一种基于领域驱动设计划分微服务的方法

宁小庚 黄晓芳
(西南科技大学计算机科学与技术学院 四川绵阳 621010)

摘要: 随着微服务(Microservices)架构在后端领域逐渐被人接受,开发人员在将传统单一架构(Monolithic)应用拆分为微服务化过程中,或者是在设计一个全新的微服务架构应用中,难以把握服务粒度大小,无法保证采用的拆分策略能够在服务不断迭代之后仍然能够适应系统的复杂度。针对此问题,应用领域驱动设计方法,结合微服务架构的特点,提出一种寻找微服务边界的方法,摆脱了传统建模方式依赖数据库而导致根据数据库隐式建模,实现了服务的良好划分,提高系统的弹性,减少缺陷,提高开发效率。

关键词: 微服务 领域驱动设计(DDD) 应用系统设计
中图分类号: TP311 **文献标志码:** A **文章编号:** 1671-8755(2019)01-0080-06

Approach to Dividing Microservices Based on Domain Driven Design

NING Xiaogeng, HUANG Xiaofang
(*School of Computer Science and Technology, Southwest University of Science and Technology, Mianyang* 621010, *Sichuan, China*)

Abstract: As the Microservices architecture is gaining acceptance in the back-end world, developers are splitting traditional Monolithic applications into microservices or designing a new microservices architecture application. It is difficult to grasp the size of the service, and there is no guarantee that the split strategy can still adapt to the complexity of the system after the service is iterated. Based on this problem, a method to find the micro-service boundary was proposed with the application of domain-driven design method, combined with the characteristics of the micro-service architecture. This method is free from the traditional modeling method which relies on the database for implicit modeling, thus achieving a good division of services, improving the flexibility of the system, enhancing development efficiency and reducing defects.

Keywords: Microservices; Doamin driven design; Application system design

微服务[1]在后端开发领域已经席卷起一股风潮。微服务是一种架构模式,它提倡将单一应用程序划分为一组小的服务,服务之间相互协调、互相配合,为用户最终提供价值。每个服务运行在其独立的进程中,服务与服务间采用轻量级的通信机制互相协作(通常是基于HTTP协议的RESTful API)。每个服务都围绕着具体业务进行构建,并且能够被独立的部署到生产环境、类生产环境等。

但是,微服务天生的分布式特性,在传统单一架构上无须考虑的问题在微服务架构中就会很常见,在划分服务过程中开发人员可能会倾向于用熟悉的SOA(Service Oriented Architecture,SOA)标准进行划分。微服务虽然由SOA概念演化而来,但是两者之间却不可划等号[2],SOA首先要解决的是异构应用的服务化,而微服务强调的是服务拆分尽可能小,最好是独立的原子服务。服务的划分结果会直接影响系统功能,一个划分不成熟的微服务体系可能在后期开发过程中为了保证数据一致性而导致分布式锁的出现,这将导致系统复杂度急剧上升,不利于后期的迭代开发。

收稿日期: 2018-09-07
第一作者简介: 宁小庚(1992—),男,硕士研究生,E-mail: 1807035805@qq.com

领域驱动设计[3]（Domain Driven Design，DDD）是由 Eric Evans 最早提出的综合软件系统分析和设计的面向对象建模方法，如今已经发展成为了一种针对大型复杂系统的领域建模与分析方法。它完全改变了传统软件开发工程师针对数据库进行的建模方法，从而将要解决的业务概念和业务规则转换为软件系统中的类型以及类型的属性与行为，通过合理运用面向对象的封装、继承和多态等设计要素，降低或隐藏整个系统的业务复杂性，并使得系统具有更好的扩展性，以应对纷繁多变的现实业务问题。

本文将在领域驱动设计的研究基础上，结合微服务架构的特点，提出一种寻找微服务边界的方法，改进现有微服务架构设计存在的潜在的服务间耦合等问题，寻找更加良好的服务划分方法。

1　领域驱动的核心元素及设计概念

1.1　领域驱动的核心元素

图 1 为领域驱动设计的元素组成，本文在使用领域驱动寻找微服务边界的过程中不会全部使用这些元素，仅会使用符合微服务概念的元素。可以看到领域驱动设计是围绕着领域模型进行设计，通过分层结构把每个领域独立出去。用实体、值对象以及领域服务来标识领域模型对象。这种严格的设计原则可以将业务逻辑约束在领域层内。

