Nutz 官方手册

1. 写在前面的话

1.1. 写在前面的话

1.1.1. Java 为什么不能是一门敏捷的语言呢？

起码 Java 是一门优美的编程语言，经过10多年的发展，它几乎已经被应用到了任何地方，在高端的企业服务器上，手持设备的芯片里，车载设备，智能家电甚至火星车上。在功能上，它广泛的涉足到了软件应用的各个领域，现在，它开始向桌面和图像处理方面频频发力。 从历史上看它是一门成熟的语言，从现在来看，它是世界上使用最广泛的语言，从将来看，它是最有前途的语言（现在它依然充满了活力和创新）。

但是，在越来越流行的 Web 开发领域， Java 似乎逐渐露出了疲态。是的，脚本语言们卷起了一场热潮，似乎已经快可以用“革命”两个字来形容了。而Java，被“脚本小子们”诟病最多的便是 **低下的开发效率**。

但是，真的是这样的吗？

从本质上来说， Java 语言本身为程序员提供的帮助只有两点，一是语言的语法，二是内置的类库。 现在从事 Web 开发，大多是采用第三方的类库（或者说是框架），比如流行的 SSH。 所谓 Java 在Web 开发的低效，不妨这样说比较贴切一些: **采用第三方类库进行开发比较低效**。

为了证明这一点，我写作了 Nutz， 它是一组轻便小型的框架的集合， 各个部分可以被独立使用。 而 Nutz 的目标就是

在力所能及的情况下，最大限度的提高Web开发人员的生产力。

是的，提高生产力是这个框架唯一的目标。 就像 Java 设计之初，考虑最多的是程序员的易用性和运行时效率的权衡， Nutz 也是这样。希望通过 Nutz， Java 的开发人员可以获得更快的开发速度，更少的代码量，并且这些以不损害运行时效率为前提。

1.1.2. Nutz 可以做什么？

* Dao -- 针对 JDBC 的薄封装，事务模板，无缓存
* Ioc -- JSON 风格的配置文件，声明时切片支持
* Mvc -- 注解风格的配置，内置多文件上传功能
* Json -- 解析和渲染
* Castors -- Java 对象类型转换
* Lang -- 更简洁的 Java 函数以及更丰富的反射支持
* Aop -- 轻便快速的切面编程支持
* Plugin -- 轻便的插件机制
* Resource -- 资源扫描

它所有的功能均不依赖第三方 jar 包

这就意味着：

* 如果一个 Web 应用，你在 WEB-INF/lib 下只 需要放置一个 nutz.jar 就够了
* 当然你要使用连接池，数据库驱动等功能，还需要自行添置 jar 包。

1.1.3. Nutz 为谁而设计？

* 如果你觉得 Hibernate 控制比较繁琐，iBatis 编写SQL又比较麻烦，Nutz.Dao 专为你设计。
* 如果你觉得在多个服务器部署或者修改 Spring 配置文件很麻烦，Nutz.Ioc 专为你设计
* 如果你觉得直接写 XML 配置文件很麻烦，可视化编辑器又没控制感，Nutz.Mvc 专为你设计
* 如果你觉得 JSON 转换很麻烦（要写超过一行以上的代码），Nutz.Json 专为你设计
* 如果你觉得 Java 语法不如 Ruby 便捷， Nutz.Castor 以及 Nutz.Lang 专为你设计
* 如果你以前根本没接触过 SSH ，只使用 JDBC 编程， 整个 **Nutz** 专门为你设计

1.1.4. Nutz 的质量

截至到现在为止，Nutz 的 JUnit 用例覆盖率大概是这样的

并且这个数字还在不断增加。

在一个功能告一段落以后，我通常会花1 - 2个晚上在一边咂着廉价的红酒一边颇有成就感的书写JUnit测试。通常我会用 JUnit 把我自己击溃，紧接着的那几天我都努力让那个该死红条变绿，之后，又想方设法写出新的Junit测试试图让它再度变红。并且我还要保证所做的修改不能让代码膨胀，这的确让我死掉了不少脑细胞。这些测试中，不仅涵盖各种功能上的测试，也涵盖了一些跨越线程的测试。在以后，我会针对代码执行的效率加入一些新的测试。

我能保证的就是Nutz的代码是小巧的，以及通过尽可能多的 JUnit 测试。但是所有我能做的也就只是这些了，找出剩下的那些代码上的缺陷，就应该交给 Nutz 第一批“小白”们了。“小白”们，加油！我支持你们...

小白： 是小白鼠的简称。小白鼠是实验室的最爱，实验室是产生新东西的地方。

1.1.5. Nutz 的未来

Nutz 是一个新兴的开源项目

它没有过去，只有未来

我可以保证，在未来，Nutz 的代码 **绝对不会膨胀**。 所有的功能设计的出发点就是最大限度给予程序员实惠。

1.1.6. 你完全可以让这个项目变得更加美好

* 不编写代码，你可以能为这个项目做很多事情 @hilliate
* 你可以随时 提交你发现的问题
* 任何人 用 任何方式 => 指出 Nutz 的缺陷，都是被欢迎的
* 如果你想成为 Nutz Committers 中的一员，请 阅读这里

1.1.7. 我使用的时候如果出现问题怎么办？

1. 加入 讨论区，讨论组一般总是有人在线，你的问题会很快得到响应
2. 加入 **GTalk 聊天群** -- 感谢 **Van** *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*
    * 或者访问 http://www.chatterous.com/nutzam/
3. 还有新浪微群
    1) Nutz 在微笑
4. 还有豆瓣小组
    1) Nutz 的微笑
5. 或者加入 QQ 群
    * Nutz & XBlink （58444676 超级群） -- 感谢 **E-Hunter** 的支持
    * Nutz在微笑 （60504323）超级群 -- 感谢 **milk cat!** 和 **yanick** 的支持
    * Nutz ② 群 （68428921）超级群 -- 感谢 **c.A** 的支持

考虑到，现在这个特殊时期，讨论区 即使用 https 也会不时被墙。所以你还可以给项目成员直接发信

* 一般情况下，你的问题都会得到答复
* 但是你必须知道，这是个开源项目，每个 Contributor 和 Committer 没有义务回答问题
* 所以，最好你能翻墙

2. 1.b.47 发行注记

2.1. 1.b.47 发行注记

2.1.1. 1.b.47 发行注记

前一版 1.b.46 的时候有一个小问题，就是 JSON 文件的解析更严格了，如果对象尾部多了一个逗号（","）（当然，JSON 语法上，这个的确是错的），JSON 解析会报错。详细请参看 Issue 349

考虑到大家都是粗心的人（多打一个逗号很平常），并且之前 Nutz 的 JSON 解析检查并没有这么严格，所以我们想最好还是紧急发布一版，这个版本和 1.b.46 区别就是能容忍更多一点 JSON 的语法错误。

这样，升级到最新的 **1.b.46** 的同学如果 JSON 解析遇到问题，愤然质问我们的时候，我们可以蛋定的答道： “同学，你 out 了，你应该用 1.b.47”

---------- Nutz 的下载地址的分隔线 ----------

* 稳定版下载地址 : http://code.google.com/p/nutz/downloads/list
* 日编译下载地址 : http://build.sunfarms.net/nutz/
* Nutz 的主页： http://nutzam.com

2.1.2. 问题修复

* Issue 348 当一个IocBean的字段注入失败时, 第二次获取这个bean,会得到一个不完整的对象(部分字段为注入) by **wendal**
* Issue 349 1.b.46 加载json配置文件时尾部多个“,”会出错 by **zwt**

2.1.3. 质量

共通过了 826 个单元测试用例,代码覆盖率达到 70%(按line计算)

Nutz.Dao 经测试在如下数据库上可以工作正常

* H2
* SQLite -- 仅有限支持事务操作
* hsqldb
* MySql
* Oracle
* Postgresql
* SqlServer2005

* SqlServer2000
* DB2
* Derby

2.1.4. 文档

-- 木有任何改动 --

2.1.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 NutzDemo 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献列表，我已经写了一个小程序，根据 Issue 列表来自动统计...

贡献者	问题	博客	支持	代码	示例	文档	测试
wendal	O	O	O	O	O	-	O
zwt	O	-	-	-	-	-	-

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz & XBlink （58444676 超级群）
* Nutz在微笑 （60504323）超级群
* Nutz ② 群 （68428921）超级群
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*
* Nutz的新浪微群
* Nutz的豆瓣小组
* Nutz的聊天室

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^!

欢迎访问官网,以获取 最新的快照版 和用户手册

3. 关于源码

3.1. 从 SVN 编译 - Eclipse

3.1.1. 遗留文档

这文档已经过时,因为Nutz的代码已经托管在github. 请使用从 Git编译源码

3.2. 从 Git 编译源码

3.2.1. 我们转移到了 Git

现在 (Nutz-1.b.38) 之后的版本，源码的即时更新都会在 Github 上进行。当然，我们起码会在每次发布前更新一下 Google Code 的 svn 服务，满足那些 svn 用户的需求。但是，我们还是得说， **Nutz 最鲜活的更新**，只有从 Github 上才能看的到哦。

如何使用 Git，什么是 Git？

如果你问出上面的问题，建议你 Google 一下，基本上各种介绍满天飞，如果你想尝试装个玩玩，我觉得这篇文章 还不错。

如果你已经是 Github 的用户（是的，为什么不是呢？）请把眼睛凑近屏幕一些，再近一些，再近一些，我必须告诉你：

喜欢 Nutz ，就 Fork 它 ^_^

然后把你认为得意的修改给我们发个 pull request，我们很乐意看到你的杰作

当然，我个人认为 Github 的界面貌似华丽，但是很多细节设计的很脑残，但是只要你是它的用户，只要你稍微有点耐心，其实它比 Google Code 要好玩的多

3.2.2. 从 Github 获取Nutz

简单的要命，执行下面的命令（前提是你装了 Git）

```
[CODE]
cd xx/xxx/xx/xx    <- 意思是，到你打算放 Nutz 的那个目录
git clone git://github.com/nutzam/nutz.git --depth=0
```

稍微等个1分钟不到，因为 github 线路问题，有点慢，原因你懂的 ...

屏幕上开始显示...

```
[CODE]
Cloning into nutz...
remote: Counting objects: 26249, done.
remote: Compressing objects: 100% (6919/6919), done.
Receiving objects:  38% (10050/26249), 8.46 MiB | 306 KiB/s
```

耐心等待，直到 ...

```
[CODE]
Cloning into nutz...
remote: Counting objects: 26249, done.
remote: Compressing objects: 100% (6919/6919), done.
remote: Total 26249 (delta 16407), reused 26160 (delta 16319)
Receiving objects: 100% (26249/26249), 28.10 MiB | 275 KiB/s, done.
Resolving deltas: 100% (16407/16407), done.
```

恭喜你，你拿到了 Nutz 最新的代码。

PS：强烈建议第一次通过Git取得代码之后，运行git gc命令以减小磁盘占用空间

```
[CODE]
cd nutz
git gc
```

而且，Git 一个好处就是，每个目录下都没有万恶的 .svn 目录，只是在项目的根目录下有个 .git 目录。这让我觉得整个世界清爽了许多 ^_^

3.2.3. 编译 Nutz

从 SVN 编译 这篇文章的后一半给出详细的图文介绍，告诉大家如何在 Eclipse 上搭建一个Nutz 的编译环境。至于 Git 用户，鉴于你们大多是骨灰级玩家，随便编译个项目对大家来说是毛毛雨，小意思。但是，这里还是列一下 Nutz 编译的必要条件，无论你是用 NetBean 还是 Eclipse 还是 JDK 手动编译，都会有点帮助：

Nutz 编译依赖如下 jar:

* Log4j-1.2.15 以上版本： 仅编译时需要，运行时不需要
* servlet-api.jar： 编译时需要，运行时，如果用 Nutz.Mvc 就需要，随便找一个 JSP/Servlet 容器，里面就会给你这个 jar, 比如 TOMCAT

如果你想运行 Nutz 的单元测试，你需要依赖更多的 Jar

* 一个数据库驱动，比如我用的 mysql-connector-java-5.1.7-bin.jar
* 一个数据库连接池，比如我用的 commons-dbcp-1.4.jar + commons-pool-1.5.4.jar
    > 更多关于数据库连接池的介绍请看 如何创建 DataSource
* 一个特殊的数据库链接池 bonecp-0.7.0.jar， 因为我们为其专门写了测试用例
* 于是你还需要
    > slf4j-api-1.6.1.jar
    > slf4j-simple-1.6.1.jar
    > google-collect-1.0.jar

上面这些 jar 是运行 Nutz 的单元测试才需要的，你如果不打算运行 Nutz 的单元测试，你不一定需要它们。关于 Nutz 的单元测试，你还需要看这篇文章：运行 JUnit 测试，它告诉如何修改一个配置文件就能成功运行 Nutz 的单元测试.

3.3. 运行 JUnit 测试 - Eclipse

3.3.1. 搭建 Eclipse 编译环境

如果你对于 Eclipse 以及 SVN 不是特别熟悉，请参看 从 SVN 编译。如果你认为自己有足够的经验，你需要知道一下几点就能顺利编译：

1. SVN 的 check out 信息在 http://code.google.com/p/nutz/source/checkout
    * 如果你只想阅读最新代码，从这里检出：http://nutz.googlecode.com/svn/trunk
    * 如果你想提交代码，从这里检出： https://nutz.googlecode.com/svn/trunk （当然你得有提交权限）
2. 项目编译依赖
    * log4j 用来编译 Nutz.log，在运行时，则不是必须的。关于 Nutz.log 更多介绍，请参看这里
    * commons-dbcp 和 commons-pool 是运行 Junit 测试所必须的。并且你还需要一个 JDBC 驱动
    * Servlet API 的支持
    * JUnit 4
3. 在编译前，请确定你的项目是用 UTF-8 方式编码的。因为我的源码以及文档用的都是 UTF-8 格式的文件

你可以参看 从 SVN 编译，这是我们推荐的项目构建和编译方式。

3.3.2. 如何运行 JUnit 测试

那么如何运行 JUnit 测试呢？

3.3.2.1. 1. 创建测试数据库

首先你需要建立一个测试数据库，比如，我们叫 zzhtest. 建议不要使用test.

```
[CODE]
mysql:> create database zzhtest;
```

3.3.2.2. 2. 创建连接配置文件

在 Eclipse 项目里，增加一个新的 source folder ：右击项目 > New > Source Folder

1. 随便给这个 Folder 起个名字，比如 **properties**
2. 将这个目录设置成为 Source Folder
3. 然后在里面创建一个文本文件 **nutz-test.properties**
    * **请注意**，一定要叫这个名字，否则运行测试一定是不过的，详细原因请看 /test/org/nutz/Nutzs.java 的源代码

这个文件的正文为：

```
[CODE]
driver=com.mysql.jdbc.Driver
url=jdbc:mysql://127.0.0.1:3306/zzhtest
username=root
password=123456
```

一共四行。当然，不用我说，大家一定知道怎么修改吧,建立好文件的项目结构为：

请注意，= 前后不要有空格

我之所以这样做而不直接把 nutz-test.properties 文件放到 src 或者 test 下面，是因为，这个文件通常在不同的机器上时不一样的，我家，公司，以及笔记本上的 nutz-properties 都不一样。所以，不宜将这个文件放到 svn 上。

3.3.2.3. 3. 增加 log4j.properties

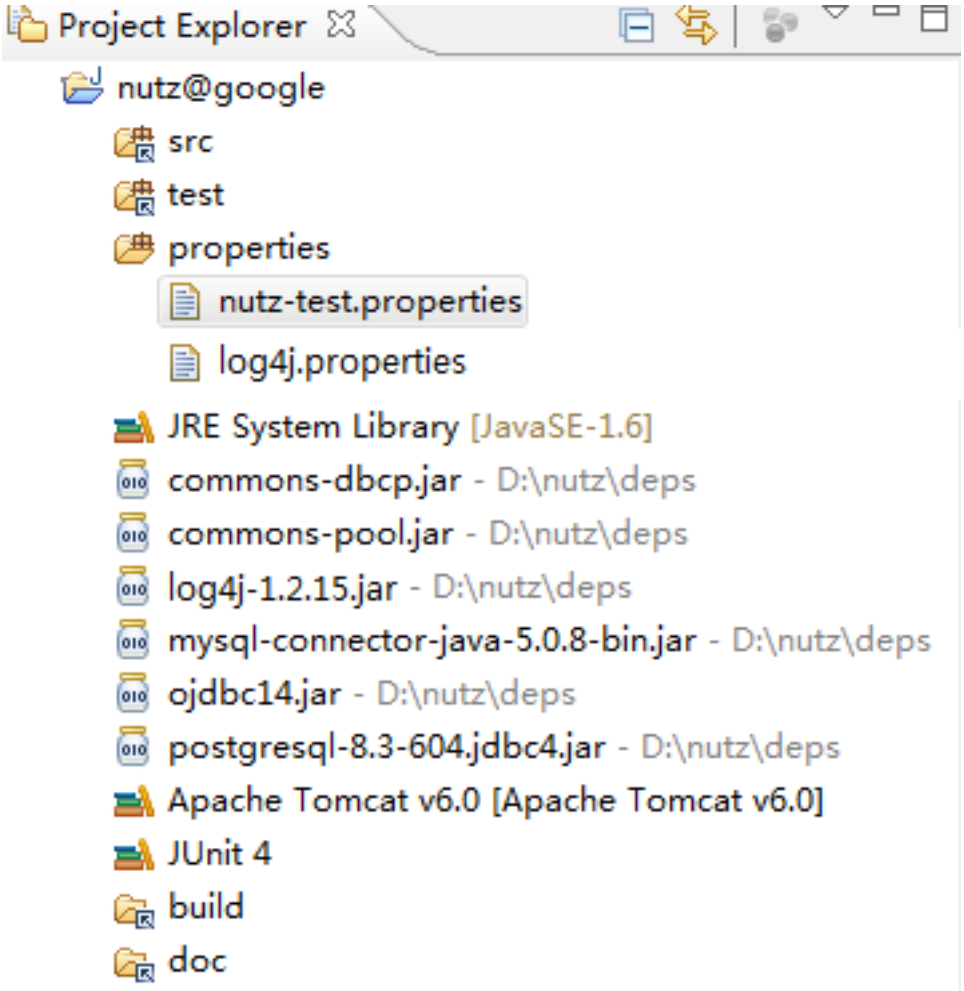
这步，是为了能让你在控制台上打印 Nutz 内部的运行信息：

在刚才创建的 Source Folder 下创建一个 log4j.properties 文件，正文为：

```
[CODE]
log4j.category.org.nutz=debug, NUTZ
log4j.additivity.org.nutz=false
#Appenders ...
log4j.appender.NUTZ=org.apache.log4j.ConsoleAppender
log4j.appender.NUTZ.layout=org.apache.log4j.PatternLayout
log4j.appender.NUTZ.layout.ConversionPattern=%d [%t] %-5p %c - %m%n
```

3.3.2.4. 4. 检查

你的项目应该看起来是这个样子：



3.3.2.5. 5. 增加 Jetty 的依赖

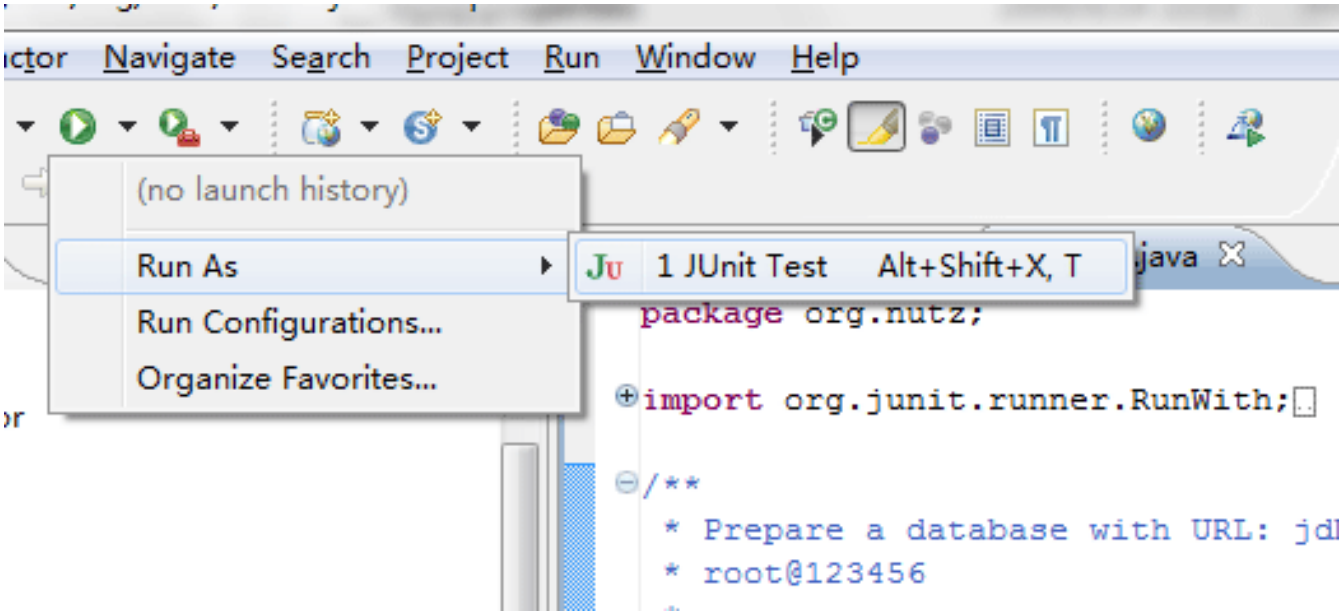
因为 Nutz 的 Mvc 自动测试，适用嵌入式Servlet容器 Jetty，因此你需要下一个最新的版本并加入你的 Eclipse 工程的类路径中

我用的是 'jetty-all-7.4.1.v20110513.jar' 这个版本，我想只要你的版本比我这个新，应该都能兼容。对于不熟悉 Jetty 的朋友，一开始，能找它的下载地址都要费半天劲，让我替你省点事，直接从这个地址:http://repo2.maven.org/maven2/org/eclipse/jetty/aggregate/jetty-all/ 下载就好了。

另外请注意: - 可能 Jetty 同 Tomcat 有点冲突，你把它们一起放在你的工程里是不明智的，前面的文章曾提示你将Tomcat 加入你的类路径，仅仅是为了编译，因为 Nutz 需要 Tomcat 中的 servlet-api.jar， 如果你现在下了 Jetty，你就不需要Tomcat 了，再把 Jetty 给你提供的，或者随便谁提供的 servlet-api.jar 加入你的类路径就好了。

3.3.2.6. 6. 运行

之后，打开 test/org/nutz/TestAll.java ，然后用 JUnit 运行

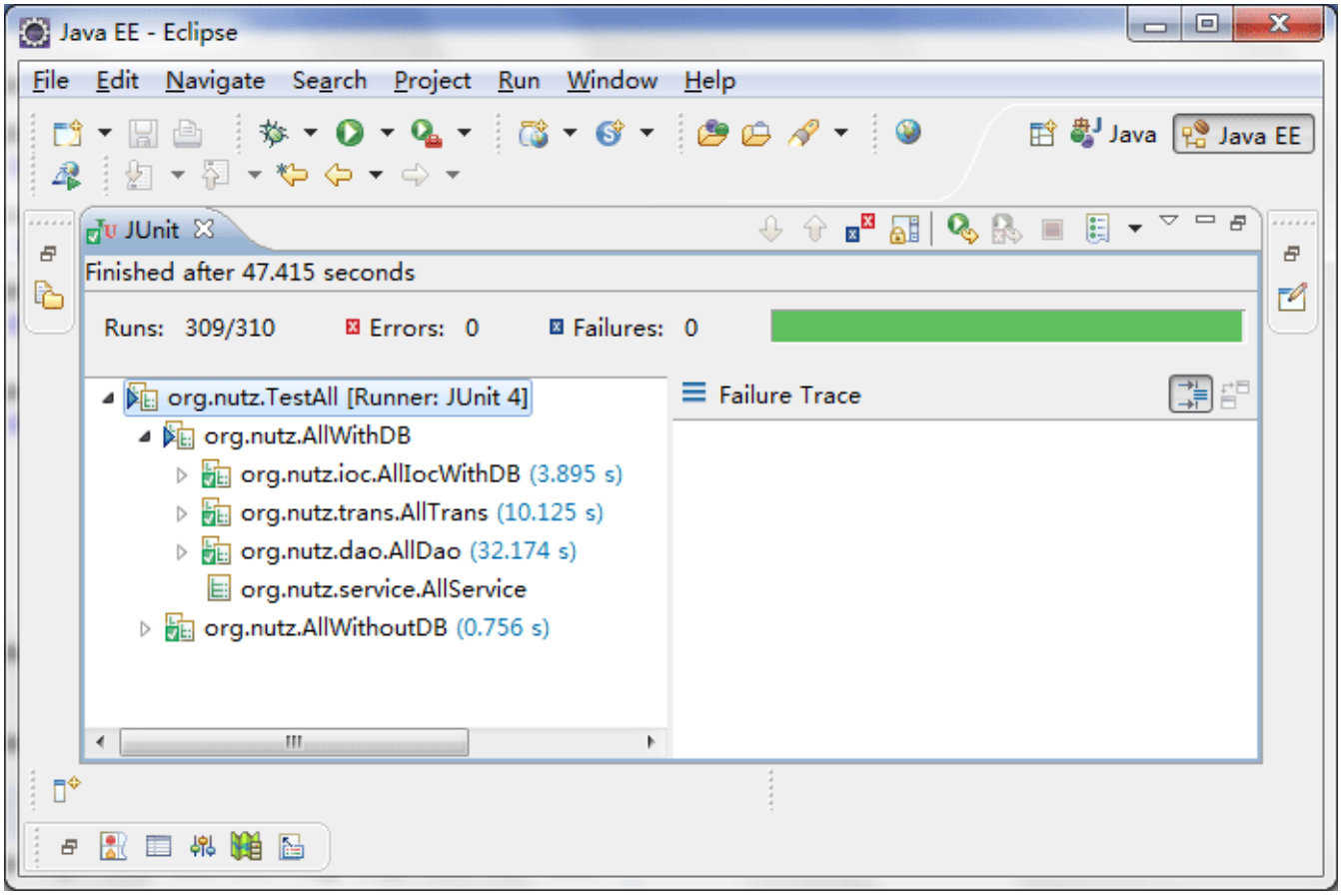


3.3.2.7. 7. 查看结果

如果上述操作没出什么岔子，那么你会看到正确的运行结果：

3.3.3. 最后 ...

* Nutz 是开源项目，你可以修改源代码，并运行JUnit 测试，来验证你的修改会不会导致错误
* 你还可以增加 JUnit 测试，来看看 Nutz 的代码品质到底如何
* 当然，如果你写的 JUnit 测出了 Nutz 的 bug，或者修正了Nutz代码的缺陷， 请发信到这个地址 nutzam@googlegroups.com ,并附上你的代码。
* 我们会非常感谢你的付出。如果你提出的修改超过 5 次（包含）被接受，你会收到一封邀请信，邀请你成为 Nutz 的 Commiter



* 关于如何成为 Nutz 的 Commiter，请 参看这里

还有：

* 在 build 目录下，有 build.xml， 你可以用 Ant 编译
* 如果你建立一个基于 Nutz 的项目，将 Nutz 的 jar 包 加入到项目的build path 即可

3.4. 使用 Ant 编译

3.4.1. 简单介绍

如果你不知道怎么从 svn 服务器下载 Nutz 的源代码，请看从 SVN 编译 一节

编译通常是个麻烦的问题，依赖关系啊，环境变量啊，等等，全是些没有技术含量的东东，但是却能搞的你非常头疼。虽然编译 Nutz 是一个相对简单的工作 -- 它编译的时候不需要依赖第三方 jar 包，但是还是有一些工作要做的。

从我个人的情况来看，我经常在如下三个环境下进行 Nutz 的开发：

* 公司 - Vista
* 家 - Ubuntu
* 笔记本 - Win XP

我这三台机器由于安装的时候不同，加上我是一个比较随意的人，所有依赖库的位置均有所不同，但是我配置编译环境基本上能在5分钟搞定。写这个介绍的目的就是分享我的一些经验，帮助大家远离痛苦，选择欢乐。

Nutz 提供 Ant 编译的脚本，在 Ant 运行前，你需要配置几个环境变量，在你的命令行客户端输入 ant，查看一下你的 ant 安装没有，如果没有请先从它的官方网站下载。你还需要检查：

1. Java 是否安装，Nutz 需要至少 1.5 以上版本
2. Tomcat（Nutz 需要它的 servletAPI），
3. Log4j 在编译时需要它。 当然，运行时，我们不一定需要 Log4j

有了这些我们就可以顺利编译 Nutz 了。

我们提供了两个 ant 脚本：

* build.xml - 快速编译，只编译 nutz.jar
* build2.xml - 全面的编译 nutz 的 jar，文档，并运行单元测试，等等

3.4.2. build.xml - 快速编译脚本

这个是我在开发中最常用到的编译脚本，我其他的基于 Nutz 的项目在，在 Nutz 代码更新时，我就运行一下，其他的项目一刷新，就升级到这个 Jar 文件了。

在 /trunk/build 目录下你能看到 build.bat（写给 windows的）以及 build（写给 Linux）的。你可以将其拷贝到你的 PATH 目录下，以便你在控制台执行。执行前，请先打开脚本文件，修改其中的内容。因为 build.xml 需要如下环境变量：

* **TOMCAT_HOME** : 你的 tomcat 安装目录，这目录下应该有 lib 目录，下面包括了 servlet-api.jar
* **JAVA_HOME** : 你的 JDK 安装目录，这目录下应该有 jre/lib/rt.jar
* **DEPS** : 第三方 jar 包存放的目录，这个目录下应该有 log4j.jar
* **OUTPUT** : 成功编译后 nutz 的 jar 包的输出目录
* **PROJECT_HOME** : Nutz 的项目目录，这个目录下应该有 src 目录

正确的设置这些环境变量后，运行这个脚本，Nutz 的 jar 就会出现在你的输出目录中

3.4.3. build2.xml - 全面编译脚本

每次发布前，我们都会运行一下这个脚本，因为它要运行单元测试，所以你需要更多的依赖包

这个脚本现在只是用作内部使用，Wendal 在里面做了详细的描述，根据我的经验，只要你把需要的 jar 包都放在依赖目录下，它会正常运行的。

3.5. Nutz 的包结构

3.5.1. org.nutz.*

包名	描述
aop	拦截器框架
castors	帮助你把任何类型转换成任何类型，从而让 Java 的类型不是那么的"强"
dao	数据库访问接口，更薄，更简单轻便
el	嵌入式表达式引擎
filepool	文件池接口以及默认实现
http	简单的Http客户端工具类，帮助你发送 HTTP 请求
ioc	Ioc 容器，你懂得
json	JSON 的解析器和渲染器
lang	让 Java 的语法更加友好，这个包，是 Nutz 最基础的包
log	轻便的 Log 打印的功能
mvc	基于注解的 Mvc 框架，可以和Ioc容器很好的结合使用
service	数据库业务逻辑访问接口，它提供了几个很基础的类，你可以扩展它们，成为你自己的 CRUD
trans	提供了事务能力，它让 dao 包里的类不用关心事务，你如果需要事务，就用找个包里的 Trans.run 就好了
plugin	轻便的插件机制，我们提供了所谓"热部署"的概念，它比"热插拔"似乎更实用

3.6. Nutz 的版本命名规范

3.6.1. Nutz 每个版本 jar 包的命名遵照如下格式

```
[CODE]
nutz[.模块名]-主版本号.质量级别.[发布序号].jar
```

1. **模块名** : 可选，如不声明，则表示包括所有功能
2. **主版本号** : 从 1 开始顺序递增，除非程序结构或者接口发生重大改动，否则保持不变
3. **质量级别** : **a** 表示 alpha 品质， **b** 表示 beta 品质， **r** 表示 release 品质
    * **a** alpha: 表示这个版本，接口仍然不稳定，每次发布仍然有可能做微小的调整
    * **b** beta: 表示这个版本，接口已经稳定，主版号一致的 Beta 版会一直向前兼容，直至遇到一个 alpha 版
    * **r** release: 表示这个版本是一个非常稳定的版本。
4. **发布序号** : 从 1 开始顺序递增

这就意味着，如果是如下的一系列版本

1.b.30 -> 1.a.31 -> 1.b.32 -> 1.b.33 -> 1.b.34

* 1.b.32 可能不会兼容 1.b.30
* 1.b.34, 可以兼容 1.b.33 和 1.b.32

3.6.2. 你还需要知道

如果你看到 nutz-1.a.20.jar 和 nutz.1.b.18.jar

* 1.a.20 比较新，因为它的发布序号更新
* 1.b.18 更稳定，它可以兼容之前所有的 beta 版本
* 1.b.19 之后的 beta 版本，将兼容到 1.b.18 （不包括），因为期间的 alpha 版将兼容序列破坏了

3.6.3. 持续集成

每当有代码提交,Nutz的构建服务器会在15分钟内构建一个快照版本
http://build.sunfarms.net/nutz/

3.7. Nutz Java 编码规范 (V1.0)

3.7.1. 0. 规范的规范

1. 本规范的每一条目必须无二义性，并且可执行。否则作废
2. 本规范的条目分为两个级别:
    * 规则 - R

    * 建议 - S
3. 本规范所有的“规则”条目必须被遵守

3.7.2. 1. 代码格式

1. R-使用统一的 Eclipse 的代码格式: https://github.com/nutzam/nutz/blob/master/doc/eclipse/nutz-eclipse-java-code-format-1.0.xml
    * 请从 github项目中 获得此文件
2. S-非 Eclipse 用户请阅读上述 XML 代码自行遵守
    * *基本上我们没有为非 Eclipse 用户指定规范，我们还没有一个好办法*

3.7.3. 2. 命名

3.7.3.1. 2.1 包

1. R-包名必须全部小写，2个以内单词。
    1) S-最好为 1 个单数名词
2. R-所有项目的包要以 “org.nutz” 为父 包 。

3.7.3.2. 2.2 类和接口

1. S-最好为名词
2. R-命名类和接口时，需要将所有单词的首字母大写。
3. R-接口的命名不采用首字母为 I 或加上 IF 后缀的命名方式 。例 如 ：IBookDao 、BookDaoIF 等 。
4. R-抽象类必须使用 Abstract 作为类名的前缀，而接口建议使用 Interface 作为 接口名后缀 。
5. R-异常类应该使用 Exception 做为 名称 后缀。
6. R-如果是运行一次就抛弃的类，以 ing 结尾，比如Rendering
7. R-类名尽量短，但是最好不要缩写，如果缩写，必须为特别常用的类，比如 org.nutz.dao.Cnd
    * 因为调用者书写你的类名太长，他(她)的IDE会自动替他(她)换行，他会觉得有点不爽
8. R-不要和 Java 的标准库中的类名冲突，比如 Class, Object, String 等
    * 如果冲突，就表示你极其藐视 Java 标准库中的那个的设计
    * 调用者需要花更多的时间和代码来明确他使用的是你的类, 而不是标准库中的那个
9. S-以下情况可以允许写奇怪类名 -- 名称简短，让人一眼不知道什么意思，用了以后一眼就能知道什么意思
    * 类特别常用
    * 类非常特殊，难以归类
    * 私有类或内部类
    * 不推荐其他人调用的 公有、保护、默认类
        > 起个奇怪的名字，就是不想让你关心这个类的代码
10. R-缺省接口实现应该使用 Default 名称 前缀 。例 如 ： DefaultEntityMaker。
    * 也可以采用 Impl 作为后缀，表示这个实现为此接口的最优实现或者唯一实现

3.7.3.3. 2.3 成员变量

1. S-最好为单数名词
2. R-能 private 就不要 default，能 default 就不要 protected，最好不要 public
3. R-如果是集合或数组，用复数名词
    * Map pets， 比 Map petMap 要好
4. R-不要用一个字母，尤其是 i，你可以用 index 或者 cursor 来代替

3.7.3.4. 2.4 常量

1. R-命名常量（带有 final 修饰符的域）时需分隔。如 ： public final int MAX_VALUE = 30 。

3.7.3.5. 2.5 局部变量

1. R-局域变量名要尽量短，推荐用缩写，比如 StringBuilder sb
2. R-总的来说局部变量请随意命名，越短越好

比如这个就不好

```
[Java]
public String abc(String str){
   AbcObjectSet abcObjectSet = new AbcObjectSet();
   abcObjectSet.setName(str);
   return abcObjectSet.getBrief();
}
```

而这个就很容易阅读了：

```
[Java]
public String abc(String str){
   AbcObjectSet aos = new AbcObjectSet();
   aos.setName(str);
   return aos.getBrief();
}
```

3.7.3.6. 2.6 成员函数和静态函数

1. R-除了 setter / getter，其他的函数采用动词或者动名短语

2. S-以下情况可以允许写奇怪函数 -- 名称简短，让人一眼不知道什么意思，用了以后一眼就能知道什么意思
    1) 函数特别常用
    2) 函数非常特殊
    3) 私有函数或默认函数
3. S-支持链式赋值的 setter 允许写成 ，并且也可以支持同名 getter

```
[JAVA]
// Setter
public Pet name(String name){
   this.name = name;
   return this;
}
// Getter
public String name(){
   return this.name;
}
```

3.7.4. 3. 注释

1. R-注释必须和代码保持同步 。
2. R-注释中的第一个句子要以（英文）句号、问号或者感叹号结束。 Java成工具会将注释中的第一个句子放在方法汇总表和索引中。
3. R-如果注释中有超过一个段落，用 <P> 标签 分隔。
4. R-如果注释中有多个章节，用 <H2> 标签声明每个章节的标题。
5. R-如果注释需要换行，用
 标签。
6. R-示例代码以 <PRE></PRE> 包裹。

3.7.4.1. 3.1 类 Java Doc

1. R-要著名作者，格式为 @Author XiaoMing(xm@gmail.com)
2. R-继承的方法可以省略注释，但是被继承方法必须有注释。

3.7.4.2. 3.2 函数 Java Doc

1. R-简单的 get/set 方法可以省略注释。
2. R-继承的方法可以省略注释，但是被继承方法必须有注释。

3.7.4.3. 3.3 字段 Java Doc

* R-没有更多说明了

3.7.4.4. 3.4 函数内部注释

* R-行注释和块注释都是可以被接受的
* R-不要写 JAVA DOC，没意义
* R-代码质量不好但能正常运行，或者还没有实现的代码用 “ //TODO: ”
* R-在 if ... else .. 分支上做注释格式应该如下：

```
[CODE]
// comments for case A
if(xxxx){
   //TODO you code here
}
/*
 * Multipline comments for case B
 */
else if(xxxxx){
   //TODO you code here
}
// comments for default case
else{
   //TODO you code here
}
```

3.7.5. 4. 编程

1. R-你的每一次提交，必须都是编译通过的
2. R-你的每一次提交，最好都是通过 JUnit 测试的
    * 除非有特别的情况 -- 比如你要和其他人分享的修改
3. R-无论任何时候，同样的功能，一段更短的代码，总比更长的代码要好
    * 这里的“短”，主要指的是“逻辑”短，而不是“字符长度”短
4. R-删掉一段代码的贡献，比增加一段代码的贡献要大，至少不比它小
5. R-避免过度设计
    * 先让代码能工作，然后重构成为优美的代码
    * 你需要知道，“接口”固定了架构，“类” 不是，当它进化为接口的时候就固定了
    * 代码结构设计请遵循《草坪原则》

3.7.6. 5. 单元测试

1. R-用例名请用 "**长名**" - 一句话，用下划线_代替空白
    * 通过这个名字，基本可以了解测试是干什么的
2. R-主要接口和实现类要尽可能多的被用例覆盖

3.8. 成为代码提交者

3.8.1. 如果你已经是代码提交者

3.8.1.1. 一定要注意

* 请确保你修改的类上的 JDoc, 有你的大名，比如 "@author zozoh(zozohtnt@gmail.com)"
    > 首先，体现了你的贡献
    > 否则，出了问题，不知道由谁主要负责
* 请尽量用统一的 code formatter， 导入 /nutz/doc/eclipse/nutz-eclipse-java-code-format-1.0.xml，否则我们同步代码是很难比较
* 去掉的代码不要注释掉，直接删掉。我们会用文件比较工具比较两个版本的不同

3.8.1.2. 你需要知道

* Nutz 特点是代码短（以不影响阅读为前提）
    > 无论任何时候，同样的功能，一段更短的代码，总比更长的代码要好
    > 这里的“短”，主要指的是“逻辑”短，而不是“字符长度”短
* 如果你指出，某一个函数或者类其实无用，是最大的贡献
* 删掉一段代码的贡献，比增加一段代码的贡献要大，至少不比它小
* 如果你有机会能将代码减少一行，既不影响阅读，也不影响效率，你的贡献很杰出
* 我希望代码越来越优美，我认为简单，直接，就是优美
* 实现什么样的功能，由社区决定
* 如何实现，由你决定
* 如果有可能，请尽量为 public 的类和方法写 Java Doc，中文或者英文都可以
* 主要的函数，需要由 JUnit 来保证质量
* 你必须遵守这样的编码原则: 请阅读 Nutz Java 编码规范

3.8.1.3. 关于代码重写

* 接口的实现可以随时被重写，没关系，我们有版本控制，可以回滚
* 你的每一次提交，必须都是编译通过的
* 你的每一次提交，最好都是通过 JUnit 测试的
    > 除非有特别的情况 -- 比如你要和其他人分享的修改
* 我希望服务器上的代码通不过 JUnit 测试的情况，不要持续 3 天以上。最好，它时时刻刻都是可以通过全部 JUnit 测试的。

3.8.1.4. 关于翻天覆地的重写

* 这是不可避免的
* 如果有充足的理由，我很高兴这样做
* 我很喜欢我现在写出的代码，但是我不能保证一直喜欢它们
* 实际上，我认为在相当长的一段时间里，我们没有精力维护两个或两个以上的版本，Nutz 应该只有 master

3.8.2. 如果你不是，但是想成为代码提交者

* 你需要申请加入社区， 发信至： nutzam@googlegroups.com
* 通过社区，你可以让我们了解你，如果你发的贴很有价值，我们会考虑邀请你成为代码提交者
* 或者你可以直接给任何一个 Nutz 的代码提交者发信， 指出 Nutz 代码哪里存在不足，并提供修改建议。
    > 如果连续五次，你的修改建议被接受，我们会邀请你成为 Nutz 的代码提交者 -- 当然你可以拒绝
* 除此之外，没有其他的途径

4. Dao 手册

4.1. Hello world

4.1.1. 检查运行环境

1. 首先，你必须安装 JDK1.5 或者以上版本：
    1) 如果你直接使用编译好的 jar 包，请保证你使用对应版本的jar,因为Nutz分成 JDK5 和 JDK6 两种编译级别的jar。我们推荐使用JDK6,因为它更快!
    2) 最好有 Log4J 1.2.12或以上，如果没有，也没关系
2. 其次，为了运行 Nutz.Dao，你必须要有一个数据库软件：
    1) Mysql,Postgresql, SQLServer, Oracle, DB2，H2都可以。
3. 建立一个普通的 Java 项目：
    1) 你可以采用任何自己觉得舒适的 IDE 环境，或者编辑器。
    2) 你要将数据库的 JDBC Driver 和你喜欢的连接池加入项目的 classpath。
4. 在数据库里建立一张你自己的数据表， 这里我们用 t_person 来举例：

4.1.2. 创建数据库

此处代码适用于PostgreSQL，如果你使用了不同的数据库，请自行修改相应建库语句：

```
[sql]
CREATE TABLE t_person (
  id  SERIAL PRIMARY KEY,
  name  VARCHAR(50) NOT NULL UNIQUE,
  age INT
);
//对于最新版的Nutz,你可以用简单的dao.create(XXX.class, false)来完成建表
```

4.1.3. 创建 POJO

在 Java 项目中建立一个POJO，比如 com.zzh.demo.Person：

```
[java]
import org.nutz.dao.entity.annotation.*;
@Table("t_person")  // 声明了Person对象的数据表
public class Person { //无需继承任何Nutz的类
    @Column  // 表示该对象属性可以映射到数据库里作为一个字段
    @Id      // 表示该字段为一个自增长的Id,注意,是数据库表中自增!!
```

```
    private int id;
    @Column
    @Name    // 表示该字段可以用来标识此对象，或者是字符型主键，或者是唯一性约束
    private String name;
    @Column
    private int age;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

4.1.4. 创建 main 函数

随便建立一个有 main 函数的类，在 main 函数里这么写：

```
[java]
Dao dao = new NutDao(dataSource);
Person p = new Person();
p.setName("ABC");
p.setAge(20);
dao.insert(p);
```

* 执行完毕后，在数据表中会多出一条记录。

* 如果你不知道如何创建 Datasource，请参看 附录：如何创建 Datasource
* 这个例子足够作为 Hello World 了， 祝你玩的开心 :)

4.1.5. 如何进阶

在你看完上述这个例子以后，你可以看看 Nutz.Dao 的基本操作，这篇短短的文章会让你很快对 Nutz.Dao 建立一个大体上的印象。但是实际上，你的 POJO 在真正的项目中可能不会这么简单，你需要操作也不会只是增删改查。下面有三个建议：

* Nutz.Dao 全部的文档并不很多，花上1-2个晚上，你就可以看完。
* 你如果想快速了解 Nutz.Dao 到底支持多少注解，请看这里
* 在 Nutz 的 Demo Site 上，我们也会不断增加有针对性的演示，你可以 访问它的首页，看看里面有没有你需要的例子代码

4.2. Dao 接口的基本操作

4.2.1. 概述

传统关系型数据库定义了四种数据操作：

1. 插入 Insert
2. 删除 Delete
3. 更新 Update
4. 查询 Query

可以说，这四种操作涵盖了所有的数据操作。并且，除了 插入 操作，所有的操作都是可以一次针对多条记录的。

但是，Nutz.Dao 认为从使用者的角度来看，这四种操作还是有所不同的。比如，查询返回的结果，很多时候仅仅是一条记录。我们需要为这种情况进行优化。所以，Nutz.Dao 在传统关系型数据库数据操作的基础上定义了如下的数据操作：

插入	Insert	一条 SQL 插入一条记录或者多条记录
插入	FastInsert	一条 SQL ,通过batch插入多条记录
删除	Delete	一条 SQL 删除一条记录
更新	Update	一条 SQL 更新一条或者多条记录
获取	Fetch	一条 SQL 获取一条记录

查询	Query	一条 SQL 根据条件获取多条记录
清除	Clear	一条 SQL 根据条件删除多条记录

请注意： 这里我是说 “一条” SQL。 如果通过 Dao 接口，你传入的是一个集合或者数组，它会为每一个元素都生成一条 SQL 并执行，并更新操作:

```
[java]
Pet[] pets = xxxx;
dao.update(pets);// 可以是数组，当然 pets 也可以是集合
```

同理，delete 和 insert 也支持传入数组和集合

4.2.2. 示例的前提条件

* 我们假设已经创建了实体类 **com.zzh.demo.Person** 和实体表 **t_person**
* 在文档 Nutz.Dao 入门 中，我们已经声明了这个实体
* 下述所有的操作都是假设已经有了 dao 变量，并且它指向一个 Dao 的实例。文档 Nutz.Dao 入门 中，我们给出了如何创建 Dao 实例，以及如何搭建运行环境

4.2.3. 创建数据表

为 Pet 创建数据表，如果数据表存在，先 DROP 掉，再创建

```
[java]
dao.create(Pet.class, true); //生产环境你可千万别这样写!!
```

为 Pet 创建数据表，如果数据表存在，忽略

```
[java]
dao.create(Pet.class, false); //一般我们都这样写
```

4.2.4. 删除数据表

删除 Pet 的数据表

```
[java]
dao.drop(Pet.class);//全部删掉哦,没条件的,慎用!!
```

4.2.5. 插入 Insert

```
[java]
Person p = new Person();
p.setName("Peter");
p.setAge(22);
dao.insert(p);
System.out.println(p.getId());
```

Person 对象的 Id 被自动更新了。

* 更多的关于 @Id 注解的描述，请参看 关于主键 以及 在插入前后的为字段设值

4.2.6. 取得 Fetch

根据名称获取（如果你的实体声明了 @Name 字段）

```
[java]
Person p = dao.fetch(Person.class,"Peter");
System.out.println(p.getId());
```

根据 ID 获取（如果你的实体声明了 @Id 字段）

```
[java]
Person p = dao.fetch(Person.class,2);
System.out.println(p.getName());
```

4.2.7. 更新 Update

```
[java]
Person p = dao.fetch(Person.class,2);
p.setAge(32);
dao.update(p)
```

4.2.8. 更新多条

```
[CODE]
dao.update(Person.class, Chain.make("dead",true),
Cnd.where("age",">",150));
```

4.2.9. 删除 Delete

根据名称删除（如果你的实体声明了 @Name 字段）

```
[java]
dao.delete(Person.class,"Peter");
```

根据 ID 删除（如果你的实体声明了 @Id 字段）

```
[java]
dao.delete(Person.class,2);
```

4.2.10. 查询 Query

4.2.10.1. 查询全部记录

```
[java]
List<Person> people = dao.query(Person.class,null);
```

4.2.10.2. 按条件查询

```
[java]
List<Person> people = dao.query(Person.class, Cnd.where("name",
"like", "P%"));
```

* Cnd 类的全名是 org.nutz.dao.Cnd
    > 它主要是用来快速替你建立一个 org.nutz.dao.Condition 接口的实现类
    > where() 函数 第一个参数是字段名，要和 Java 类里面的字段名相同。
    > where() 函数 第二个参数遵循 SQL 的标准，可以是 >, <, >=, <= 等等
    > 提供了一个 wrap 函数，你可以直接写 SQL 的条件
* 如果你愿意，你完全可以自己实现一个 Condition，来做更复杂灵活的判断
* 关于更多的查询条件的说明，请参看 复杂条件

4.2.10.3. 分页查询

```
[java]
List<Person> people = dao.query(Person.class, Cnd.where("age", ">",
18), dao.createPager(2, 4));
```

* dao.createPager 第一个参数是第几页，第二参数是一页有多少条记录
* 关于分页更多的说明，请参看 分页查询

4.2.11. 清除 Clear

4.2.11.1. 清除所有记录

```
[java]
dao.clear(Person.class); //还是那句,慎用
```

4.2.11.2. 按条件清除

```
[java]
dao.clear(Person.class,Cnd.where("id", ">", 35));
```

* 关于更多的清除条件的说明，请参看 复杂条件

4.2.12. 插入和更新集合

无论是插入 (Insert) 还是更新 (Update)，你传入的对象都可以不仅仅是一个 POJO，你可以传入：

* 集合 (Collection<?>)
* Map<?,?>
* 数组 (T[])

Nutz.Dao 会自动替你拆包，对集合成员依次执行相应操作。 对于 Map，它会迭代每一个值。

4.3. 关于主键

4.3.1. 简要介绍

为使用 Dao 接口的 fetch(Class<?>, long) 以及 fetch(Class<?>, String)，需要为一个 POJO 指明它的主键。主键可以是整数型的，也可以是字符型的。同时它也可以支持复合主键。

整数型主键	注解 @Id	声明在字段上
字符型主键	注解 @Name	声明在字段上
复合主键	注解 @PK	声明在类上

注意： 对于一个 POJO，你可以同时为其声明 @Id 和 @Name，它们都能正常工作。你只需要保证 @Name 对应的字段在数据库里有唯一性约束即可。 但是通常， Nutz.Dao 并没有假设你同时在一个 POJO 里应用 @Id, @Name 和 @PK，如果你这么做了，可能会引发一些奇怪的问题。事实上，你也不可能这么做，不是吗？

4.3.2. 整数型主键

```
[CODE]
@Table("t_pet")
public class Pet{
    @Column
    @Id
    private int id;
...
```

通过 @Id 声明了一个整数型主键后，你可以：

