添加正则首位限制符号, 底层已经添加。

50. Answer:

(1). BDD 可以解决第一个问题, TDD 可以解决第二个问题。

(2). 原因: BDD 是行为驱动开发。它是一种敏捷软件开发的技术, 鼓励软件项目中的开发者、QA 和非技术人员或商业参与者之间的协作。主要是从用户的需求出发, 保证程序实现效果与用户需求一致。

TDD 是测试驱动开发。TDD 的原理是在开发功能代码之前, 先编写单元测试用例代码, 测试代码确定需要编写什么产品代码。它的基本思路就是通过测试来推动整个开发的进行, 但测试驱动开发并不只是单纯的测试工作, 而是把需求分析, 设计, 质量控制量化的过程。TDD 首先考虑使用需求 (对象、功能、过程、接口等), 主要是编写测试用例框架对功能的过程和接口进行设计, 而测试框架可以持续进行验证。

51. Answer:

命令如下:

```
git add -A
git checkout master
git merge login
git push -u origin master
```

52. Answer:

(1). 在登陆界面输入账户和密码后进入选课界面

(2). 选课界面内展示了课程按照学院顺序排列的表 (指明顺序)

(3). 在搜索栏可以通过课程名、教室、学院等选项搜索课程 (指明用什么搜索)

(4). 搜索结果与实时课程都可以在 1 秒内返回 (指明时间)

(5). eg: 可以看到课程评价 (最后一个模块与功能无关, 可以直接删去或修改)

53. Answer:

- 1). 真正的项目很少遵循顺序模型。随着项目团队的进行, 更改可能会导致混乱。
- 2). 对于客户来说, 明确地陈述所有的需求是很困难的。难以适应许多项目开始时存在的自然不确定性。
- 3). 顾客必须有耐心。程序的工作版本将在项目时间跨度的后期才可用。一个重大失误, 如果不及时发现, 可能是灾难性的。

54. Answer: Hi Albert! (2 分)

Bye Albert, come

55. Answer:

Creating a new MyName

O

Y

56. Answer:

(1) 软件开发中的重构是什么: (1.5 分)

在不影响软件外部表现的前提下, 对其内部实现进行的调整和修改。

(2) 为什么要重构: (2 分)

- 1) 重构的代码更加清晰, 更易于理解
- 2) 提高应用程序的性能
- 3) 方便对旧版的库或者框架进行更新
- 4) 重构可以改进系统设计

(3) 有哪些基本原则: (2.5 分)

- 1) 以一致性为目标
- 2) 避免深度嵌套
- 3) 采取单独关注 SRP 原则
- 4) 避免重复
- 5) 高内聚、低耦合

57. Answer:

指的是先构建测试案例来描述应用程序的行为, 当测试完成后, 开发人员才开始编码, 一旦测试通过, 那软件就完成了测试中描述的某种行为。传统的先代码后测试是“由里到外”, 弊端是内部功能在不断迭代的情况下, 外部测试无法完善、外部测试受内部开发思路限制等。而 BDD 开发模式使用用户和开发人员都可以理解的一门语言, 把需求转换成了软件的功能测试, 先写功能测试再驱动出产品代码, 保证软件行为正确性。

58. Answer:

- (1) 每一个 URI 代表 1 种资源
- (2). 客户端使用 GET、POST、PUT、DELETE 4 个表示操作方式的动词对服务端资源进行操作: 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
- (3). 通过 操作资源的表现形式来操作资源
- (4). 资源的表现形式是 XML 或者 HTML
- (5). 客户端与服务端之间的交互在请求之间是 无状态的, 从客户端到服务端的每个请求都必须包含 理解请求所必需的信息

59. **Answer:** 相同点:

- (1). Flask 和 Django 都是 Python Web 开发框架。
- (2). 都是已被用户大量使用的开发框架, 有完善的用户使用文档。

不同点:

- (1). Flask 相比 Django 是更加轻量的 Web 框架, 适合中小型项目, Django 相比
- (2). Flask 是一个更全面的重量级框架, 适合大中型项目的快速开发。
- (3). Flask 的可扩展性相对强, 用户在使用 Flask 进行 Web 开发时既可以自己造轮子, 也能利用已有的 Python 库。Django 的灵活度就相对较低, 它自带大量的常用工具和组件 (比如数据库 ORM 组件), 不过 Flask 可以通过添加 flask-admin 以及 ORM 拓展弥补这一点, 同时这也体现了 Flask 框架的高自由度。
- (4). Django 项目的结构布局是刚性的, 最终的项目结构大体类似, Flask 是很灵活的, 用户可以随意地组织自己的代码

