**Hasor 使用手册**

**手册版本：1.0.0**

**软件版本：0.0.1**

**作者：赵永春(zyc@hasor.net)**

**日期：2013-08-16**

目录

[第一章 介绍 4](#_Toc364703975)

[1.1 介绍 4](#_Toc364703976)

[1.2 概述 5](#_Toc364703977)

[1.3 开源协议 5](#_Toc364703978)

[1.4 内置组建及其授权协议 5](#_Toc364703979)

[1.5 获取和贡献 6](#_Toc364703980)

[1.6 类库引用 6](#_Toc364703981)

[第二章 快速上手 7](#_Toc364703982)

[2.1 创建项目 7](#_Toc364703983)

[2.2 启动Hasor 8](#_Toc364703984)

[2.3 第一个模块(HelloWord) 9](#_Toc364703985)

[2.4 Bean 9](#_Toc364703986)

[1)注册Bean： 9](#_Toc364703987)

[2)获取Bean 10](#_Toc364703988)

[3)单例Bean 10](#_Toc364703989)

[4)比较注册Bean非注册Bean 10](#_Toc364703990)

[2.5 Aop拦截器 10](#_Toc364703991)

[2.6 Ioc依赖注入 10](#_Toc364703992)

[2.7 事件监听器 10](#_Toc364703993)

[2.8 一个模块依赖另外一个模块 10](#_Toc364703994)

[2.9 使用配置文件 10](#_Toc364703995)

[2.10 创建Web项目 10](#_Toc364703996)

[2.11 Filter 10](#_Toc364703997)

[2.12 HttpServlet 10](#_Toc364703998)

[2.13 Session监听器 10](#_Toc364703999)

[2.14 截获服务器异常 11](#_Toc364704000)

[2.15 Action 11](#_Toc364704001)

[2.16 RESTful映射 11](#_Toc364704002)

[2.17 获取Request和Response 11](#_Toc364704003)

[2.18 返回Json数据 11](#_Toc364704004)

[2.19 Action结果处理 11](#_Toc364704005)

[2.21 把Web资源打入Jar包 11](#_Toc364704006)

[第三章 模块 11](#_Toc364704007)

[3.1 定义 11](#_Toc364704008)

[3.2 模块生命周期(HasorModule) 11](#_Toc364704009)

[3.3 模块运行状态(ModuleInfo) 11](#_Toc364704010)

[3.4 模块依赖(Dependency) 11](#_Toc364704011)

[3.5 一个简单的模块示例(Simple) 11](#_Toc364704012)

[3.6 复杂模块Demo演示 11](#_Toc364704013)

[第四章 环境支持 11](#_Toc364704014)

[4.1 环境(Context) 11](#_Toc364704015)

[4.1.1 StandardInitContext 11](#_Toc364704016)

[4.1.2 DefaultAppContext 11](#_Toc364704017)

[4.1.3 AnnoAppContext 11](#_Toc364704018)

[4.1.4 AnnoWebAppContext 11](#_Toc364704019)

[4.2 类路径扫描范围 11](#_Toc364704020)

[4.3 事件(EventManager) 11](#_Toc364704021)

[4.4 Guice与ApiBinder 11](#_Toc364704022)

[4.5 Bean服务(@Bean) 11](#_Toc364704023)

[4.6 启动和销毁 11](#_Toc364704024)

[第五章 配置文件 11](#_Toc364704025)

[5.1 简介 11](#_Toc364704026)

[5.2 独立性&随意性(Settings) 11](#_Toc364704027)

[5.3 主配置文件(hasor-config.xml) 11](#_Toc364704028)

[5.4 静态配置文件(static-config.xml) 11](#_Toc364704029)

[5.5 配置名称映射文件(config-mapping.properties) 11](#_Toc364704030)

[5.6 配置文件监听器 11](#_Toc364704031)

[5.7 命名空间(NameSpace) 11](#_Toc364704032)

[5.8 配置文件简单操作(Simple) 11](#_Toc364704033)

[5.9 自定义Xml配置文件解析器 11](#_Toc364704034)

