设计模式完全手册

讲师：丁宋涛 夏曹俊

创建型模式之简单工厂

- 学习简单工厂
- 一：简单工厂模式的介绍-定义、结构、参考实现、场景问题
- 二：简单工厂的典型疑问与优缺点评价
- 三：简单工厂的应用案例与思考

接口？什么是针对接口编程？

- 接口从语法层面上来说，是一种特殊的抽象类，是一个纯虚的类。从软件设计的意义上来说，我们通常用接口来定义实现类的外观，就相当于一份契约，根据外部应用需要的功能，约定了实现类应该要实现的功能。
- 软件开发中永恒的主题是“变化”，“只有变化才是永恒不变！”，接口最重要的一个设计语义就是封装变化。所谓“封装变化”就是隔离变化。
- 从软件的整体结构上看，只要接口不变，内部实现的变化就不会影响到外部应用，从而使得系统更灵活，具有更好的扩展性和可维护性。

场景

- 比如，通常的，我们都知道Api，我们设计了一组API，一个程序需要使用这些Api完成功能。我们如何规划Api的设计。

简单工厂的定义

- 提供一个创建对象实例的功能，而无需关心其具体实现。被创建的类型可以是接口、抽象类，也可以是具体的类。

简单工厂在实际开发中的应用案例

- 简单工厂可以帮助我们实现具体类的选择，如果这个时候，在需要添加一个新的实现类怎么办?
- 动态添加：简单工厂与客户端的完全解耦

单子（单例）模式

- 学习单例模式
- 一：单例模式的介绍-定义、结构、参考实现、场景问题
- 二：单例模式的典型疑问与优缺点评价：饿汉，懒汉与多线程安全
- 三：单例的应用案例与思考：缓存

单子（单例）模式

- 保证一个类仅有一个实例，并提供一个访问它的全局访问点。
- 懒汉，额汉与多线程安全

单子（单例）模式的扩展与工程应用-缓存

- 既然我们可以控制全局生成一个对象，那么有没有需要生成1个以上的对象呢?
- 缓存设计与Singleton的扩展

工厂模式

- 学习工厂模式
- 一：工厂模式的介绍-定义、结构、参考实现、场景问题
- 二：工厂模式的典型疑问与优缺点评价
- 三：工厂模式的应用案例与思考

场景

- 考虑这样一个实际应用：实现一个导出数据的应用框架，来让客户选择数据的导出方式，并真正执行数据导出。
- 通常这种系统，在导出数据上，会有一些约定的方式，比如导出成：csv格式、数据库备份形式、Excel格式、Xml格式等等。

如何设计

- 从封装的角度来说，我们希望导出数据的业务功能对象创建ExportFileApi的具体实例，目前他只知道接口，怎么办？

工厂模式定义

- 1：工厂方法模式的功能
- 工厂方法的主要功能是让父类在不知道具体实现的情况下，完成自身的功能调用，而具体的实现延迟到子类来实现。
- 2：实现成抽象类
- 工厂方法的实现中，通常父类会是一个抽象类，里面包含创建所需对象的抽象方法，这些抽象方法就是工厂方法
- 3：实现成具体的类
- 也可以把父类实现成为一个具体的类，这种情况下，通常是在父类中提供获取所需对象的默认实现方法，这样就算没有具体的子类，也能够运行 。
- 4：工厂方法的参数和返回值
- 工厂方法的实现中，可能需要参数，以便决定到底选用哪一种具体的实现。
- 一般工厂方法返回的是被创建对象的接口对象，当然也可以是抽象类或者一个具体的类的实例。

工厂模式的UML示例与代码

工厂模式的本质-依赖倒置原则-让子类选择实现

- 依赖倒置原则告诉我们“要依赖抽象，不要依赖于具体类”：不能让高层组件依赖于低层组件，而且不管高层组件还是低层组件，都应该依赖于抽象。
- 何时选用工厂方法模式
- 1：如果一个类需要创建某个接口的对象，但是又不知道具体的实现，这种情况可以选用工厂方法模式，把创建对象的工作延迟到子类去实现
- 2：如果一个类本身就希望由它的子类来创建所需的对象的时候，应该使用工厂方法模式

