极客班第一期 C++专业期末综合测试题

学生编号：G2015010262

1. 设计一个多边形 Polygon，由若干个节点练成线。为其实现构造函数，拷贝构造函数，复制构造函数，析构函数

解答:源程序请参加 OOP_Exam1.cpp

2. 为 Polygon 类设计一个数据绑定机制。

解答:源程序请参加 OOP_Exam2.cpp

3. STL 与泛型编程 1。

解答:源程序请参加 STL_Exam1.cpp

4. STL 与泛型编程 2。

解答:源程序请参加 STL_Exam2.cpp

5. 算法与系统设计 1。

解答:源程序请参加 SystemAlgorithm_Exam01.cpp

6. 算法与系统设计 2。

解答:源程序请参加 SystemAlgorithm_Exam02.cpp

7. 算法与系统设计 3。

解答:源程序请参加 SystemAlgorithm_Exam03.cpp

8. 算法与系统设计 4: Design Shopping Cart

如何设计一个购物车，可以从商品列表中添加商品，修改数量，生成订单，如果商品数量在1亿以上，如何设计架构保证安全稳定的 (multi-tier, MVC, SOA)

解答:

1) 系统的整体框架

整套系统框架采用 Service Oriented Architecture(SOA) 和 Multi-Tier (MVC) 的组合架构，其基本构架如**Figure 1** 和 **Figure 2** 所示。

1) 基于SOA，应用程序，数据库以及系统分布式文件系统都在服务器端运行，当然它们分属于不同的服务器。
2) 基于Multi-Tier(MVC)概念。用户的浏览器作为MVC架构中的View，应用程序服务器是MVC架构中的Controller，而数据库和文件服务器则是MVC架构中的Model。
3) 应用程序服务器中的应用程序具体实现了购物车的功能，它可以使用基于NoSQL的内存型数据库来实现cache，比如开源的redis和mongodb。这两种NoSQL数据库都可以对数据进行持久化处理。
4) 数据库服务器存贮了所有商品以及相关信息。应用程序服务器可以读写数据库服务器的数据，而数据库服务器要负责保证数据同步。
5) 分布式文件系统服务器负责文件的管理，它也可以被应用程序服务器中的应用程序访问。

6) 为了增加整体框架的稳定性，还可以添加CDN服务，Rate Limit 以及 Load Balance负载均衡。

   a. CDN系统能够实时地根据网络流量和各节点的连接、负载状况以及到用户的距离和响应时间等综合信息将用户的请求重新导向离用户最近的服务节点上。其目的是使用户可就近取得所需内容，解决网络拥挤的状况，提高用户访问网站的响应速度。