[第六章 Web支持 11](#_Toc364704035)

[6.1 简介 11](#_Toc364704036)

[6.2 Web模块(WebApiBinder) 11](#_Toc364704037)

[6.3 Filter过滤器(@WebFilter) 12](#_Toc364704038)

[6.4 HttpServlet(@WebServlet) 12](#_Toc364704039)

[6.5 Session监听器(@WebSessionListener) 12](#_Toc364704040)

[6.6 Servlet异常截获(@WebError) 12](#_Toc364704041)

[第七章 架构及扩展 12](#_Toc364704042)

[7.1 概述 12](#_Toc364704043)

[7.2 Settings体系 12](#_Toc364704044)

[7.3 Module体系 12](#_Toc364704045)

[7.4 Context体系 12](#_Toc364704046)

[7.5 Binder概念 12](#_Toc364704047)

[第八章 核心技术 12](#_Toc364704048)

[8.1 12](#_Toc364704049)

[8.2 12](#_Toc364704050)

[第九章 Controller模块(Hasor-MVC) 12](#_Toc364704051)

[8.1 12](#_Toc364704052)

[第十章 Resource模块 (Hasor-MVC) 12](#_Toc364704053)

[9.1 12](#_Toc364704054)

# 介绍

## 1.1 介绍

Hasor是一款开源框架。它是为了解决企业模块化开发中复杂性而创建的。Hasor遵循简单的依赖、单一职责，在开发多模块企业项目中更加有调理。然而Hasor的用途不仅仅限于多模块项目开发。从简单性、松耦合性的角度而言，任何Java应用都可以从中受益。Hasor与Struts,Hibernate等单层框架不同，它可以提供一个以统一、高效的、友好的方式构造整个应用程序。并且可以将这些单层框架建立起一个连贯的体系，可以说Hasor是一个搭建开发环境的框架。这一点与Spring比较相似。Hasor目前包含8个可选的子模块。

特点：

* 清晰：在Hasor体系中每一个模块都被封装到一个jar或者classpath路径中。
* 简单：少量的代码开发关键的部件，Hasor在开发上提供了强有力的粘合作用。此外，由于Hasor对外开发都是以接口形式提供开发者避免接触到大量无用的API。
* 容器：Hasor包含并管理每个模块对象的配置和生命周期。
* 轻量：利用Guice3.0强大的DI支持使得Hasor的运行效率很高，另外Hasor的依赖也只有5个Jar。
* 友好：Hasor的所有功能仅在几个核心API接口和注解上实现.
* 兼容：对Web情况下进行了特殊制定，在开发Web项目时候可以得到更加友好的API支持。由于Hasor仅仅是一个轻量化容器，这又使得它可以很方便的和任何框架整合到一起。

设计思想：

模块：

Hasor-Core

核心包，所有Hasor模块都必须依赖它。对模块提供生命周期管理；以Settings接口形式提供配置文件获取服务；对主配置文件提供检测修改的支持；提供事件服务；提供环境变量操作接口；提供了Timer支持；提供了IoC/Aop。

Web支持下，可以注解声明Filter、HttpServlet、Session监听器、Aop拦截器；还提供了Servlet异常拦截器。通过Guice支持JSR-330。

Hasor-MVC

一个专门用于Web开发的模块，它提供了一个用于MVC模式下开发的请求控制器，通过它可以定义Action，并且可以将这个Action映射为RESTful，其API部分实现了JSR-311。

还有一个类路径资源装载器，用以加载位于Jar包中的资源。

## 1.2 概述

第一章是用以说明Hasor这个项目的功能、目的以及设计思想。同时说明了Hasor所使用的开源协议、类库以及如何贡献和索取Hasor代码。

第二章是引导读者使用Hasor进行项目开发，其中会包含丰富的Demo和API讲解。这一章节不会深入介绍Hasor部内部实现机制和原理。但是在第二章上会有许多知识点链接到第三章.