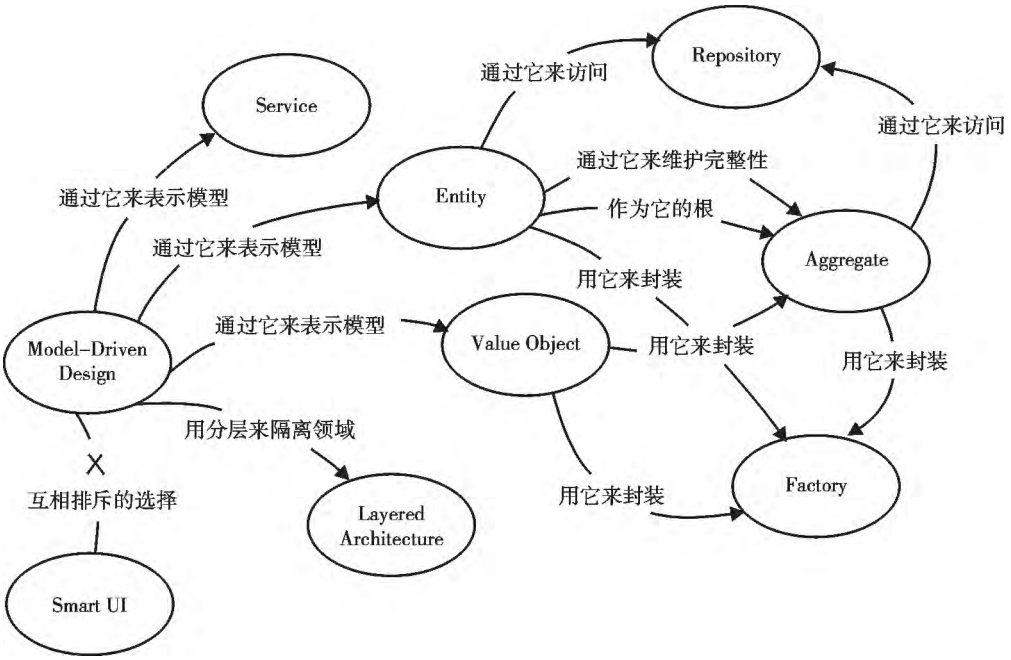

图 1　领域驱动的核心元素

Fig. 1　The core elements of domain driven design

1.2　本文用到的几个领域驱动设计的概念

1.2.1　统一语言（Ubiquitous Language）

统一语言可以帮助领域专家、开发团队、客户能在互相理解彼此的意图，在讨论时避免产生歧义。利用统一语言，抽象出领域专家与开发人员的共识，是后继沟通的基本准则。

1.2.2　领域事件（Domain Event）

领域事件泛指在我们所建模领域中发生过的事件[4]。严格来说，领域事件本身也属于统一语言的一个子集，在项目建模过程中，领域事件也可以成为项目组员的交流用语。

1.2.3　聚合（Aggregate）

聚合是由实体（Entity）组成的，实体是一个对象附带着唯一标识。例如在一个银行应用系统中，客户就是实体。实体中附带的唯一标识符可以是数据库中的主键，也可以是 UUID，这个标识符可以跨越多个子域，甚至在应用程序结束之后仍然有效，例如公民身份证号。聚合是一个或多个实体的一致性边界。一个聚合如果只包含一个实体，那这个实体称为此聚合的聚合根（Aggregate Root），每个聚合中只能有一个聚合根，非聚合根实体都被聚合根实体引用。

1.2.4 限界上下文(Bounded Context)

限界上下文是领域驱动设计中最重要的元素。上下文(Context)是领域业务目标语境,限界(Bounded)则是保护和隔离上下文的边界,避免业务目标的多样而带来的混乱和概念的不一致。限界上下文可以类比细胞膜,因为细胞膜定义了什么在细胞内,什么在细胞外,而且确定了哪些物质可以通过细胞膜。因此,团队组织中必须首先明确地定义模型所应用的上下文,标记出不同模型之间的边界和关系,通过团队的组织、软件系统的各个部分的用法以及物理表现(代码和数据库模式等)来设置模型的边界,在这些边界中严格保持模型的一致性,而不要受到边界之外问题的干扰和混淆。因此,限界上下文的理念与微服务的每个服务边界范围正好相符,这也正是领域驱动设计与微服务概念天生相契合的原因之一。