```
[CODE]
Pet pet = dao.fetch(Pet.class,23);
```

4.3.2.1. 默认自增

默认的，Nutz.Dao 认为一个整数型主键默认就是自增的。所以他会在：

```
[CODE]
dao.insert(pet);
```

之后，为你插入的对象执行

```
[CODE]
SELECT MAX(id) FROM t_pet;
```

并设置到 pet 对象中。当然，不同的数据库，获得自增值的方式是不一样的，请参看 在插入前后的为字段设值里面描述了，通过 @Next 注解，声明数据库本地方言，来获取自增值。注意!! 这里的自增是由数据库表来实现的,而非NutDao的内部自增机制!!

4.3.2.2. 手工设值

由于默认的，@Id 字段被认为是自增的，所以在插入时，Nutz.Dao 会忽略这个字段。但是，有些时候，你的整数主键不是自增的，你希望手工为其设值，怎么办呢？ 你可以给 @Id 设一个属性: auto=false

```
[CODE]
@Table("t_pet")
public class Pet{
    @Column
    @Id(auto=false)
    private int id;
...
```

Nutz.Dao 在插入对象是，就不会忽略你这个主键的值了，并且在插入完毕后，它也不会执行 SELECT MAX(id)。

4.3.2.3. 快速插入

无论你是不是 @Id(auto=false)，通过 Dao.fastInsert ，都不会执行 SELECT MAX(id)

```
[CODE]
dao.fastInsert(pet)
```

它只会单纯的拼出一条 INSERT XXX，然后执行。 在一个循环里，一次插入多个对象时，很适合使用。

4.3.3. 字符型主键

```
[CODE]
@Table("t_pet")
public class Pet{
    @Column
    @Name
    private String name;
...
```

通过 @Name 声明了一个字符型主键后，你可以：

```
[CODE]
Pet pet = dao.fetch(Pet.class,"XiaoBai");
```

4.3.3.1. 忽略大小写

```
[CODE]
@Table("t_pet")
    public class Pet{
        @Column
        @Name(casesensitive=false)
```

```
    private String name;
...
```

因此

```
[CODE]
Pet pet = dao.fetch(Pet.class,"XiaoBai");
同
Pet pet = dao.fetch(Pet.class,"xiaobai");
运行的结果就会没有区别
```

4.3.4. 复合主键

```
[CODE]
@Table("t_pet")
@PK( {"masterId", "petId"} )
public class Pet{
    @Column
    private int masterId
    @Column
    private int petId;
...
```

通过 @PK 声明了复合主键，你就可以:

4.3.4.1. 通过变参获取和删除

获取

```
[CODE]
Pet pet = dao.fetchx(Pet.class, 23, 12);
```

删除

```
[CODE]
Pet pet = dao.deletex(Pet.class, 23, 12);
```

注意

* 这里你给的变参的顺序必须按照你 @PK 里声明的顺序。
* 是 fetch**X** 和 delete**X**

4.3.4.2. 其他操作

至于 update, clear, insert 则和平常的对象一样。 不过 Update 的时候，你的 POJO 所有的复合主键字段需要被设上值，才能正确被更新。

4.4. 复杂的SQL条件

4.4.1. 概述

4.4.1.1. 什么是 Nutz.Dao 中的复杂SQL条件

* 对于 Nutz.Dao 来说，它本质上就是将你的 Java 对象转化成 SQL，然后交给 JDBC 去执行。
* 而 SQL 中，当执行数据删除和查询操作时，最常用的就是 WHERE 关键字。
* WHERE 关键字后面的就是所谓的复杂查询条件

4.4.1.2. Nutz.Dao 将如何如何使用这个条件

* Dao 接口的 clear 方法和 query 方法的第二个参数，就是为了生成 WHERE 后面那段字符串设计的
* 这个参数是一个 org.nutz.dao.Condition 接口的实现类
* 通过该接口的 toString(org.nutz.dao.entity.Entity) 方法， Nutz.Dao 将获得 WHERE 后面那段字符串
* 当然也包括 ORDER BY

4.4.2. Condition 接口

* 这个接口只有一个方法 toString(Entity entity)
    > 这个方法带一个参数 org.nutz.dao.entity.Entity
    > 通过这个参数，实现者可以获得当前需要操作的实体的配置信息
    > 当然，如果你不需要的话，你可以不理睬这个参数
* Nutz.Dao 会将 toString(Entity entity) 的返回直接拼接到SQL 的 WHERE 关键字后面
    > 如果你返回的字符串以 WHERE 或者 ORDER BY 开头，它会直接使用，否则会补上一个 WHERE 关键字
    > 这个判断会忽略前后空白以及大小写

4.4.3. Nutz 给你的快速实现

* 如果你的数据库字段被假设不会发生变化，用直接硬编码是个很好的选择
* 如果在开发期，你的数据库字段变化非常频繁，用 Cnd 工具类则是更好的选择

4.4.3.1. 直接硬编码

最简单直接的方法就是直接输出 WHERE 关键字后面的 SQL 代码了。比如查询一个 Person 对象

```
[JAVA]
List<Person> crowd = dao.query(Person.class, Cnd.wrap("name LIKE 'J%' AND age>20"), null);
```

这句话，就会将所有名称以 J 开头，并且年龄超过20岁的人全部查询出来。参看 Nutz.Dao 入门 针对 Person 对象的描述，实际上，上面那句话执行的 SQL 代码为：

```
[SQL]
SELECT * FROM t_person WHERE name LIKE 'J%' AND age>20;
```

当然你要是写

```
[JAVA]
Cnd.wrap("name LIKE 'J%' AND age>20 ORDER BY name ASC");
```

就会按照 name 字段排序。

4.4.3.2. 一个友好的工具类 -- Cnd

有些情况，数据库中的字段同 Java 对象中的字段并不同名，所以就需要给 Java 字段上的数据库字段注解加上参数 @Column("数据库字段名")如果你通过 Cnd.wrap() 硬编码某个字段，那么当这个字段数据库字段名发生改变时，你就需要改动很多。因此你希望仅仅将对于数据库的变动限制在 Java 对象的源文件里所以 Nutz 提供了 Cnd.where() 方法

```
```

```
[Java]
Condition c = Cnd.where("age",">",30).and("name", "LIKE",
"%K%").asc("name").desc("id");
```

这个条件将生成 SQL

```
[SQL]
WHERE age>30 AND name LIKE '%K%' ORDERBY name ASC, id DESC
```

你也可以嵌套表达式

```
[JAVA]
SqlExpressionGroup e1 = Cnd.exps("name", "LIKE", "P%").and("age",
">", "20");
SqlExpressionGroup e2 = Cnd.exps("name", "LIKE", "S%").and("age",
"<", "30");
Condition c = Cnd.where(e1).or(e2).asc("name");
```

这个条件将生成 SQL

```
[SQL]
WHERE (name LIKE 'P%' AND age>'20') OR (name LIKE 'S%' AND
age<'30') ORDER BY name ASC
```

4.4.4. 拼装更加复杂的条件

上面的例子的 Cnd.where 函数，在大多数情况下可以快速的生成一个简单的查询条件。但是，如果查询条件非常复杂，用它可能就比较费劲了。是的，它的设计初衷就是 " **查询条件应该一行搞定**"。

有些时候，查询条件很复杂，一行确实搞不定，怎么办？ **Nutz-1.b.38** 以后，提供了 Criteria 接口，它继承自Condition 接口，它的设计目的有两个：

1. 让程序员更容易的拼装复杂逻辑的条件
2. 让生成的 SQL 可以被参数化，更好的支持 PreparedStatement

这个接口的使用也很简单，它基本符合 "IDE 的所见即所得" 接口设计原则。 就是说，如果你的 IDE 有智能提示的话，你使用这个接口是不需要文档的。

```
[CODE]
// 创建一个 Criteria 接口实例
Criteria cri = Cnd.cri();
// 组装条件
if(...){
    cri.where().andIn("id", 3,4,5).andIn("name", "Peter", "Wendal",
"Juqkai");
}else if(...){
    cri.where().andLT("id", 9);
}
if(...){
    cri.where().andLike("name", "%A%");
}
cri.getOrderBy().asc("name").desc("id");
// 执行查询
List<MyObj> list = dao.query(MyObj.class, cri, null);
```

Criteria 的 where() 函数返回的是 SqlExpressionGroup，主要由它来提供各种 SQL 条件的组合方法。这里需要给出一点提示，比如方法名 andGT，表示的是 andGreatThan，即"大于" 的意思，同理:

* LT : 小于 (LessThan)
* GTE : 大于等于 (GreatThanEquale)
* LTE : 小于等于 (LessThanEquale)

4.5. 分页查询

4.5.1. 概述

使用数据库的应用程序，多数情况下都需要使用 “分页” 这个功能。尤其是在 Web 应用程序中，后端的分页查询尤其的普遍。在以往的使用经验中，一个分页查询，除了能获取到一个列表外，我们通常需要如下几个信息才能在客户端显示出一个完整的翻页条。

* 当前页数 -- 第几页
* 页大小 -- 每页有多少条记录

* 总页数 -- 一共多少页
* 总记录数 -- 如果不分页，一共有多少条记录

当我们获得了这四条信息后，对于维护一个翻页查询就足够。

Nutz.Dao 的查询接口天然就支持分页查询。

4.5.2. Dao 接口的第三个参数

让我们先看看 Nutz.Dao 接口查询函数的声明：

```
[JAVA]
<T> List<T> query(Class<T> classOfT, Condition condition, Pager
pager);
```

这个接口有三个参数

* classOfT 告诉 Nutz.Dao 需要查询的实体类型
* condition 告诉 Nutz.Dao 查询出的列表需要符合的条件。详细请看 复杂条件。
* 最后一个参数，就是告诉 Nutz.Dao 将结果如何分页的了。

Pager 对象有如下几个注意事项：

* 如果 pager 被传入了 null，则不分页
* 生成 Pager 对象的时候需要传入 “当前页数” 和 “页大小”
* Pager 虽然有 getRecordCount() 和 getPageCount() 方法，但是它不会自动被设值 -- 因为考虑到效率
* 通过 Pager.setRecordCount() 可以为 Pager 设置结果集的总数，Pager 会通过 getPageCount() 返回总页数

4.5.3. 将分页信息和查询结果一起返回

```
[CODE]
public QueryResult getPetList(Dao dao, int pageNumber, int
pageSize){
    Pager pager = dao.createPager(pageNumber, pageSize);
    List<Pet> list = dao.query(Pet.class, null, pager);
    pager.setRecordCount(dao.count(Pet.class));
    return new QueryResult(list, pager);
```

```
}
```

Nutz 会自动为各种不同的数据库，根据你传入的 Pager 生成翻页方言。但是考虑到效率因素，Nutz.Dao 并不会为 Pager 计算结果集的总数。我知道这那么一点点让人觉得有点不方便，但是我想给你控制权，我想让你可以: "只在必要的时候才计算结果集的总数"，我想，这个控制权对你来说，比少写一行CRUD代码更加重要，对吗？

4.6. 插入前后的设置

4.6.1. 插入之前 - @Prev

在插入之前，你想通过一段 SQL 为你的 POJO 某一个字段设值。你可以通过 @Prev 属性

```
[CODE]
@Table("t_pet")
public class Pet{
    @Name
    private String name;
    @Column("photo")
    @Prev( @SQL("SELECT txt FROM t_pet_photo WHERE
pname=@name") )
    private String photoPath;
...
```

@Prev 注解接受一组 @SQL 作为参数，它遵守如下约定：

* @SQL 声明了一条 SQL 语句，支持动态占位符。
    > 变量 - 形式如： $变量名
        - 其值由 org.nutz.dao.TableName 来设置，具体使用方式请参看 动态表名
        - 特殊占位符不需要手工设值，Nutz.Dao 自动为你设置，它们是：
            - **$view** - 表示当前实体对象的视图名称
            - **$field** - 表示注解所在字段数据库名称
        - 其他特殊占位符，会被对象自身的同名属性值替换
    > 参数 - 形式如：@参数名
        - 其值直接使用 POJO 自身的属性值
        - **比如上例**，将会参考对象自身的 name 字段的值
* 如果 SQL 执行没有结果，即结果集合为空，将不会为相应字段设置
* 如果 SQL 执行的结果集包含多条记录，只有第一条记录的第一列的值会被使用
* @SQL 更详细的语法规范请参看 自定义 SQL

4.6.2. 插入之后 - @Next

在插入之后，你想通过一段 SQL 为你的 POJO 某一个字段设值。你可以通过 @Next 属性

```
[CODE]
@Table("t_pet")
public class Pet{
    @Column
    @Id
    @Next( @SQL("SELECT currval('t_pet_id_seq')") )
    private long id;
...
```

如上例，执行插入后，你的 Pet 对象的 id 会被数据库中新的值更新。

@Next 的规则和 @Prev 是一样的

4.6.3. 以 @Prev 来举例

下面让我们举两个例子，详细说明一下 **变量** 和 **参数** 的异同点。

4.6.3.1. 使用变量的例子

```
[CODE]
@Prev(@SQL("SELECT pet_name FROM t_user_pet WHERE
ownm='$ownerName'"))
private String name;
```

在执行 dao.inert 操作时， Nutz.Dao 会预先执行这段 SQL, 执行之前，变量 **$ownerName** 会被对象本身的 ownerName 字段的值替换，如果对象本身的 ownerName 字段的值恰好是 "zzh"，那么执行的 SQL 会变成：

```
[CODE]
SELECT pet_name FROM m_user_pet WHERE ownm='zzh';
```

这段 SQL 执行的结果会复制给对象的 name 字段。

4.6.3.2. 使用参数的例子

```
[CODE]
@Prev(@SQL("SELECT pet_name FROM t_user_pet WHERE
ownm=@ownerName"))
private String name;
```

在执行 dao.inert 操作时， Nutz.Dao 会预先执行这段 SQL, 执行之前，参数 **@ownerName** 会被 '?' 替换，并根据这段 SQL 生成 PreparedStatement 对象：

```
[CODE]
SELECT pet_name FROM m_user_pet WHERE ownm=?;
```

然后，根据对象本身的 ownerName 字段的值，为这个 PreparedStatement 设置参数，执行之后，这段 SQL 执行的结果会复制给对象的 name 字段。

4.6.4. 数据库方言

无论是 @Prev 还是 @Next，你都是通过 @SQL 声明的数据库方言。但是，假设你并不确定你的 POJO 将会工作在哪一个数据库上，比如你的项目有两个数据源，一个是 Oracle 一个是 Postgresql，那么你的 POJO 该如何写方言呢？

```
[CODE]
@Table("t_pet")
public class Pet{
    @Column
    @Id
    @Next({
        @SQL(db = DB.PSQL,    "SELECT currval('t_pet_id_seq')"),
        @SQL(db = DB.ORACLE, "SELECT t_pet_id_seq.currval FROM
dual')"),
        @SQL(db = DB.OTHER,   "SELECT MAX(id) FROM t_pet")
    })
    private long id;
...
```

现在 Nutz.Dao 支持这些数据库:

```
[CODE]
public enum DB {
    H2, DB2, PSQL, ORACLE, SQLSERVER, MYSQL, OTHER
}
```

特别说一下Oracle的seq

```
[CODE]
@Table("t_pet")
public class Pet{
    @Column
    @Id(auto=false)
    @Prev({
        @SQL(db = DB.ORACLE, "SELECT t_pet_id_seq.currval FROM
dual')")
    })
    private long id;
...
```

4.7. 使用视图

4.7.1. 为什么需要视图

数据库将数据存储在数据表中，有些时候，为了展现某些数据，还需要跨越多个表进行计算。比如你有两个数据表

1. 雇员表 t_employee
2. 任务表 t_task

每个雇员都会有多个任务，所以任务表有一个字段 eid 指向雇员的主键 id。 我们需要随时知道一个雇员有多少个任务，所以在雇员的的POJO 里面，有一个 Java 字段叫做 taskCount 来描述雇员当前总的任务数量。

那么，如何获取这个 taskCount 呢？ 每次都执行 SELECT COUNT(*) 吗？比较理想的一个解决方法就是使用视图。在视图的 SQL 语句中我们可以利用数据库的方言，发挥数据库的效率。并且在程序代码里也会更加简洁。比如我们建立一个视图 v_employee

```
[SQL]
CREATE VIEW v_employee AS (
SELECT * , taskcount = (SELECT COUNT(id) FROM t_task AS t WHERE
t.eid = e.id) FROM t_employee AS e;
)
```

这段代码仅仅是个示意，在不同的数据库上，有不同的写法

那么实际上，我们建立的这个 Employee 的 Java 对象就是要从 v_employee 获取，但是执行 insert, update 或者 delete 的时候，却是操作 t_employee 的。为了这个特殊的需求，Nutz.Dao 提供了 @View 和 @Readonly 这两个注解。

4.7.2. 在 POJO 对象上使用视图

我们直接来表述一下 Employee 这个 POJO，这里仅仅是个示意

```
[JAVA]
@Table("t_employee")
@View("v_employee")// <- 这里声明了视图
public class Employee{
    @Column
    @Id
    private int id;
    ...
    @Column("taskcount")// 其实可以不用声明数据库字段名
"taskcount"，因为多数数据库忽略大小写
    @Readonly// <- 这里声明了只读字段，即视图里增加的字段
    private int taskCount;
    ...
}
```

* 通过 @View， Nutz.Dao 知道了当查询数据时候，应该从哪里获取数据
* 通过 @Readonly 注解 Nutz.Dao 知道，当修改或者删除数据的时候，那些字段应该忽略。

接下来，你就可以通过 org.nutz.dao.Dao 接口随意操作这个 POJO 了。

4.8. 过滤字段

4.8.1. 为什么需要过滤字段

某些时候，尤其是在更新对象的时候，用户希望忽略某些字段。通过注解 Nutz.Dao 实体注解 @Readonly 可以达到 这个目的。但是更多的时候，对于 POJO 对象，只有在运行时，用户的程序才能决定哪些字段更新，哪些不更新。@Readonly 注解可就达不到这个 目的了。

怎么办呢？ Nutz.Dao 提供了类似于 动态表名 的解决办法。

4.8.2. 如何过滤字段的例子

如下代码，将只更新 Pet 的 id 和 name 字段：

```
[CODE]
FieldFilter.create(Pet.class, "^id|name$").run(new Atom() {
    public void run() {
        Pet pet = dao.fetch(Pet.class, 24);
        pet.setName("ABC");
        pet.setNickname("XiaoBai");
        dao.update(pet);
    }
});
```

4.8.3. 字段过滤器的原理

字段过滤，不过是要在一个地方记录一下下面两个信息：

1. 对什么实体
2. 过滤哪些字段

并且它希望 Nutz.Dao 自行能获取到这些信息。当然，ThreadLocal 就是一个很好的选择。实际上，你如果看看 FieldFilter 里面的方法，你其实就能猜到。为了能为多个实体保存字段过滤配置信息，它实际上在ThreadLoacal 里保存了自身的一个实例，同时，它自己有一个私有的 Map<Class<?>, FieldMatcher>，具体的，你可以看 FieldFilter 这个类的定义:

```
[CODE]
public class FieldFilter {
    ...
    private static ThreadLocal<FieldFilter> FF = new
ThreadLocal<FieldFilter>();
    ...
```

```
    private Map<Class<?>, FieldMatcher> map;
    ...
```

而且既然在 ThreadLocal 设置了数据，它就不得不考虑如何让你清除这个数据。因此，它的写法也保证了你一定会清掉你的数据了。

```
[CODE]
FieldFilter ... run(new Atom(){   <-- 开始将自身加入 ThreadLocal
    public void run(){
        // 这里是你的代码，你的 Dao 调用都会得到 ThreadLocal 中你对于实体字段过滤的设置
    }
});   <-- run 方法结束前，会从 ThreadLocal 清除自身
```

4.8.4. 字段过滤器的创建

下面是一个最简单和常用的例子：

```
[CODE]
FieldFilter.create(Pet.class, "^id|name$").run(new Atom(){
    public void run(){
        // TODO 你的 DAO 操作代码
    }
});
```

* 这样，无论你查询或者更新等操作，对 Pet 这个实体只会针对能被正则表达式 "id|name" 匹配的字段进行操作
    > 实际上，上例的正则表达式表示: 所有包括 id 和 name 字符的字段
* 如果你想仅仅让 id 和 name 字段受到匹配，你的正则表达式最好写的严格一些，比如 "^id|name$"
* 当然，SQL 的条件部分不会受到字段过滤器的影响

如果你读完上面的介绍，你应该就很了解字段过滤器如何使用了，但是你可能还有个几个小疑问:

* 如果我字段比较多怎么办呢？
* 如果我想忽略所有之为空的字段怎么办呢？

* 如果我想同时为多个实体设置字段过滤怎么办呢？

4.8.4.1. 忽略少数字段

```
[CODE]
FieldFilter.lock(Pet.class, "^last|age$").run(new Atom(){
    public void run(){
        // TODO 你的 DAO 操作代码
    }
});
```

4.8.4.2. 忽略空值

```
[CODE]
FieldFilter.create(Pet.class, true).run(new Atom(){
    public void run(){
        // TODO 你的 DAO 操作代码
    }
});
```

4.8.4.3. 保留几个字段且忽略空值

```
[CODE]
FieldFilter.create(Pet.class,"^id|name|age|last$", true).run(new
Atom(){
    public void run(){
        // TODO 你的 DAO 操作代码
    }
});
```

4.8.4.4. 忽略少数字段且忽略空值

```
[CODE]
FieldFilter.create(Pet.class, null, "^age|last$", true).run(new Atom(){
    public void run(){
        // TODO 你的 DAO 操作代码
```

```
    }
});
```

4.8.4.5. 为多个实体设置字段过滤

```
[CODE]
FieldFilter.create(Pet.class, true)
          .set(Master.class, "^id|name$")
          .run(new Atom(){
    public void run(){
        // TODO 你的 DAO 操作代码
    }
});
```

4.9. 动态表名

4.9.1. 为什么需要动态表名

当数据量比较大的时候，为了提高数据库操作的效率，尤其是查询的效率，其中一种解决方案就是将数据表拆分。拆分的数据表，结构完全一致，只不过是表的名字，按照某种规律，而成为一组。

4.9.2. 动态表名的常用形式

通常情况下动态表名都是通过一个后缀来表示的。比如我们要记录全中国所有的公司以及其雇员，通常的设计是建立两张数据表， t_company 记录公司，t_employee 记录雇员。由于考虑到 t_employee 的数量可能太过庞大，我们可以将 t_employee 进行拆分，为每个公司建立一张雇员表。 比如 t_employee_1 记录 ID 为 1 的公司所有雇员，t_employee_19 记录 ID 为 19 的公司所有雇员。

当然，我们也不能排除动态表名的其他形式，比如，如果公司也是动态表名： t_company_3 表示在 ID 为 3 的国家的公司。那么雇员表有可能被设计成 t_employee_3_10， 在 ID 为 3 的国家且 ID 为 10 的公司所有的雇员记录

另外，表名的中的变量可能不只是数字，也可能不只是后缀。考虑到这样的情况，同时也希望不增加 org.nutz.dao.Dao接口的复杂程度， Nutz.Dao 将怎样为这样的数据库操作方法提供支持的呢？

4.9.3. Nutz对于动态表名的支持

4.9.3.1. 在 POJO 中声明动态表名

毫无疑问，首先，需要配置你的 POJO。 Nutz.Dao 提供的 @Table 注解本身就支持动态表名，比如：

```
[JAVA]
@Table("t_employee_${cid}")
public class Employee{
    // The class fields and methods...
}
```

@Table 注解支持字符串模板的写法，在你希望放置动态表名变量的位置插入 ${变量名} ，如 ${cid}，那么${cid} 会在运行时被 Nutz.Dao 替换。

如何替换？用什么替换？请看下面一节。

4.9.3.2. 在调用时应用动态表名

Nutz.Dao 提供了一个小巧的类: org.nutz.dao.TableName。 通过这个类你可以随意设置你的动态表名：

```
[JAVA]
public void demoTableName(final Dao dao) {
    TableName.run(3, new Runnable() {
        public void run() {
            Employee peter = dao.fetch(Employee.class, "peter");
            System.out.println(peter);
        }
    });
}
```

通过创建一个匿名 java.lang.Runnable 对象，你可以像静态 POJO 一样使用 Dao 接口的一切方法。因为通过你传入的参数 3 (TableName.run 方法的第一个参数)， 以及前面的 @Table 声明，Nutz.Dao 已经很清楚如何操作数据库了，它会用 3 替代 null。

如果细心一些，你会发现在 TableName.run 方法的声明是：

```
[JAVA]
public static void run(Object refer, Runnable atom);
```

是的，第一个参数是个 Object，也就是说，你可以传入任何对象，上面的例子我们传入的是个整数，Java编译器会自动将其包裹成 Integer 对象。考虑到动态表名可能存在的复杂性（在前面一节“动态表名的常用形式”提到），你还可以传入一个 Map 或者一个POJO， Nutz.Dao 会根据你的传入为动态表名的多个变量同时赋值。下面我列出一个完整的**动态表名赋值规则**

4.9.3.3. 动态表名赋值规则

* 当传入参数为数字或字符串
    > 所有的动态表名变量将被其替换
* 当传入参数为 Map
    > 按照动态表名变量的名称在 Map 中查找值，并进行替换
    > 大小写敏感
    > 未找到的变量将被空串替换
* 当传入参数为 任意Java对象(POJO)
    > 按照动态表名变量名称在对象中查找对应字段的值，并进行替换
    > 大小写敏感
    > 未找到的变量将被空串替换
* 当传入参数为null
    > 所有变量将被空串替换

4.9.3.4. 更灵活的应用动态表名

使用 TableName.run 提供的动态表名模板的方式设置动态表名是很好的做法，也很安全。因为不会在 ThreadLocal里面留下垃圾动态表名变量值。但是通过模板的写法另外一方面也限制了一定线程灵活性。所以上述例子还有另外一个写法:

```
[JAVA]
public void demoTableName2(final Dao dao) {
    TableName.set(3);
    Employee peter = dao.fetch(Employee.class, "peter");
    System.out.println(peter);
    TableName.clear();
}
```

在 TableName.set 和 TableName.clear 之间的代码，就是动态表名变量的生命周期。当然，这个写法存在两个潜在的危险。

1. 可能在 ThreadLocal 留下垃圾动态表名变量
2. 会清除掉 ThreadLocal 所有的动态表名变量

关于第一点，可以用 try...catch...finally 来解决:

```
[JAVA]
public void demoTableName3(final Dao dao) {
    try {
        TableName.set(3);
        Employee peter = dao.fetch(Employee.class, "peter");
        System.out.println(peter);
    } finally {
        TableName.clear();
    }
}
```

这样，永远都不会留下垃圾动态表名了

关于第二点，是的，如果在 TableName.set(3) 之前你曾经设置了另一个动态表名变量，比如

```
[JAVA]
public void demoTableName_multi(final Dao dao) {
    TableName.set(10);
    Employee john = dao.fetch(Employee.class, "john");
    System.out.println(john);
    TableName.set(3);
    Employee peter = dao.fetch(Employee.class, "peter");
    System.out.println(peter);
    TableName.clear();
    Employee mary = dao.fetch(Employee.class, "Mary");
    System.out.println(mary);
    TableName.clear();
}
```

当执行到 Employee mary = dao.fetch(Employee.class, "Mary"); 一定会出错，不是吗？ 所以比较安全的写法是

```
[JAVA]
public void demoTableName_multi(final Dao dao) {
    TableName.set(10);
    Employee john = dao.fetch(Employee.class, "john");
```

```
    System.out.println(john);
    Object old = TableName.get();
    try {
        TableName.set(3);
        Employee peter = dao.fetch(Employee.class, "peter");
        System.out.println(peter);
    } finally {
        TableName.set(old);
    }
    Employee mary = dao.fetch(Employee.class, "Mary");
    System.out.println(mary);
    TableName.clear();
}
```

比较麻烦是吗？ 是的，使用 TableName.get ... TableName.set ... TableName.clear 虽然带来更大的灵活性，但是写起来有点麻烦，这也是为什么我要提供模板写法，上面的例子用模板的写法看起来是这个样子的：

```
[JAVA]
public void demoTableName_multi_temp(final Dao dao) {
    TableName.run(10, new Runnable() {
        public void run() {
            Employee john = dao.fetch(Employee.class, "john");
            System.out.println(john);
            TableName.run(3, new Runnable() {
                public void run() {
                    Employee peter = dao.fetch(Employee.class, "peter");
                    System.out.println(peter);
                }
            });
            Employee mary = dao.fetch(Employee.class, "Mary");
            System.out.println(mary);
        }
    });
}
```

看，虽然行数并没有减少，但是你不会犯错了。是的，模板写法主要的目的是 **为了让你不会出错**。Java 语法上的局限让上述写法不可避免的有点显得臃肿，但是，层次的确清晰了一些，不是吗？在这个方面，我也期待这 Java 能向函数式编程靠近一些，提供闭包或者匿名函数，当然，前提是不能损害现在 Java 语言结构清晰易于调试的优点。

4.9.3.5. 在映射中的动态表名

通过Java注解 @One, @Many, @ManyMany，Nutz.Dao 支持对象间的映射。很自然的，对象间的映射自动的会支持动态表名。比如，如果 Company 对象有个成员变量 private List<Employee> employees; 由于 Employee 是动态表名的所以当获取 employee 的时候，同样也能支持动态表名。

4.9.3.5.1. 一对一映射

比如，我们的 Company 对象需要一个新的字段存储 CEO 的 ID，以及另外一个字段存储 CEO 对象本身。按照Nutz.Dao 对于一对一映射的定义：

当对象A有字段f1指向对象B的主键，且在对象A上有字段b类型为B，且声明了@One(target=B.class,field="f1"，则称A.b为对于B的一对一映射

那么，我们的 Company 对象必然会有类似如下的代码：

```
[JAVA]
@Table("t_company")
public class Company {
    @Column
    @Id
    private int id;
    @Column
    @Name
    private String name;
    @Column
    private int ceoId;
    @One(target = Employee.class, field = "ceoId")
    private Employee CEO;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getCeoId() {
```

```
        return ceoId;
    }
    public void setCeoId(int ceoId) {
        this.ceoId = ceoId;
    }
    public Employee getCEO() {
        return CEO;
    }
    public void setCEO(Employee ceo) {
        CEO = ceo;
    }
}
```

比如，我们要在控制台上打印"nutz" 的公司的 CEO 的名字时，代码将为：

```
[JAVA]
public void demoTableName_links_one(final Dao dao) {
    final Company nutz = dao.fetch(Company.class,"nutz");
    TableName.run(nutz.getId(), new Runnable(){
        public void run() {
            dao.fetchLinks(nutz, "CEO");
        }
    });
    System.out.println(nutz.getCEO().getName());
}
```

4.9.3.5.2. 一对多映射

下面我们来增加一对多映射，是的，一个公司有很多雇员，不是吗？那么我们就为 Company 类增加一个新的字段

```
[JAVA]
@Table("t_company")
public class Company {
    // ... another fields ...
    @Many(target = Employee.class, field = "companyId")
    private List<Employee> employees;
    public List<Employee> getEmployees() {
```

```
        return employees;
    }
    public void setEmployees(List<Employee> employees) {
        this.employees = employees;
    }
    // ... another methods ...
}
```

可以看到，在字段 employees 增加了 @Many 说明，就像一般的一对多映射一样， field 项声明了 Employee 类必须有一个名叫（companyId）的字段指向 Company 的主键。所以 Employee 类的代码为：

```
[JAVA]
@Table("t_employee_${cid}")
public class Employee {
    @Column
    @Id
    private int id;
    @Column
    @Name
    private String name;
    @Column("comid")
    private int companyId;
    public int getCompanyId() {
        return companyId;
    }
    public void setCompanyId(int companyId) {
        this.companyId = companyId;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
```

```
    }
}
```

一切设置完毕，我们就可以这么调用：

```
[JAVA]
public void demoTableName_links_many(final Dao dao) {
    final Company nutz = dao.fetch(Company.class, "nutz");
    TableName.run(nutz.getId(), new Runnable() {
        public void run() {
            dao.fetchLinks(nutz, "employess");
        }
    });
    for (Employee e : nutz.getEmployees())
        System.out.println(e.getName());
}
```

上面的程序会逐行打印出 nutz 公司所有的雇员名称。

4.9.3.5.3. 多对多映射

多对多映射是通过一个中间表进行的数据关联，比如数据库中有数据表 A,B， 可以在建立一张表 C，描述 A 表和 B 表的数据关联。一般的关联表至少有两个字段，一个是 A 表的主键，另一个记录 B 表的主键。如果是复合主键或者要记录关联的权重，关联表的设计将会更加复杂。

Nutz.Dao 提供了 @ManyMany 注解帮助你的 POJO 来声明多对多关联，比如在 Employee 类中可以增加一个新的字段，表示某个雇员的下属。

```
[JAVA]
@ManyMany(target = Employee.class, relation =
"t_employee_staff_${cid}", from = "eid", to = "sid")
private List<Employee> staffs;
```

请注意

* relation 项就是关联表的名称，这个名称也可以写成动态的。

* from 项是关联表字段的名称，将对应到本 POJO 的主键，这里的 eid 将对应 Employee.id
* to 项也是关联表字段的名称，将对应到 target 项声明的主键，这里的 sid 也将对应 Employee.id

在调用代码里可以这样调用

```
[JAVA]
public void demoTableName_links_manymany(final Dao dao) {
    final Company nutz = dao.fetch(Company.class, "nutz");
    TableName.run(nutz.getId(), new Runnable() {
        public void run() {
            Employee peter = dao.fetch(Employee.class,"Peter");
            dao.fetchLinks(peter, "staffs");
            for (Employee e : peter.getStaffs())
                System.out.println(e.getName());
        }
    });
}
```

这段代码将 nutz 公司雇员 Peter 的所有下属逐行打印出来。

4.9.4. 总结一下

关于动态表名的这一节写的比较长，因为我认为动态表名的支持，Nutz.Dao 是比较独特的。它基本做到了这两个效果

1. 如果你不希望使用动态表名，你根本不会看到 Nutz.Dao 关于动态表名的设计
2. 如果你希望使用动态表名，你并不需要学习更多的配置方法

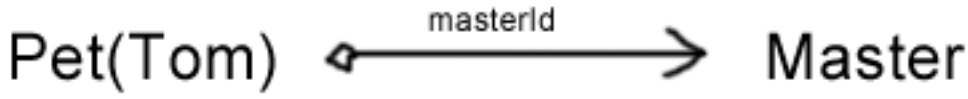
并不是因为我是 Nutz 的作者而努力的在为自己吹嘘，如果你细心体会 Nutz 各个模块的设计，所有的设计基本是本着上述两个原则，即需要的时候你会看见，不需要的时候尽量让你看不见。由于我是个职业界面设计师，所以我会自然而然的将我设计 UE 时很多原则应用在编程接口的设计上，事实上我发现，这的确在某种程度上让程序接口更简洁更轻便了，所以自然也就会对程序员更友好了。当然我并不否认 Nutz 可能依然存在一些脑残设计，我和你们一样不能忍受它们，如果发现它们请第一时间通知我。在讨论区发个贴就是个很好的办法。

4.10. 一对一映射

4.10.1. 什么是一对一映射

有两张数据表，其中A表的某个字段的值指向B表的主键。因为A表的任何一条记录只能对应B表的一条且唯一一条记录，所以称这种映射为A表对B表数据的一对一映射。(当然，反过来，你也可是说，是B表对A表的一对多映射)。

上述结构，如果用 POJO 来表示的话，可以参看下图：



如上图， Pet 中就可以有一个字段 master，通过自身的 masterId 指向一个Master 对象，那么我们说 Pet.master 就是 Pet 对Master 的一对一映射。

4.10.2. 在 POJO 中配置一对一映射

在 POJO 类中字段中增加注解 **@One**：

```
[JAVA]
@Table("t_pet")
public class Pet extends Pojo {
    @Column
    public int masterId;
    @One(target = Master.class, field = "masterId")
    public Master master;
}
```

在 Pet 对象中必须存在一个 Master 类型的字段，你的一对一映射就需要配置在这个字段上。通过 **@One** 注解告诉 Nutz.Dao 对象 Pet 和 Master 对象的关系，其中：

* target 表示你要映射的对象类型
* field 表示你打算依靠**本对象**的哪一个字段来映射目标对象的主键

因此：

* POJO 类中**必须**存在一个字段，本 POJO 将通过该字段同目标 POJO 类的主键 关联
* 该字段必须同目标 POJO （Master）的主键类型相同
* **注意**，这里是大小写敏感的。

4.10.3. 插入操作

如果你已经实现准备好了这样的对象：

```
[JAVA]
Pet pet = new Pet();
pet.setName("XiaoBai");
Master master = new Master();
master.setName("Peter");
pet.setMaster(master);
```

那么你可以一次将 pet 以及它对应的 master 一起插入到数据表中

```
[JAVA]
dao.insertWith(pet,"master");
```

Nutz.Dao 会根据正则表达式 "master" 寻找可以被匹配上的映射字段（只要声明了 @One, @Many, @ManyMany 任何一个注解，都是映射字段）并根据注解具体的配置信息，执行相应的 SQL。比如上面的操作，会实际上：

```
[CODE]
执行 SQL : INSERT INTO t_master (name) VALUES("Peter");
执行 SQL 获取 最大值： SELECT MAX(id) FROM t_master  // 假设返回的值是 29
将该最大值 29 赋给 master 对象的主键 id
将该最大值 29 赋给 pet.masterId 字段
执行 SQL : INSERT INTO t_pet (name,masterId) VALUES("Xiaobai",29)
```

这里通过 SELECT MAX 来获取插入的最大值，是默认的做法，如果你想修改这个默认做法，请参看 关于主键一章。

* 这里因为是一对一映射，所以会首先插入映射对象，以便用新的主键值更新主对象的映射字段
* 如果你的对象中包括多个 @One 字段，被你的正则式匹配上，那么这些字段对应的字段（如果不为null）都会被匹配，并首先被插入

当然，你要想选择仅仅只插入映射字段的话，你可以：

```
[JAVA]
dao.insertLinks(pet,"master");
```

那么上述操作实际上会执行：

```
[CODE]
执行 SQL : INSERT INTO t_master (name) VALUES("Peter");
执行 SQL 获取 最大值： SELECT MAX(id) FROM t_master  // 假设返回的值是 29
将该最大值 29 赋给 master 对象的主键 id
```

看，并不会插入 pet 对象。

4.10.4. 获取操作

仅仅获取映射对象：

```
[CODE]
Pet pet = dao.fetch(Pet.class, "XiaoBai");
dao.fetchLinks(pet, "master");
```

这会执行操作：

```
[CODE]
执行 SQL: SELECT * FROM t_pet WHERE name='XiaoBai';  // 如果 pet.masterId 是 29
```

```
执行 SQL: SELECT * FROM t_master WHERE id=29;
```

但是 Nutz.Dao 没有提供一次获取 pet 对象以及 master 对象的方法，因为，你完全可以把上面的两句话写在一行上：

```
[JAVA]
Pet pet = dao.fetchLinks(dao.fetch(Pet.class, "XiaoBai"), "master");
```

然后，你可以通过 pet.getMaster() 得到 Nutz.Dao 为 pet.master 字段设置的值。

4.10.5. 更新操作

同时更新 pet 和 master

```
[JAVA]
dao.updateWith(pet, "master");
```

这会执行

```
[CODE]
执行SQL: UPDATE t_master ....
执行SQL: UPDATE t_pet ...
```

仅仅更新 master

```
[JAVA]
dao.updateLinks(pet, "master");
```

这会执行

```
[CODE]
执行SQL: UPDATE t_master ....
```

4.10.6. 删除操作

同时删除 pet 和 master

```
[JAVA]
dao.deleteWith(pet, "master");
```

仅仅删除 master

```
[JAVA]
dao.deleteLinks(pet, "master");
```

清除 master

```
[JAVA]
dao.clearLinks(pet, "master");
```

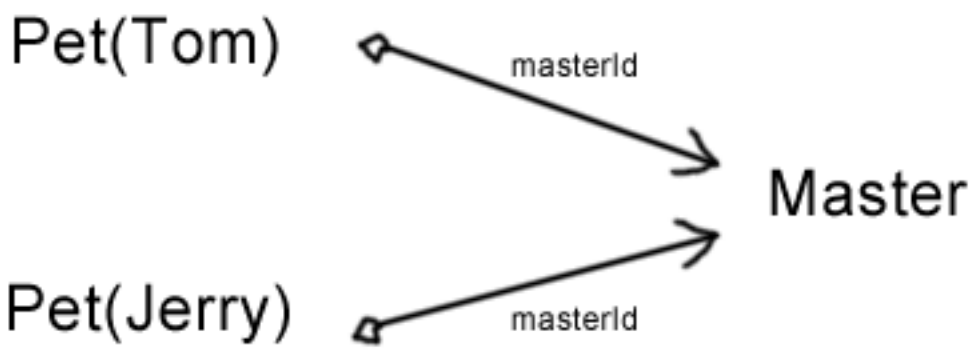
对于一对一映射来说其实清除和删除是等价的操作，对于一对多和多对多映射来说，就又区别了，因为清除只会执行一条 SQL 删除一批，而且删除会逐个调用 dao.delete 来删除对象

4.11. 一对多映射

4.11.1. 什么是一对多映射

有两张数据表，其中A表的某个字段的值指向B表的主键。因为B表的任何一条记录理论上可以对应A表的多条记录，所以称这种映射为B表对A表数据的一对多映射。(当然，反过来，你也可是说，是A表对B表的一对一映射)。

上述结构，如果用 POJO 来表示的话，可以参看下图：



如上图，一个 Master 自然就能对应多个 Pet ，所以， Master.pets （一个 List<Pet> ）就可以指向多个 Pet 对象，那么我们说 Master.pets 就是 Master 对 Pet 的一对多映射。

4.11.2. 在 POJO 中配置一对多映射

在 POJO 类中字段中增加注解 **@Many**：

```
[JAVA]
@Table("t_master")
public class Master extends Pojo {
    @Many(target = Pet.class, field = "masterId")
    private List<Pet> pets;
    public List<Pet> getPets() {
        return pets;
    }
    public void setPets(List<Pet> pets) {
        this.pets = pets;
    }
}
```

在 Master 对象中必须存在一个 List<Pet> 类型的字段，你的一对多映射就需要配置在这个字段上。通过 **@Many** 注解告诉 Nutz.Dao 对象 Pet 和 Master 对象的关系，其中：

* target 表示你要映射的对象类型
* field 表示你打算依靠**目标对象**的哪一个字段来映射本对象的主键

因此：

* 目标 POJO 类 （Pet）中**必须**存在一个字段，用来同本 POJO POJO 类的主键关联
    > **还要注意**，这里的名称是 目标 POJO 的 **JAVA** 字段的名称。
    > **注意**，这里是大小写敏感的。
* 该字段必须同本 POJO 类的主键类型相同

4.11.3. 你不仅可以在集合类型字段上声明一对多映射

本 POJO 类的 @Many 映射，可以不止声明在 List 对象上，它还可以声明在

* 数组
* Map
* POJO

4.11.3.1. 数组

例如：

```
[JAVA]
@Table("t_master")
public class Master extends Pojo {
    @Many(target = Pet.class, field = "masterId")
    private Pet[] pets;
    // ... 省略其余代码
```

当采用 fetchLinks 获取值的时候，会自动填充此数组

4.11.3.2. Map

如果采用 Map 类型， 我们还需要你为 @Many 注解多添加一个参数，通过这个参数, Nutz.Dao 才能知道采用目标 POJO 对象的哪一个字段来作为 Map 的键。

```
[JAVA]
@Table("t_master")
public class Master extends Pojo {
    @Many(target = Pet.class, field = "masterId", key="name")
    private Map<String,Pet> pets;
    // ... 省略其余代码
```

其中：

* key 所指的字段 name，表示 Pet 对象的 name 字段， Nutz.Dao 将采用这个字段作为 Map 的键
    > 为目标 POJO 类的 JAVA 字段名
    > 大小写敏感
* 请注意，将 Map 的第一个泛型参数设置正确，同 key 所指向的字段类型相同即可。

4.11.3.3. POJO

例如：

```
[JAVA]
@Table("t_master")
public class Master extends Pojo {
    @Many(target = Pet.class, field = "masterId")
    private Pet pet;
    // ... 省略其余代码
```

则会从 Pet 对象的数据表中，选取**第一个** masterId 为当前对象主键值的 Pet 对象。至于什么是“**第一**”不同的数据库有所不同。总之，就是 SQL 语句：

```
[SQL]
SELECT * FROM t_pet;
```

选出的结果集中的第一个记录。

4.11.4. 插入操作

如果你已经实现准备好了这样的对象：

```
[JAVA]
Master master = new Master();
master.setName("Peter");
List<Pet> pets = new ArrayList<Pet>();
pets.add(new Pet("XiaoBai");
pets.add(new Pet("XiaoHei");
master.setPets(pets);
```

那么你可以一次将 master 以及它对应的 pets 一起插入到数据表中

```
[JAVA]
dao.insertWith(master, "pets");
```

Nutz.Dao 会根据正则表达式 "pets" 寻找可以被匹配上的映射字段（只要声明了 @One, @Many, @ManyMany 任何一个注解，都是映射字段）并根据注解具体的配置信息，执行相应的 SQL。比如上面的操作，会实际上：

```
[CODE]
执行 SQL : INSERT INTO t_master (name) VALUES("Peter");
执行 SQL 获取 最大值： SELECT MAX(id) FROM t_master  // 假设返回的值是 29
将该最大值 29 赋给 master 对象的主键 id
循环 master.pets，将该最大值 29 赋给每一个 pet 对象的 pet.masterId 字段
执行 SQL : INSERT INTO t_pet (name,masterId) VALUES("XiaoBai",29)
执行 SQL : INSERT INTO t_pet (name,masterId) VALUES("XiaoHei",29)
```

这里通过 SELECT MAX 来获取插入的最大值，是默认的做法，如果你想修改这个默认做法，请参看 关于主键一章。

* 这里因为是一对多映射，所以会首先插入主对象，以便用新的主键值更新映射对象的映射字段
* 如果你的对象中包括多个 @Many 字段，被你的正则式匹配上，那么这些字段对应的字段（如果不为null）都会被匹配，并首先被插入

当然，你要想选择仅仅只插入映射字段的话，你可以：

```
[JAVA]
dao.insertLinks(master,"pets");
```

那么上述操作实际上会执行：

```
[CODE]
循环 master.pets，将该master.id (主键) 赋给每一个 pet 对象的
pet.masterId 字段，我们假设该值为 29
执行 SQL : INSERT INTO t_pet (name,masterId) VALUES("XiaoBai",29)
执行 SQL : INSERT INTO t_pet (name,masterId) VALUES("XiaoHei",29)
```

看，并不会插入 master 对象。

4.11.5. 获取操作

仅仅获取映射对象：

```
[CODE]
Master master = dao.fetch(Master.class, "Peter");
dao.fetchLinks(master, "pets");
```

这会执行操作：

```
[CODE]
执行 SQL: SELECT * FROM t_master WHERE name='Peter'; // 如果
master.id 是 12
执行 SQL: SELECT * FROM t_pet WHERE masterId=12;
```

但是 Nutz.Dao 没有提供一次获取 master 对象以及 pets 对象的方法，因为，你完全可以把上面的两句话写在一行上：

```
[JAVA]
Master master = dao.fetchLinks(dao.fetch(Master.class, "Peter"),
"pets");
```

然后，你可以通过 master.getPets() 得到 Nutz.Dao 为 master.pets 字段设置的值。

4.11.6. 更新操作

同时更新 pet 和 master

```
[JAVA]
dao.updateWith(master, "pets");
```

这会执行

```
[CODE]
执行SQL: UPDATE t_master ....
循环 master.pets 并依次执行SQL: UPDATE t_pet ...
```

仅仅更新 pets

```
[JAVA]
dao.updateLinks(master, "pets");
```

这会执行

```
[CODE]
循环 master.pets 并依次执行SQL: UPDATE t_pet ...
```

4.11.7. 删除操作

同时删除 master 和 pets

```
[JAVA]
dao.deleteWith(master, "pets");
```

仅仅删除 pets

```
[JAVA]
dao.deleteLinks(master, "pets");
```

清除 pets

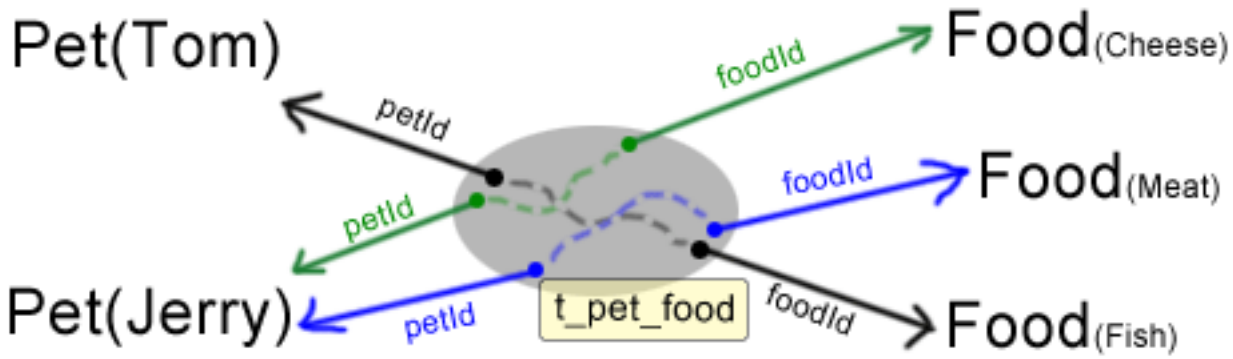
```
[JAVA]
dao.clearLinks(master, "pets");
```

清除同删除的区别在于，清除只会执行一条 SQL 删除一批映射对象，而且删除会逐个调用 dao.delete 来删除对象

4.12. 多对多映射

4.12.1. 什么是多对多映射

有两张数据表，通过第三张数据表来表示关联关系，我们称之为多对多的映射



如上图，通过一个中间数据表的两个字段，分别指向两个对象的主键，可以实现多对多映射。所以，Pet.foods（一个 List<Food>）或者 Food.pets（一个List<Pet>）就是多对多映射。

4.12.2. 在 POJO 中配置多对多映射

在 POJO 类中字段中增加注解 **@ManyMany**：

```
[JAVA]
@Table("t_food")
public class Food extends Pojo {
    @ManyMany(target = Pet.class, relation = "t_pet_food", from =
"foodid", to = "petid")
    private List<Pet> pets;
    public List<Pet> getPets() {
        return pets;
    }
    public void setPets(List<Pet> pets) {
        this.pets = pets;
    }
}
```

在 Food 对象中必须存在一个 List<Pet> 类型的字段，你的多对多映射就需要配置在这个字段上。通过 **@ManyMany** 注解告诉 Nutz.Dao对象 Food 和 Pet 之间的关系，其中：

* target 表示你要映射的对象类型
* relation 为中间数据表的表名，它也支持动态表名
* from 是中间数据表的字段名，这个字段将储存主对象的主键（上例的 Food 的主键）
* to 是中间数据表的字段名，这个字段将储存映射对象的主键（上例的 Pet 的主键）

因此：

* 数据库中**必须**存在一个中间表 t_pet_food
    > 该表有一个字段 foodid 对应到 Food 对象的主键
    > 该表有一个字段 petid 对应到 Pet 对象的主键
* Nutz.Dao 通过 @ManyMany 这四个属性了解到：
    > 目标的 POJO 类 ： Pet
    > 关联表（或者说：中间表）：t_pet_food
    > 关联表的 foodid 字段对应到是本 POJO （Food）主键
    > 关联表的 petid 字段对应到是目标 POJO （Pet） 主键

4.12.3. 你不仅可以在集合类型字段上声明一对多映射

本 POJO 类的 @Many 映射，可以不止声明在 List 对象上，它还可以声明在

* 数组
* Map
* POJO

详情，可以参看 一对多映射 的相关描述

4.12.4. 插入操作

如果你已经实现准备好了这样的对象：