   b. Rate Limit 用来限制网络流量。

   c. 使用 Load Balance 负载均衡 来在多个服务器中分配负载，共同完成任务，以达到最佳化资源使用、最小化响应时间、同时避免过载的目的。

更加详细和具体的系统框架可以进一步参见 **Figure 3**。

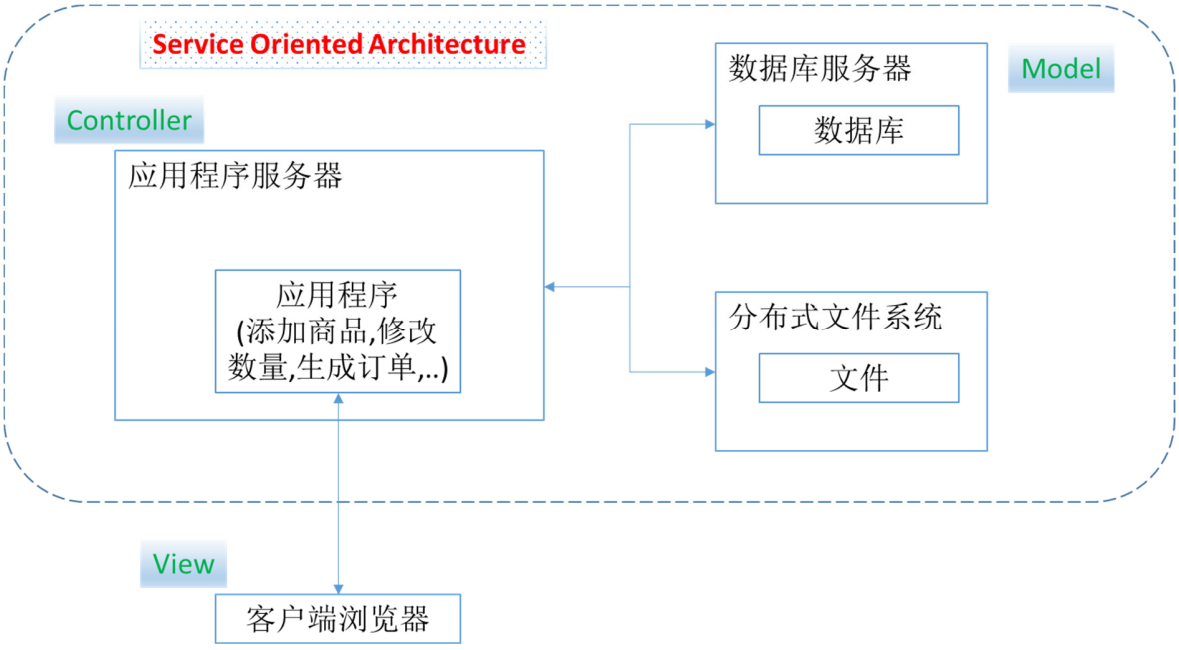

Figure 1 MVC 框架

Service Oriented Architecture + Multi-Tier (MVC)