2 解决方案

事件风暴[5](Event Storming)是落实领域驱动的一种常用方法,使用事件风暴能够通过领域事件来识别出聚合根,组合的聚合根则又组成限界上下文,限界上下文则正是我们寻找的"微服务"的概念。本文将借助电子签章SaaS应用实例,介绍如何在业务中准确定义领域驱动设计的核心要素,继而找出微服务的近似界限。

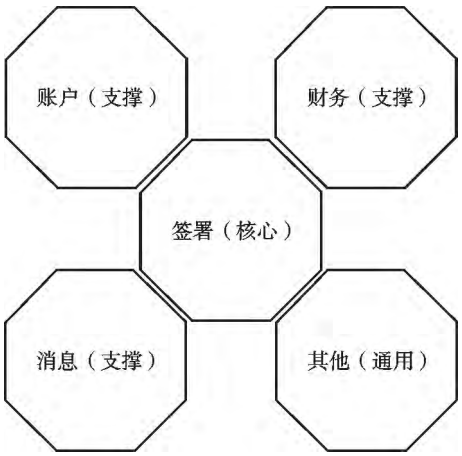

图2 电子签章应用领域草图

Fig. 2 The draft of E-signature application

图2是此电子签名SaaS应用的领域草图。可以看到,签署属于核心模块,关系图中的其他模块都是为了支撑此模块而存在,其主要负责系统的核心签署功能。账户也是业务的核心功能,其主要用于管理系统中的用户数据。财务模块主要负责用户消费行为的管理。消息为系统的整个系统推送模块。其他模块供将来扩展使用。

2.1 确定统一语言

在开发过程中,开发人员更注重从数据库、通信机制等技术的角度进行叙述,而精通业务的领域专家可能对此完全不了解。例如,在电子签章系统中,用户的一个订单对于开发人员可能就是数据库中一条记录,附带着一个UUID标识,对于领域专家而言,可能只是用户的一次购买行为。所以在筹划阶段,开发人员应该暂时屏蔽订单这一概念的技术属性,只保留其领域事件的属性。

2.2 确定领域事件

如图3所示,用户下订单,就是一个典型的领域事件。用户生成一个订单的消息可以被订单模块发向消息中间件,财务系统可以订阅它,也可以是感兴趣的第三方系统,这都会触发系统下游事件接收者的相应动作。但是需要注意的是,在此例中如果一条订单记录成功插入数据库则不属于领域事件,因为它不属于领域事件的生命周期,只是技术层面的行为。用户下单成功的领域事件将通过消息中间件发布出去,则不关心谁会收到消息,财务系统订阅了此消息则会进行相应的结算处理。领域驱动设计系统就是由这样一个个领域事件组成的,这样的消息驱动的系统更易于扩展,模块化的系统健壮性极强。

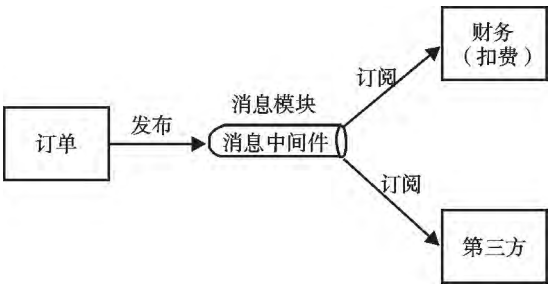

图3 领域事件之间的消息驱动

Fig. 3 The message drive between domain events

同时,订阅方服务要对接受到的消息做幂等(idempotent)处理,因为相同的消息可能会重复发送,就要保证多次收到相同的消息只需处理一次,再次收到就要忽略。

为了便于在后续的限界上下文中寻找共同点,要尽可能简明语言描述领域事件,在语法上采取动

宾形式。在描述时尽量避免使用诸如“管理”、“维护”等过于抽象用词,抽象用词容易使我们忽略隐藏的领域语言,缺少对领域的精准表达[6]。以电子签章应用为示例,图 4 是一个用领域事件驱动的完整的签署流程示例,图中方框中的便是领域事件,很难在流程图中体现的领域事件,比如“保存签名草稿”。