```
[JAVA]
Food food = new Food("Fish");
List<Pet> pets = new ArrayList<Pet>();
pets.add(new Pet("XiaoBai");
pets.add(new Pet("XiaoHei");
food.setPets(pets);
```

那么你可以一次将 food 以及它对应的 pets 一起插入到数据表中，并在关联表中插入对应的记录

```
[JAVA]
dao.insertWith(food, "pets");
```

Nutz.Dao 会根据正则表达式 "pets" 寻找可以被匹配上的映射字段（只要声明了 @One, @Many, @ManyMany 任何一个注解，都是映射字段）并根据注解具体的配置信息，执行相应的 SQL。比如上面的操作，会实际上：

```
[CODE]
执行 SQL : INSERT INTO t_food (name) VALUES("Fish");
执行 SQL 获取 最大值： SELECT MAX(id) FROM t_food  // 假设返回的值是 6
循环 food.pets
    执行 SQL: INSERT INTO t_pet (name) VALUES("XiaoBai");
    执行 SQL 获取 最大值： SELECT MAX(id) FROM t_pet  // 假设返回的值是 97
    执行 SQL 插入关联: INSERT INTO t_pet_food (foodid, petid) VALUES(6, 97);
    ...
```

这里通过 SELECT MAX 来获取插入的最大值，是默认的做法，如果你想修改这个默认做法，请参看 关于主键一章。

* 这里因为是多对多映射，所以会首先插入主对象并循环插入映射对象，以便获得双发的主键
* 如果你的对象中包括多个 @ManyMany 字段，被你的正则式匹配上，那么这些字段对应的字段（如果不为null）都会被匹配，一次被执行

当然，你要想选择仅仅只插入映射字段的话，你可以：

```
[JAVA]
dao.insertLinks(food,"pets");
```

如果 food.id 的值为 6，那么上述操作实际上会执行：

```
[CODE]
循环 food.pets
    执行 SQL: INSERT INTO t_pet (name) VALUES("XiaoBai");
    执行 SQL 获取 最大值： SELECT MAX(id) FROM t_pet  // 假设返回的值是 97
    执行 SQL 插入关联: INSERT INTO t_pet_food (foodid, petid) VALUES(6, 97);
    ...
```

看，并不会插入 food 对象。

如果你已经存在了 food 和 pets 对象，你仅仅打算将它们关联起来，那么你可以

```
[JAVA]
dao.insertRelation(food,"pets");
```

如果 food.id 的值为 6，那么上述操作实际上会执行：

```
```

```
[CODE]
循环 food.pets
    执行 SQL 插入关联: INSERT INTO t_pet_food (foodid, petid)
VALUES(6, 97);
    ...
```

看，仅仅只会插入 food 和 pets 的关联

4.12.5. 获取操作

仅仅获取映射对象：

```
[CODE]
Food food = dao.fetch(Food.class, "Fish");
dao.fetchLinks(food, "pets");
```

这会执行操作：

```
[CODE]
执行 SQL: SELECT * FROM t_food WHERE name='Fish'; // 如果 food.id
是6
执行 SQL: SELECT * FROM t_pet WHERE id IN (SELECT petid FROM
t_pet_food WHERE foodid=6)
```

但是 Nutz.Dao 没有提供一次获取 food 对象以及 pets 对象的方法，因为，你完全可以把上面的两句话写在一行上：

```
[JAVA]
Food food = dao.fetchLinks(dao.fetch(Food.class, "Fish"), "pets");
```

然后，你可以通过 food.getPets() 得到 Nutz.Dao 为 food.pets 字段设置的值。

4.12.6. 更新操作

同时更新 food 和 pet

```
[JAVA]
dao.updateWith(food, "pets");
```

这会执行

```
[CODE]
执行SQL: UPDATE t_food ....
循环 food.pets 并依次执行SQL: UPDATE t_pet ...
```

仅仅更新 pets

```
[JAVA]
dao.updateLinks(food, "pets");
```

这会执行

```
[CODE]
循环 food.pets 并依次执行SQL: UPDATE t_pet ...
```

4.12.7. 删除操作

同时删除 food 和 pets

```
[JAVA]
dao.deleteWith(food, "pets");
```

仅仅删除 pets

```
[JAVA]
dao.deleteLinks(food, "pets");
```

清除 pets

```
[JAVA]
dao.clearLinks(food, "pets");
```

清除同删除的区别在于，清除只会执行一条 SQL 删除 t_pet_food 的记录，但是 t_pet 和 t_food 表中的数据不会被删除而删除则不仅会清除 t_pet_food 里的记录，还会逐个调用 dao.delete 来删除对象。

4.13. 自定义 SQL

4.13.1. Nutz.Dao 自定义 SQL 的概述

Nutz.Dao 提供了大多数简单的操作，在80%以上的情况下，你并不需要编写 SQL，因为 Nutz.Dao 会自动替你生成可以使用的 SQL。但是，在某些特殊的情况下，尤其是考虑到效率等问题，直接写作 SQL 仍然是程序员们的一个杀手锏，有了这个杀手锏，程序员们永远可以针对任何数据库做他们想要的任何操作。

在之前的时代，很多程序员将 SQL 代码同 Java 代码混杂在一起，即所谓的**硬编码**。硬编码通常是不好的，所以很多程序员都采用了各种办法将 SQL 提炼出来存放在一个独立的文件中。其中比较著名的一个框架就是 iBatis。这个小巧的 SQL 映射框架（Nutz.Dao 比它还小）在这个领域里干的不错。缺省的它将 SQL 存放在 XML 文件中，现在最新的 iBatis3 也提供了JAVA注解的写法。但是我并不认为 XML 文件或者是 JAVA注解是存放我的 SQL 语句好地方，我认为 SQL 存放的地方，应该是可以用 Eclipse 的 SQL 编辑器打开并且能够被正确语法高亮的一种文本文件。

著名的 Hibernate 提供 HQL， 虽然语法近似于 SQL 但是它必然会有两个不可避免的缺点

1. 对于数据库方言支持的不好
2. 必然会增加使用者的学习曲线

因此，Nutz.Dao 的自定义 SQL 部分的解决方案是：

1. 用户可以硬编码 SQL 语句，比如：

```
[JAVA]
Sql sql = Sqls.create("DELETE FROM t_abc WHERE
name='Peter'");
```

2. 支持占位符的书写方式，比如：

```
[JAVA]
Sql sql = Sqls.create("DELETE FROM $table WHERE
name=@name");
sql.vars().set("table","t_abc");
sql.params().set("name","Peter");
```

    * $table 将会被替换成 t_abc
    * @name 将会被替换成 ?，用来创建 PreparedStatement

3. 用户可以将所有的 SQL 语句存放在一个或者多个文件中，语句的间隔可以通过注释，比如：

```
[CODE]
/* delete.data */
DELETE FROM $table WHERE name LIKE @name
/* update.data */
UPDATE FROM $table SET name=@name WHERE id=@id
```

```
[CODE]
Sql sql = dao.sqls().create("delete.data");
```

4. 你可以为你的 SQL 任意定制回调，后面会有详细讲解

下面我们就由 org.nutz.dao.sql.Sql 接口入手，详细讲解一下 Nutz.Dao 的自定义 SQL 解决方案

4.13.2. Sql 对象 -- org.nutz.dao.sql.Sql

我几乎是不加思索的将 SQL 的实现封装在一个接口后面。现在想想这到也没什么坏处。接口的默认实现是 org.nutz.dao.impl.sql.NutSql。你可以直接 new 这个对象，当然，我也提供了构造 Sql 对象的静态方法:

4.13.2.1. 如何创建 Sql 对象

通过 org.nutz.dao.Sqls 类提供的静态方法 create，你可以很方便的构建你的 Sql 对象

```
[JAVA]
Sql sql = Sqls.create("INSERT INTO t_abc (name,age)
VALUES('Peter',18)");
```

Sqls 提供的

* fetchEntity
* fetchInt
* fetchString
* queryEntity

方法来帮助你构建 Sql 对象。它们之间的区别在稍后会详细说明。

通常的情况，你需要构建某些 **动态** 的 SQL，所以我也允许你为你的 SQL 设置占位符，占位符分两种：

* 变量(var)占位符 - 形式为 $名称
    > 以字符 $ 开头，名称为英文字母，数字，下划线，减号和句点。
    > 正则表达式为：

```
[CODE]
 [$][a-zA-Z0-9_-]
```

    > 在执行 SQL 前，该占位符会被用户设置的值替换
    > 类似 C 语言中的 **宏**
* 参数(param)占位符 - 形式为 @名称
    > 以字符 @ 开头，名称为英文字母，数字，下划线，减号和句点。
    > 正则表达式为：

```
[CODE]
[@][a-zA-Z0-9_-]+
```

    > 在执行 SQL 前，该占位符会被字符 "?" 替换，用来创建 PreparedStatement
    > Nutz.Dao 会自动计算 PreparedStatement的索引值

所有的占位符可以同样的名称出现的多个地方。并且变量占位符和参数占位符的名称不互相干扰，比如:

```
[JAVA]
Sql sql = Sqls.create("INSERT INTO $table ($name,$age,$weight)
VALUES(@name,@age,@weight)");
// 为变量占位符设值
sql.vars().set("table","t_person");
sql.vars().set("name","f_name").set("age","f_age").set("weight","f_weig
ht");
// 为参数占位符设值
sql.params().set("name","Peter").set("age",18).set("weight",60);
```

通过上例，我们可以看出，变量占位符和参数占位符的确可以重名且不相互干扰的。

4.13.2.2. Sql 的逃逸字符

有些时候，有的朋友给出的 SQL 包括特殊字符 '@' 或者 '$'，比如:

```
[CODE]
Sql sql = Sqls.create("INSERT INTO t_usr (name,email)
VALUES('XiaoMing','xiaoming@163.com');"
```

这个时候，因为有关键字 '@'，所以 SQL 不能被正确解析，因为你的本意是给一个 'xiaoming@163.com' 这个字符串。但是 Nutz.Dao 却认为这个是个语句参数。

这时候你可以使用逃逸字符:

```
[CODE]
Sql sql = Sqls.create("INSERT INTO t_usr (name,email)
VALUES('XiaoMing','xiaoming@@163.com');"
```

即

* 输入 "@@" 表示一个 '@'
* 输入 "$$$$" 表示一个 '$'

4.13.2.3. 如何执行 Sql 对象

当你顺利的创建了一个 Sql 对象，执行它就相当简单了，比如：

```
[JAVA]
public void demoSql(Dao dao){
    Sql sql = Sqls.create("SELECT name FROM t_abc WHERE name LIKE
@name");
    sql.params().set("name", "A%");
    dao.execute(sql);
}
```

这就完了吗？我怎么取得查询的结果呢。是的，同 UPDATE, DELETE, INSERT 不同， SELECT 是需要返回结果的，但是 Nutz.Dao 也不太清楚怎样为你自定义的 SELECT 语句返回结果，于是，就需要你设置回调。

4.13.2.4. 回调的用处

接上例，你需要这么改造一下你的函数：

```
[JAVA]
List<String> demoSql(Dao dao) {
    Sql sql = Sqls.create("SELECT name FROM t_abc WHERE name LIKE
@name");
    sql.params().set("name", "A%");
    sql.setCallback(new SqlCallback() {
        public Object invoke(Connection conn, ResultSet rs, Sql sql)
throws SQLException {
            List<String> list = new LinkedList<String>();
            while (rs.next())
                list.add(rs.getString("name"));
            return list;
        }
    });
    dao.execute(sql);
    return sql.getList(String.class);
    // Nutz内置了大量回调, 请查看Sqls.callback的属性
}
```

看到熟悉的 ResultSet 了吧。 当然，如果你执行的不是 SELECT 语句，你依然可以设置回调，但是 ResultSet 参数就是 null了。

总结一下：

1. 回调对象实现接口 org.nutz.dao.sql.SqlCallback，事实上，就像上例所示，这种场景非常适合使用匿名类。
2. 你的回调函数的返回值会存放在 Sql 对象中
3. 调用 sql.getResult() 可以直接返回这个对象
4. sql.getList() 以及 sql.getObject() 方法会泛型安全的替你转型
    * 如果你的对象类型符合要求，则直接返回，否则会通过 Nutz.Castors 替你转换。
    * 对于 getList()，泛型参数用来描述集合内部元素的类型
5. sql.getInt() 会安全的替你将结果转成 int，如果它可以被转成 int 的话，以下是我能想到的列表：
    * 字符串
    * 各种数字类型
    * 字符
    * 布尔类型

4.13.2.5. 获取一个列表的回调

为了更加详细的说明一下回调的用处，我们再下面这个例子：

```
[Java]
Sql.create("SELECT m.* FROM master m JOIN detail d ON
m.d_id=d.d_id WHERE d.name='aa'");
sql.setCallback(Sqls.callback.entities());
sql.setEntity(dao.getEntity(Master.class));
dao.execute(sql);
List<Master> list = sql.getList(Master.class);
```

只要你保证你的 Master 类声明了 @Table 并且每个字段上的 @Column 可以同你的 ResultSet 配置起来那么，上面的代码可以很方便的帮你获取一个 List<Master>.

4.13.2.6. 批量执行

在 Nutz 1.b.38 之后的版本，自定义 SQL 可以支持批量操作

```
[CODE]
Sql sql = Sqls.create("UPDATE t_pet SET name=@name WHERE
id=@id");
sql.params().set("name","XiaoBai").set("id",4);
sql.addBatch();
```

```
sql.params().set("name","XiaoHei").set("id",5);
sql.addBatch();
dao.execute(sql);
```

4.13.3. Nutz.Dao SQL 文件的格式

我们了解了如何构建 Sql 对象，但是一个应用通常由很多 SQL 语句构成，如何管理这些语句呢？前面我说过，我希望：" 用户可以将所有的 SQL 语句存放在一个或者多个文件中，语句的间隔可以通过注释 "。是的这是一种非常简单的纯文本文件，文件里只包含三种信息：

1. SQL 语句
2. SQL 语句的名称 (或者说是键值)。你的程序可以通过语句的名称获取到某一条或几条 SQL 语句
3. 注释 (通常包括在 /* 与 */ 之间)

请注意： 你的 SQL 文件必须为 "**UTF-8**" 编码。

下面是一个例子

```
[SQLs]
/*
这里是这个 SQL 文件的注释，你随便怎么写
*/
/* sql1 */
DROP TABLE t_abc
/* 你可以随便写任何的注释文字，只有距离 SQL 语句最近的那一行注释，才会被认为是键值 */
/* getpet*/
SELECT * FROM t_pet WHERE id=@id
/* listpets*/
SELECT * FROM t_pet $condition
```

4.13.4. 加载 SQL 文件

如何使用上述的 SQL 文件呢，可以将数个 SQL 文件加载到 Dao 对象中。在之后，只要得到 Dao 的对象，可以使用 dao.sqls() 方法获得org.nutz.dao.SqlManager 接口，从这个接口中你就可以获得你预先定义好的 Sql 对象了。

对于 Dao 接口的默认实现， org.nutz.dao.impl.NutDao，提供两个方法，一个是通过构造函数，另一个是 setter 函数。

4.13.4.1. 在构造时加载

```
[JAVA]
Dao dao = new NutDao(datasource,new
FileSqlManager("demo/sqls/all.sqls"));
System.out.println(dao.sqls().count());
```

上述代码将打印出 all.sqls 文件中 SQL 语句的数量。路径 "demo/sqls/all.sqls" 是一个存在在 CLASSPATH 的文件。

* FileSqlManager 的构造函数接受数目可变的字符串对象，每个对象就是 SQL 文件的路径。
* 如不是存在在 CLASSPATH 中的文件，则需要写上绝对路径。
* 如果你给出的 path 是一个目录，那么该目录下所有后缀为**.sqls** 的文件都会被加载

4.13.4.2. 在构造之后的任何时加载

```
[JAVA]
Dao dao = new NutDao(datasource);
((NutDao)dao).setSqlManager(new
FileSqlManager("demo/sqls/all.sqls"));
System.out.println(dao.sqls().count());
```

4.13.5. 条件变量占位符

我认为 Nutz.Dao 比较吸引人的一个函数就是 Dao.query，它允许你用多种方法传入一个条件关于复杂的条件，请参看 复杂的SQL条件

在 Sql 对象中，我在接口里也设计了一个方法：

```
[JAVA]
Sql setCondition(Condition condition);
```

是的，你的 Sql 对象也可以使用 Condition，但是这个 Condition 要如何同你自定义的 SQL 拼装在一起呢，这里，我提供了一个特殊的变量占位符 -- 条件变量占位符 $condition

4.13.5.1. 特殊的占位符 -- null

唯一需要说明的是，在你写作的 SQL 中，需要声明一个特殊的占位符，比如下面的代码输出所有 id 大于 35 的 Pet 对象的名称

```
[JAVA]
Sql sql = Sqls.create("SELECT name FROM t_pet $condition");
sql.setCondition(Cnd.where("id", ">", 35)).setCallback(new
SqlCallback() {
    public Object invoke(Connection conn, ResultSet rs, Sql sql) throws
SQLException {
        List<String> list = new LinkedList<String>();
        while (rs.next())
            list.add(rs.getString("name"));
        return list;
    }
});
dao.execute(sql);
for (String name : sql.getList(String.class))
    System.out.println(name);
```

请主要看看这两行代码：

```
[JAVA]
Sql sql = Sqls.create("SELECT name FROM t_pet $condition");
sql.setCondition(Cnd.where("id", ">", 35));
```

第一行的占位符 $condition 已经被 Nutz.Dao 保留。声明了该占位符的 SQL 都可以使用 setCondition 函数。否则，你设置的条件将无效。

另外一个例子 - **将所有的 id 大于 35 的 Pet 对象的 masterId 设置为 45**

```
[JAVA]
void demoCondition2(Dao dao){
    Sql sql = Sqls.create("UPDATE t_pet SET masterid=@masterId
$condition");
    sql.params().set("masterId", 45);
    sql.setCondition(Cnd.wrap("id>35"));
```

```
    dao.execute(sql);
}
```

4.13.5.2. 使用 Entity<?>

Nutz.Dao 会将你的 POJO 预先处理，处理的结果就是 Entity<?>。你可以通过 Dao 接口的 getEntity() 方法获取。你通过实体注解配置的信息，尤其是 @Column 中配置的数据库字段的名字（当数据库字段名同 Java 字段名不同时）尤其有用。Condition 接口的 toString(Entity<?>) 方法是你唯一要实现的方法，如果你将一个 Condition 赋个了 Sql 对象，在生成真正 SQL 语句的时候，这个 Entity<?>又要从那里来呢？答案是，(*你要预先设置}。

如果你不设置 Entity<?>，那么你的 Condition 的 toString(Entity<?>) 参数就是 null。你可以通过 Dao接口随时获取任何一个 POJO 的Entity<?>，但是如何设置给你的 Condition 呢，答案是，通过 Sql 对象。

```
[JAVA]
void demoEntityCondition(Dao dao) {
    Sql sql = Sqls.create("UPDATE t_pet SET masterid=@masterId
$condition");
    Entity<Pet> entity = dao.getEntity(Pet.class);
    sql.setEntity(entity).setCondition(new Condition() {
        public String toString(Entity<?> entity) {
            return String.format("%s LIKE 'Y%'",
entity.getField("name").getColumnName());
        }
    });
    dao.execute(sql);
}
```

很多时候，大量的 SQL 语句就是为了能够查出一些 POJO 对象，因此，我给你内置了两个回调，这两个回调都需要你为你的 Sql 设置一个正确的 Entity<?>

4.13.5.2.1. 获取实体的回调

```
[JAVA]
Pet demoEntityQuery(Dao dao) {
    Sql sql = Sqls.create("SELECT * FROM t_pet $condition");
    sql.setCallback(Sqls.callback.entity());
    Entity<Pet> entity = dao.getEntity(Pet.class);
```

```
    sql.setEntity(entity).setCondition(Cnd.wrap("id=15"));
    dao.execute(sql);
    return sql.getObject(Pet.class);
}
```

为了方便起见，你可以直接使用 Sqls.fetch 来创建你的 Sql 对象，这个函数会自动为你的 Sql 设置获取实体的回调

```
[JAVA]
Pet demoEntityQuery(Dao dao) {
    Sql sql = Sqls.fetchEntity("SELECT * FROM t_pet $condition");
    Entity<Pet> entity = dao.getEntity(Pet.class);
    sql.setEntity(entity).setCondition(Cnd.wrap("id=15"));
    dao.execute(sql);
    return sql.getObject(Pet.class);
}
```

4.13.5.2.2. 查询实体的回调

```
[JAVA]
List<Pet> demoEntityQuery(Dao dao) {
    Sql sql = Sqls.create("SELECT * FROM t_pet $condition");
    sql.setCallback(Sqls.callback.entities());
    Entity<Pet> entity = dao.getEntity(Pet.class);
    sql.setEntity(entity).setCondition(Cnd.wrap("id=15"));
    dao.execute(sql);
    return sql.getList(Pet.class);
}
```

那么，我提供了一个 Sqls.queryEntity 函数也就不奇怪了吧。 :)

4.13.6. 分页

这是1.b.43才添加的功能!

4.13.6.1. 一直被投诉,终于被实现!

```
```

```
[JAVA]
Sql sql = Sqls.queryEntity("SELECT * FROM t_pet");
sql.setPager(dao.createPager(2,20));
sql.setEntity(dao.getEntity(Pet.class));
dao.execute(sql);
return sql.getList(Pet.class);
```

提醒一下,SqlServer数据库下, 用于分页的sql对象,不要重复使用!!

4.14. 事务模板

4.14.1. 为什么提供事务模板

截至到现在为止，除非你使用 dao.execute(Sql ...) ，一次执行多个 SQL，是事务安全的，其他的情况均是事务不安全的，比如如下代码：

```
[JAVA]
Pet pet1 = dao.fetch(Pet.class,"XiaoBai");
Pet pet2 = dao.fetch(Pet.class,"XiaoHei");
pet1.setNickname("BaiBai");
pet2.setNickname("HeiHei");
dao.update(pet1);
dao.update(pet2);
```

尤其是请关注最后两句:

```
[JAVA]
dao.update(pet1);
dao.update(pet2);
```

当第二句话抛出异常的时候，第一句话不能被回滚。这两条调用就是不事务安全的。如果我想让 pet1 和 pet2的更新操作是原子性的，它们必须一同成功，一同失败，怎么办呢？

4.14.2. 使用事务模板

Nutz.Dao 提供了简单的解决办法： **事务模板**

4.14.2.1. 一段示例代码

上一节的例子可以修改为:

```
[JAVA]
final Pet pet1 = dao.fetch(Pet.class,"XiaoBai");
final Pet pet2 = dao.fetch(Pet.class,"XiaoHei");
pet1.setNickname("BaiBai");
pet2.setNickname("HeiHei");
// Begin transaction
Trans.exec(new Atom(){
    public void run() {
        dao.update(pet1);
        dao.update(pet2);
    }
});
// End transaction
```

提供一个 org.nutz.trans.Atom 接口的匿名实现，在里面执行的所有的 Dao 操作都是原子性的，因为它们在同一个 “**原子** (Atom)” 里。

4.14.2.2. 事务的关键就是原子的界定

事务最核心的是原子的界定，在 Nutz.Dao中，界定原子的方法出奇的简单，借助匿名类，你可以随时将一段代码用你的原子实现包裹住。而 Trans.exec() 方法接受**数目可变的原子**，每个原子都是事务性的。

Trans.exec 的函数声明

```
[JAVA]
public static void exec(Atom... atoms);
```

被原子实现包裹住的代码就是事务安全的，无论它同时操作了多少个 DataSource。Nutz.Dao 提供的原子接口非常简单，实际上它就是 java.lang.Runnable 的一个别名，下面就是这个接口的全部代码：

```
[JAVA]
```

```
package com.zzh.trans;
public interface Atom extends Runnable {}
```

这几乎是我写过的最简单的 Java 类了，正是因为它简单，所以才有无限的威力。你如果查看过 Nutz 的源代码包，在和数据库操作的地方，你总会和 Atom 不期而遇。很多朋友曾经很不适应匿名类的写法，是的，我在早期写 Java 的时候也比较讨厌匿名类，但是熟悉了以后，你会真正喜欢上这个东西，就像你写 Javascript 的一段时候以后，多数人都会喜欢上“闭包”一样。你可以把匿名类当作 Java 给你的闭包。

采用事物模板的来界定事物有一个缺点，这是 Java 语言带来的限制：你有可能需要将一些相关的变量声明成 final 的。并且在 run 函数中，你只能向外抛 RuntimeException 或其子类。

4.14.2.3. 设置事务的级别

在 JDBC 的 java.sql.Connection 接口中定义的 setTransactionIsolation 函数可以设置事务的级别Nutz.Dao 也提供另外一个静态函数，允许你设置事务的级别:

Trans.exec 的函数声明

```
[JAVA]
public static void exec(int level, Atom... atoms);
```

这里的第一个参数 level 和 java.sql.Connection 接口中的 setTransactionIsolation 规定的 level 是一样的。下面是在 java.sql.Connection 里面关于 level 参数的 JDoc 说明：

它可以是下列常量中的任意一个值：

* Connection.TRANSACTION_READ_UNCOMMITTED
* Connection.TRANSACTION_READ_COMMITTED
* Connection.TRANSACTION_REPEATABLE_READ
* Connection.TRANSACTION_SERIALIZABLE

注意： 不能使用常量 Connection.TRANSACTION_NONE，因为它的意思是“不支持事务”

关于 level 参数的更多说明，请参看java.sql.Connection 的文档

不同的数据库，对于 JDBC 事务级别的规范，支持的力度不同。请参看相应数据库的文档，已确定你设置的数据库事务级别是否被支持。

4.14.2.4. 事务的嵌套

Nutz 的事务模板可以嵌套吗？ 答案是肯定的。事实上，Nutz 支持事务模板的无限层级嵌套。

这里，如果每一层嵌套，指定的事务级别有所不同，不同的数据库，可能引发不可预知的错误。所以，嵌套的事务模板的事务，将以最顶层的事务为级别为标准。就是说，如果最顶层的事务级别为 'TRANSACTION_READ_COMMITTED'，那么下面所包含的所有事务，无论你指定什么样的事务级别，都是 'TRANSACTION_READ_COMMITTED'， 这一点，由抽象类 Transaction 来保证。其 setLevel 当被设置了一个大于 0 的整数以后，将不再 接受任何其他的值。

你可以通过继承 Transaction 来修改这个默认的行为，当然，这个行为修改一般是没有必要的。

另外，你还可能需要知道，通过 Trans.setup 方法，能让整个虚拟机的 Nutz 事务操作都使用你的 Transaction 实现

下面我给出两个例子:

最外层模板决定了整个事务的级别:

```
[CODE]
Trans.exec(Connection.TRANSACTION_READ_COMMITTED, new
Atom(){
    public void run(){
        dao.update(xxx);
        dao.update(bbb);
        // 在下层模板，虽然你指定了新的事务级别，但是这里的事务级别还
是
        // 'TRANSACTION_READ_COMMITTED'。在一个事务中，级别一旦
设定就不可更改
        Trans.exec(Connection.TRANSACTION_SERIALIZABLE, new
Atom(){
            public void run(){
                dao.update(CCC);
                dao.update(EEE);
            }
        });
    }
});
```

让整个函数都是事务的:

```
[CODE]
public void updatePet(final Pet pet){
    Trans.exec(new Atom(){
        public void run(){
            dao.update(pet);
            dao.update(pet.getMaster());
        }
    });
}
// 在另外一个函数里，可以这么使用
public void updateDogAndCat(final Pet dog, final Pet cat){
    Trans.exec(new Atom(){
        public void run(){
            updatePet(dog);
            updatePet(cat);
        }
    });
}
```

4.14.3. 扩展实现

org.nutz.trans.Trans 类的 exec()方法，接受数目可变的 Atom 实例，足够方便了吧。 但是它默认只能支持在一台机器上保证事务性，就是在一个 JVM 里保证代码的事务性。如果跨越多个 JVM一起组合的 Service，如何保证事务性呢，很抱歉，Nutz.Dao 的第一版的实现里不包括跨越多个JVM依然保证事务性的功能，但是你如果真的需要这个功能也没关系，你可以自己写一个 org.nutz.trans.Transaction 的实现，然后在你的应用启动时，通过

```
[JAVA]
org.nutz.trans.Trans.setup(你的实现类)
```

替换 Nutz.Dao 的默认实现。

4.14.4. 总结一下 Nutz.Dao 事务

* org.nutz.trans.Trans 类提供了两个函数 exec
    > 一个接受数目可变的 Atom 对象
    > 一个接受一个整型值用以界定本事务的级别，以及一个数目可变的 Atom 对象
* Atom 类就是 java.lang.Runnable 的一个别名
* 在一个 Atom 里，无论同时操作多少 DataSource，都是事务安全的(由于不是使用 XADataSource,无法100%保证)

* 你可以通过实现自己的 Transaction 实现类，扩展 Nutz.Dao 对于事务的支持

4.15. 交叉事务

```
[CODE]
函数 A
    数据操作 1;
    数据操作 2;
函数 B
    数据操作 3;
    -> 函数 A();
函数 C
    数据操作 4;
    -> 函数 A();
```

则称，A 为 B,C 的交叉操作。如果，A,B,C 都需要保证事务性，则 A 为 B, C 的交叉事务

Nutz.Dao 的原子操作支持事务嵌套，所以你可以这么实现这三个函数：

```
[CODE]
函数 A
    Trans.exec(new Atom(){
        public void run(){
            数据操作 1;
            数据操作 2;
        }
    });
函数 B
    Trans.exec(new Atom(){
        public void run(){
            数据操作 3;
            -> 函数 A();
        }
    });
函数 C
    Trans.exec(new Atom(){
        public void run(){
            数据操作 4;
            -> 函数 A();
```

```
        }
    });
```

那么，这三个函数都是事务性的。就是说，只有最外层的事务是起作用的，被包裹的事务会“融化”在上层事务里

4.16. 更底层定制NutDao

4.16.1. Dao 接口的默认实现 NutDao

Nutz 的 Dao 接口是它最大的一个接口，这个接口封装了一组更便利的数据库操作，同时 Nutz 也提供了一个默认的实现 : org.nutz.dao.impl.NutDao，基本上这个实现足够你使用。但是有些时候你希望对它进行更加深刻的，触及灵魂的定制，比如你打算让 Dao 的使用 Spring 的事务管理器等等，那么你可以读读这篇文档，它告诉你几个 NutDao 类的几个扩展点，可以让你更加底层定制 NutDao

4.16.2. 完全控制SQL语句的执行

NutDao 中无论任何操作，最终都要生成一个 SQL 语句，并传入给 JDBC 来执行。

* PojoMaker 接口负责语句的生成
* DaoExecutor 接口负责语句的执行

我们很鼓励你自己实现自己的 DaoExecutor 接口替换掉默认的实现，但是 PojoMaker 接口则暂时不建议你这么做，因为里面的逻辑稍微有点复杂，你非常容易弄错。

DaoExecutor 接口的源码如下：

```
[CODE]
public interface DaoExecutor {
    void exec(Connection conn, DaoStatement st);
}
```

它的实现类面对的是一个 DaoStatement，无论你是用自定义SQL，还是直接操作 POJO，最后NutDao都要生成DaoStatement 接口的一个实例，然后交付给 DaoExecutor 来运行.

Nutz 默认提供的 NutDaoExecutor 也不太复杂，算上空行和注释，240多行，你如果有兴趣看看它的源码，它倒是能成为你自定义 DaoExecutor 的一个很好的参考。

总之，通过 DaoExecutor 接口，你可以完全控制单条SQL语句的执行，当然，到现在，似乎还没有人明确的希望控制这个接口，大家都在用 Nutz 的默认实现。

4.16.3. 终极扩展点 - DaoRunner

如果你使用的是 Dao 接口的默认实现类 (org.nutz.dao.impl.NutDao)，你觉得任何事情都很满意，但是就是单单事务这部分你很不喜欢。你很不喜欢事务模板的写法：

```
[JAVA]
public void doSomething(final Pet pet){
    Trans.exec(new Atom(){
        public void runt(){
            dao.update(pet);
            dao.update(pet.getMaster());
        }
    });
}
```

看看，这样写代码太多了。因此，你甚至开始怀念 Spring，它的声明式事务管理，可以让你的代码完全不用这么繁琐。怎么办呢？

虽然 Nutz 也提供了声明式事务，但是你的项目是个老项目，一直在用 Spring，你所有的 CRUD 都是由 Spring来管理的，但是你很想试试 Nutz.Dao，你可以做到吗？

我们提供了一个扩展点。通过修改这个接口，你可以为 Dao 的默认实现类 NutDao 彻底定制事务行为 它就是 DaoRunner，它只有一个方法：

```
[JAVA]
public interface DaoRunner {
    public void run(DataSource dataSource, ConnCallback callback);
}
```

你可以根据自己的喜欢实现这个接口，然后

```
[CODE]
dao.setRuner(yourRuner);
```

当然，你可以通过 IOC 容器，将你的 runner 注入进 NutDao 对象中

4.16.4. 同 Spring 事务整合的例子

这个特性是在 1.a.27 之后，由知足常乐(hzzdong) 在博客《Nutz DAO与spring集成讨论》提出的，我们因此得到了 Issue 162。

在做了一点修改后，现在(1.a.28)，只要你提供一个这样的 DaoRunner

```
[JAVA]
import org.springframework.jdbc.datasource.DataSourceUtils;
public class SpringDaoRunner implements DaoRunner {
    public void run(DataSource dataSource, ConnCallback callback) {
        Connection con = DataSourceUtils.getConnection(dataSource);
        try {
            callback.invoke(con);
        }
        catch (Exception e) {
            if (e instanceof RuntimeException)
                throw (RuntimeException) e;
            else
                throw new RuntimeException(e);
        } finally {
            DataSourceUtils.releaseConnection(con, dataSource);
        }
    }
}
```

任何时候你可以通过:

```
[CODE]
dao.setRunner(new SpringDaoRunner());
```

修改 NutDao 的默认事务行为。 当然，如果你的 Dao 对像本身就是通过 Ioc 容器获得，我想你一定知道如何为你的对象设置一个属性，对吧。

4.17. 实体解析

4.17.1. 什么是实体解析

对于任何一个 ORM 工具，大都是根据配置者约定了解你打算怎么把一张数据表与你的 Java 对象映射到一起。默认的 Nutz.Dao 采用 Java 注解(Annotation) 的方式描述这个映射，但是，当然世界上还有很多其它的映射方式，比如用各种配置文件，比如 JPA 的注解，或者你很想把这个映射关系写在数据库的几张表里，甚至一个 Excel 表格里（你就觉得这样很酷）

从 1.b.38 之后，Nutz.Dao 开放了自己的 Entity 接口，你就可以定义自己的映射存放方式了。

```
[CODE]
Class<?>  --->  Entity<?> ---> PojoMaker ---> DaoStatement --->
DaoExecutor ---> JDBC
        |           |           |            |            |
        \----------------------- NutDao 的实现流程 ----------------------
-/
```

从上图，我们可以清楚的看到，Nutz.Dao 会首先分析你传入的 Class<?>，然后将将映射关系保存成一个 Entity<?>，也就是说通过找个 Entity<?>，NutDao 可以了解到调用者打算怎么映射数据表和Java类。

通过 EntityMaker 接口，Nutz 封装了这个过程，你可以重载 NutDao 的一个函数:

```
[JAVA]
public class MyDao extends NutDao {
    protected EntityMaker createEntityMaker() {
        return new MyEntityMaker();
    }
}
```

之后，如果使用 MyDao，那么实体的生成方式就是你说了算啦 ^_^

4.17.2. 默认的注解解析

通过在你的 POJO 类上标注注解，可以让 Nutz 来理解你打算怎么映射字段，这里我们举几个例子

4.17.2.1. 声明实体对应的表名

```
```

```
[CODE]
@Table
public class PetBean {
    ...
```

* 将对应到数据表 "pet_bean"
* @Table("t_pet") 将对应到 "t_pet"

4.17.2.2. 描述字段

```
[CODE]
@Column //提醒一下,这一行是多余的
@Name
@ColDefine(type=ColType.VARCHAR, width=20)
private String name;
```

* 将对应到数据表的 "name" 字段 -- 默认用 field 的名字
* 如果 @Column("nm") 将对应到数据表的 "nm" 字段
* @Name 表示这个字段有唯一性约束，你可以 Pet pet = dao.fetch(Pet.class,"abc");
* @Name(casesensitive=false) 表示依靠这个字段 fetch 时，忽略大小写
* @ColDefine 是给出这个字段最精确的描述，比如上例就是 VARCHAR(20)
    > @ColDefine 不是必须的，你如果没有声明，Nutz.Dao 会根据字段类型进行猜测
    > 通常，它能猜对，但是对于 String 类型的字段，它实在不知到长度为多少为好
    > 所以它会给一个默认的长度，但是很有可能你希望给一个别的长度
    > @ColDefine 主要是为 dao.create(XXX.class) 设计的，因为它要生成建表语句

通常一个字段，你只需要:

```
[CODE]
@Column
private int age;
@Column("pname")
private String parentName;
```

如果你的 POJO 没有任何一个字段标注了 @Column，那么相当于你所有的字段都是数据库字段。否则，仅仅是标注了 @Column 的字段才被认为是数据库字段

4.17.3. EntityMaker 接口

EntityMaker 接口负责具体的 Entity 生成，你可以参看一下它的源代码:

```
[CODE]
package org.nutz.dao.entity;
public interface EntityMaker {
        <T> Entity<T> make(Class<T> type);
}
```

Entity<T> 也是一个接口，当然，默认实现类 NutEntity<T> 应该能满足你大多数需求，我想至力于想扩展 Entity 生成方式的同学，会直接阅读相关的源代码作为参考，所以这里就不啰嗦了。

4.17.4. 动态实体

在 1.b.38 之前的版本，曾经记得有个朋友提出一个意见，他希望:

```
[CODE]
Map<String,Object> map = new HashMap<String,Object>();
map.put("name", "abc");
map.put("age", 18);
dao.update(map);    // 这个是不会有编译错误的
```

我记得当时我们的理由是：“搞不定呀，我们不知道表名呀””

现在，我们已经支持了这个特性:

```
[CODE]
Map<String,Object> map = new HashMap<String,Object>();
map.put(".table", "t_person");
map.put("name", "abc");
map.put("age", 18);
dao.update(map);
```

你给的 Map 只要多一个固定的名值对 ".table"，那么我就能知道你想操作的数据库表名。所以，你还可以:

```
[CODE]
Map<String,Object> map = new HashMap<String,Object>();
map.put(".table", "t_person");
map.put("name", "abc");
map.put("age", 18);
dao.insert(map);
```

实际上，NutDao 是根据给定的 Map，先构建了一个 Entity<?>，然后之后的事情就顺理成章了。这个特性从另外一方面也验证了现在的实体机制，它的确工作的还不错 :)

4.18. 内置的服务类

4.18.1. 什么是服务类，为什么需要它

Nutz.Dao 接口可以针对任何 POJO 的进行操作。因为是通用的 Dao 操作，所以多数接口函数都需要一个参数类说明 POJO 的类型，比如：

```
[JAVA]
Pet pet = dao.fetch(Pet.class,"XiaoBai");
```

单独的调用一行接口，多传入一个参数到没什么，但是如果频繁的被使用，每次都要多写一个参数毕竟很是麻烦。为此我又提供一层非常简单的针对 org.nutz.dao.Dao 接口的封装。譬如：

```
[JAVA]
IdNameEntityService<Pet> pets = new
IdNameEntityService<Pet>(dao){};
Pet pet = pets.fetch("XiaoBai");
```

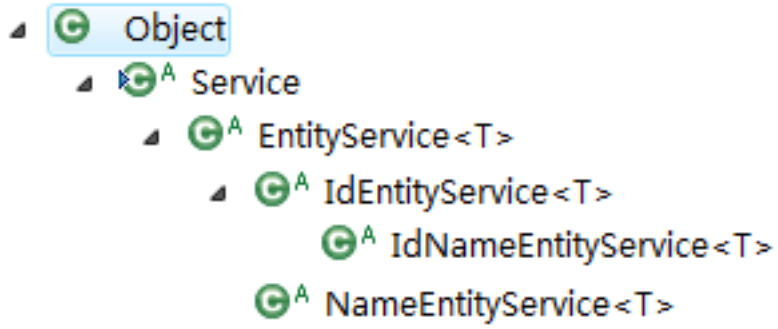
这样，调用的时候，就不用每次都传入参数了。

当然服务类并不是强制你使用的，只不过多数时候，采用 Nutz 提供的服务类会比较方便。你可以从 org.nutz.service.Service 继承自己的服务类，或者你干脆重新建立自己的服务类（提供更多的数据操作方法，进行 Crud 操作），因为 org.nutz.service.Service并不复杂，所以你是否

从中继承你自己的类对你的代码影响不大。你可以参考它的源代码。

4.18.2. Nutz 内置的服务类

在包 Service 中，提供了一些实现：



请重新温习一下Nutz.Dao 实体注解(Entity Annotation)中的注解 @Id 和 @Name 这两个注解，这里我称一个 POJO 对象为一个 **实体** （ Entity ）：

* 如果 POJO 即声明了 @Id 又声明了 @Name，那么适合采用 IdNameEntityService
* 如果 POJO 仅声明了 @Id，那么适合采用 IdEntityService
* 如果 POJO 仅声明了 @Name，那么适合采用 NameEntityService
* 如果 POJO 即没声明了 @Id 又没声明了 @Name，那么适合采用 EntityService

4.18.3. 关于服务类的总结

* 这四个内置的服务类，仅仅提供了一些基本的操作。
* 这些服务类都是支持泛型。
* 你可以直接使用，或者你可以从这四个服务类继承你自己的实现。
* 如果你继承这些服务类,请务必声明泛型

正确的写法

```
[JAVA]
public class RegisterService extends IdEntityService<UserBean> {
    public RegisterService(Dao dao){
        super(dao);
    }
}
```

错误的写法,你在log中会看到一条警告信息.

```
[JAVA]
public class RegisterService extends IdEntityService {
public RegisterService(Dao dao){
super(dao);
}
}
```

即，你的 Service 必须通过模板参数，告诉自己的父类，自己要操作的 POJO 类是什么，否则父类会很茫然的

4.19. 更快的构建你的POJO

4.19.1. 为什么需要提供更多构建 POJO 的手段

当查询以及获取一个或者是多个 POJO 时，Nutz.Dao 采用反射来创建对象以及为对象的各个相关字段填充值。为了提高速度，Nutz.Dao 对对象采取一些缓存的策略，所以基本上不用去查询每个对象的构造函数以及getter，setter 或者公共字段,所有的时间开销都集中在反射的 invoke 方法上。事实证明，这让 POJO 的构建速度提高了不少，但是对于那些还想对速度进行优化的应用，怎么办呢？ Nutz.Dao 还提供了更加极限的手段。

4.19.2. 将构建的过程掌握在你自己的手里

有些人说JDK1.5 以后，反射更快了。并且由于 Nutz.Dao 对于实体的反射做了缓存，所以它创建对象比一般的反射更快，但是，还是有些人不满意，他们还需要更快。所以我提出了一个约定：

4.19.2.1. 最优先 -- 带一个参数的静态工厂方法

如果你的实体对象有一个静态的函数，返回类型的是你的实体对象(有点类似于工厂方法)，输入参数是 java.sql.ResultSet， 那么在创建实例的时候，会直接调用你这个方法。

例如:

```
[JAVA]
public class Pet {
    public static Pet getInstance(ResultSet rs) throws SQLException{
        Pet pet = new Pet();
        pet.id = rs.getInt("id");
        pet.name = rs.getString("name");
        pet.nickname = rs.getString("nick");
        return pet;
    }
```

```
    // ... 省略后面代码，包括字段声明以及 getter 和 setter
```

只要你声明了一个可访问的，带有一个参数其类型为ResultSet，且返回值为 Pet 的静态函数，名称无所谓， 那么 Nutz.Dao 在构建 Pet 对象时，就会优先使用这个函数。 因为 Nutz.Dao 认为你打算亲自来做 ResultSet 到 POJO 字段的映射工作。你可以将这个过程写的很快，因为如果交由 Nutz.Dao 来做的话，它不得不做一些字段类型的判断以及转换的工作。

4.19.2.2. 第二优先：带一个参数的构造函数

如果你的实体中有一个构造函数，参数是 java.sql.ResultSet，那么创建实例的时候会直接使用这个构造函数。

例如:

```
[JAVA]
public class Pet {
    public Pet(){}
    public Pet(ResultSet rs) throws SQLException{
        id = rs.getInt("id");
        name = rs.getString("name");
        nickname = rs.getString("nick");
    }
    // ... 省略后面代码，包括字段声明以及 getter 和 setter
```

只要你声明了构造函数 Pet(ResultSet rs)， 那么 Nutz.Dao 在构建的时候，就不会使用默认构造函数，无论你是否声明它。

4.19.2.3. 第三优先：无参数静态工厂方法

如果你的实体对象有一个静态的函数，返回类型的是你的实体对象(有点类似于工厂方法)，不需要输入参数， 那么在创建实例的时候，会直接调用你这个方法。再通过反射为各个字段填充值。

例如:

```
[JAVA]
public class Pet {
    public static Pet getInstance(){
        return new Pet();
```

```
    }
    private Pet(){}
    // ... 省略后面代码，包括字段声明以及 getter 和 setter
```

你的静态函数名称是无所谓的，只要它的返回类型是当前的类，并且无参数。 如果出现多个符合改条件的函数，Nutz.Dao 会挑取任意一个。

4.19.2.4. 最不优先：默认构造函数

如果你的 POJO 仅仅有默认构造函数可用，那么，Nutz.Dao 就会采用JAVA反射机制来为你构建 POJO 对象。但是你必须保证默认构造函数是可被访问的。

多数 POJO 都会采用这种形式，因为它最方便，比如：

4.19.2.4.1. 编译器自己为你增加默认构造函数

```
[JAVA]
public class Pet {
    @Column
    @Id
    private int id;
    @Column
    @Name
    private String name;
    @Column
    private String nickname;
    // ... 这里是 getter 和 setter
}
```

4.19.2.4.2. 自己定制的默认构造函数

```
[JAVA]
public class Pet {
    public Pet(){
        this.nickname = "Unknown";
    }
    @Column
    @Id
    private int id;
    @Column
```

```
    @Name
    private String name;
    @Column
    private String nickname;
    // ... 这里是 getter 和 setter
}
```

4.19.3. 总结一下这四个约定

通过头两个约定，你可以让你的数据库操作同直接调用 JDBC 接口一样快。别忘了，在数据库操作的时候，拼装SQL 这点小开销几乎可以忽略不计。第三个约定，适用于在你不希望暴露POJO的构造函数的前提下。最后一个约定则是 Nutz.Dao 的默认期望的工作方式。

4.20. 不构建 POJO 访问数据库

4.20.1. POJO 之苦

很多时候，程序员在访问数据库时，预先构建一个 POJO 对象，会为之后的编程带来很大的便利。但是有些时候，构建 POJO 是不可能的，或者是很麻烦的，比如：

* 字段可以动态增删的表
* 表是被动态创建出来的
* 非常少用到的表，为此维护一个 POJO 很不划算

当然上述三个问题都可以通过 自定义 SQL 来实现，但是它毕竟有一点点复杂。想想其他的动态语言是怎么做的吧。他们为每条数据库记录直接返回一组名值对。很简单不是吗？

名值对，非常像 JDBC 中的 ResourceSet 对象，但是不幸的是，你不能把它保存到任意的地方，因为一旦 Connection 关闭了，它就 Over 了。为此，我想很多人都会想到从 ResultSet 读取一组 Map。Nutz.Dao 也提供了一个，它不是 Map 而是一个 Map 的包裹类 -- Record。它比 Map 提供了更多的一些包裹方法，便于你取值。

对于 Update, Insert 操作，你可以使用 Chain （值链）。 这个我们后面会给你举几个例子，一看便知。

至于 Delete 操作， 通过 Dao.clear 操作，并配合上 Condition，你可以很容易删除数据

后面的例子，我们都假设有这样一个数据表：

```
[CODE]
t_pet {
    id INT PK,
```

```
    name VARCHAR(20) UNIQUE,
    birthday TIMESTAMP
}
```

4.20.2. 获取一条记录

```
[CODE]
// 根据主键获取
Record re = dao.fetch("t_pet", Cnd.where("id","=",2);
// 根据名字获取
re = dao.fetch("t_pet", Cnd.where("name","=","XiaoBai");
// 打印名字
System.out.println(re.getString("name"));
// 打印生日
System.out.println(re.getTimestamp("birthday"));
// 打印字段总数
System.out.println(re.getColumnCount());
// 转换成 Pet 类，如果你有 Pet 类的话
Pet pet = re.toPojo(Pet.class);
// 转换成 Json 字符串
String json = re.toJson(JsonFormat.nice());
```

4.20.3. 查询多条记录

```
[CODE]
// 查询所有以 A 开头的 Pet
List<Record> list = dao.query("t_pet",
Cnd.where("name","LIKE","A%"), null);
// 查询所有以 A 开头的 Pet，返回前10个
List<Record> list = dao.query("t_pet",
Cnd.where("name","LIKE","A%"), dao.createPager(1,10));
```

4.20.4. 插入

```
[CODE]
dao.insert("t_pet", Chain.make("name","XiaoBai").add("birthday", new
```

```
Timestampe(System.currentTimeMillis()));
```

它会执行 SQL

```
[CODE]
INSERT INTO t_pet (name,birthday) VALUES ('XiaoBai', '2010-4-28
14:27:12')
```

值链实际就是通过链式赋值的方式构建出的一组名值对，当然如果值链较长你可以：

```
[CODE]
Chain ch = Chain.make("字段名",字段值);
ch.add("字段名", 字段值);
ch.add("字段名", 字段值);
ch.add("字段名", 字段值);
...
```

4.20.5. 更新

```
[CODE]
dao.update("t_pet",
        Chain.make("name","XiaoBai").add("birthday", new
Timestampe(System.currentTimeMillis())),
        Cnd.where("id","=",2));
```

它会执行 SQL

```
[CODE]
UPDATE t_pet SET name='XiaoBai', birthday='2010-4-28 14:27:12'
WHERE id=2;
```

4.20.6. 删除