60. **Answer:**

Workspace: 工作区, 就是你平时存放项目代码的地方

Index / Stage: 暂存区, 用于临时存放你的改动, 事实上它只是一个文件, 保存即将提交到文件列表信息

Repository: 仓库区 (或版本库), 就是安全存放数据的位置, 这里面有你提交到所有版本的数据。其中 HEAD 指向最新放入仓库的版本

Remote: 远程仓库, 托管代码的服务器, 可以简单的认为是你项目组中的一台电脑用于远程数据交换

61. **Answer:**

依赖倒置原则 (Dependence Inversion Principle)

依赖倒置原则说的是 高层模块 不应该 依赖底层模块, 两者都应该

依赖 其抽象 这个原则其实是在指导如何实现接口隔离原则, 也就是

前文提到的，高层的消费者不应该依赖于具体实现，应该由消费者定义并依赖于 Role interface，底层的具体实现也依赖于 Role interface，因为它要实现此接口。

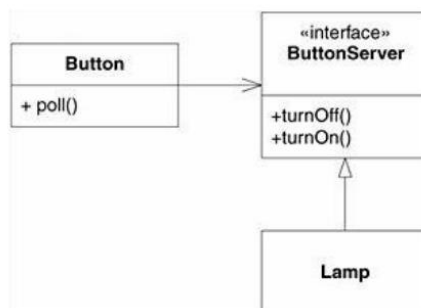
依赖倒置原则是区分过程式编程和面向对象编程的分水岭。过程式编程的依赖没有倒置。



上图的关系中，当 Button 直接调用灯的开和关时，Button 就依赖于灯了。其代码完全是过程式编程：

```
public class Button {
    private Lamp lamp;
    public void Poll() {
        if (/*some condition*/)
            lamp.TurnOn();
    }
}
```

如果 Button 还想控制电视机，微波炉怎么办？应对这种变化的办法就是抽象，抽象出 Role interface ButtonServer：



不管是电灯，还是电视机，只要实现了 ButtonServer，Button 都可以控制。这是面向对象的编程方式。

62. Answer:

从语言角度来看，基于Ruby 的Web框架可以为用户创造以下价值：

- 1) Ruby良好的面向对象特性，使得用户能够很好地组织代码。
- 2) Ruby的动态类型特性，使得用户更容易入门并上手使用。
- 3) Ruby灵活的语法，可以让用户以非常简洁的方式，创建一个Web应用。
- 4) Ruby在多数时候支持用多种方式，实现同一功能，使得用户在编写代码时不会受到语言的限制。

