atitit 高扩展性解决方案.docx

[1.1.       人无远虑，必有近忧。       系统也一样。 1](#_Toc24521)

[1.2. 预留接口 2](#_Toc12536)

[1.3. 原件最小封装 2](#_Toc1251)

[1.4. 通用接口 多功能接口 就类似usb接口那样通用 2](#_Toc30188)

[1.5. 接口的层次化 功能 模块 子系统 2](#_Toc21219)

[1.6. Dsl接口和ast接口 2](#_Toc2175)

[1.7. 接口转换器 adapter模式 2](#_Toc11286)

[1.8. 无中心节点组件设计 避免庞大组件出现， 2](#_Toc13886)

[1.9. 使用配置化 ioc提升扩展性，配置lazy化 2](#_Toc12904)

[1.10. 脚本。脚本是扩展复杂功能的利器 2](#_Toc16824)

[1.11. 插件系统或者模块化平台。插件系统平台从理论上提供了无数的可扩展性。比如Eclipse和NetBeans平台。这儿是抽象的最高点， 3](#_Toc19792)

[1.12. 灵活性和可扩展性 其易用性差的三角关系 3](#_Toc28441)

[1.13. 稳定性。越是灵活和可扩展 3](#_Toc10365)

[1.14. 扩展性与性能的关系 3](#_Toc6852)

[1.15. Vm+dsl vm 其实就是添加了一个layer，无所不能 3](#_Toc14470)

[1.16. 功能使用和功能定义的关注点分离 4](#_Toc20909)

[1.17. 参考资料 4](#_Toc32353)

## 人无远虑，必有近忧。       系统也一样。

2）可扩展性不是一蹴而就的，是需要随着你对业务领域理解的深入而不断重构获得。一般三次可以达到比较理想的程度。

     3）在系统演化过程中，时刻准备着，保持对复杂性的关注。确保这些复杂性得到消化。

     4）要想获得可扩展性，你需要看得远一点，对上下文做充分的了解。

      人无远虑，必有近忧。       系统也一样。

## 预留接口

## 原件最小封装

## 通用接口 多功能接口 就类似usb接口那样通用

## 接口的层次化 功能 模块 子系统

分别在不同的层次上面的接口

## Dsl接口和ast接口

使用ast接口预先分词会更加方便。减少工作量。。

当然dsl接口也是急需要的额。。可以在ui层实现。。。

## 接口转换器 adapter模式

## 无中心节点组件设计 避免庞大组件出现，

类似于主板一样，接口众多，难依然难以升级

因为一旦升级，就要牵扯到众多接口同时升级。。

## 使用配置化 ioc提升扩展性，配置lazy化

## 脚本。脚本是扩展复杂功能的利器

## 插件系统或者模块化平台。插件系统平台从理论上提供了无数的可扩展性。比如Eclipse和NetBeans平台。这儿是抽象的最高点，

## 灵活性和可扩展性 其易用性差的三角关系

灵活性和可扩展性也是其易用性差

.易用性。越是灵活，用户的使用难度就越大，如果没有产品的现场实施人员的服务，对于非计算机人专业用户来说，这实际增加了他们使用的难度。Eclipse的平台正在展现这种问题，虽然它的用户主要是开发者，即时插件文档做得很好，但是很多人不得不抱怨Eclipse需要太多得配置和技巧。另外典型的例子是Windows和Linux要求有不同能力的人来使用，原因在于，Linux的灵活性和可扩展性也是其易用性差的原因之一

可以通过分装一层易用性layer来解决

## 稳定性。越是灵活和可扩展

， 。这主要是灵活性带来的现实世界的排列组合太多了，版本之间的兼容性

这个可以通过版本管理来实现，不会

## 扩展性与性能的关系

性能。微内核/插件模式设计的缺点表现在不同插件之间的交互性能损失上。Unix系统之所以没有采取现代操作系统所提倡的微内核模式，恐怕原因也在于此。

## Vm+dsl vm 其实就是添加了一个layer，无所不能

****基于中间语言。**** 框架通常定义了一些Hotspot（热点），在这些点上，可以进行扩展。平台，可扩展性是最强的，比如window 平台，你可以开发应用程序运行在上面。windows提供了几千个API，你可以使用他们来开发应用程序

。由此推断，从技术上来说，要获得最大的可扩展性，就要通过一种或多种中间语言来进行扩展。比如现在流行的OpenSocial API和Facebook都是这个思路。这样说来，DSL的出现就是顺理成章的了，它是为了满足在某个业务领域的扩展而设计的。思路还是和前面说的一样。 说白了，就是顺序，选择和循环可以表达所有逻辑，这是证明过的，这是语言能够带来灵活性的本质。

## 功能使用和功能定义的关注点分离

自行开发两个扩展，并注册到扩展点上。张三在运行时收集扩展，进行处理。实现了功能使用和功能定义的关注点分离。大家只要遵守扩展的契约就行了。

## 参考资料

程序的灵活性与可扩展性 - 嘿 - 博客频道 - CSDN.NET.html

可扩展性（Extensibility）——构建灵活系统的思考 - me.think(everything.about(\_software\_)).serialize(this);\_\_古路刀客 - ITeye技术网站.html