```
[CODE]
dao.clear("t_pet", Cnd.where("id", "=" ,2);
```

它会执行 SQL

```
[CODE]
DELETE t_pet WHERE id=2;
```

4.21. 注解列表

以下的列表，是 Nutz.Dao 支持的全部注解，如果你想深入了解每个注解的意义，你可以

* 请参看相关的 JDoc
* 直接浏览源代码: https://github.com/nutzam/nutz/tree/master/src/org/nutz/dao/entity/annotation
* 用 Git 工具检出源代码: https://github.com/nutzam/nutz.git
* 用 SVN 工具检出源代码（因只在新版本发布的时候更新该地址，不推荐）: http://code.google.com/p/nutz/source/checkout
* 下载源代码: https://github.com/ywjno/yuchberry-server-plus/zipball/master

@Column	字段	@see JDoc: org.nutz.dao.entity.annotation.Column
@ColDefine	字段精确定义	@see JDoc: org.nutz.dao.entity.annotation.ColDefine
@Default	默认值	@see JDoc: org.nutz.dao.entity.annotation.Default
@EL	字段表达式宏	@see JDoc: org.nutz.dao.entity.annotation.SQL

@Id	数字主键	@see JDoc: org.nutz.dao.entity.annotation.Id
@Name	字符主键	@see JDoc: org.nutz.dao.entity.annotation.Name
@PK	复合主键	@see JDoc: org.nutz.dao.entity.annotation.PK
@Many	一对多映射	@see JDoc: org.nutz.dao.entity.annotation.Many
@ManyMany	多对多映射	@see JDoc: org.nutz.dao.entity.annotation.ManyMany
@One	一对一映射	@see JDoc: org.nutz.dao.entity.annotation.One
@Prev	自动设置	@see JDoc: org.nutz.dao.entity.annotation.Prev
@Next	自动获取	@see JDoc: org.nutz.dao.entity.annotation.Next
@Readonly	只读声明	@see JDoc: org.nutz.dao.entity.annotation.Readonly
@SQL	字段SQL宏	@see JDoc: org.nutz.dao.entity.annotation.SQL
@Table	表名	@see JDoc: org.nutz.dao.entity.annotation.Table
@View	视图名	@see JDoc: org.nutz.dao.entity.an

		notation.View

5. Ioc 手册

5.1. Hello world

5.1.1. Ioc 的概念

Nutz.Ioc 从概念上是很简单：将一部分关于对象的依赖关系单独存储在某种介质里，并且提供一个接口帮助使用者获得这些对象。

但是将依赖关系存储在什么地方呢？ Spring 选的是 XML， Guice 选的 Java （硬编码）

Nutz.Ioc 核心逻辑并没有限定配置信息的存储方式，但它还是提供了一个默认的配置文件编写方式 -- JSON。因为

* 省却了 XML 书写的烦恼
* 避免了硬编码 -- 修改配置，不需要重新编译工程

当然，你可以扩展它，提供自己的配置文件加载方式， Nutz.Ioc 不反对你这样，它甚至有点鼓励你这样，虽然 JSON 方式的配置文件书写方式已经工作的很好了。

下面，我先以 JSON 文件为例，给大家一个 Hello World

5.1.2. 一个简单的例子

在这个例子中，你需要一 POJO，以及一个 JSON 配置文件。例子的源代码，你可以访问 http://nutzdemo.googlecode.com 获取

5.1.2.1. POJO 源代码

```
[CODE]
package nutz.demo.ioc.book;
import java.util.Calendar;
public class Pet {
    private String name;
    private Calendar birthday;
    private Pet friend;
    public Pet() {}
    public Pet(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
```

```
    public void setName(String name) {
        this.name = name;
    }
    public Calendar getBirthday() {
        return birthday;
    }
    public void setBirthday(Calendar birthday) {
        this.birthday = birthday;
    }
    public Pet getFriend() {
        return friend;
    }
    public void setFriend(Pet friend) {
        this.friend = friend;
    }
}
```

这个对象有两个构造函数

5.1.2.2. Json 配置文件

与 POJO 在同一包路径下

```
[CODE]
/*
 * 开始写上 var ioc = { ， 是为了利用 eclipse 的 javascript 编辑器的自动格式化功能
 */
var ioc = {
    /*
     * 默认的，你仅仅需要直接声明每个字段的值即可，Nutz.Ioc 会为你转型
     */
    xiaobai : {
        name : 'XiaoBai',
        birthday : '2009-10-25 15:23:40'
    },
    /*
     * 你当然也可以做更细致的设置
     */
    xiaohei : {
```

```
        type : 'nutz.demo.ioc.book.Pet', // 类型
        singleton : false, // 是否为单件
        args : [ 'XiaoHei' ], // 构造函数参数
        fields : {
            birthday : '2009-11-3 08:02:14',
            friend : {refer : 'xiaobai'}// 指向容器里另外一个对象
        }
    }
}
```

5.1.2.3. 调用代码

```
[CODE]
package nutz.demo.ioc.book;
import org.nutz.ioc.Ioc;
import org.nutz.ioc.impl.NutIoc;
import org.nutz.ioc.loader.json.JsonLoader;
public class HelloPet {
    public static void main(String[] args) {
        Ioc ioc = new NutIoc(new
JsonLoader("nutz/demo/ioc/pet/pets.js"));
        Pet pet = ioc.get(Pet.class, "xiaobai");
        System.out.printf("%s - [%s]\n", pet.getName(),
pet.getBirthday().getTimeZone().getID());
    }
}
```

5.1.2.4. 控制台输出

```
[CODE]
XiaoBai - [Asia/Shanghai]
```

5.1.2.5. 如果配置文件中声明了类型，则可不传入类型

```
[CODE]
Pet xh = ioc.get(null, "xiaohei");
System.out.printf("%s's friend is %s\n", xh.getName(),
```

```
xh.getFriend().getName());
```

控制台输出：

```
[CODE]
XiaoHei's friend is XiaoBai
```

5.1.2.6. 声明了 singleton: false，那么它每次获取，都会生成一个新的实例

```
[CODE]
Pet p1 = ioc.get(null, "xiaohei");
Pet p2 = ioc.get(null, "xiaohei");
System.out.println(p1==p2);
```

控制台输出：

```
[CODE]
false
```

5.1.3. 关于进阶

我可以负责任的告诉你：你已经掌握了 Nutz.Ioc 在你开发的时候 80% 情况下所需要的知识。当然，它还提供了更多的功能，有些功能是其他 Ioc 容器所不具备的，你可以根据自己需要来阅读，他们包括：

* 匿名对象
* 事件监听
* 你都可以注入什么
* 事件的监听
* 定义自己的配置文件格式
* AOP模型 -- NutAop的实现思路
* AOP -- 声明式切片
* 对象生命范围 -- 级联的上下文环境
* 使用XML作为配置文件格式 -- 使用XML作为配置文件格式

* 使用注解配置Ioc --使用注解配置Ioc
* 在json配置文件中声明Aop -- 在json配置文件中声明Aop及声明式事务

5.2. 匿名对象

5.2.1. 鸭子法则

如果它走路像鸭子，说话像鸭子，样子像鸭子，叫声也像鸭子，那它就是一只鸭子。

对于 Nutz Ioc 来说，它面对的配置文件就是层层嵌套的 "名值对集合"，或者说是 Map 集合。事实上，它是先把整个配置文件解析成 Map 再做判断的。

如果一个 Map 仅包括如下的键，则被认为是一个注入对象：

type	对象类型
singleton	是否单例
scope	生命周期范围
events	监听事件
args	构造函数参数
fields	字段设定

否则，这个 Map 被认为，是在声明对象的每个字段。

5.2.2. 匿名对象

如果，一个字段的值就是这样一个 “鸭子 Map” 呢？ 那么，自然会被认为是另外一个对象。这个对象没有名字，你不能通过 Ioc 接口直接获得，它隐藏在某个对象的某个字段里面。所以我们称这个对象为 **匿名对象**

匿名对象，没有所谓的单例，你声明了 singleton: true 也没有用。如果它的宿主是单例，它自然也只会被创建一次。否则，每次宿主被创建的时候，它都会被创建。

5.2.2.1. JSON 配置文件：

```
[CODE]
var ioc = {
    xb : {
        name : 'XiaoBai',
        // 请注意，在这里， friend 字段，直接声明了另外一个对象
        friend : {
```

```
            type : 'nutz.demo.ioc.book.Pet',
            fields : {
                name : 'XiaoHei'
            }
        }
    }
}
```

5.2.2.2. 调用代码

```
[CODE]
Ioc ioc = new NutIoc(new
JsonLoader("nutz/demo/ioc/book/inner.js"));
Pet pet = ioc.get(Pet.class, "xb");
System.out.println(pet.getFriend().getName());
```

5.2.2.3. 控制台输出

```
[CODE]
XiaoHei
```

5.3. 事件监听

5.3.1. 都有哪些事件

Nutz.Ioc 容器有三种事件：

* 对象被创建
* 对象被从容器中取出
* 对象被销毁

在这三种时刻，你如果想做一些特殊的操作，比如，当一个数据源被销毁时，你希望能够关闭所有的连接，你可以在 JSON 配置文件中，声明一下，你想监听什么事件，以及怎么监听。

5.3.2. 怎么监听

我们为 Pet 对象增加一个新的属性:

```
[CODE]
public class Pet {
    private int fetchCount;
    ... // 省略其他属性以及 getter 和 setter
```

5.3.2.1. 通过实现一个触发器

5.3.2.1.1. 触发器

```
[CODE]
package nutz.demo.ioc.book;//提醒: 不要把你的类放在缺省包中!!
import org.nutz.ioc.IocEventTrigger;
public class OnFetchPet implements IocEventTrigger<Pet> {
    public void trigger(Pet pet) {
        pet.setFetchCount(pet.getFetchCount() + 1);
    }
}
```

IocEventTrigger 接口只有一个方法，当它被调用的时候，就表示某个事件发生了。具体什么事件呢？完全看你在 JSON 配置文件中把这个触发器，声明在哪种事件下面

5.3.2.1.2. 在 JSON 配置文件中声明监听的事件

```
[CODE]
var ioc = {
    xb : {
        events : {
            fetch : 'nutz.demo.ioc.book.OnFetchPet'
        },
        fields : {
            name : 'XiaoBai'
        }
    }
}
```

5.3.2.1.3. 调用代码

```
```

```
[CODE]
Ioc ioc = new NutIoc(new
JsonLoader("nutz/demo/ioc/book/events.js"));
Pet pet = ioc.get(Pet.class, "xb");
ioc.get(Pet.class, "xb");
ioc.get(Pet.class, "xb");
System.out.printf("%s be fetch %d times", pet.getName(),
pet.getFetchCount());
```

5.3.2.1.4. 控制台输出

```
[CODE]
XiaoBai be fetch 3 times
```

5.3.2.2. 通过对象自身的一个函数

这个函数必须为 public，并且不能有参数

5.3.2.2.1. 为 Pet 对象增加一个函数：

```
[CODE]
public void onFetch() {
    this.fetchCount++;
}
```

5.3.2.2.2. 在 JSON 配置文件中增加新的对象

```
[CODE]
xh : {
    events : {
        fetch : 'onFetch'
    },
    fields : {
        name : 'XiaoHei'
    }
}
```

5.3.2.2.3. 调用代码

```
[CODE]
Pet xh = ioc.get(Pet.class, "xh");
ioc.get(Pet.class, "xh");
System.out.printf("%s be fetch %d times\n", xh.getName(),
xh.getFetchCount());
```

5.3.2.2.4. 控制台输出

```
[CODE]
XiaoHei be fetch 2 times
```

5.3.3. 监听其他事件

```
[CODE]
events : {
    fetch  : ... ,
    create : ... ,
    depose : ...
}
```

根据需要，你可以选择上述三个事件，为其声明触发器，或者触发函数。

5.4. 你都可以注入什么

5.4.1. 从哪里注入？

你可以向对象注入值的位置有两个

1. 构造函数参数
2. 属性

5.4.1.1. 向构造函数里注入参数

你的 JSON 配置文件会是这样

```
[CODE]
{
    xb : {
        type : 'nutz.demo.ioc.book.Pet',
        args : ['XiaoBai']
    }
}
```

args 的值时一个数组，里面每一个元素都将对应构造函数的一个参数。当然，你必须确保你得有这样的构造函数。每个参数按照 JSON 的规定，是用半角逗号分隔的。

5.4.1.2. 向属性注入参数

你的 JSON 配置文件会是这样

```
[CODE]
{
    xb : {
        type : 'nutz.demo.ioc.book.Pet',
        fields : {
            name : 'XiaoBai'
        }
    }
}
```

如果你不需要写 type，那么你可以用简写模式:

```
[CODE]
{
xb : { name: 'XiaoBai'}
}
```

5.4.2. 值可以不仅是字符串

是的，它还可以是

5.4.2.1. 布尔

```
[CODE]
{
    xb : { dead: true }
}
```

5.4.2.2. 数字

```
[CODE]
{
    xb : { age: 24 }
}
```

5.4.2.3. 内部对象

```
[CODE]
{
    xb : {
        friend: {
            type : 'nutz.demo.ioc.book.Pet',
            fields : {name : 'XiaoHei'}
        }
    }
}
```

关于内部对象 请看这里

5.4.2.4. 引用

```
[CODE]
{
    xb : { friend: {refer: 'XiaoBai' } }
}
```

{refer : '另外一个对象在容器中的名称'} 将会得到容器中另外一个对象

5.4.2.5. 容器自身

```
[CODE]
{
    xb : { myIoc : {refer: '$Ioc'} }
}
```

一种特殊的引用，大小写不敏感，值就是 Ioc 容器本身

5.4.2.6. 对象的名称

```
[CODE]
{
    xb : { myIoc : {refer: '$Name'} }
}
```

一种特殊的引用，大小写不敏感，值就是对象的名称，即 "xb"

5.4.2.7. 容器上下文

```
[CODE]
{
    xb : { myIoc : {refer: '$Context'} }
}
```

一种特殊的引用，大小写不敏感，值就是当前容器的上下文环境接口 org.nutz.ioc.IocContext

5.4.2.8. 环境变量

```
[CODE]
{
    xb : { name : {env : "JAVA_HOME"} }
}
```

{env : '环境变量名'} 将会得到系统中环境变量的值

5.4.2.9. 文件

```
[CODE]
{
    xb : { profile : {file : "/home/zozoh/tmp/name.txt"} }
}
```

{file : '文件路径'} 可以是绝对路径，也可以是 CLASSPATH 中的路径

5.4.2.10. 数组或容器

如果你对象某个字段是数组，集合，或者 Map， 用 JSON 可以很自然为其设置值，不是吗？

5.4.2.11. Java 调用

这是个极度灵活的注入方式，它几乎可以让你 **做任何事情。** 因为它允许你直接调用一个 JAVA 函数。

更详细的说明，请参看 org.nutz.ioc.val.JavaValue 的 JDoc

下面只是列出主要的几种应用方式

5.4.2.11.1. 静态属性

```
[CODE]
{
    xb : { oneField : {java: 'com.my.SomeClass.staticPropertyName'} }
}
```

5.4.2.11.2. 静态函数

```
[CODE]
{
    xb : { oneField : {java: 'com.my.SomeClass.someFunc'} }
}
```

5.4.2.11.3. 带参数的静态函数

```
[CODE]
{
    xb : { oneField : {java: 'com.my.SomeClass.someFunc("p1",true)'} }
}
```

参数可以是任何种类的值

5.4.2.11.4. 容器中的对象

```
[CODE]
{
    xb : { oneField : {java: '$xh'} } ,
    xh : { name : 'XiaoHei'}
}
```

5.4.2.11.5. 容器对象某个属性

```
[CODE]
{
    xb : { oneField : {java: '$xh.name'} } ,
    xh : { name : 'XiaoHei'}
}
```

5.4.2.11.6. 容器对象某个方法的返回值

```
[CODE]
{
    xb : { oneField : {java: '$xh.getXXX()'} } ,
    xh : { name : 'XiaoHei'}
}
```

5.4.2.11.7. 容器对象某个方法的返回值，带参数

```
[CODE]
{
    xb : { oneField : {java: '$xh.getXXX("some string", true, 34)'} } ,
    xh : { name : 'XiaoHei'}
}
```

参数可以是任何种类的值

5.4.3. 你可以增加自己的特殊类型

从上面你可以看到 JSON 语法的好处，非常轻巧

* 文件 -- {file: '路径'}
* 环境变量 -- {env: '环境变量名'}
* 引用 -- {refer: '对象名'}
* JAVA -- {java: '$对象名.方法名(参数1, 参数2)'}

还可以更多吗？

是的，你完全可以扩展，比如你如果想支持一种新的类型：

```
[CODE]
oneField : {scan : '扫描仪地址'}
```

如何支持这种新的值的类型呢？

5.4.3.1. 实现一个扩展接口

实现 org.nutz.ioc.ValueProxyMaker 接口：

```
[CODE]
package nutz.demo.ioc.book;
import org.nutz.ioc.IocMaking;
import org.nutz.ioc.ValueProxy;
import org.nutz.ioc.ValueProxyMaker;
import org.nutz.ioc.meta.IocValue;
import org.nutz.lang.Lang;
```

```
public class ScanValueProxyMaker implements ValueProxyMaker {
    public ValueProxy make(IocMaking ing, IocValue iv) {
        if ("scan".equals(iv.getType())) {
            final String address = iv.getValue().toString();
            return new ValueProxy() {
                public Object get(IocMaking ing) {
                    // 根据 address 创建一个对象
                    throw Lang.noImplement();
                }
            };
        }
        return null;
    }
    public String[] supportedTypes() {
        return new String[]{"scan"};
    }
}
```

5.4.3.2. 添加到 Ioc 容器中

```
[CODE]
Ioc2 ioc = new NutIoc(new JsonLoader("path/path/name.js"));
ioc.addValueProxyMaker(new ScanValueProxyMaker());
// 下面，你就可以正常使用 Ioc 接口了
```

注意，这里使用的是 Ioc2 接口，它继承自 Ioc 接口，提供了更高级的方法

5.5. 定义自己的配置文件格式

5.5.1. 你如果不喜欢 JSON 怎么办？

我很喜欢 JSON，因为它语法轻巧。当然，我不能保证你也很喜欢 JSON，因为你可能会说：

* 没有很好的 JSON 编辑器 -- *如果是这样，我推荐你试试 Eclipse 自带的 Javascript 编辑器*。
* 我更喜欢 XML，我不喜欢大括号
* 我就是讨厌 JSON

那么，你想自己规定自己的配置文件格式吗？ 你可以定义一个比 JSON 更酷的配置文件语法，当然你得多写一点点代码，解析你的配置文件。但是你就想做点很酷的事情，不是吗？

Nutz.Ioc 提供了一个扩展点：**org.nutz.ioc.IocLoader** 接口。实际上 JSON 的配置文件语法，不过是我预先为这个接口写的一个实现，所以你当然也可以写一个，请注意 Nutz.Ioc 接口是如何被构建的：

```
[CODE]
Ioc ioc = new NutIoc(new JsonLoader("文件路径"));
```

我提供的 JsonLoader 没有任何“特殊待遇”，它和你自己要实现的 IocLoader 地位是完全一样的，你完全可以这么写：

```
[CODE]
Ioc ioc = new NutIoc(new MyIocLoader());
```

只要你正确的实现了 IocLoader 接口，如何持有对象，如果解释对象的语义，都由 Nutz.Ioc 的标准实现流程负责，你完全不用操心。

你可能还有一个问题：“**那么 IocLoader 接口实现起来复杂吗？**”

我可以负责任的告诉你，虽然那不是极其简单，但是绝对不复杂。你只要理解下面这个知识即可：

5.5.2. 在 Nutz.Ioc 定义一个对象的数据结构是怎样的？

在 org.nutz.ioc.meta.IocObject 类中，你如果拿到它的源代码，或者是 JDoc，它描述了在容器中一个对象的全部信息，你会发现它其实也简单：

```
[CODE]
public class IocObject {
    private Class<?> type;
    private boolean singleton;
    private IocEventSet events;
    private List<IocValue> args;
    private List<IocField> fields;
    private String scope;
    // 省略所有的 getter 和 setter 函数
```

看看这个类的属性，顾名思义：

* **type** -- 对象类型
* **singleton** -- 声明对象是否为单例
* **events** -- 对象监听何种事件
* **args** -- 对象构造函数的参数列表
* **fields** -- 对象的字段

没有对象的名称，对吗？是的，通过 JSON （如果你了解一点 Javascript，那就更容易理解了），你通常要这么写：

```
[CODE]
你的 JSON 文件，下面是正文
-----------------------------------------------
{
   "对象名称" : {
      type : "对象类型",
      singleton : true | false,
      events : {
         fetch : "触发器的类型或者函数名",
         create : "触发器的类型或者函数名",
         depose : "触发器的类型或者函数名"
      },
      args : [
         参数1， 参数2 ...
      ],
      fields : {
         "字段名称1" : 字段值1 ,
         "字段名称2" : 字段值2 ,
         ...
      }
   }
}
```

那么你如果想通过其他的配置文件格式，比如 XML 类获取 IocObject， 文件格式应该如何定义，完全看你个人的喜好了。

这里需要说一下：

5.5.2.1. 事件集合 IocEventSet

描述了一个对象可以监听的事件。三个属性分别表示：

* create: 对象创建时触发
* fetch: 对象获取时触发
* depose: 对象销毁时触发

它们的值:

* 可以是一个函数名，也可以是一个 org.nutz.ioc.IocEventTrigger 的实现类全名
* 如果 是函数，那么这个函数就是对象内的一个非静态 public 的函数，而且不能有参数
* 如果是 IocEventTrigger 的实现类，你的实现类必须有一个 public 的默认构造函数

5.5.2.2. 值 IocValue

描述了对象的一个值，这个值可以是构造函数的参数，也可以是一个字段的值。 它由两个属性，一个是值的类型，另外一个是 value。

赋值约定:

* 如果 type 是 "null"，则值会被认为是 null
* 如果 value 是 字符串，数字，布尔，那么 type 必须为 "normal"或 null
* 如果 value 是 数组， Collection 或 Map，那么类型也必须是 "normal"或 null，Ioc 容器的实现类会深层递归集合的每个元素。集合内的每个元素的值也可以是 IocValue，规则符合本约定
* 如果 value 是 IocObject，则表示这个值是一个内部匿名对象，type 必须为 "inner" 或者 null
* 如果 value 是字符串，表示另外一个对象的名称，type 必须是 "refer"
* 如果 value 是字符串，表示一个环境变量，type 必须是 "env"
* 如果 value 是字符串，表示一个文件路径，type 必须是 "file"
* 如果 value 是字符串，表示一个 Java 调用，type 必须是 "java"，具体值的语法，请参看 JavaValue 类的 JDoc，当然 Ioc 容器来解析执行它，不需要 IocLoader 操心 说明
* 你的 ValueProxyMaker 可以扩展这个约定

5.5.2.3. 字段 IocField

描述了一个对象的字段，两个属性分别表示字段名，和字段值

5.5.3. 实现你的加载器

你的 IocLoader 的实现类需要实现三个方法:

```
[CODE]
public interface IocLoader {
    /**
```

```
     * @return 配置信息里所有对象的名称
     */
    String[] getName();
    /**
     * 每次这个函数被调用，则要构造一个新的 IocObject
     *
     * @param name
     * @return
     * @throws ObjectLoadException
     */
    IocObject load(String name) throws ObjectLoadException;
    /**
     * @param name
     * @return 配置信息里是否存在一个对象
     */
    boolean has(String name);
}
```

5.5.4. 最后

没什么要说的了，充分发挥你的想象力吧，如果你愿意，将配置信息放在 PDF 里，放在 excel 表格里都是可以的 如果你完成了一个IocLoader,不妨分享出来,我们将收录到 http://code.google.com/p/nutz-ioc-loaders/

5.5.5. 附:已经可用的IocLoader

* 使用Json作为配置文件 org.nutz.ioc.loader.json.JsonLoader
* 使用XML作为配置文件 org.nutz.ioc.loader.xml.XmlIocLoader
* 使用注解作为配置文件 org.nutz.ioc.loader.annotation.AnnotationIocLoader
* 使用Map作为配置文件 org.nutz.ioc.loader.map.MapLoader
* 混合使用多种配置?一样可以! org.nutz.ioc.loader.combo.ComboIocLoader

5.6. 对象生命范围

5.6.1. 高级接口容器接口

NutIoc 实现了 Ioc2 接口，它继承自 Ioc 接口，并多出了两个方法。一个允许你自行添加自定义的值类型，另一个是允许你在获取对象时，链入自己的上下文环境

请阅读 org.nutz.ioc.IocContext 的接口文档， 你可以根据需要实现这个上下文接口

5.6.2. 这个设计有什么用？

比如，在一个 Web 应用中，你希望在会话中保存一个数据源，你不希望这个数据源保存在 Application级别里。因为，只有当用户登录的时候，你才能确定你到底要连接哪个数据源。

那么，在配置文件中，我的那个数据源对象，以及所有引用它的对象，都会声明了 scope : "session"。每次启动一个会话，我们就创建一个 Session 的上下文，当请求发生时，就用通过 Ioc2 来获取对象--- 传入 session 上下文。 当会话停止时，会注销这个上下文，那么保存在里面的数据源也会彻底关闭

如果没有 Ioc2 这个接口，则做不到这一点

5.6.3. 通过 ComboContext 链接

在 org.ioc.impl.NutIoc 里，默认有一个 IocContext，它的 scope 是 "app"。

每次从 NutIoc 里获取对象，如果你想告诉容器： “嘿，我还有另外的一个缓存，请优先在里面查查” 你可以这样调用：

```
[CODE]
IocContext myCache = ...; // 获取你的缓存
MyObject obj = ioc.get(MyObjec.class, "objName", myCache);
```

Nutz.Ioc 默认提供了两个 IocContext:

* ScopeContext： 顾名思义，它只表示某一个指定的声明周期范围
    > 在其构造函数里，你必须指明这个上下文对象可以接受的生命周期范围的名称
    > 当你的配置信息中 scope 一项同它的范围名称完全匹配时，它会接受你的对象
* ComboContext：它可以把多个 IocContext 集合成一个 IocContext
    > 通过它，你可以一次向多个 IocContext 获取或者存入对象

5.7. 使用XML作为配置文件格式

格式约定: 参阅xsd文件(包含在jar中): nutz-ioc-0.1.xsd

示例文件 示例:

```
[CODE]
<ioc xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="nutz-ioc-0.1.xsd">
   <obj name="obj" singleton="true" scope="app" parent="XXX"
      type="java.lang.String">
      <args>
         <str>Tasdfadf</str> <!-- 对应具体类型 -->
         <int>23</int>
```

```
        <float>78.34</float>
        <bool>true</bool>
        <refer>abc</refer>
        <refer>$Ioc</refer>
        <java>$abc.find('YYY')</java>
        <env>TOMCAT_HOME</env>
        <file>/home/zozoh/tmp/name.txt</file>
        <obj type="java.lang.StringBuilder" name="xx"> <!-- 内部对
象 -->
          <args>
            <str>tttt</str>
          </args>
        </obj>
        <array>
          <str>XXXX</str>
          <str>XXXX</str>
        </array>
        <list>
          <int>34</int>
          <float>78.34</float>
        </list>
        <map>
          <item key="a">
            <str>XXXX</str>
          </item>
          <item key="B">
            <int>11</int>
          </item>
        </map>
      </args>
      <field name="field1">
        <env>JAVA_HOME</env>
      </field>
      <events>
        <create>onCreate</create>
        <depose>depose</depose>
        <fetch>com.you.app.OnFetchObject</fetch>
      </events>
    </obj>
    <obj name="XXX">
      <events>
        <create>xxxx</create>
        <depose></depose>
```

```
    </events>
  </obj>
</ioc>
```

5.8. Ioc - Annotation 加载器

5.8.1. 为什么需要 Ioc 注解加载器

无论是 XML 还是 JSON，都需要你创建一个新的配置文件，在里面描述你的对象依赖关系。一般的来说，一个应用大多数的对象依赖关系，是固定的，即，在项目发布以后是不必调整的。如果将这些依赖关系通通写到配置文件中，颇有点"脱了裤子放屁"的感觉，最理想的情况是，将可能变动的依赖关系写到配置文件里，而将不怎么会变动的依赖关系写成 Java 的注解（*Annotation*），如果能这样的话，一切就圆满了。

但是，真的可以吗？

我可以负责任的告诉你，完全是可以滴 ^_^

首先这篇文章，会详细讲解一下如果通过注解来配置你的容器对象，而 Ioc 复合加载器一篇，将会告诉你如何组合多个加载器，这样你就可以把你的对象依赖关系分别写到 xml,json,以及 Java 注解里，组合使用了。

5.8.2. 如何使用 AnnotationIocLoader

同 JsonLoader 一样，你可以直接 new 一个 AnnoationIocLoader

```
[Java]
Ioc ioc = new NutIoc(new
AnnotationIocLoader("com.you.app.package0",
"com.you.app.package1"));
```

当然在 Nutz.Mvc 中，你可以通过 IocProvider 来初始化 Ioc 容器，所以你可以在你的 MainModule 上这么声明

```
[Java]
@IocBy(type = AnnotationIocProvider.class,
      args = { "com.you.app.package0",
            "com.you.app.package1"})
public class MainModule {
```

```
    ....
```

这样，你在

* com.you.app.package0
* com.you.app.package1

这两包下，所有声明了 **@IocBean** 这个注解的对象，都会被认为是容器对象。是的，通过注解 **@IocBean**， AnnotationIocLoader 就能辨别你指定的包中，哪些类是可以交由容器管理的。

那么， **@IocBean**里面还能声明什么信息，我怎么为我的容器对象设置注入内容呢？ 请继续看下面内容 ^_^

5.8.3. 指定对象的名称

任何一个 Ioc 容器管理的对象，都必须有一个名字，以便通过:

```
[Java]
MyObject obj = ioc.get(MyObject.class, "myObjectName");
```

来获取对象。

因此，你可以为你的对象这么声明:

```
[Java]
@IocBean(name="myObjectName")
public class MyObject {
    ...
```

如果你的对象名字为你对象类名的首字母小写形式，你可以省略名字这个属性, 即

```
[Java]
@IocBean
public class MyObject {
    ...
```

同

```
[Java]
@IocBean(name="myObject")
public class MyObject {
    ...
```

效果是一样的。

还有另外一种方法，你可以为你的对象声明一个单独注解：

```
[Java]
@InjectName("myObjectName")
@IocBean
public class MyObject {
    ...
```

列位，看到这里，可能有人会问了，这不是脱裤子放屁吗？ @IocBean 可以有 name 属性，而你又搞了一个@InjectName 注解专门声明名字，你到底打算让我们用哪一个？你逗我们玩哪？

首先，我得跟大家声明一下，这的确是一点点历史问题，原先的 @InjectName 是给 Nutz.Mvc 用的，它如果发现了子模块声明了这个属性，就交付给 Ioc 容器管理。 后来，我们发现，介个名字和 @IocBean 的名字必须是相同的，所以在 AnnotationIocLoader 里，我们做了如下优先级的判断:

1. 如果发现 @IocBean 有 name 属性，这个对象就采用这个名字
2. 如果没有 @IocBean(name="xxxx")，哪么就看看有没有声明了 @InjectName
3. 还没有的话，就用对象的类名首字母小写形式作为这个对象的名称

因此对于一个 Nutz.Mvc 的模块类来说， @InjectName + @IocBean 是一个比较方便的写法。

但是现在我也承认， @IocBean 的 name 属性有点多余，或者 @InjectName 有点多余。 但是由于是过了几个版本以后才认识到的这个问题，所以我想，不如就留着这个设计，作为 Nutz 这个项目的一段 **盲肠**， 希望列位看官理解我们的苦衷，毕竟我们宣称了接口不会有重大变动之后，就要拿掉这个盲肠话，仿佛自己打了一记自己嘴巴。因此，人类的劣根性导致我们这么安慰自己： “没事没事，这个设计虽然有一点点臃肿，但是没人让人更难用，也过得去啦 -_-!”

5.8.4. 不要单例

默认的，Ioc 容器管理的对象都是单例的，你如果不想单例，你可以:

```
[Java]
@IocBean(name="myObject", singleton=false)
public class MyObject {
    ...
```

5.8.5. 为对象传递构造函数

当然 @IocBean 这点是不够，很多对象注入的时候，需要为构造函数声明信息，你可以这样：

```
[Java]
@IocBean(name="myObject", args={"a string", "refer:anotherObject",
true, 234})
public class MyObject {
    ...
```

看，简单不？ 你的构造函数有多少个参数，你就一并在 "args" 属性里声明就好了， 那么你都能注入什么呢？

它注入的值同字段注入的值描述方式相同，请继续看下面一节，我们有更详细的解释

5.8.6. 为对象的字段注入

这个更加简单，比如:

```
[Java]
@IocBean
public class MyObject {
    @Inject("abcc")
    private String name;
    @Inject("true")
    private boolean live;
    @Inject("refer:another")
    private AnotherObject obj;
```

```
...
```

那么你到底能注入什么呢？ 感兴趣的同学可以看这里：[你都可以注入什么]。当然，同 Json 的方式有点不同，你这里直接写 "refer:xxxx" 或者 "env:xxxx" 就好了。下面是一个列表

@Inject("Hello world")	字符串
@Inject("135897415")	数字
@Inject("true")	布尔型值
@Inject("refer:objName")	注入容器其他对象的引用
@Inject("refer:$Ioc")	容器自身
@Inject("refer:$Name")	对象的名称，即你在 @InjectName 或者 @IocBean 里声明的 name
@Inject("refer:$Context")	容器上下文
@Inject("env:JAVA_HOME")	系统环境变量
@Inject("sys:user.home")	虚拟机属性
@Inject("jndi:jndiName")	JNDI 资源
@Inject("file:/home/zzh/abc.txt")	文件对象
@Inject("java:com.my.SomeClass.staticPropertyName")	调用某 JAVA 类的静态属性
@Inject("java:com.my.SomeClass.someFunc")	调用某 JAVA 类的静态函数
@Inject("java:com.my.SomeClass.someFunc("p1",true)")	调用某 JAVA 类的带参数的静态函数
@Inject("java:$xh")	获得容器对象 xh ，相当于 "refer:xh"
@Inject("java:$xh.name")	容器对象某个属性
@Inject("java:$xh.getXXX()")	容器对象某个方法的返回值

@Inject("$xh.getXXX("some string", true, 34)")	容器对象某个方法的返回值

基本上，看到上面的表格，你同时也能完全明白怎么 #为对象传递构造函数，不是吗？

5.8.7. 声明对象的事件

在 Nutz.Ioc 容器里，一个对象有三种事件：

1. create - 当对象被创建时触发
2. fetch - 当对象被从容器中取出时触发
3. depose - 当对象被销毁时触发

同 JSON 配置一样， Annotation 方式的配置也允许你声明这三种事件的处理方式

```
[JAVA]
@IocBean(
    create = "init",
    fetch = "com.myapp.MyObjectOnFetchTrigger",
    depose = 'onDepose'
)
public class MyObject {
    ...
```

同 JSON 的配置方式相同的是，你可以为该对象处理该事件的一个函数，比如上面的例子，MyObject 需要有一个函数**init** 来处理创建时的事件，还需要一个函数 **onDepose** 来处理注销时的事件。当然这两个函数不能时静态的，也不能有任何参数。 对一个事件，你还可以声明一个 **IocEventTrigger** 的实现类，来处理一个事件。比如上面的例子，我们就用 com.myapp.MyObjectOnFetchTrigger 来处理 fetch 事件的。

看到这里，可能有的同学会弱弱的问了，这个功能到底是干虾米滴？ 我认为所有需要问这个问题的同学都可以无视这个功能，因为你根本不需要它。但是如果你想在对象创建的时候，做点初始化工作，比如为某几个字段设设值；或者你希望在容器注销你的对象时，你想同时释放点资源，比如数据连接池对象被销毁时，你需要释放一下池内的连接；又或者你想为你的对象做一个计数，每次从容器获取的时候，计数 + 1，用来统计你对象被使用的频率，我想你也很需要 fetch 这个事件。无论你采用对象的函数，还是自己实现 IocEventTrigger 这个接口，你会拿到容器里的对象实例，然后你想做什么完全随你喽 :)

5.8.8. 如果要注入的字段在超类怎么办

答

```
[Java]
@IocBean(fields={"dao"})
public AbcService extends Service {
    ...
```

比如上例，你的 AbcService 从 Service 继承，但是 Serivce 有一个私有字段为 "private Dao dao"， 你怎么描述它的注入呢？ 你可以通过 @IocBean 注解提供 fields 属性，描述你要注入超类那个字段，比如上面的例子，我们会为超类的 "dao" 字段注入一个名为 "dao" 容器对象。

但是，如果我想注入的容器对象同超类的那个字段名字不一样怎么办？ 或者我不打算注入一个容器对象，我想注入一个字符串，或者布尔值怎么办？ 呵呵，抱歉，截止到现在，我们还没有解决这个问题，不过很快我们会给出一个设计，并且我可以负责的说，给出新的设计一定会兼容现在的这个用法的。

5.9. Ioc - 复合加载器

5.9.1. 为什么需要复合加载器

Ioc 实际上一种将应用的耦合集中在一起管理的一种程序结构设计方式。耦合集中的具体形式一般是各种格式的配置文件。比如 Spring 就有它自己的配置文件格式的规定。

Nutz.Ioc 在设计之初就没有特别假定用户会把配置文件存成某种特定的格式，关于这点，各位可以参看 定义自己的配置文件格式 一文。

同时，我们发现，在实际应用的时候，应用程序的耦合大概分做两种：

1. 千秋万载，基本不变
2. 部署之时，可能改变

第一种耦合关系，其实更适合写成 Annotation，这样程序发布之后就不必担心人为的错误修改了。而第二种则更适合写在配置文件里，你们的部署工程师可以看着你提供的小文档，根据客户现场的情况调整你的应用的各种参数。

如果将这两种耦合关系都写在配置文件里，首先部署工程师会看到一大堆你都可能忘记是什么意思的配置信息，当这些符号映入眼帘之时，恐惧会瞬间将TA吞没，嚼的骨头都不剩。是的，这就是面对陌生的事物，正常人类很正常的一种反应，怪不得他不是吗？

或者可怜的部署工程师手里拿的是你几个星期吐血写出来的长长的系统参数手册，那么写这个手册之前，感到恐怖的可能就是你了，我亲爱的同学。之后，他在鼓起勇气阅读的之前，通常也会倒吸一口凉气。

当然，人的智慧就是在这过程中得到了淬炼，你的意志力，你的智慧，都会得到不同程度的升华。但是 Nutz 这个小框架开发的初中并不是希望锻炼你，折磨你，事实上，它希望尽一切可能，让你远离这种锻炼...

...为此，我们也提供一个复合加载器，你可以将你的耦合关系写在配置文件中，或者Annotation中，怎样分配则由亲爱的程序员同学你来亲自决定。

5.9.2. 复合加载器的使用方法

复合加载器非常简单，似乎只花了 Wendal 同学不到 1 个小时的时间，因为它本身并不做任何事情，它只是调用其他的加载器：

```
[Java]
Ioc ioc = new NutIoc(new ComboIocLoader(
            "*org.nutz.ioc.loader.json.JsonLoader",
            "dao.js",
            "service.js",
            "*org.nutz.ioc.loader.annotation.AnnotationIocLoader",
            "com.myapp.module",
            "com.myapp.service"
        ));
```

如上面的例子，组合加载器，组合了两个 Ioc 加载器，一个是 JsonLoader，一个是AnnotationIocLoader。

ComboIocLoader 的构造函数是个字符串形变参数组，所有的参数，如果以星号 "**" 开头，则被认为是加载器的类型，后面的参数都作为这个加载器构造函数的参数，直到遇到下一个星号 "**" 开头的参数。

上面的例子，实际上为 Ioc 容器准备了这两个加载器，第一个是 JSON 加载器，它从 dao.js 和 service.js 这两个配置文件中读取容器对象的配置信息。 而另外一个 Annotation 加载器，从会扫描包 **com.myapp.module** 以及 **com.myapp.service** 已经其下的子包，如果发现有容器对象（声明了 @IocBean 注解，详情请看 Ioc - Annotation 加载器）就会加载。

并且这两个加载器的优先级为

排在前面前面的加载器更优先，

在本节的例子中， JsonLoader 加载器比 AnnotationIocLoader 更加优先.就是说，如果两个加载器都加载了同名对象，则以 JsonLoader 的为准

5.9.3. 在 Mvc 中的用法

在 Nutz.Mvc 中，Ioc 容器是由 IocProvider 接口来提供的，所以，每个加载器都由一个类似的 IocProvier 实现。比如 ComboIocLoader，也就有一个 ComboIocProvider。

但是，请千万要注意: 对于 ComboIocProvider:

它组合的依然是其他 IocLoader

而不是

其他 IocProvider

下面给个例子：

```
[Java]
@IocBy(type = ComboIocProvider.class,
      args = {"*org.nutz.ioc.loader.json.JsonLoader",
              "dao.js",
              "service.js",
              "*org.nutz.ioc.loader.annotation.AnnotationIocLoader",
              "com.myapp.module",
              "com.myapp.service"
       })
public class MainModule {
    ...
```

它是什么意思呢？ 不解释，你懂的。

5.9.4. 现在你都可以复合什么?

* XmlIocLoader
* JsonLoader
* AnnotationIocLoader
* MapLoader
* 其他任何你自己实现的 IocLoader

5.10. Ioc 注解列表

Ioc 部分用到的 Java 注解比较少，具体请看下表

@InjectName	用来与 Mvc 容器的扩展点，声明了这个注解的模块，会被 Ioc 容器管理
@IocBean	AnnotationIocLoader 根据这个注解来判断哪些类应该被自己加载
@Inject	AnnotationIocLoader 根据这个注解来了解类中的字段，具体的注入方式

关于 @InjectName 更详细的描述，请参看 同 Ioc 容器一起工作

关于 AnnotationIocLoader (注解加载器)的详细描述请参看 Ioc - Annotation 加载器

5.11. 让Ioc容器帮你规划配置文件

5.11.1. 配置文件的痛苦

一个 Java 项目，无论大小，多半是有那么几个配置文件的，比如:

* 数据库连接啦
* 关键的文件路径啦
* 一些曝露给运维人员的配置项啦

如果我们采用了 Ioc 方式组织我们的程序，我们（程序员）会理直气壮的对运维人员说：“你去改xxxx Ioc 文件去。”

老实说，运维人员会恨死你，我列一个很简单的 JSON 配置文件:

```
[CODE]
var ioc = {
    // 数据源
    dataSource : {
        type :"org.apache.commons.dbcp.BasicDataSource",
        events : {
            depose :"close"
        },
        fields : {
            driverClassName : "com.mysql.jdbc.Driver",
            url             : "jdbc:mysql://127.0.0.1:3306/mydb",
            username        : "root",
            password        : "123456",
            initialSize     : 10,
            maxActive       : 100,
```

```
            testOnReturn    : true,
            //validationQueryTimeout : 5,
            validationQuery : "select 1"
        }
    },
    // Dao
    dao : {
        type :'org.nutz.dao.impl.NutDao',
        args : [ {refer :"dataSource"}]
    }
};
```

这个配置文件就是简单的配置了以下数据源，以及一个 Dao 对象。一个运维人员打开这个文件，首先映入眼帘的就是 "org.apache.commons.dbcp.BasicDataSource" 以及 depose :"close" 他会吓的够呛，心里嘀咕:"这TM是神马东西！"

运维人员希望看到什么呢？ 他希望看到这个:

```
[CODE]
db-driver=com.mysql.jdbc.Driver
db-url=jdbc:mysql://127.0.0.1:3306/mydb
db-username=root
db-password=123456
```

这样格式的文件，傻子也知道怎么维护。

当然，很多小JIAN人喜欢XML，这样的文件

```
[CODE]
<db>
    <driver>com.mysql.jdbc.Driver</driver>
    <url>jdbc:mysql://127.0.0.1:3306/mydb</url>
    <username>root</username>
    <password>123456</password>
</db>
```

TA 看到会欢喜的不得了。

总之，有没有什么办法，能够让运维人员之看到他们喜欢看到的文件，而程序员则继续维护自己的 Ioc 文件呢？ 怎么把这两种文件连接起来呢？

有，实际上，从很早以前，Nutz 就很好的支持了这样的做法，不过这次，郑重写个文档出来，也是为了避免有人再有这个问题

答案是：**采用 java 型注入值**

5.11.2. 用 Properties 举个例子

就用上面的例子，如果我们需要暴露给运维人员一个 properties 文件任其修改：

```
[CODE]
----------------------------------------------------[下面是
myapp.properties 文件的内容]---
db-driver=com.mysql.jdbc.Driver
db-url=jdbc:mysql://127.0.0.1:3306/mydb
db-username=root
db-password=123456
```

那么我们的 Ioc 文件就改成这样：

```
[CODE]
var ioc = {
    // 读取配置文件
    config : {
        type : "org.nutz.ioc.impl.PropertiesProxy",
        fields : {
            paths : ["myapp.properties"]
        }
    },
    // 数据源
    dataSource : {
        type :"org.apache.commons.dbcp.BasicDataSource",
        events : {
            depose :"close"
        },
```

```
        fields : {
            driverClassName : {java :"$config.get('db-driver')"},
            url             : {java :"$config.get('db-url')"},
            username        : {java :"$config.get('db-username')"},
            password        : {java :"$config.get('db-password')"},
            initialSize     : 10,
            maxActive       : 100,
            testOnReturn    : true,
            //validationQueryTimeout : 5,
            validationQuery : "select 1"
        }
    },
    // Dao
    dao : {
        type :'org.nutz.dao.impl.NutDao',
        args : [ {refer :"dataSource"}]
    }
};
```

这里有几个重点

1. Nutz 提供了一个类 org.nutz.ioc.impl.PropertiesProxy，他能读取并解析一个 properties 文件
2. 在 Ioc 容器中，我们创建一个这样的单例对象，随便起个名字，比如叫 "config"
3. 那么根据配置 "myapp.properties" 会被 **PropertiesProxy** 类加载
4. 通过 java 调用，你可以直接调用 "config" 对象的 get 方法
5. 这样，你就能将分散在各个 Ioc 文件中的值集中到一个 properties 文件里了

5.11.3. 关于XML

如果你打算给你的运维人员看 XML 怎么办呢？抱歉，Nutz 木有给出内部支持，但是简单的要命，你需要：

1. 提供一个 XML 配置解析类，从xml读取内容，然后，提供 get 方法
2. 在 Ioc 配置的任何地方，你都可以用 java 调用的方式，调用 get 方法

5.11.4. 我是用Annotation的Ioc

如果是这样，那么 @Inject 可以这样写:

```
[CODE]
```

```
@Inject("java:$config.get('xxxxx')
private String myXXXX;
```

5.11.5. 关于更多扩展的意淫

现在的云端应用，你不弄七八台机器放在一负载均衡后面，你都不好意思叫它们服务器。并且这些机器基本都是跑一样的程序，用一样的配置文件，很多运维人员（尤其是不会写脚本的鼠标派运维人员）会很痛苦：

“配置文件同步害死人呀！”

那么，如果你把关键的配置信息放到一张数据表里，然后自己提供一个类 ...