在第二章有兴趣的读者可以跟随链接深入了解Hasor内部实现机制和各部分功能原理。这些内容都在第二章以后的内容中出现。从第二章开始一直到第七章都是在详细讲解Hasor各个部件的功能和实现机制以及原理。

第八章之后会根据不用模块分别讲解各自模块下的功能。

## 1.3 开源协议

作为开源发布Hasor使用是Apache License 2.0协议。

## 1.4 内置组建及其授权协议

More

More是我在2008年之后构建的第一款开源框架，当时以失败告终。而后More的大部分代码都被拆除或者改造。目前保留下来的只有ClassCode、Xml以及一部分位于util包中的工具类。

目前util包中的工具类大部分也已被Apsche的commons-lang、commons-beans项目中的代码所替代。该项目受Apache License 2.0协议保护。

ASM 3.0

ASM是一款字节码框架，使用它可以动态的创建或修改java类文件。配合ClassLoader可以装载修改之后的类Hibernate、Spring都曾使用过它。该框架的部分完成代码位于org.more.asm软件包中，Hasor并没有使用到这个软件包。并且ClassCode组建作为More项目保留组建而存在。软件地址：(<http://www.objectweb.org/asm>)。

Ognl

Ognl是一款表达式解析引擎代码位于org.more.ognl软件包中，Hasor并没有使用到这个软件包。软件地址：(<http://commons.apache.org/proper/commons-ognl/>)

这部分内容是受Apache License 2.0协议保护。

apache-commons-lang 2.6、apache-commons-beans 1.7

org.more.convert软件包的内容是来源于apache-commons-beans-1.7，org.more.util的大部分代码是来源于apache-commons-lang-2.6。Hasor并不引用这两个软件包，但是由于org.more中包含了相关代码因此在这里需要加以说明并且列出其授权信息。这部分内容是受Apache License 2.0协议保护。

org.more.json软件包

该软件包是来源于org.eclipse.jetty.util\_8.1.3.v20120522.jar，位于Eclipse eclipse-sdk-4.2.2-win32软件中，属于Eclipse 4.2.2的一部分。Hasor-mvc项目依赖这部分功能实现Json数据转换。该代码受到受Apache License 2.0协议以及Eclipse Public License协议共同保护。

Apache License 2.0协议： (<http://www.apache.org/licenses/LICENSE-2.0>)

Eclipse Public License协议： (<http://www.eclipse.org/legal/epl-v10.html>)

ASM3.0协议： (<http://asm.ow2.org/license.html>)

## 1.5 获取和贡献

目前Hasor的代码托管于Github(<https://github.com/zycgit/hasor>)，您可以通过Git客户端获取到Hasor的最新代码。可以在(<http://msysgit.github.io/>)上获取到最新的Git客户端。当您对Hasor有一个更好的改进或想法，可以通过 ([zyc@hasor.net](mailto:zyc@hasor.net))邮箱联系到我也可以通过(<https://github.com/zycgit/hasor/pulls>)递交您的代码。

当您发现Hasor的漏洞和不足可以通过Email联系我。或者在OSChina上发表技术问答。Issues(<https://github.com/zycgit/hasor/issues>)也是一个发表问题的渠道。或者加入QQ群(293401803)我会为你解答疑问。

项目主页为：<http://www.hasor.net/> [尚在建设中...]

## 1.6 类库引用

依赖：Hasor依赖以下4个软件包：

slf4j-api-1.7.5.jar (负责输出Hasor产生的日志)

guice-3.0.jar (Guice3.0，负责提供DI相关的支持)

aopalliance-1.0.jar (Aop联盟API，Guice依赖)

javax.inject-1.jar (JSR-330标准，Guice依赖)

日志：如果使用log4j作为日志组建那么需要加入以下两个jar文件，log4j.xml是参考：

log4j-1.2.17.jar (log4j库文件)

slf4j-log4j12-1.7.2.jar (slf4j-log4j的适配器)

log4j.xml (参考的log4j配置文件)

# 快速上手

Hasor推荐你使用Eclipse Standard或者Eclipse IDE for Java EE Developers作为开发环境。前者是Eclipse的基本版，后者为Web开发版本。您也可以根据自己的喜好制定一款Eclipse开发环境。

