软件质量之路-面向组件的大规模软件架构 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/224/2021_2022__E8_BD_AF_E4_BB_B6_E8_B4_A8_E9_c41_224092.htm 在中小规模的软件中，对象和对象之间的协作关系就能够满足需要。但是当软件规模扩大，复杂度上升的时候，面向对象技术强调的协作却表现出另一个极端的特点－耦合度太高导致的复杂度。这时候就需要有一种新的方法来弥补面向对象技术的弱点。 大规模软件的特点 大规模软件主要特点是复杂度。比较典型的例子是集成性的项目。软件系统需要将各种各样的硬件、遗留系统、外部接口整合起来。其间可能遇到不同的硬件接口，不同的操作系统，不同的语言，不同的平台，不同的数据库，不同的消息中间件，不同的网络介质。这些都使得系统变得非常的复杂。 面向对象技术的特点是通过对象之间的职责分工和高度协作来完成任务。这样的好处是代码量较少，系统布局合理，重用程度高。但是当对象的个数大量增加的时候，对象之间的高度耦合的关系将会使得系统变得复杂，难以理解。 以前对于这个问题的方法是采用包（请参考拙作面向对象软件开发中对包的相关讨论）作为容器来组织对象，对象之间的依赖性将转化为包之间的依赖性。这种方法听起来有道理，但是在实际中仍会出现难以解决的问题。 包仅仅只是容器。这意味着对对象的组织可以是任意的，而包之间依赖关系的设计则还是取决于对象的依赖。此外，包的设计和对象一样，缺乏一个统一的风格。而统一的风格正是大规模软件设计所必须的，因为这样可以有效改进系统的可理解性，这一点非常重要。 面向组件编程 面向组件编程的缩写

是COP.COP是对OOP的补充，帮助实现更加优秀的软件结构。组件的粒度可大可小，需要取决于具体的应用。 在COP中有几个重要的概念：服务，服务（Service）是一组接口，供客户端程序使用。例如，验证和授权服务，任务调度服务。服务是系统中各个部件相互调用的接口；组件，组件（Component）实现了一组服务，此外，组件必须符合容器订立的规范，例如，初始化，配置、销毁。 COP 是对一种组织代码的思路，尤其是服务和组件两个概念。在下文会提到Spring框架中，就采用了COP的思路，将系统看作一个个的组件，通过定义组件之间的协作关系（通过服务）来完成系统的构建。这样做的好处是能够隔离变化，合理的划分系统。而框架的意义就在于定义一个组织组件的方式。 理解组件组件不是一个新的概念，Java中的javaBean规范和EJB规范都是典型的组件。组件的特点在于他定义了一种通用的处理方式。例如，JavaBean 拥有内视的特性，这样就可以通过工具来实现JavaBean的可视化。而EJB规范定义了企业服务中的一些特性，使得EJB容器能够为符合EJB规范的代码增添企业计算所需要的能力，例如事务、持久化、池等。 所以，组件比起对象来的进步就在于通用的规范的引入。通用规范往往能够为组件添加新的能力（就像上面所讨论的），但也给组件添加了限制，例如你需要实现EJB的一些接口。以下我们将讨论组件的一些相关问题： 组件的粒度 组件的粒度是和系统的架构息息相关的。组件的粒度确定了，系统的架构也就确定了。在小规模的软件中，可能组件的粒度很小，仅相当于普通的对象，但是对于大规模的系统来说，一个组件可能包括几十，甚至上百个对象。因此，对使用COP技术的系统来说，

需要正确的定义组件的粒度。较好的定义粒度的方法是对核心流程进行分析。 针对接口 接口和实现分离是COP的基础，没有接口和实现的分离，就没有COP.接口的高度抽象特性使得各个组件能够被独立的抽取出来，而不影响到系统的其它部分。 接口和实现分离有以下几个好处： 1.在模块/组件/对象之间解耦。 2.轻松的抽换实现，而不用修改客户端。 3.用户只需要了解接口，而不需要了解实现细节。 4.增加了重用的可能性。 