```
[CODE]
+--------+              +------------+       +----------+
| Ioc    | <<< {java:...} <<<  | Your Class |  <<<<<<  | Database |
+--------+              +------------+       +----------+
```

你只需要修改数据表，然后重启各个应用 ......

根据这个思路，你可以将你的关键配置信息汇集在:

* 数据表里
* YAML 或者其它什么运维人员喜欢的格式上
* 电子表格中

写一个解析类动态读取，于是，整个世界就会安静了 ...

6. Aop 手册

6.1. AOP模型 -- NutAop的实现思路

6.1.1. 为什么需要特别写出NutAop的模型呢?

* NutAop清晰简洁地实现了Aop中最常用,最常见的需求--控制特定方法的执行逻辑
* 了解这个模型,能够更好地理解Aop

6.1.2. 基本思路

原方法(没有返回值)

```
[CODE]
public void exe(){
    //Do something
}
```

被改造后

```
[CODE]
import org.nutz.aop.InterceptorChain;
....
public void exe(){
    try{
        new InterceptorChain(XX,XXX...).doChain();
    }catch (Throwable) {
        throw e;
    }
}
```

其中使用到的InterceptorChain,可以理解为Servlet里面的FilterChain,它携带着执行原方法时可以获取的信息,如调用对象.调用方法,参数,返回值等整个 InterceptorChain 不到100行,我相信你很快就能看完.懒人的话,只看doChain()/invoke(),你将会豁然开朗.

整体思路

* 将方法调用的信息,拦截器信息,全部封装到InterceptorChain

* 调用InterceptorChain里面第一个拦截器,并由该拦截器决定是否进行,如果继续,则调用doChain()
* 当InterceptorChain中最后一个拦截器也调用了doChain(),在开始调用原方法实现,然后返回.
* 由于doChain()返回了,堆栈开始往回走,依次通过原本的拦截器, 这时,你可以改变/替换原本的返回值

6.1.3. 具体实现

如果在你的方法中打印堆栈,你将看到类似的信息

```
[CODE]
java.lang.Throwable
    at org.nutz.aop.asm.test.Aop1.mixArgsVoid(Aop1.java:55)
    at org.nutz.aop.asm.test.Aop1$$NUTZAOP._aop_invoke(Unknown
Source)
    at org.nutz.aop.InterceptorChain.invoke(InterceptorChain.java:51)
    at
org.nutz.aop.InterceptorChain.doChain(InterceptorChain.java:39)
    at
org.nutz.aop.interceptor.LoggingMethodInterceptor.filter(LoggingMe
thodInterceptor.java:29) //拦截器B
    at
org.nutz.aop.InterceptorChain.doChain(InterceptorChain.java:42)
    at
org.nutz.aop.AbstractMethodInterceptor.filter(AbstractMethodInterc
eptor.java:10) //拦截器A
    at
org.nutz.aop.InterceptorChain.doChain(InterceptorChain.java:42)
    at org.nutz.aop.asm.test.Aop1$$NUTZAOP.mixArgsVoid(Unknown
Source)
```

mixArgsVoid 就是你原本方法

6.1.4. 注意

* 切勿自行实现org.nutz.aop.AopCallback接口,并不要使用该接口声明的方法签名
* 虽然org.nutz.aop.MethodInterceptor接口允许你抛出任何异常,但*请不要抛出原方法未声明的受检异常*

6.2. AOP -- 声明式切片

6.2.1. 单纯的AOP

6.2.1.1. 创建一个拦截器类,在方法执行前后打印一句日志

```
[CODE]
package aop;
import org.nutz.aop.ClassAgent;
import org.nutz.aop.ClassDefiner;
import org.nutz.aop.DefaultClassDefiner;
import org.nutz.aop.InterceptorChain;
import org.nutz.aop.MethodInterceptor;
import org.nutz.aop.asm.AsmClassAgent;
import org.nutz.aop.matcher.MethodMatcherFactory;
public class UserAction { //被AOP的类,必须是public的非abstract类!
    /*将要被AOP的方法*/
    public boolean login(String username, String password) throws
Throwable {
        if ("wendal".equals(username) &&
"qazwsxedc".equals(password)) {
            System.out.println("登陆成功");
            return true;
        }
        System.out.println("登陆失败");
        return false;
    }
    private static ClassDefiner cd = new
DefaultClassDefiner(UserAction.class.getClassLoader());
    public static void main(String[] args) throws Throwable {
        //无AOP的时候
        UserAction ua = new UserAction(); //直接new,将按原本的流程执行
        ua.login("wendal", "qazwsxedc");
        System.out.println("----------------------------------------------
----");
        System.out.println("----------------------------------------------
----");
        //有AOP的时候
        ClassAgent agent = new AsmClassAgent();
        LogInterceptor log = new LogInterceptor();
agent.addInterceptor(MethodMatcherFactory.matcher("^login$"), lo
g);
        //返回被AOP改造的Class实例
        Class<? extends UserAction> userAction2 = agent.define(cd,
UserAction.class);
```

```
        UserAction action = userAction2.newInstance();
        action.login("wendal", "qazwsxedc");//通过日志,可以看到方法执行
前后有额外的日志
    }
}
class LogInterceptor implements MethodInterceptor {
    public void filter(InterceptorChain chain) throws Throwable {
        System.out.println("方法即将执行 -->" +
chain.getCallingMethod());
        chain.doChain();// 继续执行其他拦截器,如果没有,则执行原方法
        System.out.println("方法执行完毕 -->" +
chain.getCallingMethod());
    }
}
```

6.2.1.2. 输出

```
[CODE]
登陆成功
------------------------------------------------------
------------------------------------------------------
方法即将执行 -->public boolean
aop.UserAction.login(java.lang.String,java.lang.String) throws
java.lang.Throwable
登陆成功
方法执行完毕 -->public boolean
aop.UserAction.login(java.lang.String,java.lang.String) throws
java.lang.Throwable
```

6.2.2. 在Ioc中使用Aop

只有被Ioc容器管理的对象,才能使用AOP!!

6.2.2.1. 声明拦截器

* 你需要有一个拦截器对象，如果你愿意，你当然可以有不止一个拦截器对象。
* 将这个对象声明在你的Ioc配置文件里，就像一个普通的对象一样

6.2.2.2. 在对象的方法中声明切片

* 在你要拦截的方法上，声明 @Aop 注解或者其他配置形式,如js/xml
* @Aop 注解接受数目可变的字符串，每个字符串都是一个拦截器的名称,即必须在ioc中声明这个拦截器

* 方法所在的对象必须是Ioc容器中的对象

6.2.2.3. 将上一个例子,改造为Ioc形式

```
[CODE]
package aop;
import org.nutz.ioc.aop.Aop;
import org.nutz.ioc.loader.annotation.IocBean;
import org.nutz.ioc.loader.annotation.AnnotationIocLoader;
import org.nutz.ioc.Ioc;
import org.nutz.ioc.impl.NutIoc;
@IocBean
public class UserAction { //被AOP的类,必须是public的非abstract类!
    @Aop({"logInterceptor"}) //这里写拦截器bean的名字
    public boolean login(String username, String password) throws
Throwable {
        if ("wendal".equals(username) &&
"qazwsxedc".equals(password)) {
            System.out.println("登陆成功");
            return true;
        }
        System.out.println("登陆失败");
        return false;
    }
    public static void main(String[] args) throws Throwable {
    Ioc ioc = new NutIoc(new AnnotationIocLoader("aop"));
    UserAction action = ioc.get(UserAction.class);
    action.login("wendal", "qazwsxedc");
    }
}
//另外一个类文件
package aop;
import org.nutz.ioc.loader.annotation.IocBean;
import org.nutz.aop.InterceptorChain;
import org.nutz.aop.MethodInterceptor;
@IocBean //声明为一个Ioc的bean,名字为logInterceptor
public class LogInterceptor implements MethodInterceptor {
    public void filter(InterceptorChain chain) throws Throwable {
        System.out.println("方法即将执行 -->" +
chain.getCallingMethod());
        chain.doChain();// 继续执行其他拦截器
        System.out.println("方法执行完毕 -->" +
chain.getCallingMethod());
```

```
    }
}
```

6.2.3. 已经为你准备好的拦截器

* org.nutz.aop.interceptor.LoggingMethodInterceptor 添加日志记录
* org.nutz.aop.interceptor.TransactionInterceptor 添加数据库事务(用于NutDao)

6.3. 用json文件声明Aop切片

6.3.1. 需使用的类

* org.nutz.ioc.aop.config.impl.JsonAopConfigration

6.3.2. 看看一个示例的ioc配置文件

配置示例:

```
[json]
var ioc = {
    log : {
        type :'org.nutz.aop.interceptor.LoggingMethodInterceptor'
    },
    myMI : {
        type : 'org.nutz.ioc.aop.config.impl.MyMI'
    },
    pet2 : {
    type : "org.nutz.ioc.aop.config.impl.Pet2"
    },
    $aop : {
        type : 'org.nutz.ioc.aop.config.impl.JsonAopConfigration',
        fields : {
            itemList : [
                ['.+','toString','ioc:log'],
                ['.+','.+','ioc:myMI'],
                ['.+','.+','org.nutz.ioc.aop.config.impl.MyMI2','false']
            ]
        }
    }
}
```

可以看到, 除了$aop这个beanName外,其他的与普通的ioc配置文件没有任何区别.

$aop ,其实是org.nutz.ioc.aop.config.AopConfigration接口的IOCNAME字段的值,只有你声明这个名字,且类型为这个接口的实现,就能轻易的配置Ioc.

估计你已经猜到,org.nutz.ioc.aop.config.impl.JsonAopConfigration就是其中一个实现!

细看这个部分代码:

```
[CODE]
        fields : {
            itemList : [
                ['.+','toString','ioc:log'],
                ['.+','.+','ioc:myMI'],
                ['.+','.+','org.nutz.ioc.aop.config.impl.MyMI2','false']
            ]
        }
```

使用JsonAopConfigration,只需要为其itemList赋值.

需要什么值? 对,一个数组.

数组的每一行,对应一条规则:

```
[CODE]
['.+','toString','ioc:log'],
['.+','.+','ioc:myMI']
['com\.wendal\.nutz\..+','get.+','org.nutz.ioc.aop.config.impl.MyMI2','f
alse']
```

6.3.2.1. 规则如下:

* 第一个值,对应className,必选,用于匹配类的全称的正则表达式
* 第二个值,对应methodName,必选,用于匹配方法名的正则表达式
* 第三个值,对应interceptorName,必选,如果以ioc:开头,则代表对于ioc容器的一个对象,否则,将认为是一个类名
* 第四个值,对应singleton,可选,仅当interceptorName为类名时有效

6.3.3. 拓展使用 -- 声明式事务

首先,声明5种事务等级对应的拦截器(使用内置的事务拦截器 org.nutz.aop.interceptor.TransactionInterceptor)

```
[CODE]
    txNONE : {
        type : 'org.nutz.aop.interceptor.TransactionInterceptor',
        args : [0]
    },
    txREAD_UNCOMMITTED : {
        type : 'org.nutz.aop.interceptor.TransactionInterceptor',
        args : [1]
    },
    txREAD_COMMITTED : {
        type : 'org.nutz.aop.interceptor.TransactionInterceptor',
        args : [2]
    },
    txREPEATABLE_READ : {
        type : 'org.nutz.aop.interceptor.TransactionInterceptor',
        args : [4]
    },
    txSERIALIZABLE : {
        type : 'org.nutz.aop.interceptor.TransactionInterceptor',
        args : [8]
    },
    //声明一个log进行日志记录
    log : {
        type :'org.nutz.aop.interceptor.LoggingMethodInterceptor'
    }
```

然后,定义哪些类的什么方法需要进行声明,继续添加 (一般来说,应该把等级高的往后放)

```
[CODE]
    $aop : {
        type : 'org.nutz.ioc.aop.config.impl.JsonAopConfigration',
        fields : {
            itemList : [
                ['com\\.service\\..+','.+','ioc:log'],
```

```
            ['com\\.service\\.status\\..+','(get|set).+','ioc:txNONE'],
['com\\.service\\.media\\..+','(get|set).+','ioc:txREAD_UNCOMMITTED'
],
['com\\.service\\.news\\..+','(get|set).+','ioc:txREAD_COMMITTED'],
['com\\.service\\.blog\\..+','(get|save|update|delete).+','ioc:txREPEATA
BLE_READ'],
            ['com\\.service\\.auth\\..+','.+','ioc:txSERIALIZABLE']
        ]
    }
}
```

6.3.3.1. 按照上述声明:

* 对于com.service包下的类的全部方法,均应用log拦截器
* 对于com.service.status包下的类的全部get/set方法,均应用txNONE拦截器,事务级别 NONE
* 对于com.service.media包下的类的全部get/set方法,均应用txREAD_UNCOMMITTED拦截器,事务级别 READ_UNCOMMITTED
* 对于com.service.news包下的类的全部get/set方法,均应用txREPEATABLE_READ拦截器,事务级别 READ_COMMITTED
* 对于com.service.blog包下的类的全部get/set/update/delete方法,均应用txREPEATABLE_READ拦截器,事务级别 READ_REPEATABLE_READ
* 对于com.service.auth包下的类的全部方法,均应用txSERIALIZABLE拦截器,事务级别 SERIALIZABLE

6.3.4. 重要提醒 -- 与@Aop同时使用

如果你既使用了@Aop注解,又配置了上述的声明式Aop,你需要ComboAopConfigration来整合两种配置,示例:

```
[CODE]
$aop : {
    type : 'org.nutz.ioc.aop.config.impl.ComboAopConfigration',
    fields : {
        aopConfigrations  : [
            {type : 'org.nutz.ioc.aop.config.impl.JsonAopConfigration',
                fields : {
    itemList : [
            ['com\\.service\\..+','.+','ioc:log'],
                    ['com\\.service\\.status\\..+','(get|set).+','ioc:txNONE'],
['com\\.service\\.media\\..+','(get|set).+','ioc:txREAD_UNCOMMITTED'
],
```

```
['com\\.service\\.news\\..+','(get|set).+','ioc:txREAD_COMMITTED'],
['com\\.service\\.blog\\..+','(get|save|update|delete).+','ioc:txREPEATA
BLE_READ'],
                    ['com\\.service\\.auth\\..+','.+','ioc:txSERIALIZABLE']
    ]
                }
            },
            {type :
'org.nutz.ioc.aop.config.impl.AnnotationAopConfigration'}
        ]
    }
}
```

同理,你也可以整合XML声明式Aop.不过,为避免不必要的麻烦,请勿在不同配置方式中对同一个方法添加同一个拦截器

7. Mvc 手册

7.1. Nutz.Mvc 概述

7.1.1. 几句话的介绍

Nutz.Mvc 是要和一个 Web 服务器（比如 Tomcat）一起工作的，它存在的意义**就是要把一个标准的 HTTP 请求，转发到某一个 Java 函数中。**

它的特点是：

* 帮你做参数的解析（*当然你自己可以随意扩展*）
* 如果你愿意，你可以不用 View 来渲染 HTTP 输出流
* 提供几个基本的渲染方式（比如 JSP, JSON）
* 除此以外，不内置更多的功能

每一种 HTTP 请求，标识就是一个 URL，而每一个 Java 函数怎样和一个 URL 关联呢？ 最直接的办法就是在函数上声明一个注解，这个注解，在 Nutz.Mvc 中，叫 @At

你如果想快速开始，来一个 Hellow World， 那么请看 Hello World 一节

7.1.2. 图解 Nutz.Mvc

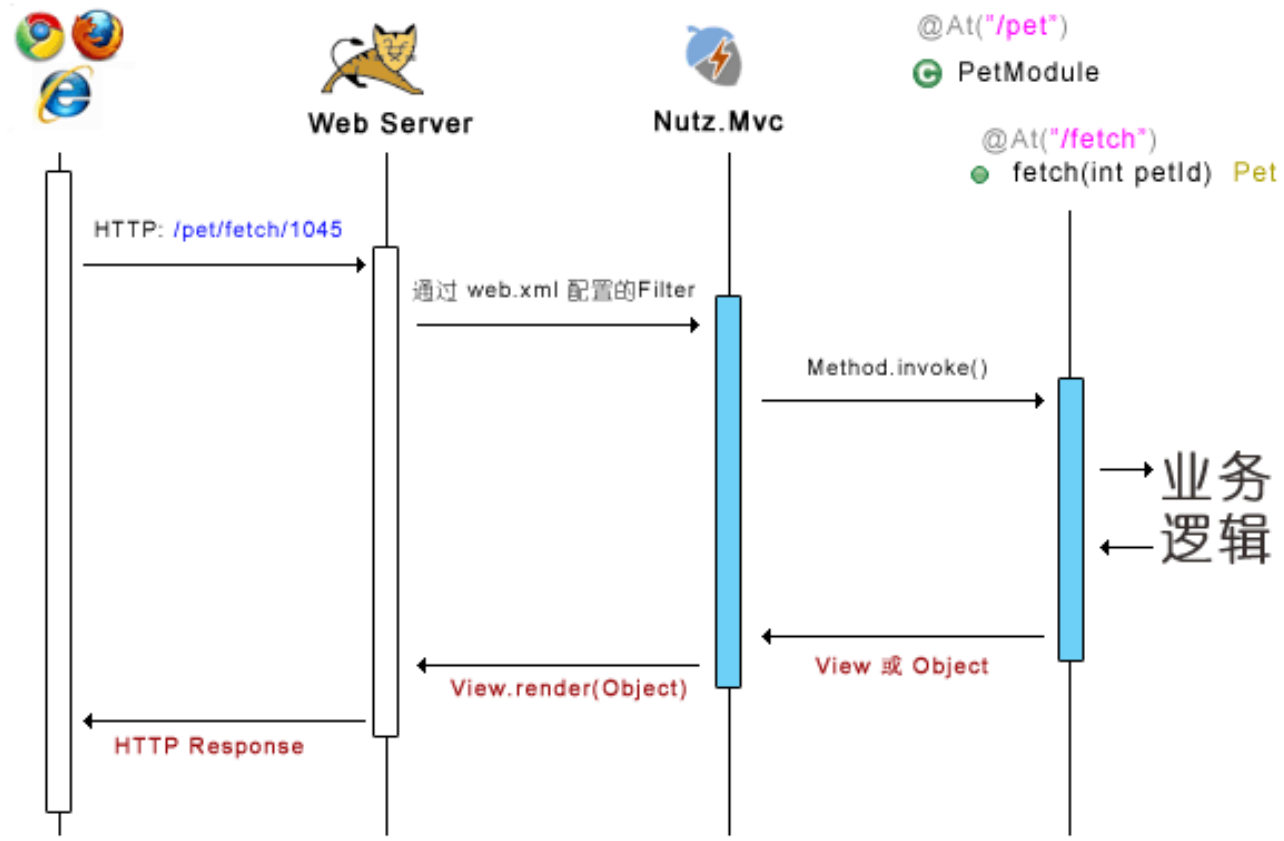
为了能让你更快速直观的了解 Nutz.Mvc 的工作方式，我提供下面两张图：

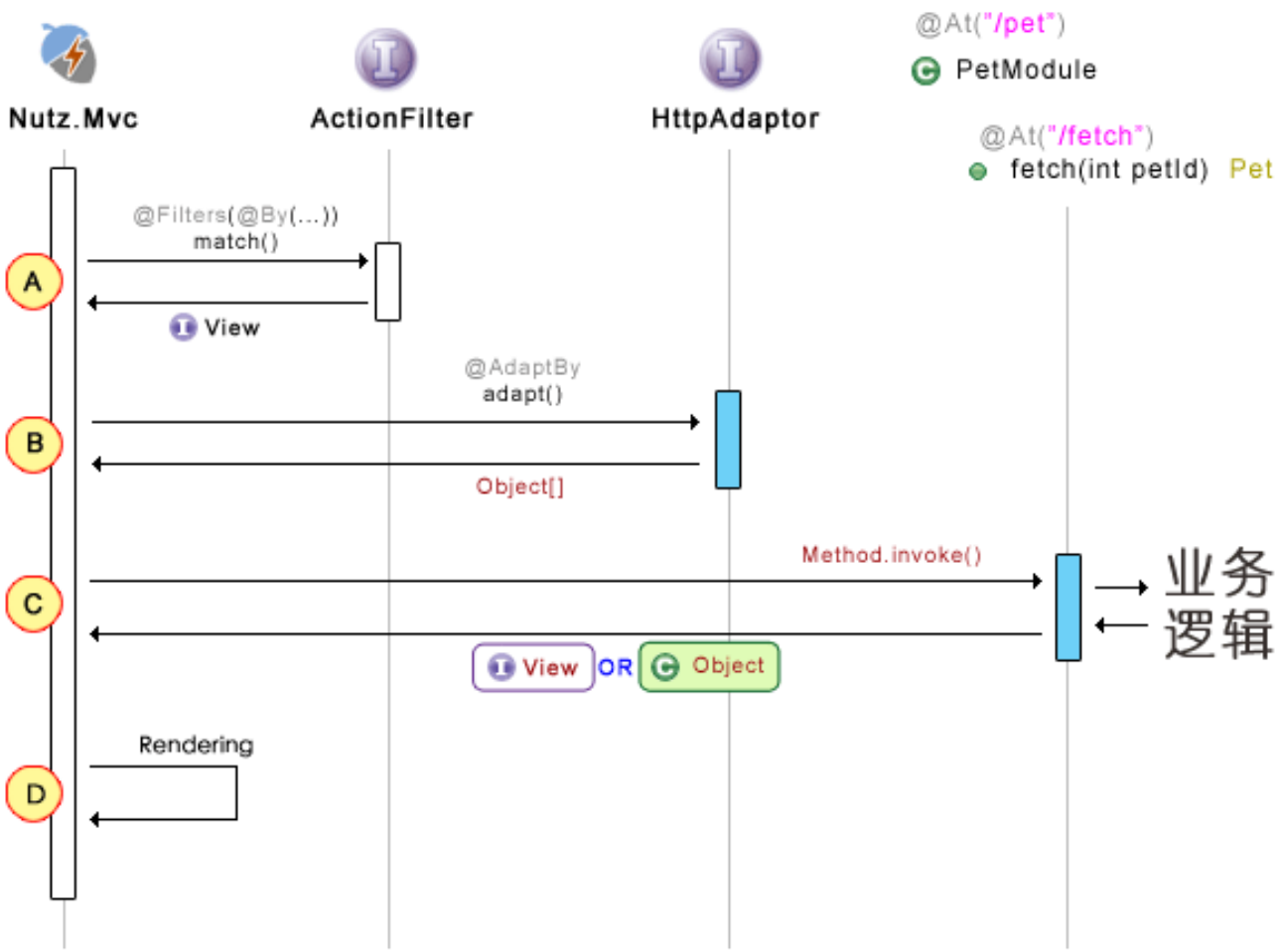
7.1.2.1. 每一次请求，都经过如下流程

Nutz.Mvc 根据 @At 注解，将一个 HTTP 请求映射到了一个函数，函数只有一个参数，会被路径参数（1045）填充。

* Nutz.Mvc 最基本的想法，就是通过注解 @At 将一个HTTP请求路径同一个 Java 函数关联起来。
* 并且，@At 支持你写多个路径

7.1.2.2. 更详细的流程


Web Server
Nutz.Mvc
@At("/pet")
PetModule
@At("/fetch")
fetch(int petId) Pet
HTTP: /pet/fetch/1045
通过 web.xml 配置的Filter
Method.invoke()
业务
逻辑
View 或 Object
View.render(Object)
HTTP Response


Nutz.Mvc
ActionFilter
HttpAdaptor
@At("/pet")
PetModule
@At("/fetch")
fetch(int petId) Pet
@Filters(@By(...))
match()
A
View
@AdaptBy
adapt()
B
Object[]
Method.invoke()
C
业务
逻辑
View
OR
Object
Rendering
D

* 声明了 @At 的函数被称为 **入口函数**
* 任何一个请求，都会经过四道工序
    1. **A - 过滤**: 你通过 @Filters 注解可以为你的**入口函数**定义任意多的过滤器
    2. **B - 适配**: 这个过程将 HTTP 输入流转换成**入口函数**的参数数组
        > 默认的，它认为输入流是传统的名值对方式
        > 更多的适配方式请参看 关于适配器
        > 文件上传也是一种适配方式，请参看 UploadAdptor
    3. **C - 调用**: 调用**入口函数**，你在里面需要调用相关的业务层代码。
        > 如果你的业务比较复杂，为了解耦合，你可能需要 Ioc 容器的帮助，请参看 同 Ioc 容器一起工作 一节
    4. **D - 渲染**: 根据**入口函数**的返回，渲染 HTTP Response。
        > 如果返回是个 View，则用这个 View 来渲染 null （null? 是的，你没看错，这种情况 View 接口第三个参数会是 null）
            - 你可以用 org.nutz.mvc.view.ViewWrapper 将你的返回对象以及要返回的视图组合在一起返回，ViewWrapper 也是一个 View
        > 否则用函数的 @Ok 注解声明的 View 来渲染**入口函数**的返回对象
        > 如果你的函数处理过程中抛出了异常，用 @Error 注解声明的 View 来渲染异常对象
        > 返回值会保存在request的attr中,名字是**obj**

7.1.3. 进阶阅读

* 如何配置 web.xml
* Nutz.Mvc 的配置语法

7.2. Hello World

7.2.1. 关于这个 Hello World

这是一个简单的小例子，通过简单的几步，即可在 Eclipse 中搭建出一个 Nutz.Mvc的运行环境，我假设你已经：

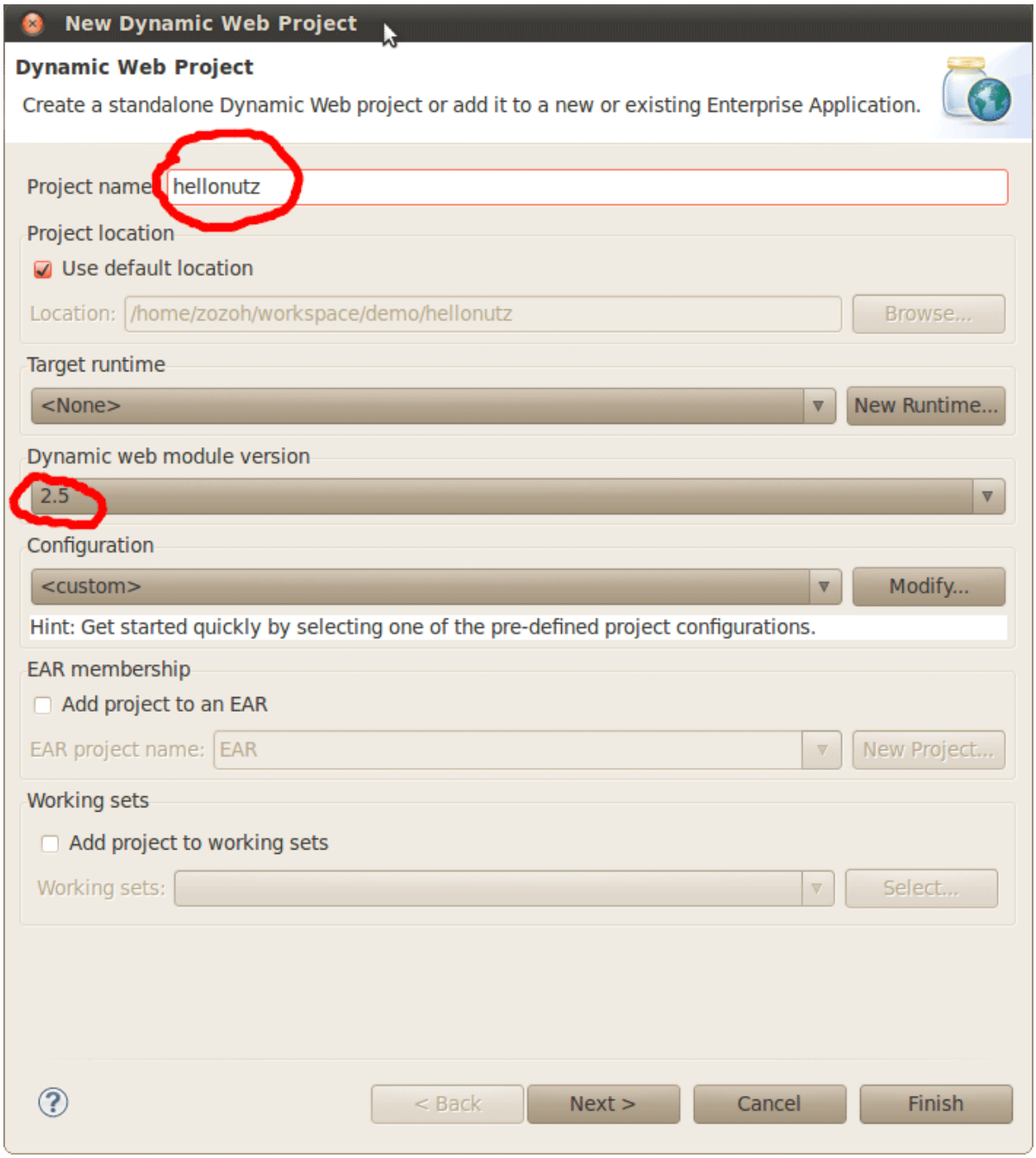
1. 有了 Eclipse J2EE 套件 -- Eclipse 3.5 以上版本
2. 安装了 Tomcat 6.0 及以上版本
3. 下载了 Nutz 的最新版本

7.2.2. 详细步骤

7.2.2.1. 创建一个 Dynamic Web Project

创建一个新的 Dynamic Web Project （File > New > Project ... > Web > Dynamic Web Project）

* 项目的名称为 hellonutz
* 这个例子使用的是 Tomcat 6.0，所以请将 Dynamic Web module version 设置成 2.5
    > 如果你的 Tomcat 是 7.0， 则可以设置成 3.0

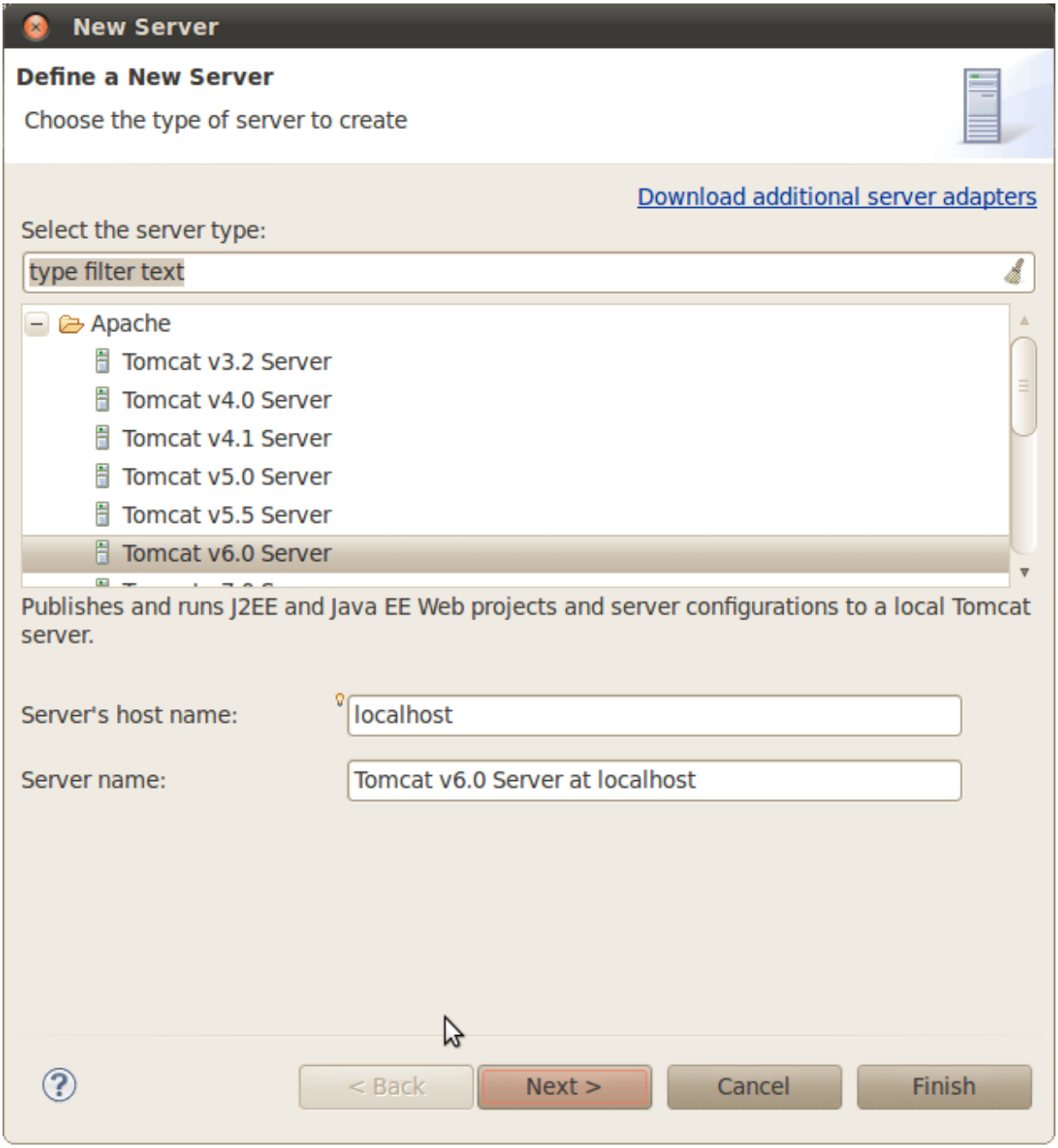


7.2.2.2. 创建一个 Tomcat 运行服务器

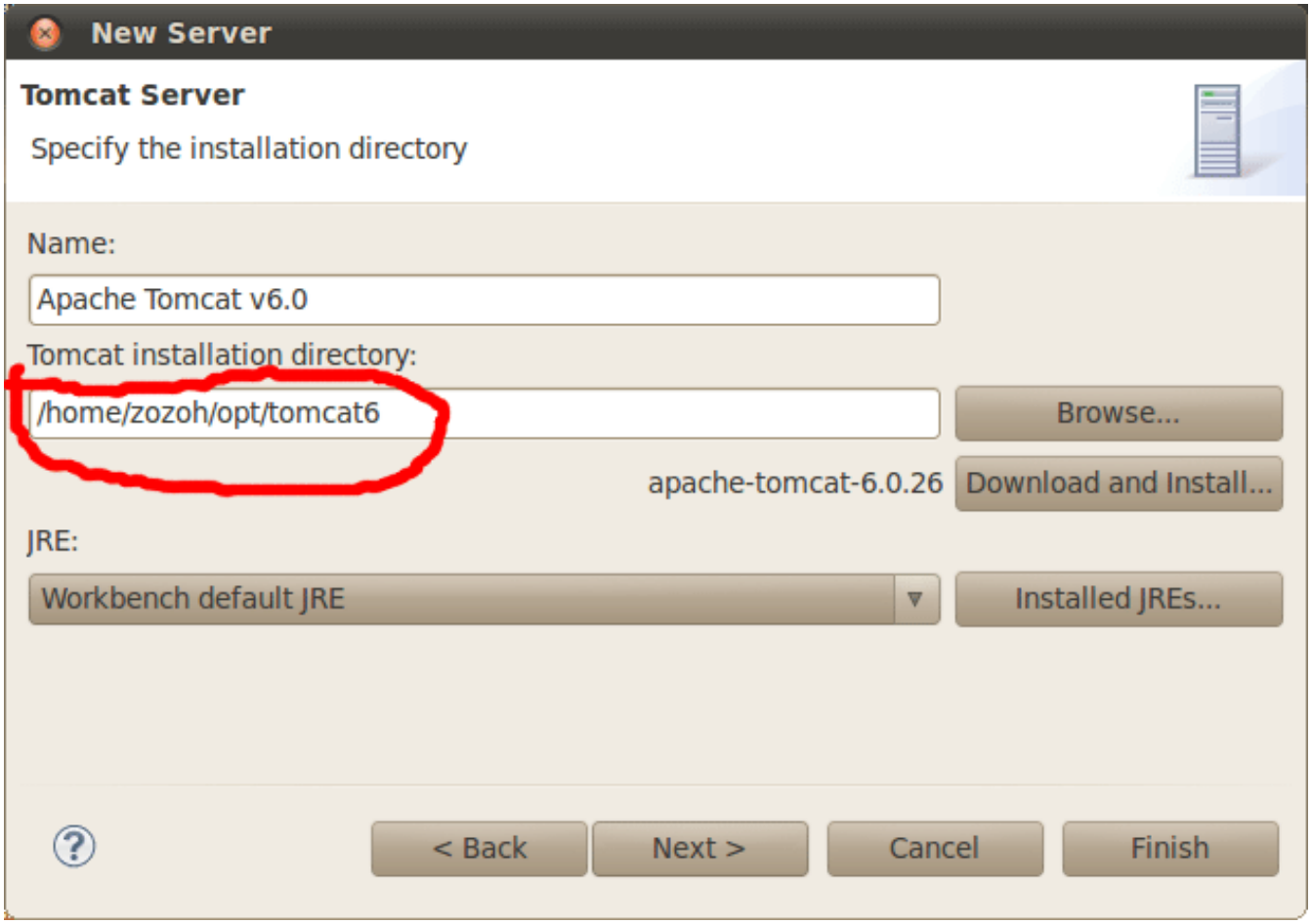
如果你已经在 Eclipse 中创建了一个 Tomcat 服务器，请跳过本步骤，直接将项目 hellonutz 添加到这个服务器中即可

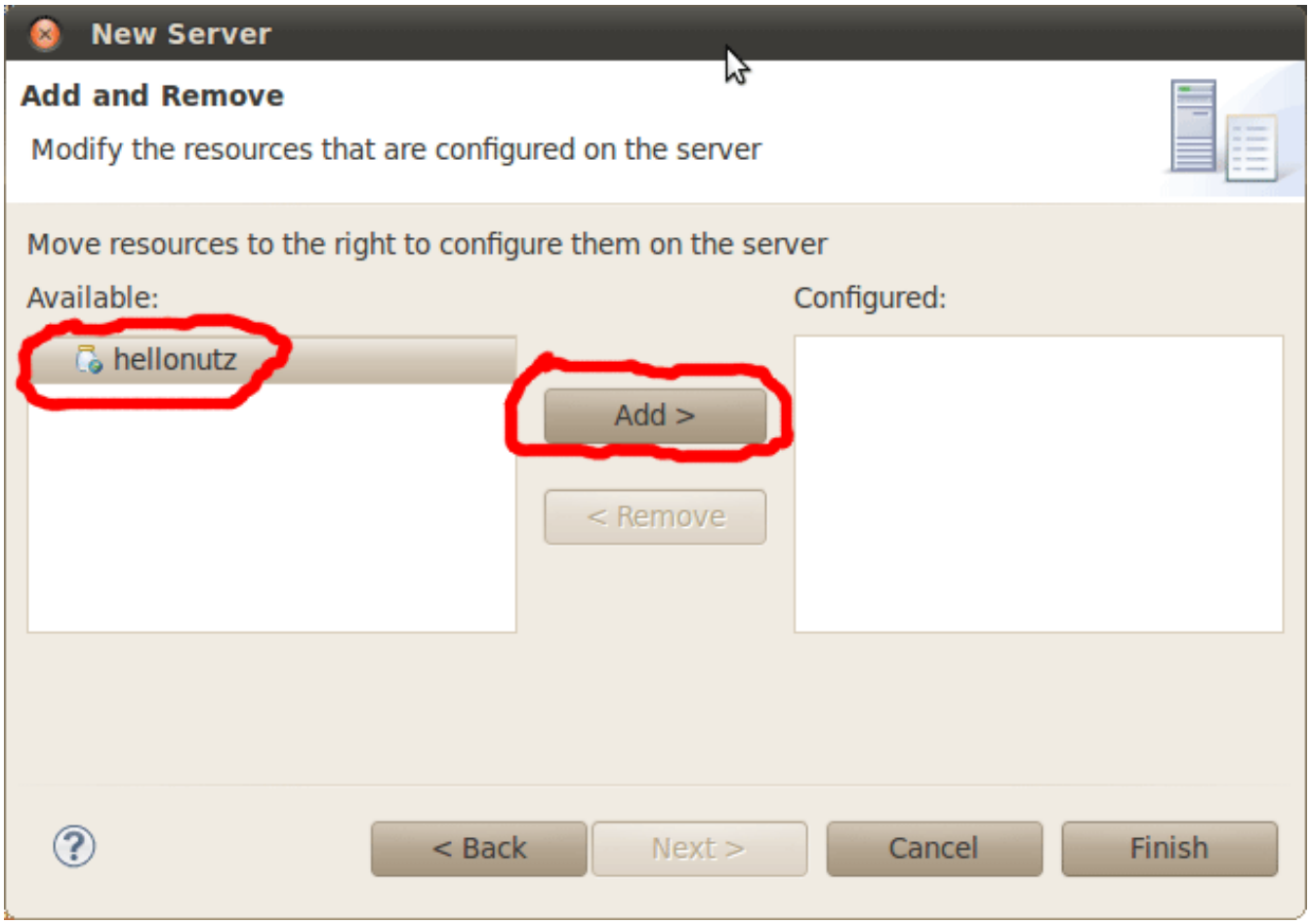
在 Eclipse 的 Servers 视图中点击右键，选择 New>Server

我们使用的 Tomcat 6.0，当然你可以根据自己的需要选用你的 Tomcat 版本，现在 7.0 已经出来了 ^_^


New Server
Define a New Server
Choose the type of server to create
Download additional server adapters
Select the server type:
type filter text
Apache
Tomcat v3.2 Server
Tomcat v4.0 Server
Tomcat v4.1 Server
Tomcat v5.0 Server
Tomcat v5.5 Server
Tomcat v6.0 Server
Publishes and runs J2EE and Java EE Web projects and server configurations to a local Tomcat server.
Server's host name:
localhost
Server name:
Tomcat v6.0 Server at localhost
< Back
Next >
Cancel
Finish

将项目加入运行服务器


New Server
Tomcat Server
Specify the installation directory
Name:
Apache Tomcat v6.0
Tomcat installation directory:
/home/zozoh/opt/tomcat6
Browse...
apache-tomcat-6.0.26
Download and Install...
JRE:
Workbench default JRE
Installed JREs...
< Back
Next >
Cancel
Finish

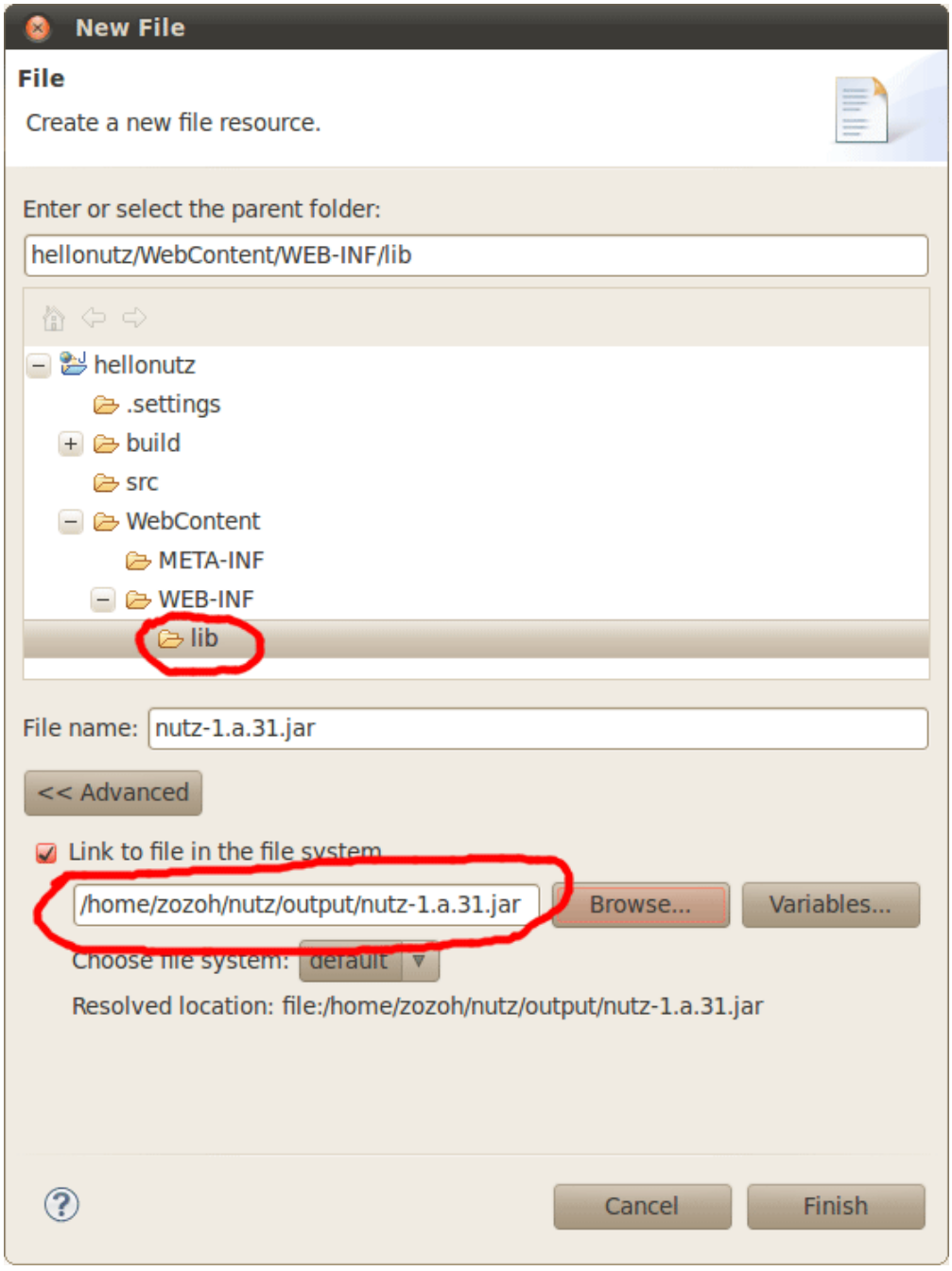

New Server
Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
Configured:
hellonutz
Add >
< Remove
< Back
Next >
Cancel
Finish

点击 'Finish' 按钮，在 Servers 视图应该出现一个 Tomcat 服务器，这个服务器下应该有一个项目 -- hellonutz

7.2.2.3. 将 Nutz.jar 加入 lib 目录中

本例子需要 Nutz-1.a.31 或者更高的版本，不过你可以自己编译一个最新版本

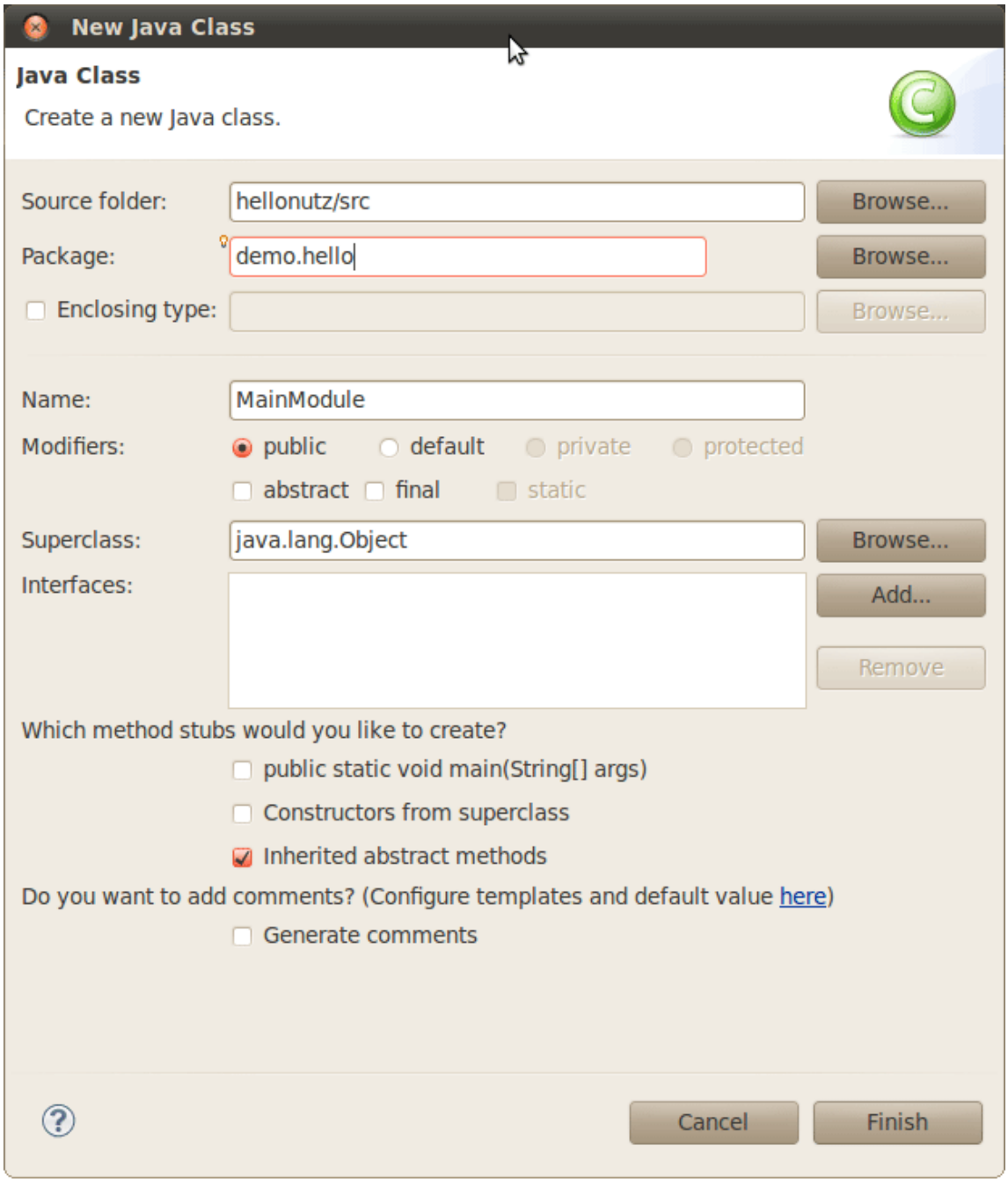
在 WebContent > WEB-INF > lib 文件夹下右键选择 New > File



点击 'Finish' Eclipse 会将 nutz 的 jar 链到项目中

7.2.2.4. 创建主模块

在 src 目录上右键选择 New > Class



直接点 'Finsih'， 后面我们会修改它的代码

7.2.2.5. 修改 web.xml

通过声明一个 Filter，将 Nutz.mvc 挂载到 Tomcat 中

```xml
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
   xmlns="http://java.sun.com/xml/ns/javaee"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
   version="2.5">
   <display-name>hellonutz</display-name>
   <filter>
      <filter-name>nutz</filter-name>
      <filter-class>org.nutz.mvc.NutFilter</filter-class>
      <init-param>
         <param-name>modules</param-name>
         <param-value>demo.hello.MainModule</param-value>
      </init-param>
   </filter>
   <filter-mapping>
      <filter-name>nutz</filter-name>
      <url-pattern>/*</url-pattern>
   </filter-mapping>
</web-app>
```

7.2.2.6. 创建入口函数

打开刚才创建的 MainModule.java，添加一个函数，整个类的源代码如下：

```
[Java]
package demo.hello;
import org.nutz.mvc.annotation.*;
public class MainModule {
   @At("/hello")
   @Ok("jsp:jsp.hello")
   public String doHello() {
      return "Hello Nutz";
   }
}
```

如果你想知道更多配置方式，请参看

* Nutz.Mvc 的配置语法

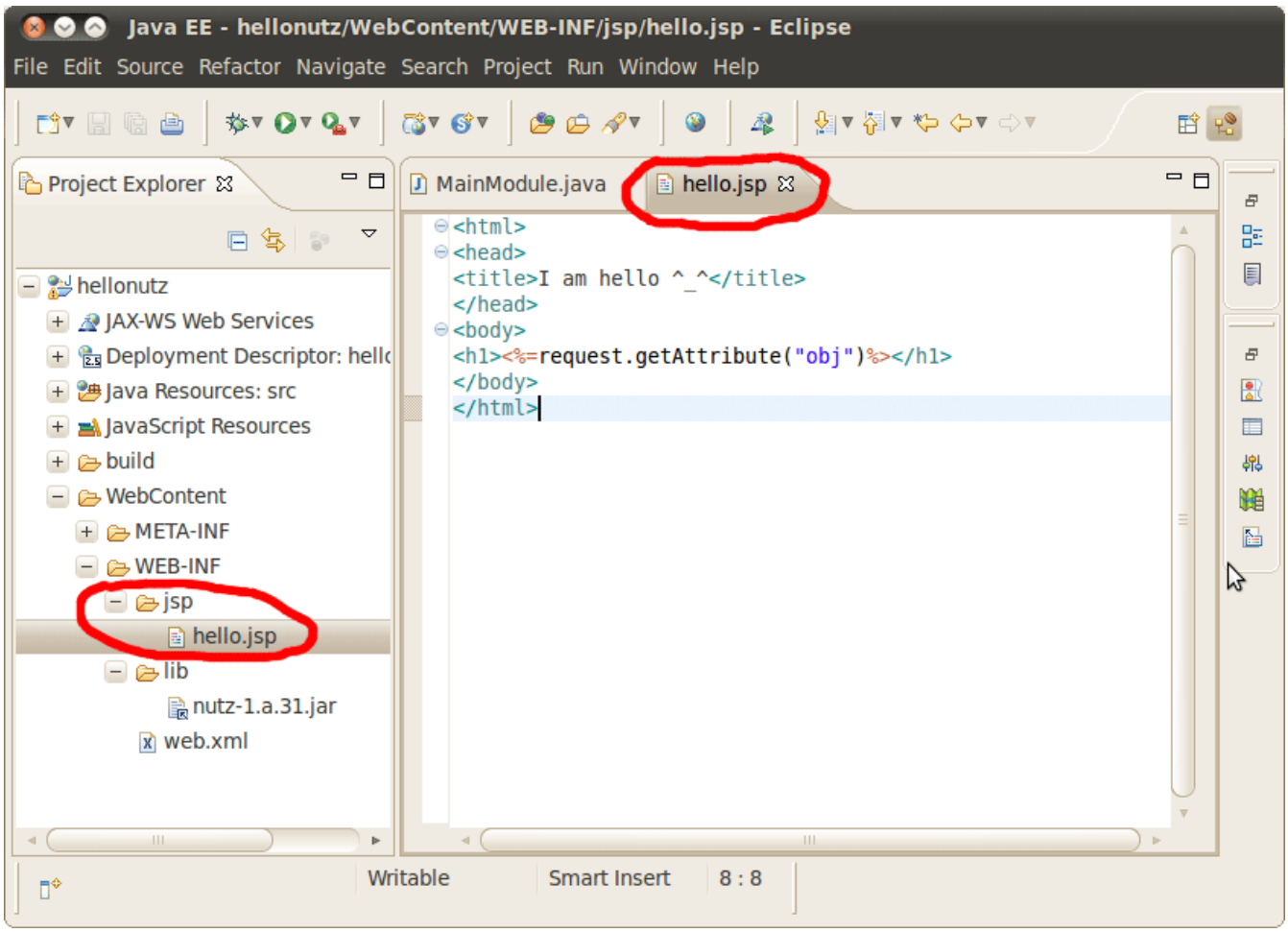
* URL 映射

7.2.2.7. 创建 jsp 页面

因为我们在入口函数里声明了一个 JSP 视图，按照视图的定义

* @Ok("jsp:jsp.hello")

我们需要在 WEB-INF 下面建立一个名为 **jsp** 的文件夹，并在里面建立一个 **hello.jsp**



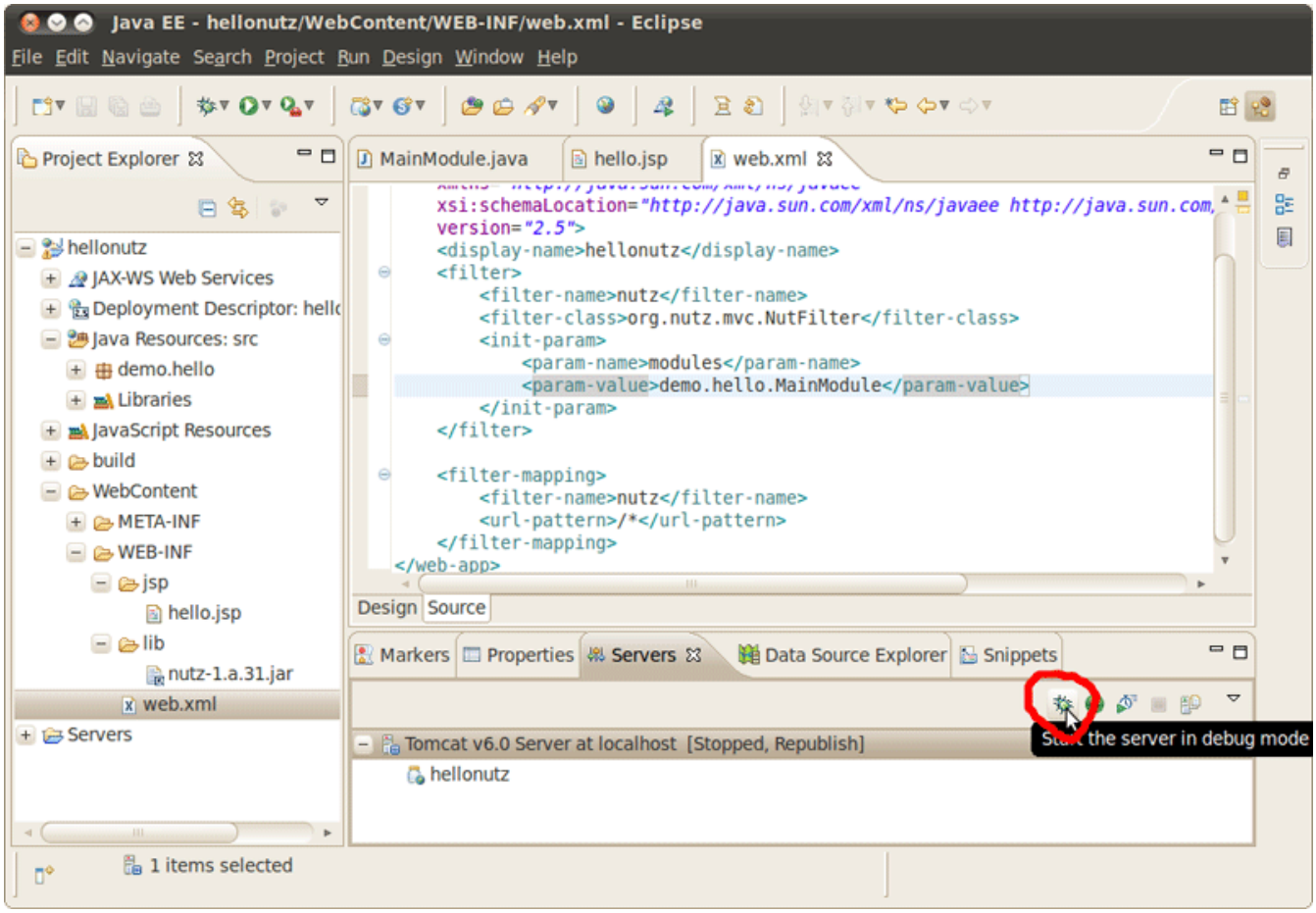
下面是 hello.jsp 的源代码：