Eclipse Standard 4.3下载地址：

(<http://eclipse.org/downloads/packages/eclipse-standard-43/keplerr>)

Eclipse IDE for Java Developers下载地址：

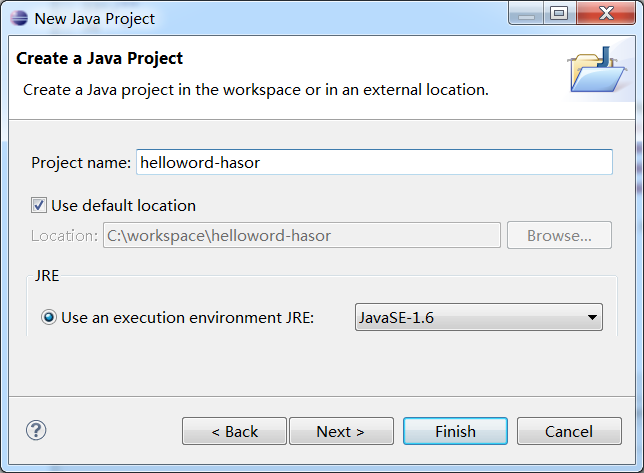
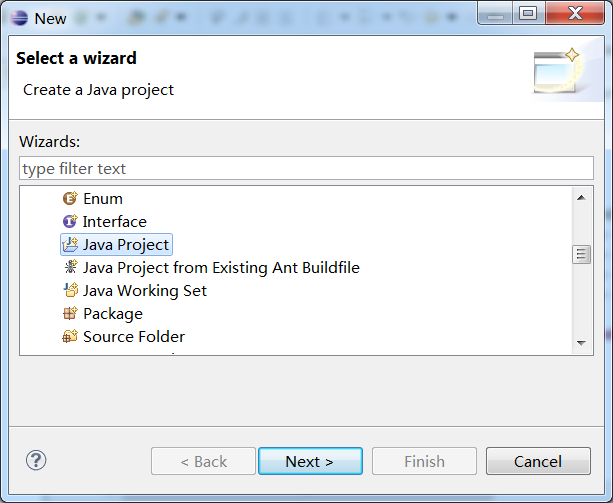
(http://eclipse.org/downloads/packages/eclipse-ide-java-developers/keplerr)

Eclipse IDE for Java EE Developers下载地址：

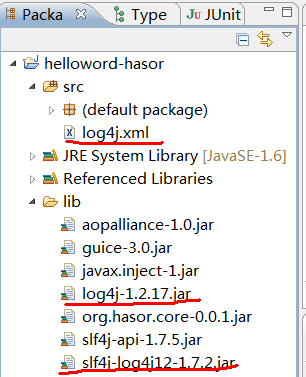
(<http://eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/keplerr>)

## 2.1 创建项目

Step 1：创建项目



Step 2：加入Hasor的Jar包，并引入类路径，图1。

图1： 图2：

Step 3：使用log4j处理Hasor输出日志，加入下面三个文件。如图2。

log4j.xml (参考的log4j配置文件)

log4j-1.2.17.jar (log4j库文件)

slf4j-log4j12-1.7.2.jar (slf4j-log4j的适配器)

## 2.2 启动Hasor

执行如下代码启动Hasor：

**import** java.io.IOException;

**import** org.hasor.context.anno.context.DefaultAppContext;

**public** **class** HelloHasor {

**public** **static** **void** main(String[] args) **throws** IOException {

//创建Hasor环境对象

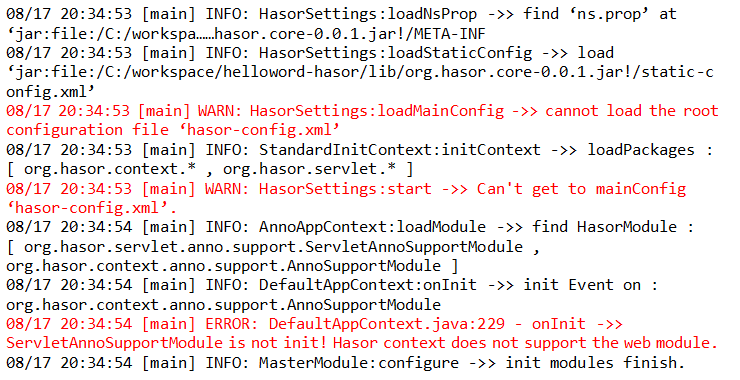
DefaultAppContext context = **new** DefaultAppContext();