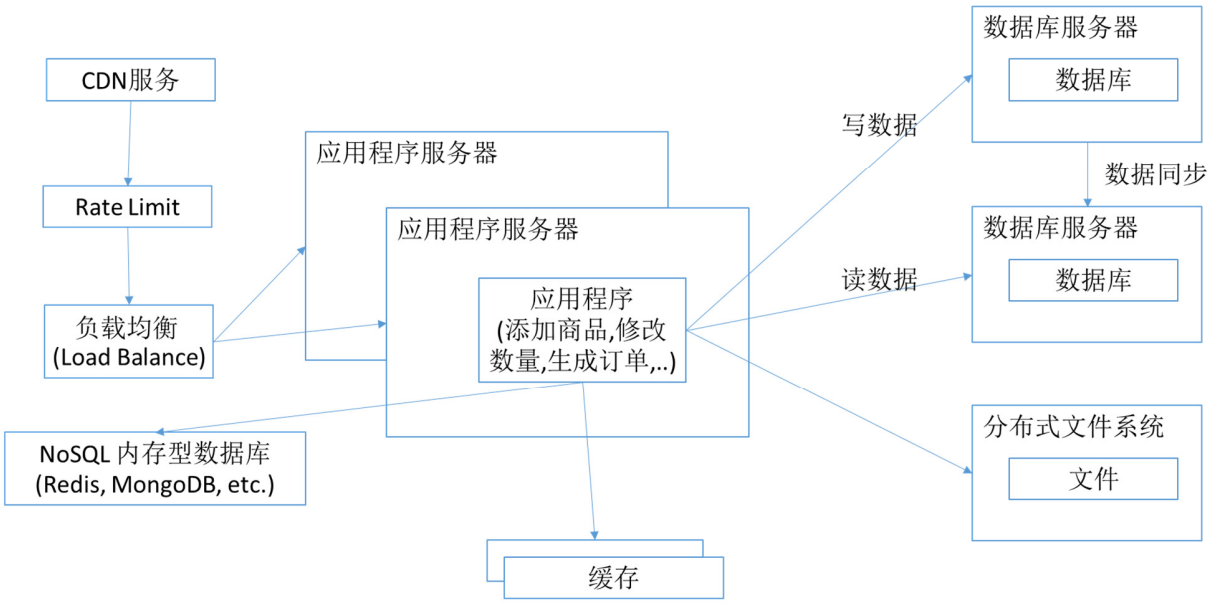

Figure 2 服务器端系统整体框架

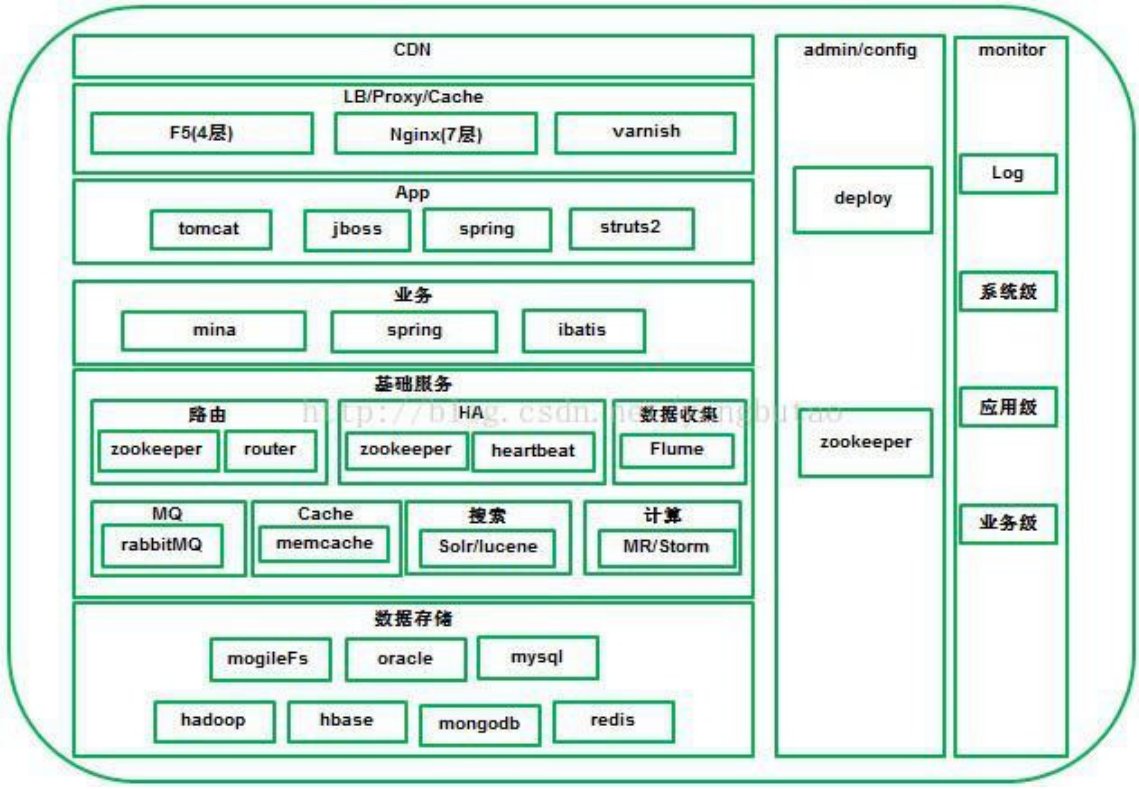

Figure 3 静态架构蓝图 [参考 http://blog.csdn.net/yangbutao]

2) 购物车在应用程序端的设计和实现

基本思想是使用Session实现“购物车”的功能。

Session方法是将数据存储在服务器端的Session中, 它存储特定用户会话所需的信息。Session对象是在每一位访问者从Web站点或Web应用程序中首次请求一个网页时创建的，它将保留到默认的期限结束或通过脚本设置中止的期限。这样，当用户在应用程序的Web页之间跳转时，存储在Session对象中的变量将不会丢失，而是在整个用户会话中一直存在下去。利用Session的功能，可以将购物信息（商品ID，购买数量，单价等等）存储到Session变量中。所以在此我们就可以将一个session 对象看作是一辆购物车， 不同的用户拥有各自的购物车。

Session 的好处是效率较高，而且也比在网页端使用Cookie安全，但它相对内存占用会较多，尤其是访问量比较大的网站. 所以我们会在必要的时候才使用Session。

(1) 只有在用户选择 "添加商品到购物车" 时才创建购物车对象。

(2) 将购物车信息临时存储到 Session 中有个好处就是，可以为没有登录的用户提供购物车服务。这样一来，只有在用户进行结算的时候才需要登录。

(3) 对于已经登录的用户，如果购物车不为空，我们会在 Session过期时将他的购物车数据存储到数据库中。这样用户下次进入网站的时候就可以继续上次挑选的商品进行结算。不过值得注意的是价格的变化，因为在实际销售中，商品的价格和日期有关，因此这种方式可能会带来价格上的混乱。当然还要为这些数据添加一个失效期，在必要的时候清除“历史数据”。