```jsp
<html>
<head>
<title>I am hello ^_^</title>
</head>
<body>
```

```
<h1><%=request.getAttribute("obj")%></h1>
</body>
</html>
```

仅仅是输出入口函数的返回值。是的，无论入口函数返回什么，都会保存在 request 对象 "obj" 属性中

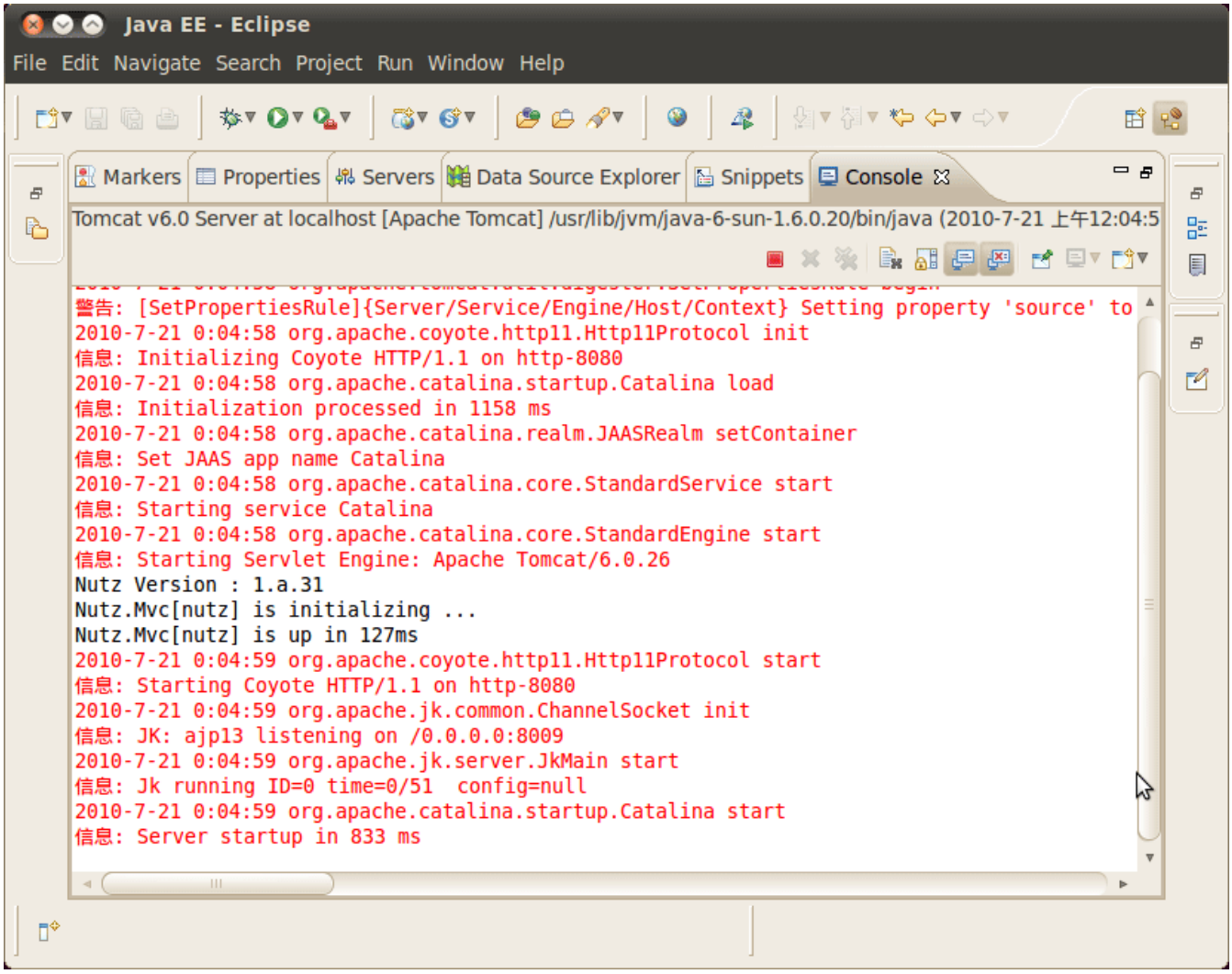
7.2.2.8. 启动服务



控制台输出

如果你想打印更详细的日志，请参看 [让_Nutz_输出日志]

7.2.2.9. 在浏览器中访问



这篇文章虽然不算短，但是实际上，你需要作的事情不多，不是吗？ ^_^

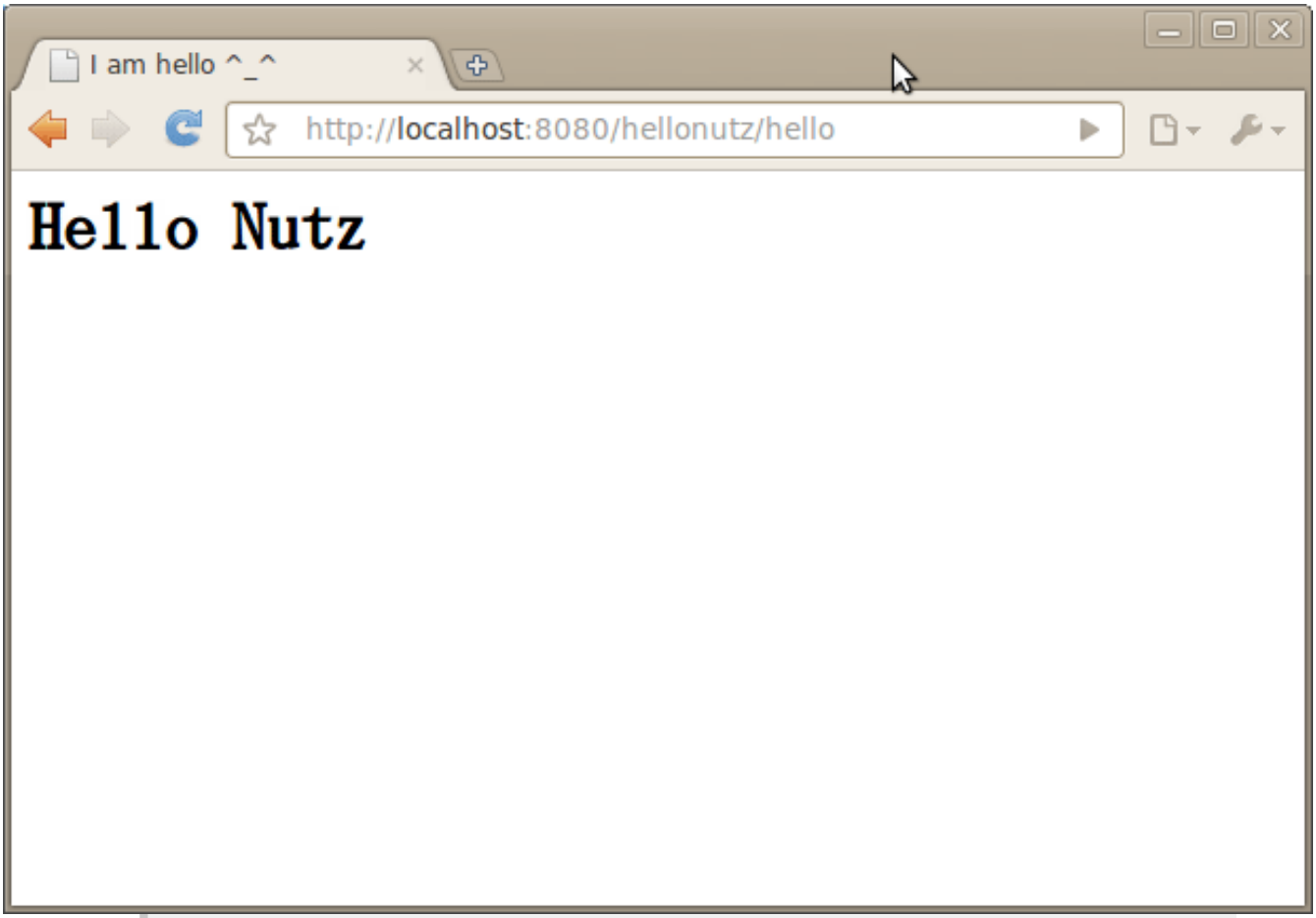
7.2.3. 如何进阶

* 访问 demo site
* 让_Nutz_输出日志
* 详细阅读 Mvc 手册

7.3. 如何配置 web.xml

7.3.1. 在 web.xml 中，一个比较典型的例子：

```xml
[xml]
<filter>
    <filter-name>nutz</filter-name>
    <filter-class>org.nutz.mvc.NutFilter</filter-class>
    <init-param>
```




```
        <param-name>modules</param-name>
        <param-value>com.mine.app.MainModule</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>nutz</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

* 如果你没有声明 '**modules**' 参数，这个过滤器将不会映射 URL，但是它并不会抛异常
* 默认的，只要被正则式 **"^(.+[.])(jsp|png|gif|jpg|js|css|jspx|jpeg)$"** 匹配上的 URL 都不会被 Nutz 过滤器处理
* 你可以为 NutFilter 设置参数 "ignore"，来重新设置这个正则表达式
* 如果你的值是 "null" （不区分大小写），所有的请求都会转发到 Nutz.mvc 进行处理
* 如果 Nutz.mvc 没有找到合适入口函数处理,将会继续 chain.doFilter

比如如果你想忽略所有的 .html 请求，你可以

```
[xml]
```

```
    <filter>
        <filter-name>nutz</filter-name>
        <filter-class>org.nutz.mvc.NutFilter</filter-class>
        <init-param>
            <param-name>modules</param-name>
            <param-value>com.mine.app.MainModule</param-value>
        </init-param>
        <init-param>
            <param-name>ignore</param-name>
            <param-
value>^(.+[.])(jsp|png|gif|jpg|js|css|jspx|jpeg|html)$</param-value>
        </init-param>
    </filter>
```

注意,如果你使用Forward视图,请这样写filter-mapping

```
[CODE]
<filter-mapping>
    <filter-name>nutz</filtername>
```

```
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

7.3.2. 比较传统的方式

```
[xml]
<filter>
    <filter-name>msgs</filter-name>
    <filter-class>org.nutz.mvc.NutFilter</filter-class>
    <init-param>
        <param-name>skip-mode</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>msgs</filter-name>
    <url-pattern>*.jsp</url-pattern>
</filter-mapping>
<servlet>
    <servlet-name>nutz</servlet-name>
        <servlet-class>org.nutz.mvc.NutServlet</servlet-class>
        <init-param>
            <param-name>modules</param-name>
            <param-value>your.package.MainModule</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>nutz</servlet-name>
        <url-pattern>*.nut</url-pattern>
    </servlet-mapping>
```

7.4. 主模块-子模块-入口函数

7.4.1. 主模块

任何一个类都可以作为主模块，只要你将其配置在 web.xml 中，这样的设计主要是为了方便一些非 JSP/Servlet 标注的 web 服务器整合 Nutz.mvc 框架。

在主模块中，你可以声明如下的注解：

* @Modules - 声明应用的所有子模块
* @IocBy - 设置应用所采用的 Ioc 容器
* @SetupBy - 应用启动以及关闭时的额外处理
* @Views - 自定义的扩展视图
* @Localization - 应用的本地化字符串设定
* 所有入口函数上支持的注解
    > 声明在主模块模块的注解，将作为所有入口函数的默认配置
    > 在子模块中的定义更为优先

7.4.1.1. @Modules - 声明应用的所有子模块

指定子模块

```
[CODE]
@Modules({ UserModule.class, PetModule.class})
public class MainModule {
    ...
```

自动搜索子模块

```
[CODE]
@Modules(scanPackage = true)
public class MainModule {
    ...
```

将自动搜索主模块所在的包（包括子包）下所有的类，如果有类包括了一个以上的入口函数将被认为是模块类

半自动搜索子模块

```
[CODE]
@Modules(value={Abc.class, Xyz.class}, scanPackage = true)
public class MainModule {
    ...
```

将自动搜索主模块类，Abc.class，Xyz.class 所在的包（包括子包）下所有的类，如果有类包括了一个以上的入口函数将被认为是模块类

7.4.1.2. @IocBy - 设置应用所采用的 Ioc 容器

声明了 Ioc 容器的获取方式，详情请参看 同 Ioc 容器一起工作 一节

7.4.1.3. @SetupBy - 应用启动以及关闭时的额外处理

在整个应用启动或者关闭时，你想做一些额外的处理工作，你可以实现一个org.nutz.mvc.Setup 接口，并将其配置在主模块上

```
[CODE]
@SetupBy(MyAppSetup.class)
public class MainModule {
    ...
```

7.4.1.4. @Views - 自定义的扩展视图

Nutz.Mvc 允许你定制自己的视图，如何使用这个注解，请参看视图>定制自己的视图 一节

7.4.1.5. @Localization - 应用的本地化字符串设定

Nutz.Mvc 允许你定制自己的本地化字符串存取方式，同时它也提供了默认的本地化字符串存取方式，请参看 本地化字符串 一节

7.4.2. 子模块

任何类都可以作为子模块，只要通过@Modules 注解声明到主模块上即可

在子模块，你可以声明

* @InjectName 使用 Ioc 容器管理本模块
* 所有入口函数上支持的注解
    > 声明在子模块的注解，将作为所有入口函数的默认配置

7.4.3. 入口函数

子模块中任何函数，只要是 public 的，且不是 static 的，都可以作为入口函数标记入口函数的方法是在其上标注注解 @At

在入口函数上，你可以声明如下注解：

* @At - 入口函数对应的 URL

* @Ok - 成功视图
* @Fail - 失败视图
* @AdaptBy - HTTP 参数适配方式
* @Filter - 过滤器
* @Encoding - 输入输出编码

7.4.3.1. @At - 入口函数对应的 URL

只有标记了这个注解的函数才被认为是入口函数，例如

```
[CODE]
@At("/my/abc")
public void someFunc(){
    ...
```

你也可以为该函数声明多个 URL

```
[CODE]
@At({"/my/abc", "/my/xyz"})
public void someFunc(){
    ...
```

* 你也可以为其声明路径参数，详细情况请参看 适配器>路径参数 一节
* 如果，你想详细了解 URL 映射的细节，请参看 URL 映射 一节。

7.4.3.2. @Ok - 成功视图

声明了入口函数的成功视图，即如果入口函数正常执行，将会通过这个视图将函数返回值渲染到 HTTP 响应中。当然，如果你的函数直接返回的就是一个视图对象，那么就不会使用成功视图（而是你返回的视图对象）来渲染HTTP 响应详细的介绍，请参看 视图 一节

7.4.3.3. @Fail - 失败视图

声明了入口函数的失败视图，即如果入口函数抛出异常，将会通过这个视图将异常渲染到 HTTP 响应中详细的介绍，请参看 视图 一节

7.4.3.4. @AdaptBy - HTTP 参数适配方式

将 HTTP 请求参数转变成你当前入口函数的参数的过程叫做 **适配**，这个注解就是声明这个适配器。如果你没有声明这个注解，默认的会采用 PairAdaptor 来适配 HTTP 请求参数。详细的介绍，请参看适配器 一节

7.4.3.5. @Filter - 过滤器

详细介绍请参看 过滤器 一节

7.4.3.6. @Encoding - 输入输出编码

定义 HTTP 请求的输入输出编码，这个注解通常是会定义在 主模块 上面，从而保证整个应用有统一的输入输出设定

如果你不定义，默认的，Nutz.Mvc 会采用 **UTF-8** 作为输入输出的编码

```
[CODE]
@Encoding(input="UTF-8",output="UTF-8")
```

7.5. URL 映射

7.5.1. 为什么需要详细描述 URL 映射

Nutz.Mvc 的核心任务就是将 HTTP 请求的 URL 映射到某一个入口函数，如果你看完了 Nutz.Mvc 概述你大概应该知道，映射的配置信息是通过注解 @At 来设置的，@At 注解也非常简单，配置起来应该没有什么障碍。

但是，依然在某些时候，你会在你的应用出现 404 错误，为了能让你在编写项目是，心里非常有底，这里将详细的解释一下JSP/Servlet 以及 Nutz.Mvc 映射部分的工作原理，在你遇到讨厌的 404 时，只要通读本文，相信就能找到问题的症结。

7.5.2. 什么是 URL

任何 URL 都由如下部分组成

http:// www.myapp.com /app /module /action .suffix

* http:// - 协议，也可以是 https://
* www.myapp.com - 域名或者 IP 地址，由 DNS 服务器负责转发
* /app - Context 的 path， 这个匹配到 server.xml 中每个 <context> 的 path 属性，由 HTTP 服务负责转发
* /module /action.suffix - 从这里以后的匹配将交给相应的 Context 的 **web.xml**，由HTTP 服务器根据 **web.xml** 来转发

因此，我们主要研究的就是 /module /action.suffix 的部分是如何转发的。

7.5.3. web.xml 中的映射 - url-pattern

通常，在 web.xml 中，我们可以声明各种 HttpServlet 子类， 为了能让某一个子类接受 URL，就需要配置映射，众所周知你可以通过 <servlet-mapping>，为你的 Servelet 增加一组至多组的配置：

```
[web.xml]
<servlet-mapping>
   <servlet-name>MyServletName</servlet-name>
   <url-pattern>/*</url-pattern>
</servlet-mapping>
```

同样，根据 Servlet 的规范，你的 <url-pattern> 可以有如下几种形式的值:

7.5.3.1. web.xml 中的全匹配 - /*

转发到本 Context 的任何请求都会调用这个 Servlet，比如：

* /abc
* /abc /dosome
* /abc /dosome.nut
* /index.jsp
* /img /logo.png

如果请求为：

```
[CODE]
http://localhost:8080/testweb/abc/getlist.nut
```

调用 request 对象不同方法将会返回的值：

req.getRequestURL()	"http://localhost:8080/testweb/abc/getlist.nut"
req.getRequestURI()	"/testweb/abc/getlist.nut"
req.getPathInfo()	"/abc/getlist.nut"
req.getServletPath()	""

7.5.3.2. web.xml 中的目录匹配 - /abc/*

转发到本 Context 的任何请求只要以 **/abc/** 开头，都会调用这个 Servlet，比如：

* /abc /dosome
* /abc /dosome.nut
* /abc /index.jsp
* /abc /logo.png

如果请求为：

> *[CODE]*
> http://localhost:8080/testweb/abc/getlist.nut

调用 request 对象不同方法将会返回的值：

req.getRequestURL()	"http://localhost:8080/testweb/abc/getlist.nut"
req.getRequestURI()	"/testweb/abc/getlist.nut"
req.getPathInfo()	"/getlist.nut"
req.getServletPath()	"/abc"

因此我们可以认为， req.getPathInfo() 的值是：

> *[CODE]*
> http://localhost:8080/testweb/abc/getlist.nut
> ----------------------------------^ 匹配 /abc/*，从这个位置之后的字符串

7.5.3.3. web.xml 中的后缀匹配 - *.nut

转发到本 Context 的任何请求只要以 **.nut** 结尾，都会调用这个 Servlet，比如：

* /abc /dosome.nut
* /abc.nut

如果请求为：

```
[CODE]
http://localhost:8080/testweb/abc/getlist.nut
```

调用 request 对象不同方法将会返回的值：

req.getRequestURL()	"http://localhost:8080/testweb/abc/getlist.nut"
req.getRequestURI()	"/testweb/abc/getlist.nut"
req.getPathInfo()	null
req.getServletPath()	"/abc/getlist.nut"

7.5.3.4. web.xml 中的精确匹配 - /abc/getlist.nut

转发到本 Context 的任何请求必须为 **/abc/getlist.nut**，才会调用这个 Servlet

如果请求为：

```
[CODE]
http://localhost:8080/testweb/abc/getlist.nut
```

调用 request 对象不同方法将会返回的值：

req.getRequestURL()	"http://localhost:8080/testweb/abc/getlist.nut"
req.getRequestURI()	"/testweb/abc/getlist.nut"
req.getPathInfo()	null
req.getServletPath()	"/abc/getlist.nut"

7.5.4. 在 Nutz.Mvc 中的映射

Nutz.Mvc 通过 org.nutz.mvc.NutFilter 类将自己同一个 JSP/SERVLET 容器挂接关于挂接的方法，详细请看 如何配置 web.xml

在设计这个框架之初，我们有一个基本的设计要求：

如果用户修改了 web.xml 或者 server.xml，不需要用户重新修改 Nutz.Mvc 相关的配置

对于任意一个请求:

* http:// www.myapp.com /app /module /action .suffix

Nutz.Mvc 匹配的时候，只会关心这个部分：

* /module /action

发现了吗？是的，它对 **.suffix** 不敏感，匹配之前，它会把 **.suffix** 切去。之后，它会通过注解 '@At' 寻找合适的入口函数，

7.5.4.1. 如何通过 @At 寻找入口函数

在Nutz.Mvc 概述里，我提到，@At 注解可以被声明在三个地方：

```
[CODE]
主模块 - @At("/a")
  子模块 - @At("/b")
    入口函数 - @At("/c")
```

最终确定了 URL **/a/b/c** 要匹配的入口函数。

所以要想匹配 **/a/b/c** 下面几种形式都是可以的

```
[CODE]
@At("/a")
public class MainModule{
  ...
@At("/b")
public class SubModule{
  ...
  @At("/c")
```

```
  public String helle(){
    ...
```

或者

```
[CODE]
public class MainModule{
  ...
@At("/a")
public class SubModule{
  ...
  @At("/b/c")
  public String helle(){
    ...
```

或者

```
[CODE]
public class MainModule{
  ...
public class SubModule{
  ...
  @At("/a/b/c")
  public String helle(){
    ...
```

当然，一般的说，很少有人在主模块上声明 @At

注: 使用不带参数的@At()注解, 默认会使用方法名/类名的小写做为入口地址!

```
[CODE]
// 比如
@At
public String getPet() {
```

```
    ...
// 与下面的代码是等效的
@At("/getpet")
public String getPet() {
    ...
```

通过上面的内容我们可以知道，只要有一个 URL，我们就知道如何设置注解 '@At'，但是你确定我们要匹配的 URL 就是

* /module /action

吗？ 不，这同时也得参考 web.xml 的匹配方式：

7.5.4.2. Nutz 中的全匹配 - /*

如果请求为：

```
[CODE]
http://localhost:8080/testweb/abc/getlist.nut
```

对于 Nutz.Mvc 我们需要匹配的部分为：

* /abc/getlist

7.5.4.3. Nutz 中的目录匹配 - /abc/*

如果请求为：

```
[CODE]
http://localhost:8080/testweb/abc/getlist.nut
```

对于 Nutz.Mvc 我们需要匹配的部分为：

* /getlist

这里需要说明的是，可能人们会怀疑，为什么目录匹配的时候，只匹配 /getlist 而不匹配 /abc/getlist 呢？因为，你在你的 web.xml 声明了这样的 url-pattern:

```
[CODE]
...
<url-pattern>/abc/*</url-pattern>
...
```

显然，你希望在 web.xml 来决定你的 URL 前面那部分，所以后面一部分就交给 Nutz.Mvc 来匹配吧。否则，你修改了web.xml 的时候，你还需要修改 Nutz.Mvc 的配置，这与显然我们设计的初衷不符，Nutz.Mvc 设计的基本要求就是：

如果用户修改了 web.xml 或者 server.xml，不需要用户重新修改 Nutz.Mvc 相关的配置

7.5.4.4. Nutz 中的后缀匹配 - *.nut

如果请求为：

```
[CODE]
http://localhost:8080/testweb/abc/getlist.nut
```

对于 Nutz.Mvc 我们需要匹配的部分为：

* /abc/getlist

7.5.4.5. Nutz 中的精确匹配 - /abc/getlist.nut

如果请求为：

```
[CODE]
http://localhost:8080/testweb/abc/getlist.nut
```

对于 Nutz.Mvc 我们需要匹配的部分为：

* /abc/getlist

这种映射方式基本是不会发生的，因为你需要让 Nutz.Mvc 控制一批 URL 而不是单个 URL。所以，你如果选择了这种模式我就没话讲了，在 Nutz.Mvc 中你就全部匹配吧，惩罚你，哼！

7.6. 动作链

7.6.1. 动作链机制概述

在新的版本中(1.b.36)之后的版本，Nutz.Mvc 统一采用动作链机制来处理每一个 HTTP 请求。我们认为:

* 对于一个 HTTP 请求的处理实际上是由一系列子处理构成的
    > 比如根据映射找到入口函数
    > 比如为入口函数生成调用参数
    > 比如调用入口函数
* 我们希望这一系列子处理可以根据 URL 的不同而不同
* 我们也希望子处理是可配置的

因此它能让你的HTTP映射处理具备更大的灵活性。

下面是一张稍微有点复杂的图，根据这张图，我们来详细的解释一下这个机制，在了解了它之后，我相信你掌握更多的复用你代码的手段，从而更合理处理你的 URL 映射关系:

这张图稍微有点复杂，你可能看起来稍微要皱一下眉头。本文的后面几节会详细为你解释，但是你首先要记住:

* 这种图的流程都是 Nutz.Mvc 在**加载时**进行的操作
* 在**运行时**，Nutz.Mvc 将根据 URL 获得 ActionChain 接口，直接执行整条动作链

7.6.2. (A)获取动作链工厂

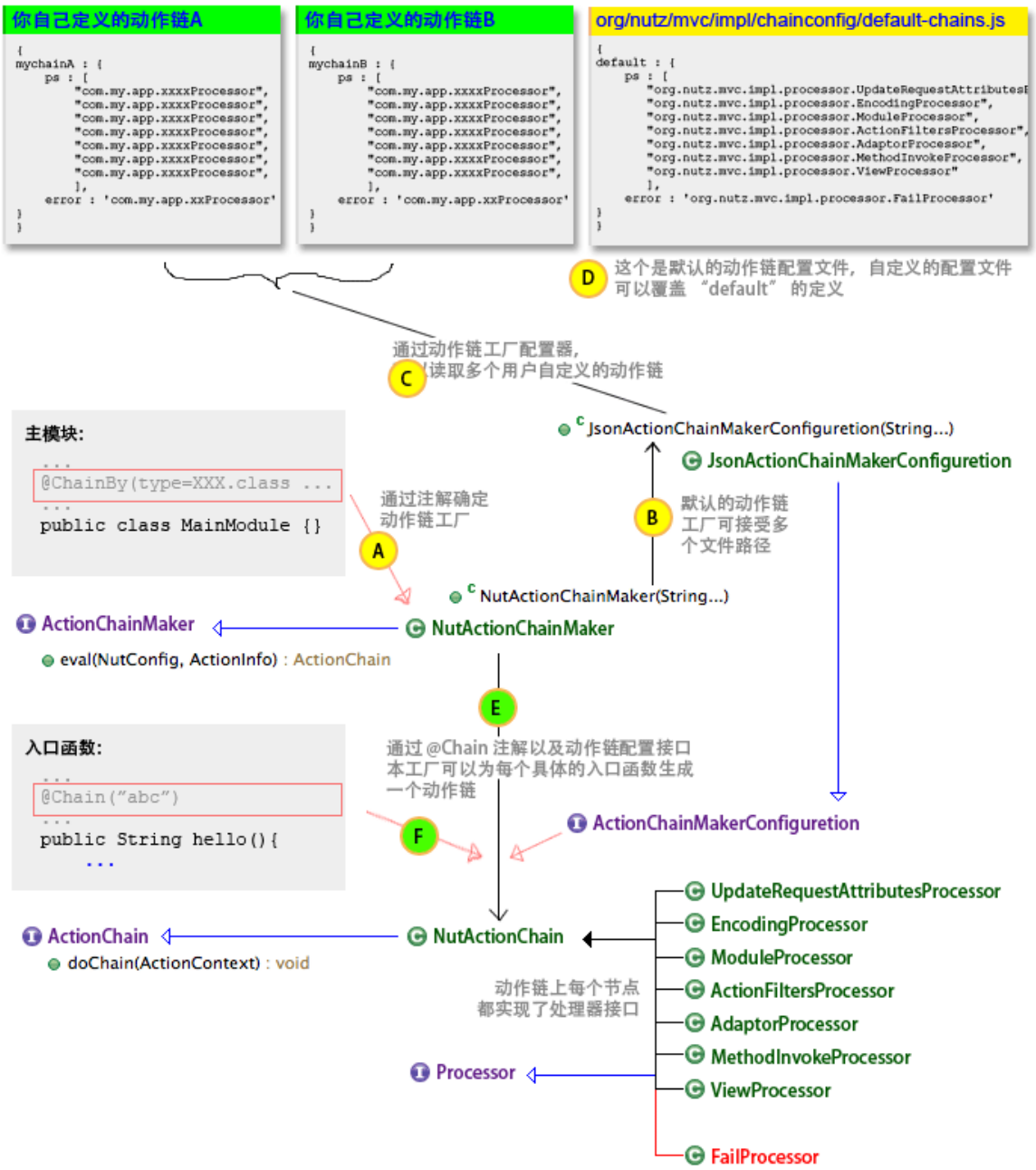
整个 Nutz.Mvc 的应用，必须有且只能有一个动作链工厂。在**主模块**上，你可以声明你自己的动作链工厂（通过 @ChainBy 注解）。当然，如果你没有声明这个注解，Nutz.Mvc 会采用默认的动作链工厂实现类(**org.nutz.mvc.impl.NutActionChainMaker**)

如果你需要定制动作链工厂，你可以通过类似下面的形式，声明自己特殊配置的动作链工厂：

```
[CODE]
@ChainBy(args={"配置文件A路径", "配置文件B路径"})
```

你也可以采用自己的动作链工厂实现类

```
[CODE]
```




```
@ChainBy(type=MyChainMaker.class, args={...})
```

如果你的动作链工厂需要更复杂的配置，你可以交给 Ioc 容器来管理

```
[CODE]
@ChainBy(type=MyChainMaker.class, args={"ioc:myChianMaker"})
```

这样，你就可以在 Ioc 容器里，声明一个 "myChainMaker" 对象，来对其做任何你想要的配置。当然，首先你需要在主函数里声明了 Ioc 容器（请参看同 Ioc 容器一起工作一文）

7.6.3. (B)获取动作链工厂配置信息

如果你采用的是 Nutz.Mvc 的默认动作链工厂，它允许你在构造函数中声明动作链的配置文件。你可以增加任意多的动作链配置文件，在一个文件中，你可以声明任意多的动作链。

它有一个默认的配置文件，声明了一个名字为 "default" 的动作链。你的入口函数如果没有声明 @Chain 注解的话，就是使用这个动作链。你可以通过自己的配置文件覆盖它。

它的构造函数定义为：

```
[CODE]
public NutActionChainMaker(String...args) {
    ...
```

接受变参数数组，每个参数，都是你配置文件的路径，可以是类路径，也可以是绝对路径，当然你也可以写成:

```
[CODE]
@ChainBy(args={"${app.root}/WEB-INF/chain/mychain.js"})
```

其中 ${app.root} 会被 Nutz.Mvc 替换成你的应用在服务器上的根目录。

7.6.4. (C)解析配置文件

每个配置文件你可以配置多个动作链，每个动作链需要一个名字，以便在入口函数通过 @Chain 注解来引用，下面让我们来看看默认动作链配置文件的内容：

```
[CODE]
{
   default : {
      ps : [
"org.nutz.mvc.impl.processor.UpdateRequestAttributesProcessor",
          "org.nutz.mvc.impl.processor.EncodingProcessor",
          "org.nutz.mvc.impl.processor.ModuleProcessor",
          "org.nutz.mvc.impl.processor.ActionFiltersProcessor",
          "org.nutz.mvc.impl.processor.AdaptorProcessor",
```

```
            "org.nutz.mvc.impl.processor.MethodInvokeProcessor",
            "org.nutz.mvc.impl.processor.ViewProcessor"
        ],
        error : 'org.nutz.mvc.impl.processor.FailProcessor'
    }
}
```

动作链的配置文件采用了 JSON 格式。在上面的文件内只有一个动作链，名字为 **"default"**。通过例子你可以很容易看出，一个动作链需要两方面的信息：

* 正常的流程是怎样的？
    > "ps" 属性是一个数组，每个值就是一个处理器接口的实现类
    > 每次该动作链执行时，会按顺序调用这些处理器
* 遇到错时怎么办？
    > 通过给出的错误处理器的实现类来处理错误

7.6.5. (D)至少还有默认配置文件

默认的配置文件优先级最低，它随着 nutz.jar 一起发布，所以它没打算让你直接修改。你可以通过自己的配置文件覆盖其唯一的动作链 **"default"**

7.6.6. (E)为入口函数创建动作链

就像前面提到的，在每个入口函数里，你可以通过注解 @Chain 来指定你的这个函数将采用哪个动作链。如果你没有指定，Nutz.Mvc 认为你是希望用 @Chain("default")} 来处理这个入口函数。

需要说明的是，考虑到效率，在 Nutz.Mvc 加载时，它就会为每个入口函数创建 URL 映射关系。即把一个 URL 映射到一个动作链实例上。所以动作链的实例，是在加载时就被创建了。所以如果你自己实现了动作链工厂，**请保证工厂生成的每个动作链是线程安全的**。

7.6.7. (F)每个入口函数的动作链都可以不同

如果你读完了上述小节，本节光看标题就足够了。

但是，我唠叨成性，这里再举个小例子：

```
[CODE]
@At("/a")
@Chain("abc")
public void funcA(){}
@At("/b")
```

```
@Chain("abc")
public void funcB(){}
@At("/c")
public void funcC(){}
@At("/d")
public void funcD(){}
```

在上面的例子中，四个入口函数，其中：

* 每个入口函数都各自有一份动作链实例
* 具体的实例是由动作链工厂决定的
* 每个动作链实例的生命周期范围是 App(ServletContext) 级别

原因不解释。

7.7. 适配器

7.7.1. 什么是适配器？

将 HTTP 参数转换成一个函数参数的过程是一个典型适配过程，执行这个过程的对象被称为适配器了。Nutz.Mvc 提供了 **org.nutz.mvc.HttpAdaptor** 接口，隔离了这种行为。

在每一个入口函数上，你都可以通过注解 @AdaptBy 来声明如何适配 HTTP 参数。当然，你没必要在每一个入口函数上都声明，在子模块类上声明，或者在整个应用的主模块上声明均可。

7.7.2. 如何使用适配器？

默认的，如果你什么也不写，Nutz.Mvc 会采用 **org.nutz.mvc.adaptor.PairAdaptor**（*也就是名值对的方式*）来适配你的 HTTP 参数。

7.7.2.1. 通过构造函数获得适配器

你可以通过 @AdaptBy 注解来改变任何一个入口函数的适配方式。比如

```
[CODE]
@AdaptBy(type=JsonAdaptor.class)
```

某些时候，你需要对一个适配器做一些复杂的设置，@AdaptBy 注解还支持一个属性 args，你可以通过这个属性为你的适配器设置构造函数参数

7.7.2.2. 通过 Ioc 容器获得适配器

更复杂的情况是，如果你希望你的适配器是交由 Ioc 容器管理的，你可以：

```
[CODE]
@AdaptBy(type=JsonAdaptor.class, args={"ioc:objName"})
```

即，如果你的参数数组长度为一，并且，由 "ioc:" 开始，那么这个适配器会交付 Ioc 容器管理，你可以在容器的配置文件中详细规定这个适配器的各个属性。当然，你需要在整个应用启用 Ioc 容器，详情，请参看 同 Ioc 容器一起工作

7.7.3. 内置的适配器

Nutz.Mvc 为你内置了 4 个最常用的适配器，可以让支持用如下四种方式适配 HTTP 参数：

7.7.3.1. 名值对 (默认) - PairAdaptor

7.7.3.1.1. 一般方式

```
[CODE]
@AdaptBy(type=PairAdaptor.class)
```

这种方式，是传统的 HTTP 参数方式。关键的问题是如何将 HTTP 的参数表同入口函数的参数对应起来。为此，它支持一个新的注解 **@Param**，你可以:

```
[CODE]
public String someFunc(@Param("pid") int petId,
                       @Param("pnm") String petName){
    ...
```

通过这个注解，为每个参数声明 HTTP 参数名。实验性,不推荐在生产环境中使用. 仅当class是通过ecj/eclipse编译的,才完整支持.请参看: 在Ant/Maven中使用ecj编译器(Eclipse内置的Java编译器)特例:你连@Param都不写(1.b.43+)

```
[CODE]
public String someFunc(int pid,
                       String petName){
```

```
    ...
```

那么,这等同于

```
[CODE]
public String someFunc(@Param("petId") int petId,
                       @Param("petName") String petName){
    ...
```

很神奇,很方便,不是吗? 前提是编译的时候,加上-debug,保留局部变量名,且大部分IDE都是默认打开这个属性的.

7.7.3.1.2. 表单方式 - Form Bean

有些时候，你需要入口函数接受一个对象，比如一个表单对象

```
[CODE]
public String someFunc(@Param("..") Pet pet){
    ...
```

值 ".." 有特殊含义，表示当前的这个对象，需要对应整个的 HTTP 参数表。 所以， Nutz.Mvc 会将 HTTP 参数表中的参数一个个的按照名字设置到 Pet 对象的字段里。 但是如果 Pet 对象的字段名同 HTTP 参数不符怎么办？ 你可以在Pet 字段上声明 @Param。

7.7.3.1.3. 前缀表单方式

进行比较复杂的 HTTP 交互是，大家都比较偏爱名值对的方式提交数据，可能是因为数据组织比较方便 -- 通过<form> 即可。但是如果在一个表单里混合上两个甚至多个表单项，那么 HTTP 的参数就会有点复杂，虽然这种情况下我更推荐采用Json 输入流，但是并不是所有人都那么喜欢它，对吗？

比如有一个表单，它希望提交两个对象的数据， User 以及 Department，这HTTP 请求的参数格式可能是这样的：

```
[CODE]
user.id = 23
```

```
user.name = abc
user.age = 56
dep.id = 15
dep.name = QA
dep.users[1].id = 23
dep.users[1].name = abc
dep.users[1].age = 56
dep.users[10001].id = 22
dep.users[10001].name = abcd
dep.users[10001].age = 26
dep.users:50001.id = 22
dep.users:50001.name = abcd
dep.users:50001.age = 26
dep.children(abc).id = 13
dep.children(abc).name = ABC
dep.children(jk).id = 25
dep.children(jk).name = JK
dep.children.nutz.id = 1
dep.children.nutz.name = NUTZ
```

怎样在入口函数内声明这样的表单项呢？我们可以采用前缀方式：

```
[Java]
public String someFunc( @Param("::user.") User user,
                        @Param("::dep.") Department dept){
    ...
}
```

关键就是这个 **@Param("::user.")** 符号 '**::**' 表示这个参数是一个表单对象，并且它有统一的前缀'**user.**' 表示前缀，Nutz.Mvc 会查看一下 User, Department 类所有的字段：

```
[CODE]
public class User {
    private int id;
    private String name;
    private int age;
}
```

```
public class Department {
    private List<User> users;
    private Map<String, User> children;
}
```

7.7.3.1.3.1. 那么， id 会对应到 HTTP 参数中的 'user.id'， 其他的字段同理.眼尖的你肯定发现了有点异样的地方, 对了, 那就是我们 nutz 对集合的支持. 在此, 你不仅可以对一般的属性进行注入, 还能对list, set, map集合以及对象数组进行注入. 在此我们提供了两种书写方式:1. 对象.list索引 = 值

对象.list索引.属性 = 值

对象.map(key) = 值对象.map(key).属性 = 值

7.7.3.1.3.2. 2. 对象.list:索引 = 值

对象.list:索引.属性 = 值

对象.map.key = 值对象.map.key.属性 = 值

两种方式是完全等价的(小声透露一下, 其实代码里面就是把第一种方式转换成第二种方式实现的哦...). 并且都可以包含多层集合. 同时需要注意的是, 在进行 list 注入的时候需要注意, 出于内存方面的考虑, 所提供的 "索引" 只做为一个组装对象的参考字段(必需,不然不能组装对象), 不做为真实list的索引使用. 因此, list 的索引可以是任意大小的数字, 以及字符, 出字符串组成.

从现在开始, nutz 参数的类型不再只支持单纯的 Object 对象注入了, 同时也提供了 List, Map, Set 以及对象数组. 亲, 还等什么? 赶快来试试吧, 不需要9998, 也不需要998, 只要98, 亲, 还等什么, 赶快拿起你手中的电话...额...请在参数前加上@Param(::前缀).

更更更强大的功能, nutz开始支持泛型了, 直接来例子, 懒得解释:

```
[CODE]
    class Abc<T>{
        T obj;
    }
    class jk{
        String name;
    }
    public void test(@Param("::abc.")Abc<jk> abc){}
```

如果要写test的参数, 你可以直接写 abc.obj.name = "nutz" , 我们的nutz就会非常智能的生成jk对象.

7.7.3.1.4. 混合方式

值得一说的是，按照这个约定，实际上，一个入口函数，是可以支持多个 POJO 的，也可以写成这样

```
[CODE]
public String someFunc(@Param("pid") int petId,
                       @Param("..") Pet pet,
                       @Param("..") Food food){
    ...
```

7.7.3.1.5. JSON 的支持

你的 HTTP 参数也可以是一个 JSON 字符串

```
[CODE]
public String someFunc(@Param("pid") int petId,
                       @Param("pet") Pet pet,
                       @Param("foods") Food[] food){
    ...
```

HTTP 参数的值都是字符串，比如上例的第二个参数，Nutz.Mvc 会看看 HTTP 参数表中的 "pet" 的值，如果它用 "" 和 ""包裹，则会试图将其按照 JSON 的方式解析成 Pet 对象。当然，如果你传入的参数格式有问题，会解析失败，抛出异常。

第三个参数，是一个数组，Nutz.Mvc 会看看 HTTP 参数表中的 "foods" 的值，如果用 "[" 和 "]" 包裹，则会视试图将其转换成一个数组。 如果你 JSON 字符串的格式有问题，它也会抛出异常。

参数类型如果是列表（java.util.List），同数组的处理方式相同。但是它不知道列表元素的类型，所以转换出的元素只可能是

* 布尔
* 数字
* 字符串
* 列表
* Map

7.7.3.2. JSON 输入流 - JsonAdaptor

如果你要通过 HTTP 传给服务器一个比较复杂的对象，通过名值对的方式可能有点不方便。因为它很难同时传两个对象。并且一个对象如果还嵌入了另外一个对象，也很难传入，你必须要自己定义一些奇奇怪怪的格式，在 JS 里组织字符串，在服务器端，手工解析这些字符串。

针对这个问题， JSON 流是一个比 XML 流更好的解决方案，它足够用，并且它更短小。

如果你的 HTTP 输入流就是一个 JSON 串，你可以这样：

```
[CODE]
@AdaptBy(type=JsonAdaptor.class)
public String someFunc( Pet pet ){
    ...
```

如果你的 JSON 流是一个数组

```
[CODE]
@AdaptBy(type=JsonAdaptor.class)
public String someFunc( Pet[] pet ){
    ...
```

如果你的 JSON 流类似：

```
[CODE]
{
    fox : {
        name : "Fox",
        arg : 30
    },
    fox_food : {
        type : "Fish" ,
        price : 1.3
    }
}
```

你希望有两个 POJO （ Pet 和 Food ） 分别表示这两个对象，你可以：

```
[CODE]
@AdaptBy(type=JsonAdaptor.class)
public String someFunc(@Param("fox") Pet pet,
                       @Param("fox_food") Food food){
    ...
```

实际上，Nutz.Mvc 会将 HTTP 输入流解析成一个 Map，然后从 Map 里取出 "fox" 和 "fox_food" 这两个子 Map，分别转换成 Pet 对象和 Food 对象。

7.7.3.3. 什么都不干 - VoidAdaptor

某些特殊的情况，你需要彻底控制输入流的解析，同时你又不想使用任何适配器，你可以

```
[CODE]
@AdaptBy(type=VoidAdaptor.class)
public String someFunc(HttpServletRequest req){
    ...
```

VoidAdaptor 什么都不会干，不会碰 HTTP 请求对象的输入流。

7.7.3.4. 上传文件 - UploadAdaptor

NutzMvc 内置了 org.nutz.mvc.upload.UploadAdaptor。关于文件上传详细的说明，请参看：文件上传

7.7.4. 特殊参数

某些时候，你可能需要得到 HttpSession，或者你需要得到 Ioc 容器的一个引用。因为你想做点更高级的事情，你想出搞掂小花样。Nutz.Mvc 完全支持你这样做。

你只要在你的入口函数里声明你希望得到的对象类型即可，比如：

```
[CODE]
@At("/myfunc")
public String someFunc(@Param("pid") int petId,
                       Ioc ioc,
```

```
                    HttpServletRequest req){
    ...
```

* 第一个参数会从 HTTP 参数表中取出赋给入口函数
* 第二个参数，Nutz.Mvc 会把自身使用的 Ioc 容器赋给入口函数，
* 第三个参数，当前请求对象也会直接赋给入口函数。

那么 Nutz.Mvc 到底支持多少类似这样的特殊参数类型呢？

7.7.4.1. Nutz.Mvc 支持的特殊参数类型

* ServletRequest & HttpServletRequest
* ServletResponse * HttpServletResponse
* HttpSession
* ServletContext
* Ioc & Ioc2

7.7.4.2. 还有就是@Attr注解,可以用于获取req或session的attr

* 默认先查找Request,然后找Session
* 找不到就返回null

示例代码:

```
[java]
@Ok("json")
public Object listAllUser(@Attr("me")User user) {
    if (user == null || !user.isAdmin())
        return new HttpStatusView(500);
    return dao.query(User.class, null);
}
```

如果你还想支持更多的类型，那么你就需要定制你自己的适配器了，稍后会有详细描述。

7.7.5. 路径参数

某些时候，你可能觉得这样的 URL 很酷

```
[CODE]
```

```
/my/article/1056.nut
```

起码比

```
[CODE]
/my/article.nut?id=1056
```

看起来要顺眼一些。

Nutz.Mvc 支持将路径作为参数吗？ 你可以在路径中增加通配符，在运行时，Nutz.Mvc 会将路径对应的内容依次变成你的入口函数的调用参数。通配符有两种：

* '?' - 单层通配符，后面你可以继续写路径和其他的通配符
* '*' - 多层通配符，后面个不能再有任何内容

7.7.5.1. 单层通配符

```
[CODE]
@At("/topic/?/comment/?")
public String getComment(int topicId, int commentId){
    // 如果输入的 URL 是： /topic/35/comment/171
    // 那么 topicId 就是 35
    // 而 commentId 就是 171
}
```

如果你有这种需求，我想不用我废话了，不解释，你懂的。

7.7.5.2. 多层通配符

```
[CODE]
@At("/article/*")
public String getArticle(String author, int articleId){
    // 如果输入的 URL 是： /article/zozoh/1352
    // 那么 author 就是 "zozoh"
    // 而 articleId 就是 1352
}
```

Nutz.Mvc 在一层一层解析路径的时候，碰到了 '*'，它就会将这个路径从此处截断，后面的字串按照字符 '/' 拆分成一个字符串数组。为入口函数填充参数的时候，会优先将这个路径参数数组按照顺序填充成参数。之后，如果它发现入口函数还有参数没有被填充完全，它才应用适配器的内部逻辑，填充其余的参数。

7.7.5.3. 单层多层通配符混用

```
[CODE]
@At("/user/?/topic/?/comment/*")
public String getComment(String author, int topicId, int commentId){
    // 如果输入的 URL 是： /user/zozoh/topic/35/comment/171
// 那么 author 就是 "zozoh"
    // 而 topicId 就是 35
    // 而 commentId 就是 171
}
```

7.7.5.4. 通配符的限制

总之，在 @At 注解中通过通配符，你可以声明你的路径参数，但是你的通配符必须是一层路径，但是它们有限制：

```
[CODE]
你不能这么写
/article/a?/topic/*
也不能这么写
/article/y*
```

如果你这么写了，匹配的时候很可能出一些奇奇怪怪的问题。因此你记住了，通配符如果在路径中出现：

* 左边一定有一个字符 '/'
* 右侧可能没有字符，但是如果有，也一定是 '/'

当然，通配符声明的路径参数仍然可以同 @Param 以及 特殊参数 混用，只是请记得，将入口函数中的路径参数排在前面

7.7.6. 错误处理

这是1.b.45及之后的版本才有的功能

在以前的版本中,由用户输入导致的类型转换错误(例如字符串转数字,非法日期),都只能通过@Fail处理

故,现在引入了AdaptorErrorContext,用于解决这一直以来被骂的缺陷

仅当入口方法的最后一个参数为AdaptorErrorContext(其子类也行),才会触发这个错误处理机制

看以下代码:

```
[java]
// 传入的id,会是一个非法的字符串!!
@At({"/err/param", "/err/param/?"})
@Fail("http:500")
public void errParam(@Param("id") long id, AdaptorErrorContext
errCtx) {
    TestCase.assertNotNull(errCtx); // 当没有异常产生时, errCtx为null
    TestCase.assertNotNull(errCtx.getErrors()[0]);
}
```

当用户输入的参数id,为"Nutz"时,自然会导致异常, 而这个方法的最后一个参数是AdaptorErrorContext,所以,仍将进入这个方法, 且errCtx参数不为null

AdaptorErrorContext类本身很简单, 但它也是一个很不错的扩展点. 因为最后一个参数只要求是AdaptorErrorContext或其子类,所以,你可以自定义一个AdaptorErrorContext,覆盖其核心方法setError,以实现你需要的纠错逻辑

7.7.7. 定制自己的适配器

先来看看适配器的接口：