工厂模式的工程应用-IOC/DI

- 工厂方法模式与IoC/DI
- 依赖注入：应用程序依赖容器创建并注入它所需要的外部资源
- 控制反转：容器控制应用程序，由容器反向的向应用程序注入应用程序所需要的

抽象**工厂**

- 学习抽象工厂模式
- 一：抽象工厂模式的介绍-定义、结构、参考实现、场景问题
- 二：抽象工厂模式的典型疑问与优缺点评价
- 三：抽象工厂模式的应用案例与思考

场景

- 考虑这样一个实际应用：我需要完成两套GUI的表示层，一个是在PC机，另一个是在平板、手机上完成这个应用程序的界面。
- 通常一个显著的设备差别就在于分辨率。

如何设计

- 从封装的角度来说，我们希望根据布局器配置我们的控件，我们开放布局器接口和控件接口，供客户端调用。

抽象工厂模式

- 1：模式的功能
- 抽象工厂的功能是为一系列相关对象或相互依赖的对象创建一个接口。
- 从某种意义上看，抽象工厂其实是一个产品系列，或者是产品簇。
- 2：实现成接口
- AbstractFactory实现成为接口
- 3：使用工厂方法
- AbstractFactory定义了创建产品所需要的接口，具体的实现是在实现类里面，通常在实现类里面就需要选择多种更具体的实现，所以AbstractFactory定义的创建产品的方法可以看成是工厂方法，而这些工厂方法的具体实现就延迟到了具体的工厂里面。也就是说使用工厂方法来实现抽象工厂。
- 4：切换产品簇
- 抽象工厂定义了一个产品簇，因此切换产品簇的时候提供不同的抽象工厂就好了

工厂模式的UML示例与代码

抽象工厂模式的工程应用-控件式开发

- 业务功能封装成控件：
- 微软有一个叫做asp.net的控件组，它认为对于一个Web应用来说，虽然搜索框、广告条、导航条、页面主题的设计与实现有很大的差别，但是绝大多数的Web应用的基本布局还是大致相同的。最后都可以组装成一个页面的过程也是大致相同的。
- 这种具有统一属组的产品簇特别适合使用抽象工厂进行装配。

抽象工厂的本质-选择产品簇的实现

- 抽象工厂模式的优缺点
- 1：分离接口和实现
- 2：使得切换产品簇变得容易
- 3：不太容易扩展新的产品
- 4：容易造成类层次复杂

抽象工厂的本质-何时选用抽象工厂模式

- 1：如果希望一个系统独立于它的产品的创建，组合和表示的时候，换句话说，希望一个系统只是知道产品的接口，而不关心实现的时候
- 2：如果一个系统要由多个产品系列中的一个来配置的时候，换句话说，就是可以动态的切换产品簇的时候
- 3：如果要强调一系列相关产品的接口，以便联合使用它们的时候

生成器/构建器模式（Builder）

- 学习Builder模式
- 一： Builder模式的介绍-定义、结构、参考实现、场景问题
- 二： Builder模式的典型疑问与优缺点评价
- 三： Builder模式的应用案例与思考

场景

- 考虑这样一个实际应用：银行对账单导出数据的应用，通常对于具体的导出内容和格式是有要求的：
- 1导出的文件，不管什么格式，都分成三个部分，分别是文件头、文件体和文件尾
- 2 在文件头部分，需要描述如下信息：分公司编号、导出数据的日期，等等
- 3在文件体部分，需要描述如下信息：表名称、然后分条描述数据。
- 4 在文件尾部分，需要描述如下信息：输出人

	账号-册序号/文书合同号	开户行	产品类型	币种	本期余额	账户名称
1	40[illegible] ...	中国银行 都支行	单位人民币活期基本账户存款	人民币	7,752.08	有限公司

