Research debug

jli

launch 一个program的时候可以选择mode（run debug profile,etc.）

如果你选择了在launch一个program的时候debug模式，那么你就需要实现Debug模型和ui能够让用户能和程序交互

所有的artifacts都是实现IDebugElement接口的

artifacts包括

IDebugTarget 可供调试的上下文，比如一个进程或者虚拟机

IExpression 一段可以被评估产生一个值的代码

IMemoryBlock 在一个执行的上下文中的一个内存的相邻段，一个memoryblock就是代表内存开始的地址和一个长度

IRegister 在一个寄存器组中的一个命名的变量

IRegisterGroup 分配给一个堆栈帧的一组寄存器

IStackFrame 在包括本地变量和参数的一个挂起的线程中的执行的上下文

IThread

IValue 一个变量的值

IVariable 值或者实在堆栈中的一个可见的数据结构

IWatchExpression 一个表达式，当指定的表达式产生值他会更新他的值

Actions

IDisconnect 结束和目标程序的debug会话，并允许目标继续运行

IStep 从当前执行点，step into、over、and return的能力

IStepFilters 调试的时候允许过滤分步实行操作

ISuspendResume 暂停和继续执行的能力

ITerminate 终止执行上下文的能力

IValueModification 定义一个变量值的能力

不同的debug elements实现不同debug actions

DebugEvent 是用来描述正在被调试的事件，debug事件包括和这个事件有关的debug element

［breakpoint］

允许用户在制定的地方暂停程序的执行。断点通常伴随这代码在UI中显示，当执行一个程序的时候遇到一个断点，程序暂停，并触发一个以BREAKPOINT为理由的SUSPEND的debug事件。

如果你需要在UI中显示断点，你可向IBreakpointManager 中添加 IBreakpointListener 一个BreakpointManager 可以从DebugPlugin中获得：

IbreakpointManager mgr = DebugPlugin.getDefault().getbreakpointManager();

BreakPoints 是通过resource markers来实现的，通过用maker来实现断点，调试模式可是使用现有的maker的功能，查找，添加，在editor中显示。

当你创建一个断点类型的时候，你必须也指定一个相关的maker类型，每一个org.eclipse.debug.core.breakpoints扩展点都要关联一个org.eclipse.core.resources.markers扩展点

debug插件定义了一种特殊的marker类型 org.eclipse.debug.core.breakpointMarker 当你为你的调试器定义一个breakpoint marker，你应该声明用这个marker来作为super类型。这样在一个源文件中，debug model就能找到所有这个marker的子类型。

［Expressions］

一个expression是一个代码的片段，可以被evaluated，提供一个value，表达式的内容取决于特殊的debug model。Expression manager（IExpressionManager）追踪所有工作空间中的表达式，