```
[CODE]
public interface HttpAdaptor {
    void init(Method method);
    Object[] adapt( HttpServletRequest request, HttpServletResponse
response, String[] pathArgs);
}
```

你如果实现自己的适配器，你需要知道：

* 你的适配器，对每个入口函数，只会有一份实例 -- Nutz.Mvc 只会创建一遍
    > 如果你的适配器是从 Ioc 容器中取得的，那么也只会被取出一次
* init 函数是 Nutz.Mvc 在创建你的适配器以后，马上就要调用的一个方法，你可以在这个方法里初始化一些逻辑
* adapt 方法的第三个参数，是 Nutz.Mvc 为你准备好的路径参数，它有可能为 null。 你的适配器 将决定是不是应用这个路径参数

7.8. 视图

7.8.1. 什么是视图？

视图的任务就是将入口函数的返回值（一个Java对象）渲染到 HTTP 响应流中。

现在 Nutz.Mvc 自带的主要视图有

* JSP - 采用 JSP 模板输出网页
* Redirect - 客户端重定向
* Forward - 服务器端中转
* Json - 将对象输出成 Json 字符串
* void - 什么都不做

当然你还可以根据需要开发你自己的视图实现类，定制自己的视图也非常简单，请参看本文 #定制自己的视图 一节

7.8.2. 入口函数返回值

前面提到的，视图（View）就是来处理入口函数的返回值的。 当然每个视图的实现类处理的方式不同。我们先来看看视图接口的源代码，非常简单：

```
[Java]
public interface View {
    void render(HttpServletRequest req,
                HttpServletResponse resp,
                Object obj)
                throws Throwable;
}
```

你要想创建自己的视图，你主要的工作就是实现这个接口。

* 显然你的实现类能拿到这次 HTTP 请求的 request 和 response 对象
* 参数 **obj** 就是你入口函数的返回

* 如果你在 @Fail 里声明的视图， **obj** 就是抛出的异常对象

这里还需要再强调一下 Nutz 所谓视图的概念：

一种视图就是一种将 Java 对象写入 HTTP 响应流的方式，谢谢

下面，我们就这个观点，再多举几个个例子，希望大家能通过这两例子，理解不同的视图处理同样的对象，可以有多大的差异。

7.8.2.1. 比如 Jsp 视图

即你声明:

```
[CODE]
@Ok("jsp:xxxx")
```

的时候，无论入口函数返回什么对象,它会将其保存到request的 **"obj"** 属性中。比如

```
[CODE]
@At
@Ok("/test.jsp")
public String test(){
    return "this is test";
}
```

那么，你在 test.jsp 这个页面里，通过:

```
[CODE]
<%=request.getAttribute("obj")%>
```

或者用 JSTL 的 EL

```
[CODE]
```

```
${obj}
```

都会直接输出字符串 **"this is test"**

JSP 视图的更多介绍请参看 #JSP_视图

7.8.2.2. 比如 Json 视图

即你声明:

```
[CODE]
@Ok("json")
```

的时候，无论入口函数返回什么对象，都会被 Json.toJson 函数变成字符串，直接写到HTTP 响应流里。

当然，有些对象，被 Json.toJson 有点麻烦，比如

* InputStream
* Reader

等，这时候你用 Json 视图来输出本身也是自找别扭，建议你使用 #Raw视图

Json 视图的更多介绍请参看 #JSON_视图

7.8.3. 怎样使用视图？

通过注解 @Ok 和 @Fail 你可以为你的入口函数声明不同的渲染方式。当然，在你的逻辑非常复杂的情况下你可以从你的入口函数直接返回一个 View 对象。

7.8.3.1. 通过注解

执行一个业务逻辑可能有两种结果

1. 成功
2. 失败

在每个入口函数上，你都可声明这两个注解

1. @Ok
2. @Fail

仔细观察，你会发现，这两个注解的值只能是一个字符串，那么怎么通过字符串，匹配到视图呢？

无论是 @Ok 还是 @Fail，他们的值的格式总是：

```
[CODE]
"视图类型:视图值"
```

* 字符 ':' 将这个字符串隔断，左半部分是视图的类型，右半部分是视图的值。
* 不同的视图类型，值的形式可能也是不一样的

7.8.3.2. 直接返回 View 对象

你可以在你入口函数根据不同的条件，决定返回何种视图。如果你需要为你的视图声明要渲染的数据，请返回 org.nutz.mvc.view.ViewWrapper

比如你可以这样写：

```
[CODE]
...
@At("/myurl")
public View myFunction(@Param("type") type){
    if(type==0)
        return new ViewWrapper(new UTF8JsonView(), "It is zero!");
    if(type<10)
        return new ViewWrapper(new UTF8JsonView(), "It less than
10!");
    return new ViewWrapper(new JspView("mypage.showNumber"),
type);
}
...
```

7.8.4. 内置的视图

通过 **org.nutz.mvc.view.DefaultViewMaker**， Nutz.Mvc 提供了一些内置视图， 你可以通过 @Ok 和 @Fail 注解声明在你的入口函数上

7.8.4.1. JSP 视图

视图的实现类为： **org.nutz.mvc.view.JspView**

7.8.4.1.1. 一般的使用形式：

```
[CODE]
@Ok("jsp:pet.detail")
```

将会使用 ***WEB-INF/pet/detail.jsp*** 来渲染 HTTP 输出流

你可以不声明视图的值：

```
[CODE]
@Ok("jsp")
```

那么会根据这个请求的地址来寻找 JSP 文件，比如请求：

```
[CODE]
/pet/detail.nut
```

将会使用 ***WEB-INF/pet/detail.jsp*** 来渲染 HTTP 输出流

7.8.4.1.2. 使用 JSTL

如果你使用 JSTL，你还可以

* 通过 ${msg.xxxx} 输出本地字符创，参看 本地化字符串更多的介绍
* 通过 ${base} 输出当前应用的 ContextPath
* 通过 ${obj.xxx} 输出要渲染对象的某个属性值
* 你需要知道，JSP 视图，会将要渲染的对象存入 request 对象的一个属性里，属性名为 "obj"

7.8.4.1.3. JSP 文件的位置

有些人（比如我）喜欢把 JSP 文件统统放在 WEB-INF 目录里。但是更多人的习惯是将 JSP 放在 WEB-INF 外面。

* 这个将对应 *WEB-INF/abc/bbc.jsp*

```
[CODE]
@Ok("jsp:abc.bbc")
```

* 这个将对应 *abc/bbc.jsp*

```
[CODE]
@Ok("jsp:/abc/bbc")
```

* 这个也将对应 *abc/bbc.jsp*

```
[CODE]
@Ok("jsp:/abc/bbc.jsp")
```

7.8.4.1.4. 本地化字符串

在 Nutz.Mvc 入口函数里使用的 JSP 文件可以使用 ${base} 和 ${msg} 来获取应用的 URL 前缀以及本地字符串。

如果你希望放在 WEB-INF 外面如果还希望使用本地化字符串，那么你需要在 web.xml 额外声明一个 Fileter，请参考 本地化字符串 **使用过滤器**一节。

7.8.4.2. JSON 视图

视图的实现类为： **org.nutz.mvc.view.UTF8JsonView**

一般的使用形式：

```
[CODE]
@Ok("json")
```

会将入口函数返回的对象转换成 JSON 字符串

你可以对这个 JSON 字符串的格式进行更多的控制：

```
[CODE]
```

```
@Ok("json:{quoteName:true, ignoreNull:true}")
```

视图的值就是： **"{quoteName:true, ignoreNull:true}"**，这个字符串会被转换成 JsonFormat 对象。如果你想了解更多的 Json 转换的介绍哦，请参看 Json 手册

7.8.4.3. 重定向视图

视图的实现类为： **org.nutz.mvc.view.ServerRedirectView**

一般的使用形式：

```
[CODE]
@Ok("redirect:/pet/list.nut")
或者
@Ok("redirect:/article/2009/10465.html")
```

它将按照给定的视图值，发送 HTTP 重定向命令到客户端

给定参数

```
[CODE]
@Ok("redirect:/pet/detail.nut?pid=${obj.id}")
或者
@Ok("redirect:/article/2009/${articleId}.html")
```

视图会填充占位符号。填充的规则是：

1. 如果占位符名称以 "obj." 开头，则表示应该用入口函数的返回对象的某一个字段来填充
    * "obj.name" 表示对象的 "name" 字段
2. 如果占位符名称以 "p." 开头，，用 HTTP 参数表的值来填充
    * "p.abc" 就表示 HTTP 参数中的 "abc" 参数
3. 如果参数表没有值，则直接用返回的对象来填充。

下面举几个例子：

7.8.4.3.1. 用返回对象的字段填充：

```
[CODE]
@Ok("redirect:/pet/detail.nut?pid=${obj.id}")
入口函数返回： Pet 对象
则取 pet.getId() 填充 ${obj.id}
```

7.8.4.3.2. 用 HTTP 参数填充

```
[CODE]
@Ok("redirect:/pet/detail.nut?pid=${id}")
从 HTTP 参数表取 "id"，填充
如果没有这个参数，入口函数返回什么，直接 toString() 以后填充
如果入口函数是 void 或者返回 null，则用空串填充
```

7.8.4.3.3. 另外一种写法

```
[CODE]
@Ok("redirect:/pet/detail.nut?pid=${id}")
//等价于
@Ok(">>:/pet/detail.nut?pid=${id}")
```

7.8.4.4. 内部重定向视图

视图的实现类为： **org.nutz.mvc.view.ForwardView**

一般的使用形式：

```
[CODE]
@Ok("forward:/pet/list.nut")
//或者
@Ok("forward:/article/2009/10465.html")
同时也可以写成
@Ok("->:/article/2009/10465.html")
```

7.8.4.4.1. 当:后面不加值,或者不是以/开头的话,生成的路径将与Jsp视图类似,除了最后不加.jsp之外. 其实Jsp视图只是Forward视图的子类而已

* 这个将对应 *WEB-INF/abc/bbc*

```
[CODE]
@Ok("jsp:abc.bbc")
```

* 这个将对应 *abc/bbc*

```
[CODE]
@Ok("jsp:/abc/bbc")
```

* 这个将对应 *abc/bbc.jsp*

```
[CODE]
@Ok("jsp:/abc/bbc.jsp")
```

它将按照给定的视图值，服务器内部重定向到指定的地址.

7.8.4.5. HTTP 返回码视图

视图的实现类为： **org.nutz.mvc.view.HttpStatusView**

使用形式

```
[CODE]
@Ok("http:404")
```

返回 HTTP 404 错误码

7.8.4.6. 空白视图

视图的实现类为： **org.nutz.mvc.view.VoidView**

使用形式

```
[CODE]
@Ok("void")
```

对 HTTP 输出流不做任何处理

7.8.4.7. 从 Ioc 容器中获取的视图

使用形式

```
[CODE]
@Ok("ioc:myView")
```

将从 Ioc 容器中获取 myView 对象。 它必须是一个 View，否则会发生转型错误。通过这种形式，可以支持你可以在 Ioc 容器中定义自己的视图对象。

7.8.4.8. Raw视图

视图的实现类为： **org.nutz.mvc.view.RawView**

该视图将数据对象直接写入 HTTP 响应

ContentType 支持几种缩写:

* xml - 表示 text/xml
* html - 表示 text/html
* htm - 表示 text/html
* stream - 表示 application/octet-stream
* 默认的(即 '@Ok("raw")') - 将采用 ContentType?=text/plain

使用方式

```
[CODE]
@Ok("raw")
```

将方法的返回值直接写入流，数据对象可以是如下类型:

null	什么都不做
byte[]	按二进制方式写入HTTP响应流
InputStream	按二进制方式写入响应流，并关闭 InputStream?

char[]	按文本方式写入HTTP响应流
Reader	按文本方式写入HTTP响应流，并关闭 Reader
默认的	直接将对象 toString() 后按文本方式写入HTTP响应流

默认设置resp的ContentType为text/plain，当然,你可以设置ContentType的值

```
[CODE]
@Ok("raw:application/rtf")
```

同时，它也支持如下的缩写形式：

@Ok("raw:xml")	等价于@Ok("raw:text/xml");
@Ok("raw:html")	等价于@Ok("raw:text/html");
@Ok("raw:htm")	等价于@Ok("raw:text/html");
@Ok("raw:stream")	等价于 @Ok("raw:application/octet-stream");
@Ok("raw:json")	等价于@Ok("raw:application/x-javascript");
@Ok("raw:js")	等价于@Ok("raw:application/x-javascript");

7.8.5. 定制自己的视图

通过 @Ok("xxx:xxxx") 这种形式声明的视图非常简洁，易于理解。某些时候，你可能觉得 Nutz.Mvc 支持的视图类型不够，你喜欢其他的视图模板引擎，比如 FreeMarker。因此你希望你的 @Ok 能写成这样：

```
[CODE]
@Ok("freemaker:/pattern/pet/myTemplate")
```

又或者，你希望你的能想这样来输出 PDF 文件：

```
[CODE]
@Ok("pdf:/pdf/article")
```

在视图层，总会有这样或者哪样的需求，对吗？ 那么你可以自己来定制自己的视图规则：

7.8.5.1. 实现自己的视图

实现 org.nutz.mvc.View 接口，比如：

```
[CODE]
public class PdfView implements View{
    public void render(HttpServletRequest req, HttpServletResponse
resp, Object obj){
        // TODO 实现代码 ...
    }
}
```

实现 Render 函数即可，第三个参数就是你的入口函数的返回

7.8.5.2. 实现自己的视图工厂

实现 org.nutz.mvc.ViewMaker 接口，比如：

```
[CODE]
public class PdfViewMaker implements ViewMaker{
    public View make(Ioc ioc, String type, String value){
        if("pdf".equalsIgnoreCase(type)){
            return new PdfView(value);
        }
        return null;
    }
}
```

函数 make 如果返回 null，则表示你的这个视图工厂不知道如何构建这种视图。 Nutz.Mvc 会看看其他的工厂能不能创建这个视图。如果所有的工厂都不能创建这个视图，则会导致异常。

7.8.5.3. 将视图工厂整合进应用中

在主模块上，通过注解 @Views 将你的工厂声明进整个应用即可。

* 你可以声明多个 ViewMaker
* 所有的视图工厂，必须有一个 public 的默认构造函数。

7.8.5.4. 已有的,经过检验的自定义视图

Freemarker Freemarker 可以拓展为Velocity

7.9. 同 Ioc 容器一起工作

7.9.1. 内置的 Ioc 容器

一个 Mvc 框架可以通过 Ioc 接口同一个 Ioc 容器挂接，挂接的方法很简单： 在主模块上声明 @IocBy

```
[CODE]
@IocBy(type=JsonIocProvider.class, args={"/conf/core.js",
"/conf/pet.js"})
public class MainModule {
    ...
```

Nutz.Mvc 内置了 JsonIocProvider 类，帮助你同标准 NutIoc 容器挂接。下面是这个类的全部源代码：

```
[CODE]
public class JsonIocProvider implements IocProvider {
    public Ioc create(ServletConfig config, String[] args) {
        return new NutIoc(new JsonLoader(args), new
ScopeContext("app"), "app");
    }
}
```

极其简单，对吗？ 其中，@IocBy 的 "args" 属性的值将会直接传入 IocProvider 的 create 函数。

7.9.2. JSON 配置的自动扫描

通常你的应用会有不只一个 ioc 配置文件，如果一个一个的写出来非常麻烦。所以，如果你指定的是一个目录：

```
[CODE]
@IocBy(type=JsonIocProvider.class, args={"myioc"})
public class MainModule {
    ...
```

如上例，那么， Nutz 内置的 JsonLoader 会自动寻找 myioc 的目录（除了 CLASSPATH 下的目录，也可以是个绝对目录）目录内的所有 *.js 或者 *.json 文件都会被自动加载

并且，你也可以混合目录和文件，比如：

```
[CODE]
@IocBy(type=JsonIocProvider.class, args={"myioc", "abc/single.js"})
public class MainModule {
    ...
```

目录 "myioc" 下所有的 *.js 和 *.json 文件都会被加载，并且也会加载 "abc/single.js"

7.9.3. 由 Ioc 容器管理子模块

通过 @IocBy 为整个应用声明了 Ioc 容器，那么如何使用呢。 实际上，你的每一个模块都可以来自容器，只要你在模块上声明 @InjectName。 当然，在主模块声明这个注解是没有意义的。

```
[CODE]
@InjectName("petM")
public class PetModule {
   ...
```

* 如果你声明了这个注解， Nutz.Mvc 构造你的这个模块的时候，会通过 Ioc 容器获取，而不直接调用默认构造函数了。
* 如果你的 '@InjectName' 并没有值，那么默认会将你的模块类名首字母小写作为模块的注入名
    > 比如上例，你直接声明 '@InjectName' 同 '@InjectName("petModule")' 是等效

的

7.9.4. 在容器对象里获得 ServletContext

ServletContext! 是的，有些时候你需要它。比如你打算作一个 Freemarker 的工厂。总之 Java 世界的奇奇怪怪的框架和插件们或多或少的都可能会依赖这个接口。如果你正在读这段文字，你说不定也正好需要这个接口。

在 Nutz.Mvc 中，你可以很容易在入口函数里拿到 ServletContext，是的，你只要直接在入口函数的参数里声明 ServletContext类型的参数就是了，Nutz.Mvc 会老老实实的为你填充这个参数，同理你也能拿到 HttpSession, HttpServletRequest, HttpServletResponse。但是在 Ioc 容器里，你希望你的对象也能得到这个参数，这个要求很过分吗？不当然不过分，在 1.a.33之后的版本，Nutz.Mvc 在启动的时候，会为你的 Ioc 容器增加一个新的自定义值，在容器里，你可以：

```
[JSON]
...
fields : {
    servletContext : {app:'$servlet'}
}
...
```

你就能为你这个对象的 servletContext 字段注入 ServletContext 的实例。当然，如果你想获得 ServletContext 中的属性，你完全可以：

```
[JSON]
...
fields : {
    servletContext : {app:'myObjName'}
}
...
```

那么，你在 servletContext 里的属性也会被一并注入

当然，你的 Ioc 容器，必须实现了 Ioc2 这个接口，默认的 NutIoc 就是实现这个接口的。

7.9.5. 容器管理对象的生命周期范围

如果你使用的是 Nutz.Ioc 标准容器，或者你的 Ioc 容器实现了 Ioc2 接口，那么你的模块类的生命周期是可以定制的。以 Nutz 的 Json 配置语法为例，你可以为你的对象增加属性：

```
[CODE]
{
    petModule : {
        type : "com.my.PetModule",
        scope : "session"
    }
}
```

那么，你的 petModule 对象将只会存在 session里，当 session 停止后，会被 NutSessionListener 注销。当然，你需要在web.xml 中声明这个会话监听器。

对于 Nutz.Mvc，它支持如下的生命周期范围

* app
* session
* request

7.9.6. 需要注意的问题

如果你让你的 Mvc 框架同 Ioc 容器一起工作（通常你都会这样），请注意，有可能会有这样的问题 (Issue 105)：

你的模块的 @InjectName 或者 Ioc 配置文件里的对象的名称，如果同 request 或者 session 里属性名称重复，有可能会被覆盖，尤其是 @InjectName。Nutz.Mvc 的工作机制导致这个现象发生：

即，当服务器收到一个请求后，Nutz 会构建两个对象：

1. RequestIocContext
2. SessionIocContext

并且将其合并为一个 ComboContext 传入 Ioc 容器。（当然，如果你的 Ioc 容器不是 Nutz 内置，而是自己实现的，并且你的容器没有实现 Ioc2 接口，这个问题不会发生）在调用相应模块的入口函数时，Nutz 首先将模块对象从 Ioc 容器中取出，取出的顺序是：

1. RequestIocContext
2. SessionIocContext

3. Ioc 容器内置的 Context -- ScopeContext -- scope: 'app'

如果你的模块声明了注解 @InjectName("ABC") -- 事实上，你通常都会声明这个注解。但是不幸的（或者幸运的）是，你的 request或者 session 对象，在某一次操作中，被你设置了 "ABC" 属性，那么下次调用是，Ioc 容器会说： “ABC? 原来在 request 已经有了哦，那就不用我缓存里的了。”

这个特性给予应用程序极大的灵活性，如果愿意，你完全可以在一个session里保存一个数据源，然后在session注销时关闭这个数据源。当然，你的 Ioc 配置，一定要把引用到这个数据源的对象 singleton 都设为 false，或者将他们的 scope 都设成 session 或者 request

7.9.7. 自定 Ioc 容器

你很喜欢 Spring，或者你很喜欢 Guice。不管怎么说，你不想用 Nutz.Ioc，那么没关系，你可以自己实现一个 IocProvider。并用 @IocBy 声明到整个应用中即可。通过 Spring 或者 Guice 实现 Ioc 接口想必不是什么难事。

7.10. 本地化字符串

7.10.1. 基本策略

每个 Mvc 框架都有自己的本地化字符串的解决方案， Nutz.Mvc 的这个是相当简陋的。我只是个人觉得足够用了。下面我把它简单介绍一下：

* 假定所有的本地化字符串文件都会存放在某一目录下
    > 这个目录下所有的 .properties 文件，将作为默认的本地字符串文件。
    > 每一种语言都会是一个目录，目录名称对应一个 Locale 的 toString()，请参看 java.util.Locale 的 JDoc
        - 比如简体中文，就是 zh_CN
        - 比如美式英语，就是 en_US
    > 目录下所有的 .properties 文件存放着该地区的字符串信息
    > .properties 文件需要按照 UTF-8 方式编码
* 目录，通过 @Localization("全路径") 声明在主模块上
* 当应用启动时，一次读入所有的字符串，并存入 ServletContext，属性名称为：“org.nutz.mvc.annotation.Localization”
* 应用可以自行设置当前 Session 是哪一个国家和地区
    > Mvcs.setLocaleName(String localeName)
* 每次请求时，会根据 Session 中的 localeName，从 ServletContext 中将对应 Locale 的字符串取出，设入 Request 对象
    > 属性名为 "msg"
    > 如果当前会话没有被设置 Locale，则将 "msg" 设置成默认本地化字符串

7.10.2. 使用字符串

7.10.2.1. 在主模块上声明

比如：

```
[CODE]
...
@Localization("mymsg")
public class MyMainModule {
    ...
```

* 在主模块上声明 @Localization 注解，指向一个目录
* 在目录下建立文件夹，比如 **zh_CN**，每个目录下所有 **.properties** 文件都会被当作字符串文件
* **.properties** 文件 一定要是 UTF-8 编码的
* 比如 @Locallization("mymsg") 会指向 CLASSPATH 下的 **mymsg** 目录

7.10.2.2. 在 JSP 里使用

直接 Scriptlet

```
[CODE]
...
<h1><%=((Map<String,String>)request.getObject("msg")).get("my.msg.key")%></h1>
...
```

采用 JSTL

```
[CODE]
...
<h1>${msg['my.msg.key']}</h1>
...
```

7.10.2.3. 我到底支持哪些语言

请参看 **org.nutz.mvc.Mvcs** 的 JavaDoc，这里我列几个常用的方法:

Mvcs.getLocaleName(HttpSession session)	获取当前会话的 Locale 名称
Mvcs.setLocaleName(HttpSession session, String name)	为当前会话设置 Locale 的名称
Mvcs.getLocaleNames(ServletContext context)	获取整个应用可用的 Locale 名称集合
Mvcs.hasLocaleName(HttpSession session)	判断当前会话是够设置了特殊的 Locale 的名称
Mvcs.hasLocale(HttpSession session)	判断当前会话是否已经设置了本地字符串表

7.10.3. 切换本地语言

自从 1.b.44 之后，Nutz 对本地字符串语言切换的接口做了改动:

```
[CODE]
// 设置一个本地字符串
@At("/lang/change")
@Ok("redirect:/")
public void changeLocal( @Param("lang") String lang){
    Mvcs.setLocalizationKey(lang);
}
```

1.b.44 版本 之前，只能采用如下方法:

```
[CODE]
// 设置一个本地字符串
@At("/lang/change")
@Ok("redirect:/")
public void changeLocal( @Param("lang") String lang, HttpSession sess){
    Mvcs.setLocale(sess, lang);
    Mvcs.setLocaleName(sess, lang);
}
```

7.10.4. 设置应用程序的默认语言

对于任何一个支持多语言版本的应用程序，第一次启动的时候，总要采用一种语言。自 1.b.44 之后，Nutz 给出了一个明确的方法，来设置这个信息。你可以在应用程序启动的Setup 时，调用:

```
[CODE]
...
Mvcs.setDefaultLocalizationKey("zh_CN");
...
```

在 1.b.45 后，Nutz 又给出一个更直接的实现，你可以直接:

```
[CODE]
...
@Localization(value="mymsg", defaultLocalizationKey="zh_CN")
public class MyMainModule {
...
```

7.10.5. 使用过滤器

有些是由，你想让你的 JSP 文件 (不是通过 Nutz.Mvc 入口函数访问的)也可以使用"**本地化字符串**"功能，那么你需要在 web.xml 这么配置:

```
[CODE]
<!--
   Nutz.Mvc 还提供了一个过滤器，你可以用这个过滤器为所有的直接访问的 jsp 设置必要的 Request 属性。 比如 ${base}
   以及 ${msg}
-->
<filter>
   <filter-name>msgs</filter-name>
   <filter-class>org.nutz.mvc.NutFilter</filter-class>
</filter>
<filter-mapping>
   <filter-name>msgs</filter-name>
```

```
    <url-pattern>*.jsp</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>msgs</filter-name>
    <url-pattern>*.jspx</url-pattern>
</filter-mapping>
```

7.10.6. 定制自己的本地化字符串方式

你需要自己实现一个 MessageLoader 的接口，然后声明到 '@Localization' 中。比如你的实现类名字为 'MyMessageLoader'，那么你应该这么声明:

```
[CODE]
...
@Localization( type=MyMessageLoader.class,
               value="msg" )
public class MyMainModule {
...
```

对于 MessageLoader 接口，就一个方法需要你来实现:

```
[CODE]
public interface MessageLoader {
    /**
     * 本函数将根据传入的 "refer" 参数，返回一个 Map <br>
     * Map 的键是语言的名称，比如 "en_US", "zh_CN" 等，<br>
     * 你会通过 Mvcs.setLocalizationKey 来直接使用这个键值
     * <p>
     * 与键对应的是一个消息字符串的 Map, 该 Map 的键就是消息键，值就是消息内容
     *
     * @param refer
     *            参考值。来自 '@Localization.value'
     * @return 多国语言字符串的 Map
     */
    Map<String, Map<String, Object>> load(String refer);
}
```

你声明在 '@Localization' 中的 "value" 的值，会被传入这个接口，作为 refer 参数的值

7.10.7. 让 Ioc 容器管理你的 MessageLoader

这是 1.b.45 已后才有的方法:

```
[CODE]
...
@Localization( type=MyMessageLoader.class,
               beanName="myMessages",
               value="msg" )
public class MyMainModule {
...
```

提供了 "beanName" 属性，这样，Nutz.Mvc 将从 Ioc 容器中加载名字为 "myMessages" 的对象。当然它的类型必须是你声明的 "MyMessageLoader.class"

7.11. 过滤器

7.11.1. 什么是过滤器?

请你再一次回顾这张图：

一个 HTTP 请求，过滤器是第一组被执行的对象。同适配器不同的是，一个请求中，可以执行多个过滤器。

7.11.2. 如何使用过滤器？

通过 @Filters 注解。

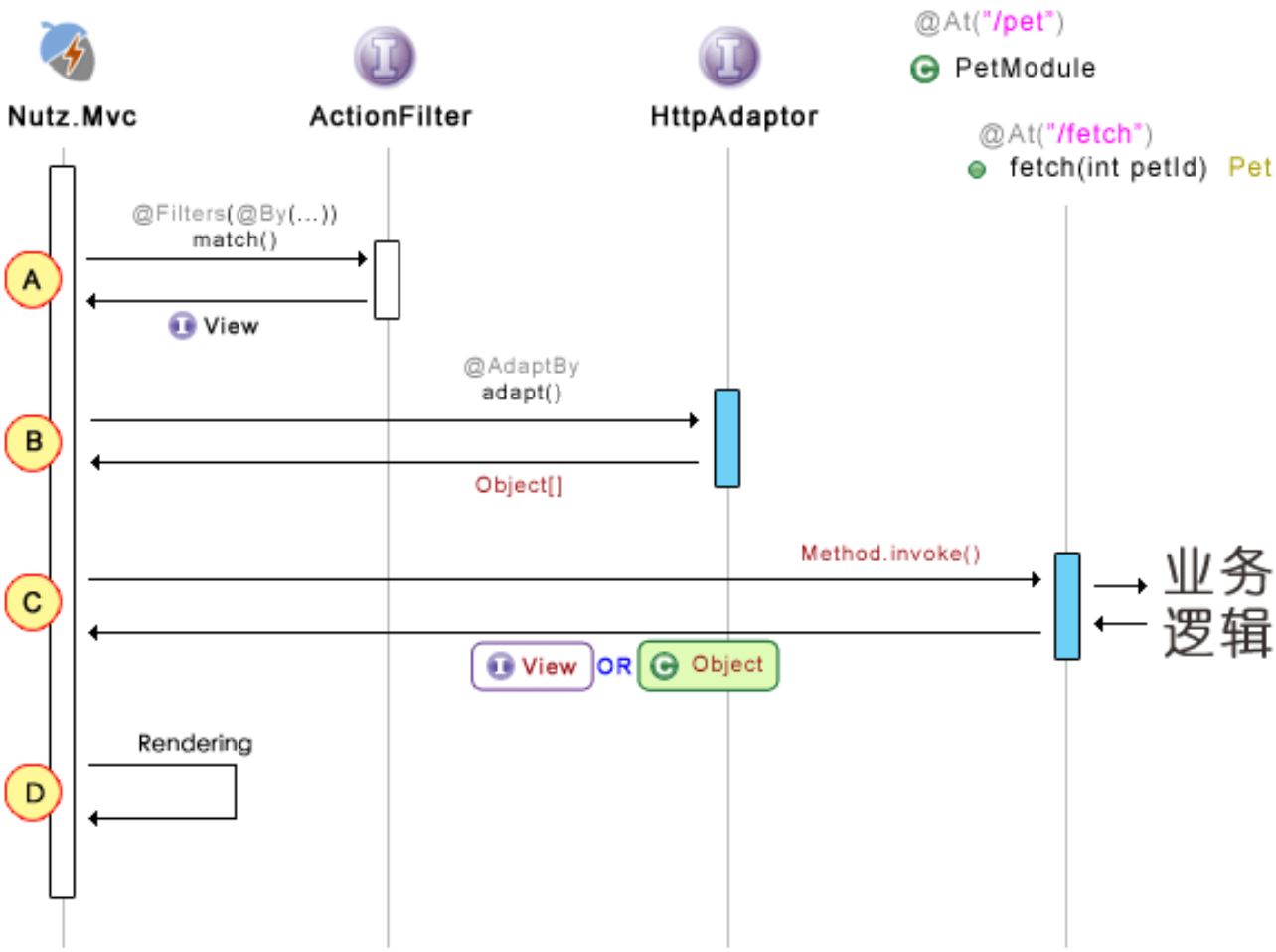
注解 '@Filters' 的值是一个 '@By' 注解的数组，它可以声明在这三个地方

1. 入口函数
2. 子模块
3. 主模块

其中入口函数的 @Filters 优先级更高，其次是子模块，最后是主模块。

就是说，你在入口模块声明了两个过滤器：

```
```




```
[CODE]
@Filters({@By(type=Filter1.class), @By(type=Filter2.class)})
public final class MainModule{
    ...
```

在某一个入口函数上声明了一个空的过滤器数组：

```
[CODE]
...
@At
@Filters
public String myFunction(){
    ...
```

那么，这个入口函数将不会应用任何过滤器。

7.11.2.1. 使用内置过滤器

```
[CODE]
@Filters(@By(type=CheckSession.class, args={"user", "/login.jsp"}))
```

CheckSession 是 Nutz.Mvc 为你内置的一个唯一的过滤器，它的构造函数需要两个参数：

* 检查 session 的什么属性？
* 如果不存在，重定向到哪里？

7.11.2.2. 定制你自己的过滤器

你当然可以定制你自己的过滤器，请注意过滤器接口代码：

```
[CODE]
public interface ActionFilter {
    View match(ActionContext context);
}
```

根据 HttpRequest 对象，你的过滤器需要决定返回值是：

* **一个 View** : 这个请求有问题，不要继续执行了，马上用这个 View 来渲染 HTTP 输出流吧
* **null** : 恩，这个请求没问题，继续吧。

如果你的过滤器返回的是 null，Nutz.Mvc 会继续执行下一个过滤器，如果全部的过滤器都返回null它就会调用适配器，进行后面的操作

7.11.3. 交给 Ioc 容器管理

如果你的 Filter 需要比较灵活的配置，你可以将它交由 Ioc 容器负责

```
[CODE]
@Filters(@By(type=MyFilter.class, args={"ioc:myFilter"}))
```

其中 "myFilter" 就是你这个过滤器在 Ioc 容器中的名字。关于 Ioc 容器更多的知识，请参看Ioc手册。

请注意 如果你要让 Ioc 容器管理你的过滤器，你的应用必须已经声明了一个 Ioc 容器。 如何在 Mvc 框架中声明 Ioc 容器，请参看同 Ioc 容器一起工作

7.12. 文件上传

7.12.1. 关于文件上传

大多数的 Web 应用都不可避免的，会涉及到文件上传。文件上传，不过是一种适配 HTTP 输入流的方式。为此，Nutz.Mvc 内置了一个专门处理文件上传的适配器
org.nutz.mvc.upload.UploadAdaptor

这个适配器专门解析形式为

```
[CODE]
<form target="hideWin" enctype="multipart/form-data">
   <input type="file">
   ...
```

的 HTML 提交表单

这个适配器的工作细节是这样的：

* 它一次将 HTTP 输入流中所有的文件读入，保存在临时文件目录里
* 表单项目会保存在内存里
* 在上传的过程中，它会向当前 Session 中设置一个对象：
  org.nutz.mvc.upload.UploadInfo
  > 属性名为 "org.nutz.mvc.upload.UploadInfo"
  > 通过静态函数 Uploads.getInfo(HttpServletRequest req) 可以很方便的获取当前会话的 UploadInfo
* 不断的读入输入流的过程，会记录在 UploadInfo 里面。
  > UploadInfo 字段 sum 是当前 HTTP 请求的 ContentLength，表示 HTTP 输入流总长度为多少字节
  > UploadInfo 字段 current 是当前会话已经上传了多少字节

7.12.2. 如何使用

如果你读过 适配器 一节，我想你已经知道怎么使用文件上传了，这里还需要多一点说明

7.12.2.1. 通过 @AdaptBy 声明

```
[CODE]
```

```
...
@AdaptBy(type = UploadAdaptor.class, args = { "${app.root}/WEB-
INF/tmp" })
public void uploadPhoto(@Param("id") int id, @Param("photo") File
f){
    ...
```

Nutz.Mvc 会为你的入口函数创建一个上传适配器，上传适配器的会将临时文件存放在***WEB-INF/tmp*** 目录下。

7.12.2.2. 更细腻的控制

更多的时候，你想控制

* 上传文件的临时文件数量
* 可以上传的单个文件最大尺寸
* 文件的类型
* 空文件跳过不保存

那么你需要把上传适配器交由 Ioc 容器来管理，首先你的入口函数需要这么写：

```
[CODE]
...
@AdaptBy(type = UploadAdaptor.class, args = { "ioc:myUpload" })
public void uploadPhoto(@Param("id") int id, @Param("photo") File
f){
    ...
```

请注意这里的 "ioc:myUpload"，Nutz.Mvc 发现你的参数为此形式时，它会从 Ioc 容器里获取你的上传适配器的实例（名字为**myUpload**），而不是直接 new 出来。

然后你需要在你的 Ioc 配置文件里（以 Json 配置为例子）配置你的 UploadAdaptor, 你需要增加下面这三个对象：

```
[CODE]
...
tmpFilePool : {
    type : 'org.nutz.filepool.NutFilePool',
```

```
    // 临时文件最大个数为 1000 个
    args : [ "~/nutz/blog/upload/tmps", 1000 ]
},
uploadFileContext : {
    type : 'org.nutz.mvc.upload.UploadingContext',
    singleton : false,
    args : [ { refer : 'tmpFilePool' } ],
    fields : {
        // 是否忽略空文件, 默认为 false
        ignoreNull : true,
        // 单个文件最大尺寸(大约的值，单位为字节，即 1048576 为 1M)
        maxFileSize : 1048576,
        // 正则表达式匹配可以支持的文件名
        nameFilter : '^(.+[.])(gif|jpg|png)$'
    }
},
myUpload : {
    type : 'org.nutz.mvc.upload.UploadAdaptor',
    singleton : false,
    args : [ { refer : 'uploadFileContext' } ]
}
...
```

这样，在配置文件中，你就可以随心所欲的进行控制了。而且这么写仅仅多写了不到 20 行的配置文件，到也没有特别麻烦，对吗？（相关详细规则描述：http_adaptor.man#通过_Ioc_容器获得适配器）

看到这里，有些同学会问了，为什么你用注解的方式可以这么写：

```
[CODE]
@AdaptBy(type = UploadAdaptor.class, args = { "${app.root}/WEB-INF/tmp" })
```

'**{app.root}**' 用起来多方便啊，那么在 Ioc 容器来，我的临时目录还想放在这里，有没有什么办法啊？

办法有。但是因为 Ioc 容器原来设计的是同 Mvc 的 Web 容器环境不相干的，所以要想绕过了这点限制，会稍微的有一点麻烦。

首先，因为 Ioc 容器设计的良好扩展性，所有实现了 'org.nutz.ioc.Ioc2' 这个接口的 Ioc容器（当然，Nutz 自带的 Ioc 容器 - NutzIoc，很好的实现了 Ioc2 这个接口）都可以被 Nutz.Mvc注入一种新的获取值的方式（详情请参看 with_ioc.man#在容器对象里获得_ServletContext),我想你只需要实现一个简单的 Java 对象即可:

```
[Java]
package com.abc.myapp;
import javax.servlet.ServletContext;
public class MyUtils {
    private ServletContext sc;
    public String getPath(String path) {
        return sc.getRealPath(path);
    }
}
```

然后，你可以在 Ioc 容器的配置文件里，增一个对象，并修改 'tmpFilePool' 对象的配置信息：

```
[CODE]
utils : {
    type : 'com.abc.myapp.MyUtils',
    fields : {
        sc : {app:'$servlet'}   // 将 ServletContext 对象注入 MyUtils
    }
},
tmpFilePool : {
    type : 'org.nutz.filepool.NutFilePool',
    args : [ {java:'$utils.getPath("/WEB-INF/tmp")'}, 1000 ]   // 调用
MyUtils.getPath 函数
},
```

7.12.2.3. 在入口函数的参数里获得上传的文件

因为 FORM 表单里的 input type=file 项都已经被 UploadAdaptor 解析并存入服务器的临时目录（你声明的那个目录，比如上例是 /WEB-INF/tmp），你在入口函数里可以直接获取：

```
[CODE]
```

```
@AdaptBy(type = UploadAdaptor.class, args = { "ioc:myUpload" })
public void uploadPhoto( @Param("photo") File f){
    ...
```

你的参数 f 将获取到表单项中:

```
[CODE]
<form target="hideWin" enctype="multipart/form-data">
    <input type="file" name="photo">
    ...
```

中的文件内容。你直接打开一个 InputStream 就能从这个 File 中读取客户端上传的文件内容了。

但是，如果你也想知道这个文件在客户端原本的名字，怎么办呢？你可以：

```
[CODE]
@AdaptBy(type = UploadAdaptor.class, args = { "ioc:myUpload" })
public void uploadPhoto( @Param("photo") TempFile tf){
    File f = tf.getFile();                    // 这个是保存的临时文件
    FieldMeta meta = tf.getMeta();            // 这个原本的文件信息
    String oldName = meta.getFileLocalName();    // 这个时原本的文件名称
    // TODO do what you wanna to do here ...
}
```

7.12.2.4. 一次上传多个同名文件

相信很多同学都会有这样的用法，就是，希望一次上传的表单为：

```
[CODE]
<form target="hideWin" enctype="multipart/form-data">
    <input type="file" name="photo">
    <input type="file" name="photo">
    <input type="file" name="photo">
```

```
    ...
```

看，一个上传表单中带有多个同名文件，那么怎么办呢？ 你在入口函数里可以这么声明：

```
[CODE]
@AdaptBy(type = UploadAdaptor.class, args = { "ioc:myUpload" })
public void uploadPhoto( @Param("photo") TempFile[] tfs){
    ...
}
```

是的，它们会被变成一个数组。顺序同 <form> 中的顺序。

但是这里，有一点**限制**

* **你只能用 TempFile[]**

也就是说，如果你用了 File[] 将会出错。原因不解释。

7.12.3. Html5流式上传(实验性)

7.12.3.1. 1.b.44及之后的版本,同时兼容Html5流式上传. 但,请注意以下事项:

* 上传的Content-Type必须是application/octet-stream
* 必须带Content-Length
* 默认情况下,对于的表单属性是filedata,除非在Content-Disposition另外声明
* 不支持Content-Type过滤,因为上传数据中通常不携带这种信息
* 支持文件名过滤,但请留意默认文件名nutz.jpg

7.13. 请求范围的模块

7.13.1. 为什么需要请求范围的模块

无论 Struts2 还是 Spring，都提供了一种控制器： 每次请求，创建实例，使用后即抛弃。这样的控制器的好处就是可以放心的吧 request 和 response 对象放心的存成它的私有属性，反正使用一次后就丢掉了。

在 Nutz.Mvc，所谓控制器，实际上就是 Module，默认的，所有的模块都是整个应用程序唯一的，除非你在 Ioc 配置文件里另有说明。

那么 Nutz 可以做到每次请求即抛弃的控制器（*Module*）吗？ 答案是肯定的，稍微有点麻烦，但是绝对不会繁琐的让你头疼，你需要自己动手做点事情，如果你对编写 JSP/Servlet 应用程序有点基本的支持，这点事情对你不会构成负担。

7.13.2. 怎样做？

我们这里给你举一个简单的例子，假设你有一个模块，通过 Nutz.Ioc 来配置：

7.13.2.1. 你的模块代码

```
[CODE]
@InjectName("myModule")
public class MyModule{
    private HttpServletRequest request;
    @At("/abc")
    public String tryIt(){
        return request.getParameter("abc");
    }
}
```

7.13.2.2. 修改 Ioc 配置文件

```
[CODE]
...
myModule : {
    type : 'com.you.app.MyModule',
    scope : 'request',
    fields : {
        request : {refer: '$request'}
    }
}
...
```

* 请注意 **scope : 'request'** 以及 fields 里 request 字段的属性 **refer: '$request'**
* scope 指明了这个对象存在的上下文环境，如果你不指明，默认是 '**app**'
    > 更多关于 scope，请参看

关键就是这个 **refer: '$request'**，很抱歉，你得自己设置它的值。怎样设置呢？通过 web.xml 的 HttpFilter

7.13.2.3. 添加 HttpFilter

注: 如果你使用的是 Nutz.1.b.37 以及之后的版本，不需要做这个操作，因为每次请求，它都会自动增加 下面的属性，而之前的版本则不会。

增加一个 HttpFilter

```
[CODE]
public class MyFilter implements Filter {
    public void destroy() {}
    public void init(FilterConfig arg0) throws ServletException {}
    public void doFilter(ServletRequest req, ServletResponse resp,
FilterChain chain)
            throws IOException, ServletException {
        req.setAttribute("$request", req);
        chain.doFilter(req, resp);
    }
}
```

将它配置在 web.xml 里，具体怎么配置我想就不用我废话了吧

7.13.2.4. 最后

这样，就可以做到 Request 级别的 Module 了，即，每次请求，都会重新创建你的 Module 实例

7.14. REST 的支持

7.14.1. 为什么要支持 REST

这个特性被喊了好几个版本了，并且先后被报了好几个 Issue:

* Issue 323
* Issue 369

加上今天是个风和日丽的日子，Nutz.Mvc 重构完毕。架构这么灵活强大，那么我就把这个特性加上吧，呵呵。

7.14.2. 如何使用 REST

暂时的，Nutz.Mvc 对于 REST 的支持，仅仅包括四个方法：

* GET
* POST
* PUT
* DELETE

比如，你有一个 User 对象，你想为其增加 "修改" 和 "删除" 的操作，在模块类里你可以定义如下两个方法

```
[CODE]
@At("/user/?")
@GET
public User getUser(int userId){
    // TODO 这里是实现代码
}
@At("/user/?")
@POST
public User updateUser(int userId, @Param("..") User user){
    // TODO 这里是实现代码
}
@At("/user/?")
@DELETE
public void deleteUser(int userId){
    // TODO 这里是实现代码
}
```

看，很简单，不是吗？你可以为你的请求设置路径参数，也可以为你的请求声明一个特殊的 HTTP 方法。路径参数的形式可以是：

1. /user/?/topic/?
2. /user/?/topic/*
3. /*

关于 "路径参数" 的具体的解释，请参看 适配器 > 路径参数

在一个入口函数上，你可以标注一个或多个下面的注解:

* @GET
* @POST
* @PUT
* @DELETE

这几个注解描述了当前入口函数仅仅相应什么样的 HTTP 请求方法。在你的应用运行时，即使 Nut.Mvc 匹配上了 URL，如果 HTTP 请求的方法和你声明的不符，它也当这个入口函数不存在。

如果你不声明这四个注解中的任何一个，则表示你希望你的这个入口函数处理发送到这个 URL 的任何HTTP 请求。

因此，你可以为一个入口函数声明 (@GET|@POST|@PUT|@DELETE) 中的一个或多个，处理其中一种或者多种HTTP 请求，而另外一个入口函数不声明注解，用来处理余下的其他 HTTP 请求方法。当然，这两个入口函数的 @At 应该是一致的。

7.15. Nutz.Mvc 注解一览表

7.15.1. 主模块上支持的注解

@Modules	整个应用有哪些**子模块**，*子模块不能再嵌套子模块*
@IocBy	整个应用，应采用何种方式进行反转注入。如果没有声明，整个应用将不支持 Ioc
@Localization	整个应用的本地地化字符串的目录
@SetupBy	应用启动的关闭时，应该进行的处理。
@Views	扩展整个应用支持的视图模板类型
@Ok	整个应用默认的成功视图
@Fail	整个应用默认的失败视图
@AdaptBy	整个应用默认的 HTTP 参数适配方式
@Filters	整个应用默认的过滤器数组
@Encoding	整个应用默认的输入输出字符编码

* 绿色的注解表示只能应用在主模块上
* 灰色的注解表示还可以应用在**子模块**以及**入口函数**上

7.15.2. 子模块上支持的注解

@InjectName	Ioc 容器中，本模块对应的名称，如果不指明，表示这个模块是通过默认构造函数创建的
@At	模块所有**入口函数**的 URL 前缀

@Ok	模块默认成功视图
@Fail	模块默认失败视图
@AdaptBy	模块默认 HTTP 参数适配方式
@Filters	模块默认的过滤器数组
@Encoding	模块默认 HTTP 请求的输入输出字符编码

* 绿色的注解表示只能应用在子模块上
* 灰色的注解表示还可以应用在**子模块**以及**入口函数**上

7.15.3. 入口函数上支持的注解

@At	函数对应的 URL
@Ok	成功视图
@Fail	失败视图
@AdaptBy	HTTP 参数适配方式
@Filters	函数的过滤器数组
@Encoding	HTTP 请求的输入输出字符编码
@GET	限定函数接受 HTTP GET 请求
@POST	限定函数接受 HTTP POST 请求
@PUT	限定函数接受 HTTP PUT 请求
@DELETE	限定函数接受 HTTP DELETE 请求

* 灰色的注解表示还可以应用在**子模块**以及**入口函数**上
* 蓝色的注解表示只能应用在**入口函数**上

8. Json 手册

8.1. Nutz.Json 简要说明

请参看： Nutz Json Book

8.2. 控制输出的格式

coming soon

8.3. 利用 Castors 按类型控制字段输出

coming soon

8.4. 高级定制

8.4.1. 动机

Json 天然的同 Javascript 融为一体，做 WEB 应用，服务器同浏览器的通信，通过 JSON 是最方便高效的选择。同时，有一个事实：**服务器比浏览器聪明**， 因为我们可以在服务器写更复杂的逻辑。

这里有一个需求，程序员希望自己的 JSON 返回是这个样子的：

```
[CODE]
{
    text : 'click me',
    handle : function(){
        alert('a function');
    }
}
```

这样的返回不符合 JSON 的语法，但是它符合 JavaScript 的语法。 默认的，Nutz.Json 并不支持这样的输出。但是它的确为类似这样的要求做了考虑:

简单的说，就是通过 @ToJson 注解，你可以控制你的对象在 JSON 字符串中的表现形式

8.4.2. POJO 的源代码

8.4.2.1. Button 对象

```
[Java]
```

```
public class Button {
    private String text;
    private Function handler;
    public String getText() {
        return text;
    }
    public void setText(String text) {
        this.text = text;
    }
    public Function getHandler() {
        return handler;
    }
    public void setHandler(Function handler) {
        this.handler = handler;
    }
}
```

8.4.2.2. Function 对象

```
[Java]
@ToJson
public class Function {
    private String body;
    public Function(String body) {
        this.body = body;
    }
    public String getBody() {
        return body;
    }
    public void setBody(String body) {
        this.body = body;
    }
    public String toJson(JsonFormat format) {
        StringBuilder sb = new StringBuilder("function(){");
        sb.append(body);
        sb.append('}');
        return sb.toString();
    }
}
```

8.4.2.3. 一点说明

上面的例子，我想你已经能基本明白了，这里再着重做几点说明：

* 将 Function 对象序列化成 JSON 字符串时，会调用 toJson 函数
  > 这个函数必须有一个参数，类型为 JsonFormat
* @ToJson 注解默认值为 "toJson"
  > 即，如果你不喜欢 "toJson" 这个名字，你可以在你的 POJO 中 @ToJson("任何你喜欢的名字")
  > 你的函数也必须有一个参数，类型为 JsonFormat

8.4.3. 调用代码

```
[Java]
Button btn = new Button();
Function func = new Function("alert('I am button');");
btn.setText("Click me!");
btn.setHandler(func);
System.out.println(Json.toJson(btn));
```

8.4.4. 控制台输出

```
[Json]
{
   text :"Click me!",
   handler :function(){alert('I am button');}
}
```

只要你高兴，通过 @ToJson 注解，你可以随意控制对象的输出内容。

8.5. Json Book

8.5.1. 目标

* 通过简单的toJson 和fromJson能完成绝大部分的互转工作, 不再需要额外的配置.
* 能够提供模板, 容易的更改Json的展现, 和反向的Java Object生成
* 适用于任何的Java对象, 包括基本类型
* 支持JDK1.5+

8.5.2. 简单开始

Java转换成Json:

```
[JAVA]
Json.toJson(6.5); ===> 输出: 6.5
Json.toJson("json"); ===> 输出: "json"
Json.toJson(new int[0]); ===> 输出: []
```

Json转换成Java:

```
[JAVA]
int intValue = Json.fromJson(Lang.inr("65"));
float floatValue = Json.fromJson(float.class, Lang.inr("65"));
int[] intArray = Json.fromJson(int[].class, Lang.inr("[65]"));
```

8.5.3. 对Map的操作

Java转换成Json:

```
[JAVA]
Map<String,Object> map = new HashMap<String, Object>();
map.put("name", "Peter");
map.put("age", 21);
map.put("friends", null);
System.out.println(Json.toJson(map));
```

这个时候会输出成:

```
[JSON]
{
    age :21,
    name :"Peter"
```

```
}
```

会自动忽略掉值为空的字段.

Json转换成Java:

```
[JAVA]
String json = "{age :21,name :\"Peter\"}";
map = Json.fromJson(HashMap.class, Lang.inr(json));
```

8.5.4. 对Bean的操作

设计这样的类:

```
[CODE]
public static class UserObject {
    public static enum Sex {
        Male, Female
    };
    private String name;
    private int age;
    private Sex sex;
    private Date birthday;
    private UserObject bestFriend;
//省略Getter/setter
}
```

Java转换成Json:

```
[JAVA]
UserObject peter = new UserObject();
peter.setName("Peter");
peter.setAge(22);
peter.setSex(UserObject.Sex.Male);
System.out.println(Json.toJson(peter));
```

这个时候会输出成:

```
[JSON]
{
    name :"Peter",
    age :22,
    sex :"Male"
}
```

会自动忽略掉值没有赋值的字段birthday和bestFriend.

更进一步, 我们来看看关联的情况, 在以上代码的基础上面:

```
[JAVA]
amy.setName("Amy");
amy.setAge(21);
amy.setSex(UserObject.Sex.Female);
amy.setBirthday(new Date());
amy.setBestFriend(peter);
System.out.println(Json.toJson(amy));
```

这个时候会输出成:

```
[JSON]
{
    name :"Amy",
    age :21,
    sex :"Female",
    birthday :"2009-04-11 21:28:59",
    bestFriend :{
    name :"Peter",
    age :22,
    sex :"Male"
    }
}
```

关联对象的关联会被自动探知.

注意:如果关联里面存在循环关联,在内部对象的关联属性里面会被置空. 如:

```
[JAVA]
peter.setBestFriend(amy);
System.out.println(Json.toJson(peter));
```

这个时候输出:

```
[CODE]
{
name :"Amy",
age :21,
sex :"Female",
birthday :"2009-04-11 21:28:59",
bestFriend :{
    name :"Peter",
    age :22,
    sex :"Male",
    bestFriend :null
    }
}
```

8.5.5. 更进一步

还可以定制JsonFormat的属性来控制输出内容. 如是否忽略null字段, 是否压缩等等.

8.5.5.1. JsonFormat默认提供了3种初始对象:

* compact() : 压缩并忽略null字段
* nice() : 不压缩;字段名不加引号;忽略null字段
* full() : 不压缩;字段名加引号;不忽略null字段

8.5.6. 对Json的过滤.

此功能已由Mapl.includeFilter(), Mapl.excludeFilter()代替.Mapl文档

8.5.7. JSON 结构转换(其实是Map,List结构, 不一定非要JSON转换的)

此功能已由Mapl.convert()代替.Mapl文档

8.5.8. Map, List结构访问.

此功能已由Mapl.cell代替Mapl文档

9. 表达式引擎

9.1. 表达式引擎简介

9.1.1. 为什么需要 EL 表达式引擎

你，对，就是你，正在看这篇文章的人，我虽然不认识你，但是 我可以负责任的说，如果你看到这个标题就心里在悄悄的呼喊： **“靠，他们连这个都有！我省事了，哇哈哈哈和”** 。那么，你绝对属于百分之一的特例。 就是说，绝大多数人的绝大多数项目，是不需要一个嵌入式的表达式引擎的。因此，提供这个功能的目的就是：

满足一小撮人的一小撮要求

但是，"一小撮人"的"一小撮要求"有很多，作为一个小众类库，为什么单单打算支持这个特性呢？下面是我的理由：

* 这个功能是其它模块功能的基础，我们需要它
* 可能因此吸引其他的开发者对 Nutz 的兴趣
    > 需要嵌入式表达式引擎的人是 Java 开发者的少数人，但是这些人也应该比 Nutz 的用户要多
    > 这些人基本上编程水平要强一些
* 其他的提交者对增加这个特性没有特别强烈的反对

9.1.2. 近一步介绍表达式引擎

那么它怎么使用呢？

是的，我想这可能会是你脑海里闪出的第一个问题。并且，我想你真正想问的是:“它好用吗？”

如果你脑海里第一个问题不是这个，而是：“表达式引擎是神马东东？” 那么建议你不用阅读本文了，反正你也用不着。等你需要的时候，再读也不迟，反正这篇文章又不长。

而关于 **"好用"**，还有下面这三层含义：

9.1.2.1. 它容易使用吗？

```
[Java]
System.out.println(El.eval("3+4*5").equals(23)); // 将打印 true，够简单吧
```

表达式接受的是字符串输入，输出则是一个Object对象，而Object对象本身是根据计算结果进行进行了自动封装的。

9.1.2.2. 它功能强大吗？

虽然在 #一些表达式的例子 这一节我有更详细的例子，但是这里我必须要先概括几点：

它支持变量，比如

```
[Java]
Context context = Lang.context();
context.set("a", 10);
System.out.println(El.eval(context, "a*10"));  // 将打印 100
```

通过 Context 接口，你可以为你的表达式随意设置变量的值。它支持如下类型的 Java 数据

* 整型 - int 或 Integer
* 浮点 - float 或 Float
* 长整 - long 或 Long
* 布尔 - boolean 或 Boolean
* 字符串 - String
* 数组 - T[]
* 列表 - Lst<Ti>
* 集合 - Collection<T>
* Map - Map<String,?>
* 普通 Java 对象

基本上，有了这些，你可以为所欲为了吧。

9.1.2.3. 它速度怎么样？

我觉得它速度不怎么样。它的工作的原理是这样的，每次解析都经过如果下三步

1. 解析成后缀表达式形式的一个队列
2. 将后缀表达式解析成一棵运算树.
3. 对运算树的根结点进行运算.

当然我也提供了一个提升效率的手段，因为如果每次计算都经过这三个步骤当然慢，所以我们可以对它先预编译：

```
[Java]
El exp = new El("a*10");  // 预编译结果为一个 El 对象
```

```
Context context = Lang.context();
context.set("a", 10);
System.out.println(exp.eval(context));  // 将打印 100
```

EI在实例化时就会对表达式进行预编译，会直接编译成运算树，当调用eval方法时，就不用再耗时的编译动作了.

它的 eval 函数是线程安全的，只要在多个线程内给它不同的 context 就是了。当然，你也可以在多个线程间共享同一个 Context，那运行起来一定很有趣，不是吗？

9.1.3. 支持什么样的操作符

我想但凡有机会和兴趣读到这篇文字的同学，一定是编程老手，即使是自称**小白**的的同学们，你们对一个编程语言应该支持的操作符基本都差不多熟的不行，所以，我就不在这里唠叨操作符的具体细节了，我只给一个列表，告诉你我现在支持什么操作符。

另外，再加上一句：

只要支持的操作符，我会让它的优先级以及行为会和 Java 的表达式一致。如果你发现不一致别犹豫，给我报 Issue 吧。

符号	权重	解释
()	100	括号，优先计算
,	0	逗号，主要是方法参数
.	1	访问对象的属性，或者Map的值，或者方法调用，或者自定义函数调用（需要结合后面是否有括号）
['abc']	1	Java 对象 Map按键值获得值
[3]	1	数字，列表，或者集合的下标访问符号
*	3	乘
/	3	整除
%	3	取模

+	4	加
-	4	减
-	2	负
>=	6	大于等于
<=	5	小于等于
==	7	等于
!=	6	不等于
!	7	非
>	6	大于
<	6	小于
&&	11	逻辑与
||	12	逻辑或
?:	13	三元运算
&	8	位运算，与
~	2	位运算，非
|	10	位运算，或
^	9	位运算，异或
<<	5	位运算，左移
>>	5	位运算，右移
>>>	5	位运算，无符号右移
&	8	位运算，与

当然，同任何编程语言一样，表达式也支持 左括号 (以及 右括号)，来控制表达式的的计算优先级别

9.1.4. 自定义函数

好吧, 你肯定要说上面的功能简直弱爆了, 就一点简单的加加减减有什么好稀奇的, 再强点的需求就没办法满足了.确实是这样, 所以, 我们在 EL 里面添加了自定义函数, 嘿嘿, 这回强了吧. 言归正传,下面详细的说说它的使用, 以及怎么自定义.

现有的自定义函数:

名称	参数	解释	例子
max	任意个Number型	取出参数中最大值	max(1, 2, 3, 4)=>4
min	任意个Number型	取出参数中最小值	min(1, 2, 3, 4)=>1
trim	一个String	去掉字符串两边的空格	trim(" 1 ")=>"1"

使用很简单吧.

当EL里面的功能无法满足你的需求时, 你可以自定义一些功能来实现, 怎么自定义呢?