从框架角度来看，Ruby on Rails本身为用户创造了以下价值：

- 1) 以MVC架构为基础，很好地分离了视图、数据、控制逻辑三部分，帮助用户拆解任

务，更容易地设计一个规范的Web应用。

2) 约定大于配置的设计哲学，让用户只需关心有意义的部分。

3) 集成大量精选功能，使得用户的大多数需求，都能找到现有的优质实现。

4) 多元化的设计模式，不会限制用户的编码方式。

5) 进步比稳定重要的设计哲学，敢于采纳更好的设计方式，应对技术发展，使得用户不会创建“过时”的 Web 应用。

■ 测试

63. Answer:

1) 功能方面，是否能按指定条件查到正确、完整的结果，具体：

1.1 录入条件为可查到结果的正常关键字、词、语句，检索到的内容、链接正确性；

1.2 录入条件为不可查到结果的关键字、词、语句；

1.3 录入条件为一些特殊的内容，如空、特殊符、标点符、极限值等，可引入等价类划分的方法等；

2) 性能方面，可利用测试工具或各种测试手段考虑功能在各方面的表现，具体：

2.1 压力测试：在不同发用户数压力下的表现（评价指标如响应时间等）

2.2 负载测试：看极限能承载多大的用户量同时正常使用

2.3 稳定性测试：常规压力下能保持多久持续稳定运行

2.4 内存测试：有无内存泄漏现象

2.5 大数据量测试：如模拟从庞大的海量数据中搜索结果、或搜索出海量的结果后列示出来，看表现如何等等。

3) 易用性方面，交互界面的设计是否便于、易于使用，具体：

3.1 依据不同的查询结果会有相关的人性化提示，查不到时告知？查到时统计条数并告知？有疑似输入条件错误时提示可能正确的输入项等等处理；

3.2 查询出的结果罗列有序，如按点击率或其他排序规则，确保每次查询出的结果位置按规则列示方便定位，显示字体、字号、色彩便于识别等等；

3.3 标题查询、全文检索、模糊查询、容错查询、多关键字组织查询（空格间格开）等实用的检索方式是否正常？

3.4 输入搜索条件的控件风格设计、位置摆放是否醒目便于使用者注意到，有否快照等快捷查看方式等人性化设计？

4) 兼容性方面，跨平台、多语言等多样性环境组合情况下测试使用的正常性，具体：

4.1 WINDOWS/LINUX/UNIX 等各类操作系统下及各版本条件下的应用

4.2 IE/FIREFOX/GOOGLE/360/QQ 等各类浏览器下及各版本条件下、各种显示分辨率条件下的应用

4.3 SQL/ORACLE/DB2/MYSQL 等各类数据库存储情况下的兼容性测试

4.4 简体中文、繁体中文、英文等各类语种软件平台下的兼容性测试

4.5 IPHONE/IPAD、安卓等各类移动应用平台下的兼容性测试

4.6 与各相关的监控程序的兼容性测试，如输入法、杀毒、监控、防火墙等工具同时使用

5) 安全性方面，往往容易被忽视的环节，具体：

5.1 被删除、加密、授权的数据，不允许被查出来的，是否有安全控制设计；

5.2 录入一些数据库查询的保留字符，如单引号、%等等，造成查询 SQL 拼接出的语句产生漏洞，如可以查出所有数据等等，这方面要有一些黑客攻击的思想并引入一些工具和技术，如爬网等。

5.3 通过白盒测试技术，检查一下在程序设计上是否存在安全方面的隐患；

5.4 对涉及国家安全、法律禁止的内容是否进行了相关的过滤和控制；

6) 异常性测试，各种破坏性的操作的影响测试，具体：

6.1 查询过程中断网、关机

6.2 查询过程中强行中断关闭页面

6.3 查询过程中强行杀死相关进程等

64. Answer:

(1) 编写自动化测试主要有三个好处：

- 1) 测试能避免回归 (regression) 问题，即由于某些原因之前能用的功能不能用了；
- 2) 有测试在重构 (改变实现方式，但功能不变) 时更有自信；
- 3) 测试代码是使用接口来调用实现功能的代码，因此帮助设计，并确定代码是如何系统中其他组件交互。

(2) 何时以及如何测试：

- 1) 首先，取决于你编写测试的熟练程度。开发者熟练之后，更倾向于先编写测试。
- 2) 其次，取决于测试代码与应用代码的难度，你对想实现的功能有多深的认识。

(3) 写测试代码的一些建议：

- 1) 与应用代码相比，如果测试代码特别简短，倾向于先编写测试。
- 2) 如果对想实现的功能不是特别清楚，倾向于先编写应用代码，然后再编写测试，并改进实现方式；
- 3) 安全是头等大事，保险起见，要为安全相关的功能先编写测试。

- 4) 只要发现一个问题，就编写一个测试重现这种问题，避免回归，然后再编写应用代码修正问题；
- 5) 尽量不为以后可能修改的代码（例如 HTML 结构的细节）编写测试；
- 6) 重构之前要编写测试代码，集中测试容易出错的代码。

我们一般先编写控制器和模型测试，然后再编写集成测试（测试模型、视图和控制器结合在一起时的行为）。如果应用代码很容易出错，或者经常变动（视图就是这样），我们就完全不测试。

65. Answer:

```
#被测试程序:
#vim calculator.rb
class Calculator
  def add(x,y)
    x+y
  end
  def subtract(x,y)
    x-y
  end
  def multiply(x,y)
    x*y
  end
  def divide(x,y)
    x/y
  end
end

#测试程序:
#vim calculator_test.rb
require 'MiniTest/autorun'
require './calculator.rb'

class TestCalculator < MiniTest::Test
  def setup
    @calculator = Calculator.new
  end

  def test_that_is_Addition
    assert_equal 3, @calculator.add(1,2)
  end

  def test_that_is_Subtraction
    assert_equal 1, @calculator.subtract(5,4)
  end

  def test_that_is_Multiplication
    assert_equal 42, @calculator.multiply(6,7)
  end

  def test_that_is_Division
    assert_equal 5, @calculator.divide(10,2)
  end
end
```

66. Answer:

该段为 Rspec 测试代码，目的是检查 /static_pages/home 路由中是否包含 "Sample App"。

67. Answer:

(2) 好处:

- 1). 测试能避免回归 (regression) 问题，即由于某些原因之前能用的功能现在不能用了；
- 2). 有测试在，重构 (改变实现方式，但功能不变) 时更有自信；
- 3). 测试是 应用代码的客户，因此可以协助我们设计，以及决定如何与系统的其他组件交互。