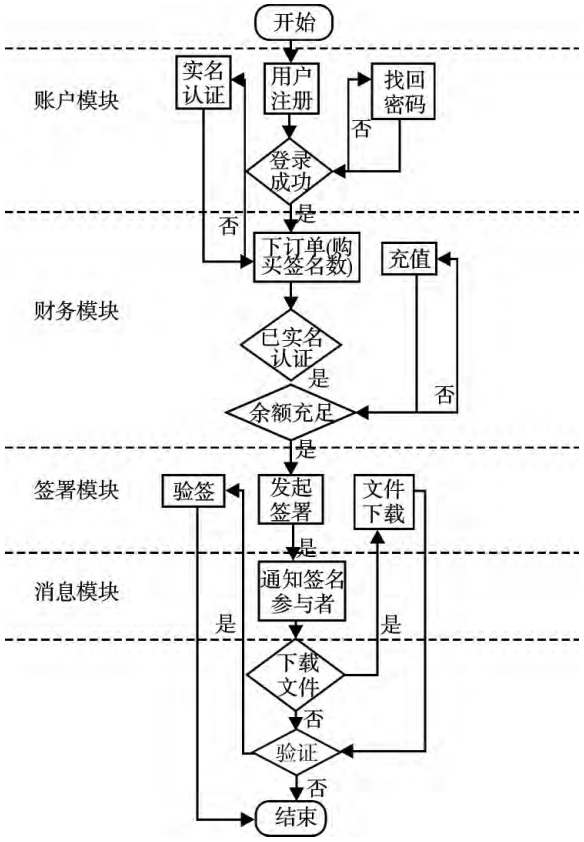

图 4　一个电子签章签署完整流程

Fig. 4　The complete process of E-signature

2.3　确定聚合

通常来说在确定聚合的时候,要以事务的边界为参考。如果用户下单购买了签名服务,则需要在财务模块中进行扣费,财务模块相关对象的属性也要随之改变。在传统单一架构应用中,我们通常使用一个事务保证数据的强一致性,但是对于天生分布式特性的微服务来说则行不通,除非使用重量级的 2PC 等分布式事务[7]方式,这种方式会造成服务之间的强耦合,也不符合微服务的思想,反而会生成一个“分布式单一架构”的畸形应用,所以一般不会考虑这种方式。对于领域驱动设计理念来说,聚合是数据一致性的保证,我们将强关联的对象封装到一个聚合中,由聚合根负责与外界沟通,在一个聚合中传统的数据库的事务将会保证数据一致性。

在上一节中我们利用动宾等语法定义了一众领域事件,我们从语义的角度去剖析领域活动的描述,如果语义相近,则可以归为一类,例如,账户注册,找回账户显然可以归为一类账户领域,可以考虑划为账户限界上下文,但是有时又会有两种描述语义相近,比如“注册账户”,“账户认证”都具有“账户”与“认证”的“语义”,认证行为是否要属于账户上下文,这时就要以哪个语义为标准就需要考虑每种语义相关性的耦合程度为划分标准。由此可以划分为同一个限界上下文中,但是同一个限界上下文中由于在不同的事务中,又需要划分为不同的聚合。如果将“认证”和“账户”划分为同一个上下文,这是就要把账户和认证分为两个聚合,账户的聚合根可以是每一个用户的唯一标识符,比如用户在数据库中的 UUID,也可以是用户的身份证号码。同样的,“认证”上下文也需要自己的聚合根。但是由于不在一个聚合中,由于领域驱动设计事务不能跨越多个聚合的原则,就需要借助消息中间件来传递消息,通过领域事件将各个模块最大程度解耦,达到基于事件的最终一致性(Eventually Consistency)。最终划分出来的聚合如表 1 所示。

表 1　模块内的聚合

Table 1 Aggregations in modules

	聚合	备注
账户	用户、认证	用于用户管理以及用户认证
签署	签名、验证签名、文件	用于签名服务,验证签名有效性以及签名文件下载
财务	订单、余额	管理订单,负责扣费
消息	系统推送、第三方服务	系统内消息推送,以及订阅第三方服务

2.4　确定限界上下文

具体划定边界的时候可以从 3 个方面入手: (1) 领域逻辑层面,它从领域模型的角度高度抽象,维护模型的一致性,降低系统的复杂度; (2) 团队合作层面,它限定了不同开发团队的的工作边界,避免团队的混乱沟通,从而降低系统的管理复杂度; (3) 技术实现层面,它限定了了系统架构的应用边界,保证系统和上下文领域层的各自一致性,从而降

