问题1：

```
public void AsycStartDispatcher(Dispatcher dispatcher)
{
    var thread = new Thread(
                            (input) =>
                            {
                                var dispatch = input as Dispatcher;
                                if (dispatch != null) dispatch.Start();
                            });
    thread.Start(dispatcher);
}
```

请考虑上面的代码，可以看到我们写了这么多行代码，其实只是完成了一件很简单的事，那就是在一个单独的线程中执行dispatch.Start()方法；那么有没有更好的方法呢，可不可以像下面这样只用一行就完成这个功能呢?

```
public void AsycStartDispatcher(Dispatcher dispatcher)
{
    dispatcher.AsycExec(obj => obj.Start());
}
```

问题1解决方案：

如果我们利用Lambda表达式、扩展方法以及泛型来为对象添加一个方法就可以解决这个问题，具体代码如下：

```
public static class ObjectExtensions
{
    public static void AsycExec<T>(this T obj, Action<T> action)
    {
        action.BeginInvoke(obj, null, null);
    }
}
```

不过在这里我们没有考虑线程异常的处理。

问题2：

```
public void StartAllDispatcher(IEnumerable<Dispatcher> dispatchers)
{
    foreach (var dispatcher in dispatchers)
    {
        dispatcher.Start();
    }
```

```
    }

    public void AsycStartAllDispatcher(IEnumerable<Dispatcher> dispatchers)
    {
        foreach (var dispatcher in dispatchers)
        {
            var thread = new Thread(
                                (input)
                                =>
                                {
                                    var dispatch = input as Dispatcher;
                                    if (dispatch != null) dispatch.Start();
                                });
            thread.Start(dispatcher);
        }
    }
```

对于上面的代码我们能不能不用Foreach而是直接对整个Enumerable对象执行一个方法呢？就下下面这样：

```
    public void TestForeach(IEnumerable<Dispatcher> dispatchers)
    {
        dispatchers.ExecEach(obj => obj.Start());

        dispatchers.AsycExecEach(obj => obj.Start());
    }
```

问题2解决方案：

```
    public static class ObjectExtensions
    {
        public static void ExecEach<T>(this IEnumerable<T> objs, Action<T> action)
        {
            foreach (var obj in objs)
            {
                action(obj);
            }
        }

        public static void AsycExecEach<T>(this IEnumerable<T> objs, Action<T> action)
        {
            foreach (var obj in objs)
            {
                action.BeginInvoke(obj, null, null);
            }
        }
```

```
    }
```

问题3：

```
    lock (s_ObjectCachePools)
    {
        if (!s_ObjectCachePools.ContainsKey(poolName))
        {
            s_ObjectCachePools.Add(poolName, new List<Object>());
        }

        s_ObjectCachePools[poolName].Add(result);
    }
```

对于上面的s_ObjectCachePools对象，每次操作都要使用lock (s_ObjectCachePools)语句，我们能不能有另一种写法呢？就像下面这样：

```
    s_ObjectCachePools.LockExec(objs =>
                {
                    if (!objs.ContainsKey(poolName))
                    {
                        objs.Add(poolName, new List<Object>());
                    }

                    objs[poolName].Add(result);
                });
```

问题3解决方案：

```
    public static class ObjectExtensions
    {
        public static void LockExec<T>(this T obj, Action<T> action) where T : class
        {
            lock (obj)
            {
                action(obj);
            }
        }
    }
```

总的解决方案

```
    public static class ObjectExtensions
    {
        public static void Exec<T>(this T obj, Action<T> action)
```

```
        {
            action(obj);
        }

        public static void AsycExec<T>(this T obj, Action<T> action)
        {
            action.BeginInvoke(obj,null,null);
        }

        public static void ExecEach<T>(this IEnumerable<T> objs, Action<T> action)
        {
            foreach (var obj in objs)
            {
                action(obj);
            }
        }

        public static void AsycExecEach<T>(this IEnumerable<T> objs, Action<T> action)
        {
            foreach (var obj in objs)
            {
                action.BeginInvoke(obj,null,null);
            }
        }

        public static void LockExec<T>(this T obj, Action<T> action) where T:class
        {
            lock (obj)
            {
                action(obj);
            }
        }
    }
```