(2) 何时

- 1). 与应用代码相比，如果测试代码特别简短，倾向于先编写测试；
- 2). 如果对想实现的功能不是特别清楚，倾向于先编写应用代码，然后再编写测试，并改进实现方式；
- 3). 安全是头等大事，保险起见，要为安全相关的功能 先编写测试；
- 4). 只要发现一个问题，就编写一个测试重现该问题，避免回归，然后再编写应用代码修正问题；
- 5). 尽量不为以后 可能修改的代码 (例如 HTML 结构的细节) 编写测试；
- 6). 重构之前要编写测试，集中测试容易出错的代码。

68. Answer:

- (1). App 稳定性测试 来避免长时间运行下的 闪退、内存泄露、性能变差 等问题
- (2). 通过 App 稳定性测试可以增强 App 本身的 稳定性和容错性
- (3). 保证产品的质量 提高用户的体验

69. Answer:

单元测试 → 集成测试 → 确认测试 → 系统测试。单元测试对源程序中每一个程序单元进行测试，检查各个模块是否正确实现规定的功能，从而发现模块在编码中或算法中的错误。该阶段涉及 编码 和 详细设计文档。集成测试是为了检查与设计相关的 软件体系结构的有关问题，也就是检查 概要设计 是否合理有效。确认测试主要是检查已实现的软件是否满足需求规格说明书中确定的各种需求。系统测试是把已确认的软件与其他系统元素 (如硬件、其他支持软件、数据、人工等) 结合在一起进行测试。以确定软件是否可以支付使用。

70. Answer:

明确要开发某个功能后，首先思考如何对这个功能进行测试，并完成测试代码的编写，

TOP

然后编写相关的代码满足这些测试用例。然后循环进行添加其他功能，直到完成全部功能的开发。

BDD

行为驱动开发是测试驱动开发的进化，但关注的核心是设计。行为驱动开发中，定义系统的行为是主要工作，而对系统行为的描述则变成了测试标准。

二者都是在真正开发之前制定测试用例以规范程序的开发，区别在于测试驱动是在功能设计完成之后使用结构化的语言制定测试用例；而行为驱动开发是在设计功能时寻找一种通用的语言，让用户、开发等多方人员可以很好地理解，并以用户的角度制定对程序功能的描述，形成对软件的验收标准。

■ 两种途径

71. Answer:

这两种方式往往是一起使用的，第一种方式是：人类发现问题，提出解决方案（构思模型），然后设计代码（程序）来让计算机执行来解决问题。第二种方式是，人类发现问题，但没法构思出有效的解决方案（模型），人类设计一些学习程序，让计算机执行学习程序，根据给出的样本数据，计算机训练出解决方案（模型），然后集成到计算机中，即把人编制的代码和机器产生的代码结合在一起运行来解决问题。

现在很多问题都采用第二种方式来解决，第二种方式虽说是数据集训练，但是也有人编写的代码的参与，比如需要对猫狗图像进行分类，这时候人无法直接构造模型，需要用大量的图片去训练，然而在这之前需要人凭借经验去搭建合适的网络，设置合适的参数，计算机在人搭建好的网络上用数据训练出模型。

相同点	不同点
对数据集特点的总结	人善于发现直观，计算机善于发现潜在规律
都有模型数据	人用代码驱动模型，机器用数据驱动模型
依赖数据，计算机，开发工具，需要评价指标	人设计方法以来人的经验，计算机以来数据和训练方法
提取特征，寻找特征，分类标签的映射规律	人找的规律具有很强主观性，计算机依赖数据更合理，但耗时易过拟合
通过不断迭代演化生成function	人总结规律集中设计编码和规则，数据集训练主要设计模型流程
都是基于数据集进行训练	人工代码效率更高，数据训练逻辑性和解释性更低
解决某个问题需要数据支持	人总结规律是透明，具体可解释；机器的规律是不透明

1. 设计一款类似于 Jupyter 的关键点在哪里？

Answer:

内核——交互性

支持本地和远程的启动

传输时候缓存对输入输出存有记录

72. Answer:

- (1) 计算机算力不够；模型性能低下；样本数量不充足和质量较低；图片处理计算不够强大；没有成熟的训练框架；人为归纳特征无法覆盖全部特征等。

- (2) 相同点：对数据集的总结；依赖数据，计算机，开发工具和需要评价指标；需要不断迭代的损失函数等。
- 不同点：人善于发现直观，计算机善于发现潜在规律；人用代码驱动模型，计算机数据驱动模型；人设计以人的经验，计算机以数据和训练方法等。
- (3) 提高模型准确性；具体应用场景；前段交互，反馈和扩展功能；平台兼容性和依赖性；硬件设备等。