如何设计

- 从封装的角度来说，我们希望根据布局器配置我们的控件，我们开放布局器接口和控件接口，供客户端调用。

Builder模式

- 1：模式的功能
- 生成器模式的主要功能是构建复杂的产品，而且是细化的，分步骤的构建产品，也就是生成器模式重在解决一步一步构造复杂对象的问题。这个构建的过程是统一的，固定不变的，变化的部分放到生成器部分了，只要配置不同的生成器，那么同样的构建过程，就能构建出不同的产品表示来。
- 直白点说，生成器模式的重心在于分离构建算法和具体的构造实现，从而使得构建算法可以重用，具体的构造实现可以很方便的扩展和切换，从而可以灵活的组合来构造出不同的产品对象。
- 生成器模式分成两个很重要的部分：
- （1）一个部分是Builder接口这边，这边是定义了如何构建各个部件，也就是知道每个部件功能如何实现，以及如何装配这些部件到产品中去；
- （2）另外一个部分是Director这边，Director是知道如何组合来构建产品，也就是说Director负责整体的构建算法，而且通常是分步骤的来执行。
- 不管如何变化，Builder模式都存在这么两个部分，一个部分是部件构造和产品装配，另一个部分是整体构建的算法。在生成器模式中，强调的是固定整体构建的算法，而灵活扩展和切换部件的具体构造和产品装配的方式。

Builder模式

- 1：模式的功能
- 生成器模式的主要功能是构建复杂的产品，而且是细化的，分步骤的构建产品，也就是生成器模式重在解决一步一步构造复杂对象的问题。这个构建的过程是统一的，固定不变的，变化的部分放到生成器部分了，只要配置不同的生成器，那么同样的构建过程，就能构建出不同的产品表示来。
- 直白点说，生成器模式的重心在于分离构建算法和具体的构造实现，从而使得构建算法可以重用，具体的构造实现可以很方便的扩展和切换，从而可以灵活的组合来构造出不同的产品对象。
- 生成器模式分成两个很重要的部分：
- （1）一个部分是Builder接口这边，这边是定义了如何构建各个部件，也就是知道每个部件功能如何实现，以及如何装配这些部件到产品中去；
- （2）另外一个部分是Director这边，Director是知道如何组合来构建产品，也就是说Director负责整体的构建算法，而且通常是分步骤的来执行。
- 不管如何变化，Builder模式都存在这么两个部分，一个部分是部件构造和产品装配，另一个部分是整体构建的算法。在生成器模式中，强调的是固定整体构建的算法，而灵活扩展和切换部件的具体构造和产品装配的方式。

Builer模式的UML示例与代码

生成器模式的本质-分离整体构建算法和部件构造

- 何时选用生成器模式
- 1：如果创建对象的算法，应该独立于该对象的组成部分以及它们的装配方式时
- 2：如果同一个构建过程有着不同的表示时

原型模式（Prototype）

- 学习Prototype模式
- 一：　Prototype模式的介绍-定义、结构、参考实现、场景问题
- 二：　Prototype模式的典型疑问与优缺点评价
- 三：　Prototype模式的应用案例与思考

场景

- 考虑这样一个实际应用：有一家制造业MRP系统里面有一个工件包的概念，制造型企业对于工件包中的每一个工件每天都有一个核查的需求，每当工件超过200项的时候，就需要将这个工件包拆成两份，如果还是超过200，继续拆分，直到单个数量不超过200。原因是后续阶段，该企业的生产部门对于工件的设计和进行人工处理，每个人每天处理工件的上限200个。起始，这个企业遇到的工件基本类型有国内企业和海外企业，我们把这个工件包需求定义成两种类型：HomeOrder和AboradOrder3

如何设计

- 我们抽象出OrderApi去处理HomeOrder和AboardOrder，然后调用OrderBussiness来处理。
- **已经有了某个对象实例后，如何能够快速简单地创建出更多的这种对象?**

原型模式

- **1：模式的功能**
- **直白点说，通过克隆来创建新的对象实例，为克隆出来的新的对象实例复制原型实例属性的值原型模式要实现的主要功能就是：通过克隆来创建新的对象实例。**