context.start();//启动Hasor容器

}

}

在控制台中你应当能得到2个警告信息和1个错误信息，它们看上去应该是这样的：



*(上面日志信息略有删减)*

【解释】

第一条警告：是说，Hasor在启动时没有找到默认的主配置文件‘hasor-config.xml’。你可以不必关心它，**使用Hasor可以不定义任何配置文件**。这条警告消息仅仅是告诉开发者，如果你定义了配置文件Hasor并没有找到它。根据这条消息你可以判断Hasor是否加载了配置文件。[<第二章：快速上手(2.9节-使用配置文件)>](#_2.9__使用配置文件)、[<第五章：配置文件(5.3节-主配置文件)>](#_5.3__主配置文件(hasor-config.xml))

第二条警告：也是因为没有定义主配置文件引起的。Hasor会动态的监听配置文件变化，因此当没有主配置文件定义时候，负责监听主配置文件的监听程序会报告这一条警告消息。[<第五章：配置文件(5.6节-配置文件监听器)>](#_5.6__配置文件监听器)

第三条Error：由于我们的第一个例子并非是Web程序，因此Hasor内部的一个Web支持模块不满足启动条件。这对于控制台例子而言可以不必关心它。[<第六章：Web支持(6.1节-简介)>](#_5.6__配置文件监听器)

## 2.3 第一个模块(HelloWord)

下面定义了一个模块。当启动时Hasor会调用模块的init方法初始化方法打印一条消息。任何继承自AbstractHasorModule抽象类或者实现了HasorModule接口的类都可以作为Hasor的模块。[<第三章：模块(3.1节-定义)>](#_3.1__定义)

**public** **class** FirstModule **extends** AbstractHasorModule {

**public** **void** init(ApiBinder binder) {

System.*out*.println("this is first module.");

}

}

然后使用下面这段代码将模块添加到Hasor。注意：添加模块操作必须在start被调用之前，否则会引发异常。这是由于模块的添加要在初始化之前进行，start方法会引发初始化阶段。[<第三章：模块(3.2节-模块生命周期)>](#_模块)

DefaultAppContext context = **new** DefaultAppContext();

context.addModule(**new** FirstModule());//添加模块

context.start();/\*启动Hasor容器\*/

再次启动项目即可看到控制台打印的“this is first module.”

## 2.4 Bean

**public** **class** CustomBean {

**public** **void** foo() {

System.*out*.println("invoke CustomBean.foo");

}

}

### **1)注册Bean：**

在Hasor中任何类都可以被视为Bean。Hasor将其区分为注册Bean和非注册Bean。注册Bean不同于非注册Bean的最大特征是Bean是否具备名字。注册Bean可以具备多个名字，并且支持根据名字从Hasor中获取实例对象。下面是注册Bean的两种方式。

方式一：在模块的init阶段使用ApiBinder的方法注册Bean，并且命名为“myBean”。这种方法比较隐蔽不会受到classpath包扫描范围影响。[<第四章：环境支持(4.2节-类路径扫描范围)>](#_4.2__类路径扫描范围_1)

**public** **void** init(ApiBinder apiBinder) {

apiBinder.newBean("myBean").bindType(CustomBean.**class**);

}

方式二：最简单的注册方式！使用@Bean注解将Bean注册到Hasor容器中。需要注意的是使用@Bean注解时需要使用AnnoAppContext替代DefaultAppContext作为Hasor容器。AnnoAppContext会启用注解扫描。[<第四章：环境支持(4.1.3节- AnnoAppContext)>](#_4.1.3__AnnoAppContext)