低系统的技术复杂度。

以上3种划分方式对应着对不同层面的控制力。在我们划分服务的时候采用领域逻辑层面的方法,最终确定下来的限界上下文以及其中的聚合如图5所示,每个限界上下文可以作为我们的一个微服务边界。我们这一步具有里程碑意义,业务随后的一切扩展变化都将在此领域模型中迭代完善。

从图5可以看到,系统分为了5个限界上下文,在文件限界上下文分别有下载和版本两个聚合,负责不同的业务边界。在原来财务模块中,分化出订单以及余额两个聚合。账户模块分化出用户和认证两个聚合。核心子域签署则有签名和验签两个聚合。消息限界上下文有系统推送和第三方两个聚合构成。

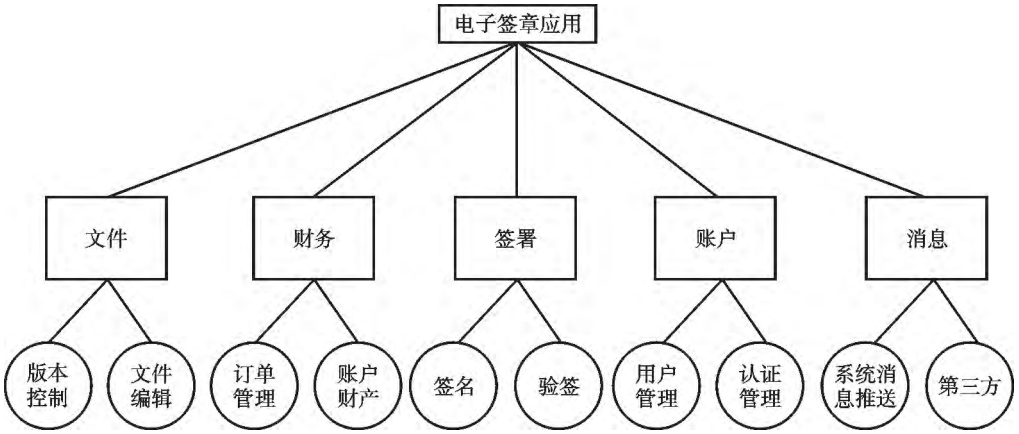

图5 电子签章的限界上下文与聚合

Fig. 5 E-signature applications bounded contexts and aggregations

对于第二种的团队合作层面,由于康威定律[8]的限制,组织和系统架构之间有一一映射的关系,如果两者不能对齐就会出现例如集中式和严格职能的组织,这样的组织结构都倾向于局部优化,无法形成有效的合作闭环。因此,结合领域设计思路,我们将领域的划分、微服务的划分和人员组织进行匹配,这也要求组织内的开发人员也要基于每个限界上下文的不同拆分分配部门。

第三种技术层面的划分要求每个服务要有单独的服务能力,要有独立的数据库。如图5,对于每一个模块都要考虑其数据的存储和部署,例如,对于消息服务模块而言,由于消息服务其存储内容的多变性[9],可以考虑MongoDB,Redis等NoSQL数据库。

经过上述3个步骤可以大致确定系统微服务的边界以及每个服务需要的开发人员配置,并且在业务变化中还可以进行拆分,达到解耦复用。

2.5 分析与比较

传统的微服务的划分方式是以模块来划分,数据库先行。根据数据库进行隐式建模,但是这种简单划分会造成潜在的服务间耦合,而且将来服务扩展会被先前的数据库设计限制。随着版本迭代,服务间耦合越来越严重,甚至在某一个服务调用失败的情况下,造成雪崩效应,导致应用的瘫痪。而采用本文提出的基于领域驱动设计的微服务划分方法,各个服务之间依赖领域事件的驱动完成通信,符合高内聚、低耦合的面向对象设计原则,将一个又一个的领域事件形成一个完整的业务闭环。同时,数据库详细设计可以在划分完毕的微服务模块和领域事件的基础上确定,整体上是一个可插拔的模块化设计,最大程度上规避了传统划分方式所带来的设计缺陷。通过对一个电子签章SaaS应用实例采用该方法,找出该SaaS应用微服务的近似界限,实现了服务的良好划分,提高了系统的弹性。