基本的购物车对象类可以用下面列出的 C# ASP.NET 程序来实现。

(1) 购物车ShoppingCart类

a. 存储一个商品列表。

b. 实现添加商品，修改商品数量，删除单件商品，计算总价等功能。

(2) 保存单件商品信息的CartItem类

a. 存储商品基本信息。

b. 实现判断购物车商品列表中是否已经存在相同商品的功能。

(3) 模拟数据库访问的Product类

```
//购物车类
//
using System.Collections.Generic;
using System.Web;

public class ShoppingCart {

  //购物车中的商品列表
  public List<CartItem> Items { get; private set; }

  // 单例实现
  public static readonly ShoppingCart Instance;

  static ShoppingCart() {
    if (HttpContext.Current.Session["ASPNETShoppingCart"] == null) {
      Instance = new ShoppingCart();
      Instance.Items = new List<CartItem>();
      HttpContext.Current.Session["ASPNETShoppingCart"] = Instance;
    } else {
      Instance = (ShoppingCart)HttpContext.Current.Session["ASPNETShoppingCart"];
    }
  }

  protected ShoppingCart() { }

  //添加商品
  public void AddItem(int productId) {
    CartItem newItem = new CartItem(productId);
   //如果已经存在，增加商品数量
    if (Items.Contains(newItem)) {
      foreach (CartItem item in Items) {
        if (item.Equals(newItem)) {
          item.Quantity++;
          return;
        }
      }
    } else {
      newItem.Quantity = 1;
      Items.Add(newItem);
    }
  }

//修改商品数量
  public void SetItemQuantity(int productId, int quantity) {
    // 如果数量是零，侧删除商品
```

```
      if (quantity == 0) {
         RemoveItem(productId);
         return;
      }

      // 查找商品并且修改数量
      CartItem updatedItem = new CartItem(productId);
       foreach (CartItem item in Items) {
         if (item.Equals(updatedItem)) {
            item.Quantity = quantity;
            return;
         }
      }
   }

//删除单件商品
   public void RemoveItem(int productId) {
      CartItem removedItem = new CartItem(productId);
      Items.Remove(removedItem);
   }

//计算总价
   public decimal GetSubTotal() {
      decimal subTotal = 0;
      foreach (CartItem item in Items)
         subTotal += item.TotalPrice;
      return subTotal;
   }

}
```

```
// 保存每一个单件商品信息的CartItem类
//
using System;
 public class CartItem : IEquatable<CartItem> {
   public int Quantity { get; set; }
   private int _productId;
   public int ProductId {
      get { return _productId; }
      set {
         _product = null;
         _productId = value;
      }
   }
   private Product _product = null;
```

```
    public Product Prod {
        get {
            if (_product == null) {
                _product = new Product(ProductId);
            }
            return _product;
        }
    }

    public string Description {
        get { return Prod.Description; }
    }

    public decimal UnitPrice {
        get { return Prod.Price; }
    }

    public decimal TotalPrice {
        get { return UnitPrice * Quantity; }
    }

    public CartItem(int productId) {
        this.ProductId = productId;
    }

   //用来判断购物车中是否已经存在该商品
    public bool Equals(CartItem item) {
        return item.ProductId == this.ProductId;
    }
}
```

```
//模拟从数据库获得商品信息的Product类
//
public class Product
{
    public int Id { get; set; }
    public decimal Price { get; set; }
    public string Description { get; set; }
    public Product(int id)
    {
        this.Id = id;
        switch (id) {
            case 1:
                this.Price = 19.99m;
                this.Description = "商品一";
```

```
            break;
        case 2:
            this.Price = 29.99m;
            this.Description = "商品二";
            break;
        case 3:
            this.Price = 9.99m;
            this.Description = "商品三";
            break;
        case 4:
            this.Price = 0.99m;
            this.Description = "商品四";
            break;
          // ……
      }
   }
}
```