@Bean("myBean")

**public** **class** CustomBean {

……

### **2)获取Bean**

对于注册到Hasor中的Bean可以使用下面这个方法获取Bean对象。

CustomBean bean = context.getBean("myBean");

对于没有注册到Hasor的Bean可以使用下面这个方法借助Hasor创建实例对象。

CustomBean bean = context.getInstance(CustomBean.**class**);

### **3)单例Bean**

代码形式：如果使用代码形式注册的Bean可以采用下面这个代码将其声明为单例。

apiBinder.newBean("myBean").bindType(CustomBean.**class**)

.asEagerSingleton();

注解形式：如果使用代码形式注册的Bean可以采用下面这个代码将其声明为单例。

**import** javax.inject.Singleton;//JSR-330，来源于javax.inject-1.jar

**import** org.hasor.context.anno.Bean;

@Bean("myBean")

@Singleton/\*声明单例\*/

**public** **class** CustomBean {

……

提示：以上两种方式可以重叠使用。

### **4)比较注册Bean非注册Bean**

下面是一些比较：

|  |  |  |
| --- | --- | --- |
| 项目 | 注册Bean | 非注册Bean |
| 具有名称 | 是 | 否 |
| 根据名称创建实例 | 是 | 否 |
| 根据类型创建实例 | 是 | 是 |
| Singleton | 是 | 是 |
| IoC/Aop | 是 | 是 |
| 被注册Bean注入 | 是 | 是 |
| 被非注册Bean注入 | 是 | 是 |

## 2.5 Aop拦截器

## 2.6 Ioc依赖注入

## 2.7 事件监听器

## 2.8 一个模块依赖另外一个模块

## 2.9 使用配置文件

## 2.10 创建Web项目

## 2.11 Filter

## 2.12 HttpServlet

## 2.13 Session监听器

## 2.14 截获服务器异常

## 2.15 Action

## 2.16 RESTful映射

## 2.17 获取Request和Response

## 2.18 返回Json数据

## 2.19 Action结果处理

## 2.21 把Web资源打入Jar包

# 模块

## 3.1 定义

## 3.2 模块生命周期(HasorModule)

## 3.3 模块运行状态(ModuleInfo)

## 3.4 模块依赖(Dependency)

## 3.5 一个简单的模块示例(Simple)

## 3.6 复杂模块Demo演示

# 环境支持

## 4.1 环境(Context)

## 4.1.1 StandardInitContext

## 4.1.2 DefaultAppContext

## 4.1.3 AnnoAppContext

## 4.1.4 AnnoWebAppContext

## 4.2 类路径扫描范围

## 4.3 事件(EventManager)

## 4.4 Guice与ApiBinder

## 4.5 Bean服务(@Bean)

## 4.6 启动和销毁

# 配置文件

## 5.1 简介

## 5.2 独立性&随意性(Settings)

## 5.3 主配置文件(hasor-config.xml)

## 5.4 静态配置文件(static-config.xml)

## 5.5 配置名称映射文件(config-mapping.properties)

## 5.6 配置文件监听器

## 5.7 命名空间(NameSpace)

## 5.8 配置文件简单操作(Simple)

## 5.9 自定义Xml配置文件解析器

# Web支持

## 6.1 简介

## 6.2 Web模块(WebApiBinder)

## 6.3 Filter过滤器(@WebFilter)

## 6.4 HttpServlet(@WebServlet)

## 6.5 Session监听器(@WebSessionListener)

## 6.6 Servlet异常截获(@WebError)

# 架构及扩展

## 7.1 概述

## 7.2 Settings体系

## 7.3 Module体系

## 7.4 Context体系

## 7.5 Binder概念

# 核心技术

## 8.1

## 8.2

# Controller模块(Hasor-MVC)

## 8.1

# Resource模块 (Hasor-MVC)

## 9.1

开发者可以摘除More代码使用Apsche的commons-lang、commons-beans软件包作为替换，但需要改写一些源程序。如果开发者这样做