3 结束语

在将领域驱动设计的战略设计模式引入到架构过程中,会发现限界上下文不仅限于对领域模型的控制,而在于分离关注点之后使得整个上下文可以成为独立部署的设计单元,这就是“微服务”的概念。上下文映射的诸多模式则对应了微服务之间的协作,因此在战略设计阶段,微服务扩展了领域驱动设计的内容,反过来领域驱动设计又能够保证良好的微服务设计。

本文采用事件风暴的领域驱动设计原则实现微

服务应用的系统分析，根据领域事件的语义相关性判断，逐步划分出限界上下文模型，最后得到每个微服务的近似边界，给划分服务过程提供了科学依据，降低了微服务设计的门槛，提高开发效率，并且能适应需求的变化，服务具备高可扩展性。

目前，领域驱动设计概念还比较晦涩，学习曲线比较陡峭，对开发人员的平均水平要求较高，因此领域驱动设计应用在大型项目上更能发挥其威力，如何让其适用于小型应用团队还需要进一步研究。

参考文献

[1] JAMES J，MARTIN F. Microservice [EB/OL]. http://martinfowler.com/articles/microservices.html，2014.

[2] RICHARDS M. Microservicesvs service-oriented architecture [M]. O'ReillyMedia Inc，2015.

[3] EVANS E. Domain-driven design：tackling complexity in the heart ofsoftware [M]. Addison-Wesley，2003.

[4] VERNON V. Implementing domain-driven design [M]. Addison-Wesley，2013.

[5] Event-Storming [EB/OL]. https://techbeacon.com/introduction-event-storming-easy-way-achieve-domain-driven-design，2018.

[6] 张逸. 领域驱动战略设计实践 [EB/OL]. https://gitbook.cn/gitchat/column/5b3235082ab5224deb750e02，2018.

[7] 罗刚. 关于 2PC 协议对分布式数据库的事务恢复机制 [J]. 软件导刊，2012，11(9)：110－112.

[8] HERBSLEB J D，GRINTER R E. Architectures，coordination，and distance：Conway's law and beyond [J]. IEEE Software，1999，16(5)，63－70.

[9] 沈姝. NoSQL 数据库技术及其应用研究 [D]. 江苏南京：南京信息工程大学，2012.

(上接第 12 页)

[3] DONG X，GUO L，WEN C，et al. Mechanism of CO preferential oxidation catalyzed by Cu_nPt ($n=3-12$)：a DFT study [J]. Res. Chem. Intermed，2015，41(12)：10049－10066.

[4] 姜洪泉，王鹏，线恒泽，等. Zn^{2+} 掺杂 TiO_2 纳米粉体的结构特性及光催化活性 [J]. 哈尔滨工业大学学报，2007，(8)：1270－1275.

[5] 吕孝江，李工. 铜锌催化剂的活性及对甲醇吸附的研究 [J]. 哈尔滨师范大学自然科学学报，1992，(4)：60－64.

[6] 马洪运，范永生，王保国. 锌-空气电池电解液 Zn^{2+} 浓度对析氢过程的影响 [J]. 化工学报，2014，65(7)：2843－2848.

[7] 李芃. Cu 基混合过渡金属团簇及其吸附特性的第一性原理计算研究 [D]. 新疆乌鲁木齐：新疆大学，2016.

[8] TANG Q L，DUANG X X，LIU B，et al. A density functional study on properties of a Cu_3Zn material and CO adsorption onto its surfaces [J]. Appl. Surf. Sci，2016，363(17)：128－139.

[9] KHAN S，EISENBACH M. Density-functional Monte-Carlo simulation of CuZn order-disorder transition [J]. Phys. Rev. B，2015，93(2)：024203.

[10] RAM R S，JARMAN C N，BERNATH P F. Fourier transform emission spectroscopy of the copper dimer [J]. J. Mol. Spectrosc，1992，156(156)：468－486.

[11] 伏春平. Zn_n ($n=7\sim14$) 团簇稳定性和电子性质的第一性原理计算 [J]. 科教导刊，2015(11)：150－151.

[12] GUVELIOGLU G H，MA P P，HE X Y，et al. First principles studies on the growth of small Cu clusters and the dissociative chemisorption of H_2 [J]. Physl. Rev. B，2006，73(15)：155436(1－10).

[13] MASSOBRIO C，PASQUARELLO A，CAR R. Structural and electronic properties of small copper clusters：a first principles study [J]. Chem. Phys. Lett，1995，238(4－6)：215－221.