```
[CODE]
# 创建一个类, 使它实现 org.nutz.el.opt.RunMethod, org.nutz.plugin.Plugin 这两个接口.
# 在配置文件中添加一个配置项:
{
"EL": {"custom":[...函数列表...]}
}
# 如果你只使用了EL这一个模块, 没有其它模块, 并且在其它地方都没有加载过配置, 请使用下面的语句加载配置:
NutConf.load("配置文件");
//加载自定义函数配置, 必须!因为默认情况下, CustomMake只在static{}块中加载一次配置, 新添加的配置在static之后的话, 那么新添加的函数配置将不会生效!
CustomMake.init();
```

完成这两步后, 你就定义了你自己的函数. 看个trim的例子:

```
[JAVA]
public class Trim implements RunMethod, Plugin{
    //处理方法, fetchParam为函数的参数. 它会将EL中函数括号后的所有内容传递过来
    public Object run(List<Object> fetchParam) {
```

```
        if(fetchParam.size() <= 0){
        throw new ElException("trim方法参数错误");
        }
        String obj = (String) fetchParam.get(0);
        return obj.trim();
    }
    //是否可以执行
    public boolean canWork() {
        return true;
    }
    //在EL表达式中的函数名
    public String fetchSelf() {
        return "trim";
    }
}
```

9.1.5. 还有些什么功能?

其实我们的 EL 很强悍的, 有好多功能, 好多使用技巧等待着你的发现. 现在简单的罗列一下, EL中的一些特性吧.

9.1.5.1. 忠于 Java

Nutz中的 EL 完全忠实于 JAVA 基本运算规则, 并没有做一些扩展, 比如最常见的, 数据类型转换,在 JAVA 中进行数值运算的过程中, 是根据运算符两边的类型而最终决定运算结果的类型的, 比如:

```
[CODE]
7/3  // 将返回int型
而
(1.0 * 7)/3    // 返回double
(1.0f * 7)/3   // 则返回float
```

为什么后面两个返回类型不一样呢? 因为在 JAVA 中默认浮点类型是 double 哦.

基于这个原因, 在 EL 中同样保留了这些特点. 所以, 亲, 要是没返回 double 别骂我们哦~~~

9.1.5.2. 支持对象方法调用

亲, EL 支持对象, 支持对象方法调用哦~~~

这有什么好大不了的? 给你看个例子, 嘿嘿:

```
[JAVA]
Context context = Lang.context();
context.set("a", new BigDecimal("7"));
context.set("b", new BigDecimal("3"));
assertEquals(10, El.eval(context, "a.add(b).intValue()"));
```

* 看到什么没? 对了, BigDecimal 你完全可以丢各种各样的对象到 context 里面去, 然后在 EL 中调用它们的方法.
* 然后你就进行各种各样的虐待, 皮鞭, 蜡烛, 想来啥来啥...额...太邪恶了...
* 同样, EL 是使用反射的, 所以会存在一些底层异常的问题.

9.1.5.3. 支持静态方法调用

没看错, EL支持静态方法调用, 不过, 在调用之前你需要设置一个变量才行...

```
[java]
Context context = Lang.context();
context.set("strings", Strings.class);
assertEquals("nutz", El.eval(context, "strings.trim(\"    nutz   \")"));
```

9.1.6. 一些表达式的例子

9.1.6.1. 普通运算

```
[JAVA]
System.out.println(El.eval("3+2*5"));
// 输出为  13
```

9.1.6.2. 字符串操作

```
[JAVA]
System.out.println(El.eval("trim(\"  abc  \")"));
// 输出为  abc
```

9.1.6.3. Java 对象属性访问调用

```
[JAVA]
Context context = Lang.content();
Pet pet = new Pet();
pet.setName("GFW");
context.set("pet",pet);
System.out.println(El.eval(context,"pet.name"));
// 输出为  GFW
```

9.1.6.4. 函数调用

```
[JAVA]
Context context = Lang.content();
Pet pet = new Pet();
context.set("pet",pet);
El.eval(context, "pet.setName('XiaoBai')");
System.out.println(El.eval(context,"pet.getName()"));
// 输出为  XiaoBai
```

9.1.6.5. 数组访问

```
[JAVA]
Context context = Lang.content();
context.set("x",Lang.array("A", "B", "C"));
System.out.println(El.eval(context,"x[0].toLowerCase()"));
// 输出为  a
```

9.1.6.6. 列表访问

```
[JAVA]
Context context = Lang.content();
context.set("x",Lang.list("A", "B", "C"));
System.out.println(El.eval(context,"x[0].toLowerCase()"));
// 输出为  a
```

9.1.6.7. Map 访问

```
[JAVA]
Context context = Lang.content();
context.set("map",Lang.map("{x:10, y:5}"));
System.out.println(El.eval(context,"map['x'] * map['y']"));
// 输出为  50
```

9.1.6.8. 判断

```
[JAVA]
Context context = Lang.content();
context.set("a",5);
System.out.println(El.eval(context,"a>10"));
// 输出为  false
context.set("a",20);
System.out.println(El.eval(context,"a>10"));
// 输出为  true
```

10. maplist结构

10.1. Mapl 结构

10.1.1. 为什么需要 Mapl 结构

一直以来都没有刻意的去思考说需要 Mapl 结构这样的东西. 所谓 Mapl 结构就是 Map-List 结构，我原来就叫 maplist 后来被灰太郎说太长，就改成 "mapl" 了

最初都是 wendal 在重写JSON的时候, 将JSON理解成了Mapl结构.然后突然在某一天发现, 咦, Mapl对象还可以这么用, 那么用, 慢慢的, 以Mapl 基础的小功能点越来越多, 已经不能够完全的与JSON概念协调, 所以才有了这样一个中间结构.

恩, 上面那段是给灰太狼, wendal看的. 我想你肯定看得头有点大, 要不我再说直白点, Mapl结构就是为Json服务的, 为什么呢? MD, Json.fromJson()忒难用了, 要是只给它一个Reader, 而不给Type, 那它给我返回的就是 Mapl结构 , 苍天呀, 大地呀, 烦都烦死了.

我要取其中的某个值, 我得遍历N层的Map, List, 每次写这种东西的时候, 我都想哭, 所以干脆对它封装吧, 越封越多, 然后就有了这玩意...

当然, 它也与EL一样 **满足一小撮人的一小撮要求**

提醒: Mapl这个名词,是MapList的缩写

10.1.2. 什么是 Mapl 结构?

一种以 Map, List 接口对象所组织起来的结构体系. 类似于JSON结构便于JAVA在内存中处理的结构. 主要提供键值对, 与列表的有机组合, 因这种结构只由Map, List组成, 因些称其为Mapl结构.

```
[java]
Map a = new HashMap();
a.put("name","a");
Map b = new HashMap();
b.put("name","b");
Map c = new HashMap();
c.put("name","c");
List list = new ArrayList();
list.add(a);
list.add(b);
list.add(c);
Map d = new HashMap();
d.put("items", list);
```

通过上面的代码我们就组织了一个Mapl结构, 它等效于以下的JSON文档:

```
[json]
{"items":[{"name":"a"},{"name":"b"},{"name":"c"}]}
```

当然, Mapl.仅可以用来表示JSON, 也可以用来表示JAVA对象的结构, 然后有了Mapl., 你会发现, 做转换, 合并, 都是非常轻松滴~~~

具体规则:

* 对象以Map存储, key为属性名, value为属性值
* 数组以List存储
* Map直接存储为Map
* List直接存储为List
* 只要不是List, Map存储的, 都认为是可以直接写入对象的

10.1.3. Mapl.转 对象

也就是根据Mapl.及Type信息转换成一个Type的实体对象了啦, 直接看例子:

```
[java]
class A{
    String name;
    Integer id;
}
class B{
    String name;
    List<A> as;
}
class C{
    public static void main(String args[]){
        String json = "{'name':'b',
'as':[{'name':'nutz','id':1},{'name':'jk','id':2}]}";
        //这样得到的就是Mapl结构的数据了.
        Object obj = Json.fromJson(json);
        B b = Mapl.maplistToObj(obj, B.class);
    }
}
```

通过上面的 Mapl.maplistToObj() 方法就可以将一个Mapl.象转换成B类型的实体对象. 我偷偷的告诉你哦, JSON里面也是这样搞的哦 ~~~先将JSON字符串转换成Mapl结构后再调用Mapl.maplistToObj()方法转换成对应的类型.

10.1.4. 对象转Maplist

除了通过JSON转换成Mapl结构以外, 还可以直接使用对象来转换成maplist结构

```
[java]
A a = new A();
a.name="a"
B b = new B();
b.name = "b";
b.as = new ArrayList();
b.as.add(a);
Mapl.toMapl.b);
```

结果:

```
[CODE]
{name:"b", as:[{name:"a", id:null}]}
```

通过toMapl.可以进行这种简单的转换

10.1.5. 访问 Maplist

就如我最开始说的那样, Json.fromJson 很难用, 就是因为在读取Mapl结构的数据时非常的繁杂, 经常需要很多层的类型转换.

```
[java]
String json = "{'name':'b', 'as':[{'name':'nutz','id':1},{'name':'jk','id':2}]}";
//这样得到的就是Mapl结构的数据了.
Object obj = Json.fromJson(json);
```

上面的obj, 如果我想取as索引为1的name的值, 怎么办? 只能这样:

```java
Map map = (Map) obj;
List list = map.get("as");
Map item = list.get(1);
String name = item.get("name");
```

亲, 看到没, 看到没~~~妈哦, 还好这里只有几层, 要是再多几次这样的, 我一定会疯的, 你肯定也跟我一样吧. 所以咯, 让我们解脱吧~~~

```java
String name = (String) Mapl.cell(obj, "as[1].name");
```

完了? 这就样? 是的, 完了, 就这样, 一句话搞定. so easy~~~

最后说说关于里面path的规则:

* map的值访问直接使用 '.', 如: abc.name
* list的访问使用 "名称[索引]", 如: as[1]. 当然要是不想写[]也可以使用 as.1.name的形式.
* 顶层为list时, 使用 "[索引].其它", 如: [1].name
* 如果想得到一个List, 而不是它某个值, 则可以使用 "名称" 不加 "[索引]". 如: as
* 如果List后加了""中间却没有索引, 则默认访问第一个元素, 如: user 等效 user1

10.1.6. maplist 合并

哇咔咔, 一个神器来鸟. 为嘛我要说它是神器呢, 看名字就知道了嘿, 当然, 这个只是一小撮的一小撮的一小撮人会觉得是神器...额...好吧, 它只是一个没啥大用的一个伪神器...

顾名思义, maplist 合并, 就是将多个maplist合并在一起, 组成一个新的 maplist .

```java
String json1 = "{'name':'nutz'}";
String json2 = "{'age':12}"
Object obj1 = Json.fromJson(json1);
Object obj2 = Json.fromJson(json2);
Object obj3 = Mapl.merge(obj1, obj2);
```

最终obj3的输出将是:

```
[CODE]
{"name":"nutz", 'age':12}
```

规则:

* 普通对象, 保存为List, 但是要去除重复.
* 合并map, 如果key值相同, 那么后一个值覆盖前面的值, 注意, 对值将会进行递归合并
* list不做递归合并, 只做简单的合并, 清除重复

10.1.7. maplist 过滤

这玩意有什么用呢, 用来剔除/筛选 maplist 中的值, 使maplist更加满足我们的需求. 还是用例子来说明吧.

```
[json]
String json = "{name:'nutz', age:12, address:[{area:1,name:'abc'},
{area:2,name:'123'}]}";
Object obj = Json.fromJson(json);
List<String> list = new ArrayList<String>();
list.add("age");
list.add("address[].area");
Object newobj = Mapl.excludeFilter(obj, list);
```

结果:

```
[CODE]
{name:'nutz', address:[{name:"abc"}, {name:"123"}]}
```

可以发现, 通过给定的过滤列表, 可以将原始的maplist结构给过滤掉满足条件的内容, 当然, 除了排除, 还有包含.

```

```

```
[java]
String json = "{name:'nutz', age:12, address:[{area:1,name:'abc'},
{area:2,name:'123'}]}";
Object obj = Json.fromJson(json);
List<String> list = new ArrayList<String>();
list.add("age");
list.add("address[].area");
Object newobj = Mapl.includeFilter(obj, list);
```

结果:

```
[CODE]
{age:12, address:[{area:1},{area:2}]}
```

excludeFilter与includeFilter是一组完全相反的功能.

path规则:

* map以 "key." 间隔
* list以"key[]."间隔, 即多一个[], 注意其中没有索引哦.

10.1.8. maplist 结构转换

好吧, 我觉得这个才是神器~~~啦~啦~~啦~~~啦~~~~完全是神一样的存在.

有没有使用过其它公司的API? 有吧, 其它公司都返回些什么格式? 它的格式与你程序的格式一样吗? 或许有, 但大部分是不一样的, 对吧. 既然这样, 那结构转换是肯定的了.

```
[java]
String json = "[{'name':'jk', 'age':12},{'name':'nutz', 'age':5}]";
String model = "[{'name':['user[].姓名', 'people[].name'],
'age':['user[].年龄', 'people[].age']}]";
String dest = "{\"people\":[{\"age\":12,\"name\":\"jk\"}, "
            + "{\"age\":5,\"name\":\"nutz\"}],"
            + "\"user\":[{\"姓名\":\"jk\",\"年龄\":12}, "
            + "{\"姓名\":\"nutz\",\"年龄\":5}]}";
Object obj = Mapl.convert(Json.fromJson(new StringReader(json)),
```

```
new StringReader(model));
assertEquals(dest, Json.toJson(obj, new JsonFormat()));
```

结果:

```
[json]
{
    "people":[
        {"age":12,"name":"jk"},
        {"age":5,"name":"nutz"}
    ],
    "user":[
        {"姓名":"jk","年龄":12},
        {"姓名":"nutz","年龄":5}
    ]
}";
```

通过一个简单的操作, 我们就将一个maplist结构转换成了一个完全不一样的结构, 是不是很神奇?

什么是 maplist 结构转换呢? 就是将一种MapList结构转换成另外一种MapList结构.例:

```
[json]
{
    "age":"123",
    "name":"juqkai"
}
```

转换成:

```
[json]
{
    "年龄":"123",
    "姓名":"juqkai"
}
```

要进行这样的转换需要预先配置一个对应关系的配置, 具体的配置关系说明如下:

* 使用原MapList一样的结构
* 有数组的, 只写第一个元素的结构
* 原结构中的值, 以字符串或字符串数组做为目标结构的对应关系
* 对应关系可以为数组
* 有数组的, 目标结构以key.abc来代替数组
* 原结构数组层次强制限定一致, 目标结构中'[]'的索引按原结构中出现先后顺序进行匹配.
* 如果原结果不存在, 那默认为0
* 未在模板中申明的不做转换

例:

例1:

```
[json]
 {
    "age":"user.年龄",
    "name":["user.name", "user.姓名"]
 }
```

例2

```
[json]
(原json:[{"name":"nutz"},{"name":"juqkai"}]):
[{
    "name":"[].姓名"
}]
```

例3:

```
[json]
{
    users:[
       {
```

```
            "name":["people[].name", "users[].name"],
            "age":"users[].name"
        }
    ]
}
```

10.1.9. MapList的增删改

只有访问, 肯定是不够的, 难免会添加, 删除, 修改某个结点. 所以, 特意的为您添加了这些功能.很简单的, 其实就三个接口而已.添加: Mapl.put, Mapl.del, Mapl.update. 具体的使用方法, 你看看注释咯, 简单得很.

除了上面的几个一次性的接口外. MapList还包含一个 MaplRebuild 类, 从名字就可以很容易知道它是干嘛的. 没错, 就是Maplist重建. 你可以根据已有的Maplist来构建它, 也可以全新的构建它, 然后你就可以对它进行添加新的结点, 修改某个结点, 或者删除某个结点. 如此反复.

下面看看Mapl.put的实现你就知道怎么用了:

```
[java]
    /**
     * 添加新的结点
     * @param obj 原始的MapList
     * @param path 路径
     * @param val 值
     */
    public static void put(Object obj, String path, Object val) {
        Object mapList = Mapl.toMaplist(val);
        MaplRebuild rebuild = new MaplRebuild(obj);
        rebuild.put(path, mapList);
    }
```

11. 甜 Java

11.1. Java 的函数糖

11.1.1. 什么是函数糖？

Java 的语法比 C/C++ 友好很多， 因为它设计之初，就是为了考虑到程序员的使用是否舒适。当然很多事情愿望是美好的，现实是残酷的。Java 语言本身的语法仍然不可避免的带有着 10年前那种的僵硬和严谨。这里是一些小小的尝试，你会发现，大多数情况，通过一些静态函数，一行代码完全可以做很多事情， 而且比“甜甜”的 Ruby 也差不了太多。

你可以查看 org.nutz.lang 下的源代码。为了便于你学习，我将里面部分最常用的用法列在文档你，便于快速学习查看。

我希望在 80% 以上的情况下，这些函数能让你有效的缩短你 代码的体积，并且增加代码的可读性。

11.1.2. 异常

11.1.2.1. 创建异常

* 根据格式化字符串，生成运行时异常

```
[CODE]
throw Lang.makeThrow("Error for %d [%s]", obj.getId(),
obj.getName());
```

* 根据格式化字符串，生成一个指定的异常。

```
[CODE]
throw Lang.makeThrow(SQLException.class, "Error for %d [%s]",
obj.getId(), obj.getName());
```

* 未实现的运行时异常

```
[CODE]
throw Lang.noImplement();
```

11.1.2.2. 包裹异常

* 用运行时异常包裹抛出对象，如果抛出对象本身就是运行时异常，则直接返回。

```
```

```
[CODE]
throw Lang.wrapThrow(e);
```

* 用一个指定可抛出类型来包裹一个抛出对象。这个指定的可抛出类型需要有一个构造函数接受 Throwable 类型的对象

```
[CODE]
throw Lang.wrapThrow(e, SQLException.class);
```

* 将一个或者多个异常组合抛出

```
[CODE]
throw Lang.comboThrow(e1,e2,e3);
```

* 将抛出对象包裹成运行时异常，并增加自己的描述

```
[CODE]
throw Lang.wrapThrow(e, "Error for %d [%s]", obj.getId(),
obj.getName());
```

11.1.3. 对象

* 比较对象: 支持数组，集合，和容器

```
[CODE]
return Lang.equals(map1, map2);
```

* 显示对象

```
[CODE]
return Dumps.obj(pojo);
```

* 是否包括

```
[CODE]
return Lang.contains(myArray, ele);
```

* 遍历

```
[CODE]
Lang.each(obj, new Each<Object>(){
    public void invoke(int i, T ele, int length){
        obj 可以是集合，数组，Map，普通对象.
        普通对象的话，则会传给本匿名类的 ele
    }
});
```

11.1.4. 容器转换

* 数组 to Map

```
[CODE]
Pet[] pets = new Pet[3];
... // 为数组赋值
// 将把 Pet 对象的 name 字段的值作为 Key， 创建一个 Map
Map<String,Pet> petMap = Lang.array2map(HashMap.class,
pets, "name");
```

* 数组 to 数组

```
[CODE]
Pet[] pets = new Pet[3];
String[] ss = Lang.array2array(pets, String.class);
```

* 集合 to 数组

```
[CODE]
List<Pet> pets = new ArrayList<Pet>();
... // 为列表赋值
Pet[] petArray = Lang.collection2array(pets);
```

* 集合 to 列表

```
[CODE]
Queue<Pet> pets = new ArrayDeque<Pet>();
... // 为队列赋值
```

```
List<Pet> list = Lang.collection2list(pets);
```

* 集合 to Map

```
[CODE]
Queue<Pet> pets = new ArrayDeque<Pet>();
... // 为队列赋值
// 将把 Pet 对象的 name 字段的值作为 Key， 创建一个 Map
Map<String,Pet> petMap =
Lang.collection2map(HashMap.class, pets, "name");
```

* Map to 对象

```
[CODE]
return Lang.map2Object(map,Pet.class);
```

11.1.5. 对象模拟

* 生成数组

```
[CODE]
String[] ss = Lang.array("A","B","C");
```

* 从字符串生成 Reader

```
[CODE]
Reader reader = Lang.inr("ABCDEF");
```

* 从字符串生成 InputStream

```
[CODE]
InputStream ins = Lang.ins("ABCDEF");
```

* 从字符串生成 Map

```
[CODE]
Map<String,Object> map = Lang.map("{a:10, b:'ABC', c:true}");
```

* 从字符串生成 List

```
[CODE]
List<Object> list =  Lang.list("[true, 23, 'ABC']");
```

11.1.6. XML

* 创建 DocumentBuilder

```
[CODE]
return Xmls.xmls();
```

* 解析 XML 文档

```
[CODE]
Document xmlDoc = Xmls.xml(Files.findFile("~/my.xml"));
```

* 读取某个子元素内容

```
[CODE]
Element ele = ...   // 假设你取到了一个 Element
String txt = Xml.get(ele, "abc");  // 获取 ele 下的 <abc> 的文本内容并去掉前后空白
```

11.1.7. 字符串操作

11.1.7.1. 判断

* 是否为空串

```
[CODE]
assertTrue(Strings.isEmpty(""));
```

* 是否为空白

```
[CODE]
assertTrue(Strings.isEmpty("\t \r\n"));
```

* 是否被特定字符包裹

```
[CODE]
assertTrue(Strings.isQuoteBy("[ABC]", '[', ']'));
// 忽略空白
assertTrue(Strings.isQuoteByIgnoreBlank("  \t [ABC]\r\n  ", '[',
']'));
```

* 集合中的最长串

```
[CODE]
assertEquals(3, Strings.maxLength((String[]){"A","ABC","BC"}));
// 集合
assertEquals(3, Strings.maxLength(Lang.list("['A','ABC','BC']")));
```

* 如果字符为null，返回特定默认值

```
[CODE]
assertEquals("abc", Strings.sNull(null, "abc"));
```

* 如果字符为空白，返回特定默认值

```
[CODE]
assertEquals("abc", Strings.sBlank(" \n\t", "abc"));
```

11.1.7.2. 修改

* 首字母大写

```
[CODE]
assertEquals("Abc", Strings.capitalize("abc"));
```

* 首字母小写

```
[CODE]
assertEquals("aBC", Strings.lowerFirst("ABC"));
```

* 切除首位空白

```
[CODE]
assertEquals("ABC", Strings.trim("\t   ABC \r\n   "));
```

* 左填充 (居右)，当无法填充时不会按照指定的长度截断原字符串

```
[CODE]
assertEquals("00FE", Strings.alignRight("FE", 4, '0'));
assertEquals("FFFF", Strings.alignRight("FFFF", 2, '0'));
```

* 右填充 (居左)，当无法填充时不会按照指定的长度截断原字符串

```
[CODE]
assertEquals("FE00", Strings.alignLeft("FE", 4, '0'));
assertEquals("FFFF", Strings.alignLeft("FFFF", 2, '0'));
```

* 固定长度左填充 (居右)，当无法填充时将按照指定的长度截断原字符串

```
[CODE]
assertEquals("00FE", Strings.cutRight("FE", 4, '0'));
assertEquals("FF", Strings.cutRight("FFFF", 2, '0'));
```

11.1.7.3. 转换

* 二进制形式字符串

```
[CODE]
assertEquals("0110", Strings.fillBinary(6, 4));
```

* 十六进制形式字符串

```
[CODE]
assertEquals("00FF", Strings.fillHex(255, 4));
```

* 拆分数组 （忽略空白元素）

```
[CODE]
assertEquals(3, Strings.splitIgnoreBlank("A,B,C"));
```

```
assertEquals(3, Strings.splitIgnoreBlank(",A,B,,C,"));
```

11.1.7.4. 创建重复

```
[CODE]
// 重复字符
assertEquals("---", Strings.dup('-', 3));
// 重复字串
assertEquals("ABCABCABC", Strings.dup("ABC", 3));
```

11.1.8. 文件操作

11.1.8.1. 查找

* 从 CLASSPATH 下或从指定的本机器路径下寻找一个文件

```
[CODE]
return Files.findFile("org/nutz/lang/Lang.class");
```

* 根据正则式，从压缩文件中获取文件

```
[CODE]
return Files.findEntryInZip(new ZipFile("D:/nutz.jar"),
"org/nutz/lang/Lang.class");
```

* 搜索一个类同一包下面的所有类

```
[CODE]
return Resources.scanClass(Lang.class);
```

11.1.8.2. 创建-删除-拷贝

* 创建新文件，如果父目录不存在，也一并创建

```
[CODE]
Files.createNewFile(new File("D:/demo/abc.txt"));
```

* 创建新目录，如果父目录不存在

```
[CODE]
Files.makeDir(new File("D:/demo/abc"));
```

* 强行删除一个目录，包括这个目录下所有的子目录和文件

```
[CODE]
Files.deleteDir(new File("D:/demo/abc"));
```

* 清除一个目录里所有的内容

```
[CODE]
Files.clearDir(new File("D:/demo/abc"));
```

* 将一个目录下的特殊名称的目录彻底删除，比如 '.svn' 或者 '.cvs'

```
[CODE]
Files.cleanAllFolderInSubFolderes(new File("D:/demo"), ".svn");
```

* 文件拷贝

```
[CODE]
Files.copyFile(new File("D:/a/b/c.txt"), new File("E:/a/b/e.txt"));
```

* 目录拷贝

```
[CODE]
Files.copyDir(new File("D:/a/b/c"), new File("E:/a/b/e"));
```

11.1.8.3. 读取-写入

* 读取文件全部内容 - UTF-8

```
[CODE]
String txt = Files.read("D:/abc.txt");
或者
```

```
String txt = Lang.readAll(Streams.fileInr("D:/abc.txt"));
```

* 重写文件全部内容 - UTF-8

```
[CODE]
// 如果 D:/abc.txt 不存在，则创建它
String txt = Files.write("D:/abc.txt", "some text");
或者
// 如果 D:/abc.txt 不存在，则什么都不做
String txt = Lang.writeAll(Streams.fileOutw("D:/abc.txt"),
content.toString());
```

* 获取 Reader - UTF-8

```
[CODE]
Reander reader =  Streams.fileInr("D:/abc.txt");
或者
Reander reader =  Streams.fileInr(new File("D:/abc.txt"));
```

* 获取 Writer - UTF-8

```
[CODE]
Writer writer = Streams.fileOutw("D:/abc.txt");
或者
Writer writer = Streams.fileOutw(new File("D:/abc.txt"));
```

* 获取 InputStream

```
[CODE]
InputStream ins =  Streams.fileIn("D:/abc.txt");
或者
InputStream ins =  Streams.fileIn(new File("D:/abc.txt"));
```

* 获取 OutputStream

```
[CODE]
OutputStream ops =  Streams.fileOut("D:/abc.txt");
或者
OutputStream ops =  Streams.fileOut(new File("D:/abc.txt"));
```

11.1.8.4. 直接在磁盘修改文件属性

* 将文件移动到新的位置

```
[CODE]
Files.move(new File("D:/demo/abc.txt"), new
File("D:/demo/def.txt"));
```

* 将文件改名

```
[CODE]
Files.rename(new File("D:/demo/abc.txt"), "def.txt");
```

11.1.8.5. 创建文件对象

* 将文件后缀改名，从而生成一个新的文件对象。但是并不在磁盘上创建它

```
[CODE]
return Files.renameSuffix(new File("D:/demo/abc.txt"), ".java");
```

* 获取文件主名

```
[CODE]
assertEquals("abc", Files.getMajorName(new
File("D:/demo/abc.txt")));
```

* 获取文件后缀名

```
[CODE]
assertEquals("txt", Files.getSuffixName(new
File("D:/demo/abc.txt")));
```

11.1.9. 秒表

```
[CODE]
Stopwatch sw = Stopwatch.begin();
...
```

```
这里是你的运行代码
...
sw.stop();
System.out.println(sw.getDuration());
```

11.1.10. 随机数据

* 随机字符串

```
[CODE]
// 生成 100 个长度不超过20的字符串
StringGenerator sg = new StringGenerator(20);
for(int i=0;i<100;i++)
    System.out.println(sg.next());
```

* 闭区间随机数

```
[CODE]
return  R.random(3,5)
```

* 随机枚举值

```
[CODE]
return (new EnumRandom<MyEnums>(){}).next();
```

* 数组不重复随机对象

```
[CODE]
Random<String> r = new
ArrayRandom<String>(Lang.array("A", "B", "C"));
int i = 0;
while (null != r.next()) {
    i++;
}
assertEquals(3, i);
```

* 数组无穷随机对象

```
[CODE]
```

```
Random<String> r = new
RecurArrayRandom<String>(Lang.array("A", "B", "C"));
for(int i=0; i<100; i++)
    System.out.println(r.next());
```

11.1.11. 其他

* 打印 java.util.regex.Matcher 的详细信息

```
[CODE]
System.out.println(Dumps.matcher(matcher));
```

* 解析 Boolean

```
[CODE]
assertTrue(Lang.parseBoolean("on"));
assertTrue(Lang.parseBoolean("1"));
assertTrue(Lang.parseBoolean("yes"));
```

11.2. 让 Java 类型不那么强

11.2.1. 类型强好，还是类型弱好？

强类型好，尤其是对编译器友好。而且 IDE 支持的比较好。 弱类型好，因为程序员写程序的时候，不需要考虑类型，只需要考虑逻辑。

其实，即使是 Javascript，这样的弱类型语言，也不能很好的将类型互相转换的很好。

我希望在我编程的时候，大多数时间都强类型。我感觉很安全，但是当我想模糊类型的时候，我能有一个工具，帮我把任何类型转成我希望的另外的类型。当然，我自己定义的对象，我并不介意写一个转换过程，但是请只让我写一次。

所以，这就是 Nutz.Castors

11.2.2. Hello Castors

举个很简单的例子：

```
[CODE]
System.out.println(Castors.me().castTo("563", int.class));
```

或者

```
[CODE]
System.out.println(Castors.me().castTo("zozohtnt@gmail.com",
org.nutz.lang.meta.Email.class));
```

你还可以：

```
[CODE]
Calendar c = Castors.me().castTo("2009-11-12 15:23:12",
Calendar.class)
```

你甚至还能：

```
[CODE]
Timestamp t = Castors.me().castTo(Calendar.getInstance(),
Timestamp.class);
```

它完全能支持容器：

```
[CODE]
List<Pet> petList = ...;
Pet[] pets = Castors.me().castTo(petList, Pet[].class);
```

11.2.3. Castors 的原理

请查看 org.nutz.castor.castor 包，这里面的一个个的实现类就是 Nutz 为你内置的类型转换器。

每当你要转换一个对象，Nutz.castors 都会看看能不能直接转换，如果不能，它就在这个包里找找，看看有没有合适的转换器。Object2Object 转换器是最终极的转换。它尝试着在目标对象里寻找构造函数，或者静态工厂方法，如果找到一个方法有且只有一个参数是源对象。那么它就会

高兴的构造目标对象，否则则抛出异常。

11.2.4. 类型转换配置

比如日期和时间，你需要定制自己的日期类型，请参看默认的 CastorSetting 的源代码：

```
[CODE]
public class CastorSetting {
    public static void setup(DateTimeCastor<?, ?> c) {
        c.setDateFormat(new SimpleDateFormat("yyyy-MM-dd"));
        c.setTimeFormat(new SimpleDateFormat("HH:mm:ss"));
        c.setDateTimeFormat(new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss"));
    }
}
```

它有一个函数，是设置 DataTimeCastor 的，那么 Nutz.Castor 会在构造这种类型的转换器前，调用一下这个函数。这个函数会初始化一下相应的 DataTimeCastor。

你可以通过：

```
[CODE]
Object yourSetting = ...;
Castors.setSetting(yourSetting);
```

替换这个默认实现。

你可以通过：

```
[CODE]
Castors.resetSetting();
```

来把转换配置，恢复到默认值

11.2.5. 添加自己的 Castor

你的项目可能会有很多自定义的 POJO，而如果你想支持它们之间流畅的转换，你可能需要写一组 A2B 以及 B2A 的转换器。这些转换器，你可以分别放在不同的包下，为了能让 Nutz.Castors 找到你的转换器，你需要在项目启动时：

```
[CODE]
Castors.addCastorPaths(A2B.class, E2D.class);
```

举个详细的例子，比如你有两组转换器：

```
[CODE]
com.you.red
    A2B
    B2A
    A2C
    C2A
    B2C
    C2B
com.you.blue
    X2Y
    Y2X
    X2Z
    Z2X
    Y2Z
    Z2Y
```

为了能让 Nutz.Castors 找到这两组转换器，你需要在项目启动时加载一下：

```
[CODE]
Castors.addCastorPaths(A2B.class, X2Y.class);
```

这样， A2B 和 X2Y 这两包下所有的转换器，都会被加载，并且他们的优先级按加载顺序，后加载的比先加载的要高。 Nutz 提供的默认的转换器，自然优先级最低。

11.3. 增强反射 -- Mirror

11.3.1. 反射的意义

Java 是静态语言。但是 JVM 却不那么静态。静态语言的好处是，IDE 可以提供很高级的重构功能。缺点是你的代码会比较僵化，像 Javascript 一样的动态语言（或者说，后绑定语言），在编写程序时的随心所欲，估计 Java 程序员是享受不到了。 但是好在 Java 还提供了“反射”。

在任何时候，你如果想在运行时决定采用哪个实现类，或者调用哪个方法，通过反射都可以实现，虽然不那么方便（你需要捕捉很多无聊的异常），虽然不那么快。

既然这样，那么能不能让反射工作的更好一些呢？org.nutz.lang 包提供了Mirror<T> 类，通过它，你可以更方便的使用反射的特性

11.3.2. 创建 Mirror

Mirror 就是 Class 的一个包裹：

```
[CODE]
Mirror<Pet> mirror = Mirror.me(Pet.class);
```

更多时候，Mirror 用不到类型：

```
[CODE]
Mirror<?> mirror = Mirror.me(Pet.class);
```

一旦你获得了 Mirror 对象，你就可以做如下操作:

11.3.3. 更方便的构造

反射的 Class.newInstance() 是个很方便的函数。但是它只能调用默认构造函数。如果你想调用类的某个带参数的构造函数，是很麻烦的。通过 Mirror 你可以

11.3.3.1. 自动判断构造函数

比如类：

```
[CODE]
```

```
public class Pet{
    private String name;
    public Pet(String name){
        this.name = name;
    }
}
```

通过如下调用：

```
[CODE]
Mirror.me(Pet.class).born("XiaoBai");
```

Mirror 会自动根据你的参数寻找到相应的构造函数的。

11.3.3.2. 自动判断工厂方法

比如类：

```
[CODE]
public class Pet{
    private String name;
    public static Pet create(String name){
        return new Pet("Pet:" + name);
    }
    private Pet(String name){
        this.name = name;
    }
}
```

通过如下调用：

```
[CODE]
Mirror.me(Pet.class).born("XiaoBai");
```

Mirror 会自动根据你的参数寻找到相应的工厂方法。

构造函数最优先，如果找不到构造函数会寻找静态工厂方法

11.3.3.3. 自动转换类型

```
[CODE]
public class Pet{
    private String name;
    private Calendar birthday;
    private Pet(String name,Calendar birthday){
        this.name = name;
        this.birthday = birthday;
    }
}
```

通过如下调用：

```
[CODE]
Mirror.me(Pet.class).born("XiaoBai", "2008-10-12 12:23:24");
```

Mirror 会尝试寻找接受两个字符串的构造函数。如果找不到，它发现构造函数 Pet(String, Calendar) 起码参数的数量是一致的，于是就尝试通过 Castors 将 "2008-10-12 12:23:24" 转换成一个 Calendar

11.3.3.4. 更快的构建 - 缓存构造方法

就像你知道的一样， Mirror 根据参数自动判断一个类型的构造函数，过程比较费时。一旦它找到了一个构造函数，或者一个静态工厂方法，我们就希望把它记下来，所以你可以：

```
[CODE]
Mirror<Pet> mirror = Mirror.me(Pet.class);
Borning<Pet> borning = mirror.getBorning("XiaoBai");
Pet xb = borning.born("XiaoBai");
Pet xh = borning.born("XiaoHei");
```

11.3.4. 更方便的调用

Java 的反射允许你在运行时决定调用一个类的某一个成员方法。 通过 Mirror 调用的过程将会变得更加简单。

11.3.4.1. 调用一个函数

比如你有一个类:

```
[CODE]
public class MyClass {
    public String getInfo(String s) {
        return "Get " + s + " @ " + System.currentTimeMillis();
    }
}
```

你可以这么调用 getInfo 方法：

```
[CODE]
Mirror<MyClass> mirror = Mirror.me(MyClass.class);
MyClass mc = mirror.born();
System.out.println(mirror.invoke(mc, "getInfo", "Hello~~~"));
```

控制台输出：

```
[CODE]
Get Hello~~~ @ 1259836169165
```

11.3.4.2. 参数自动转型

```
[CODE]
System.out.println(mirror.invoke(mc, "getInfo", new Email("zozoh",
"263.net")));
```

控制台输出：

```
[CODE]
Get zozoh@263.net @ 1259836289830
```

当然， Email 类的 toString() 需要正常工作。 实际上，它是利用 Castors 来做对象的转型的。

11.3.5. 更方便的获取和设置

11.3.5.1. 获取泛型参数

* 如果没有泛型参数，返回 null

```
[CODE]
Type[] types = Mirror.getTypeParams(MyClass.class);
```

11.3.5.2. 获取 getter

```
[CODE]
// 获取 getName() 方法
Method getter = mirror.getGetter("name");
```

11.3.5.3. 获取 setter

```
[CODE]
// 获取 setName(String) 方法
Method setter = mirror.getSetter("name", String.class);
```

11.3.5.4. 获取全部属性

获得所有的属性，包括私有属性。不包括 Object 的属性

```
[CODE]
Field[] fields = mirror.getFields();
```

11.3.5.5. 获取全部方法

获取所有的方法，包括私有方法。不包括 Object 的方法

```
[CODE]
Method[] methods = mirror.getMethods();
```

11.3.5.6. 获取全部静态方法

```
[CODE]
Method[] methods = mirror.getStaticMethods();
```

11.3.5.7. 获取字段

* 获取一个字段。这个字段可以是当前类型或者其父类的私有字段。

```
[CODE]
Field f = mirror.getField("name");
```

* 获取一组声明了特殊注解的字段

```
[CODE]
Field[] fields = mirror.getFields(MyAnnotation.class);
```

* 获取第一个声明了特殊注解的字段

```
[CODE]
Field f = mirror.getField(MyAnnotation.class);
```

11.3.5.8. 获取字段值

* 不调用 getter，直接获得字段的值

```
[CODE]
Object v = mirror.getValue(obj, mirror.getField("name"));
```

* 优先通过 getter 获取字段值，如果没有，则直接获取字段值

```
[CODE]
Object v = mirror.getValue(obj, "name");
```

11.3.5.9. 设置字段值

* 为对象的一个字段设值。 不会调用对象的 setter，直接设置字段的值

```
[CODE]
mirror.setValue(obj, mirror.getField("name"), "XiaoBai");
```

* 为对象的一个字段设置。优先调用 setter 方法。

```
[CODE]
mirror.setValue(obj, "name", "XiaoBai");
```

11.4. 代码模板

11.4.1. 什么是代码模板

通过 Java 提供的 String.format() 方法，可以很方便的声明字符串模板，以及占位符。但是，它不方便的是，如果我的字符串模板两个占位符希望的是同一个值：

```
[CODE]
XXXXXXXX $A  XXXXXX $B  XXXXXX $A XXXXX
```

通过 String.format() 需要这么写：

```
[CODE]
String.format("XXXXXXXX %s  XXXXXX %s  XXXXXX %s XXXXX",
"TxtA","TxtB","TxtA");
```

并且你不设值，会被无情抛错。

11.4.2. Nutz 的代码模板

接上例，你可以这么写：

```
[CODE]
Segment seg = new CharSegment("XXXXXXXX ${A}  XXXXXX ${B}
XXXXXX ${A} XXXXX");
seg.set("A", "TxtA").set("B", "TxtB");
System.out.println(seg.toString());
```

* 无需重复设置占位符
* 自动通过 Castors 将你的对象转成字符串
* 不设值的占位符，输出时会被空串（""）填充

11.4.3. 什么时候使用 Nutz 的代码模板

* 如果你的代码模板比较简单，还是推荐使用 String.format
* 如果你的代码模板比较复杂，有重复的占位符，或者可能有重复的占位符，推荐使用 CharSegment
* 你需要知道
  > 如果多线程共享一个 CharSegment，那么最好使用前复制一份：

```
[CODE]
Segment newSeg = seg.born();
newSeg.set("A",XXX").set("B",XXX") ....
```

    - 否则通常会有多线程内存共享的问题。
  > CharSegment 解析的速度很快
  > Clone 方法对多线程是不安全的，因为每个占位符的值不会被深层 clone

```
[CODE]
Segment newSeg = seg.clone();
```

11.5. 文件池

11.5.1. 什么是文件池

如果大家写一些稍微复杂的应用，可能或多或少的会使用一下临时文件。或者希望能有一个目录，保存用户上传的文件。你的用法可能有下面两种：

* 文件不能超过一定数量，超过了自动删除旧的
* 文件永远保存，除非程序主动删除

对于网络应用，处理用户上传的文件，有一种比较常用的做法是：

* 把文件的索引以及一些信息记录在数据库里
* 把文件内容记录在磁盘里

而 Nutz 的文件池，就是为上述这些场景设计的。

11.5.2. 怎么使用文件池

对于文件池，它有一个接口 'org.nutz.filepool.FilePool' 以及一个简单的实现 'org.nutz.filepool.NutFilePool'。当然，你可以根据需要实现自己的特殊规则的文件池实现类。

11.5.2.1. 创建文件池

```
[Java]
// 将目录 ~/tmp/myfiles 作为一个文件池的根目录，里面最多同时有2000个文件
FilePool pool = new NutFilePool("~/tmp/myfiles", 2000);
// 将目录 ~/tmp/myfiles 作为一个文件池的根目录，里面不限文件
FilePool pool = new NutFilePool("~/tmp/myfiles");
// 相当于 FilePool pool = new NutFilePool("~/tmp/myfiles", 0);
```

11.5.2.2. 在池中创建一个文件

```
[Java]
File f = pool.createFile(".png");  // 该文件本句之后，已经被创建
// TODO 为这个文件写入内容
```

实际上， NutFilePool 会将你的 id （一个 Long 值）以及后缀变成如下形式的文件路径:

```
[CODE]
~/tmp/myfiles/00/00/00/00/00/00/00/EF.png
```

也就是说，你传入的 Long 值会先变成字符串 '00000000000000EF'， 然后每两位字符加入一个 '/'，最后拼上 ".png" 就成为你这个文件在池中的路径了。

这样做的好处是，你只要在数据库记录一个自增的 ID，就能知道在池中对应文件的路径。同样，如果你知道了在池中对应文件的路径，你也能知道在数据库中记录的 ID。这个特性有时候是很有用的。

文件池会维护一个自增的 id， 每当成功创建一个文件后，它就会自加。

补充说明: NutFilePool 这个实现非常简单，没有考虑到集群，即多台主机共用一个文件池的问题。但是如果在一台主机内，虽然它没用用线程同步锁，但是线程安全也不太会是问题，因为是直接一句自加，即使是多线程竞争也不会出问题的。

了解了这个之后，之后的操作就都顺理成章了 ^_^

11.5.2.3. 从池中获取一个文件

```
[Java]
File f = pool.getFile(239, ".png");  // 如果文件不存在，将返回 null
```

11.5.2.4. 从池中获取一个文件，如果不存在，创建它

```
[Java]
File f = pool.returnFile(239, ".png");
```

11.5.2.5. 从池中删除一个文件

```
[Java]
File f = pool.removeFile(239, ".png");
```

11.5.2.6. 判断池中是否存在一个文件

```
[Java]
boolean exists = pool.hasFile(239, ".png");
```

11.5.2.7. 看看当前池中最大的文件 ID

```
[Java]
long maxId = pool.current();
```

11.5.2.8. 看看某一个池中的文件的 ID 为多少

```
[Java]
File f = pool.getFile(239, ".png");
long fId = pool.getFileId(f, ".png");
// fId should be 239
```

11.5.2.9. 清除池中所有文件

```
[Java]
pool.clear();
```

11.5.3. 在 Ioc 中使用

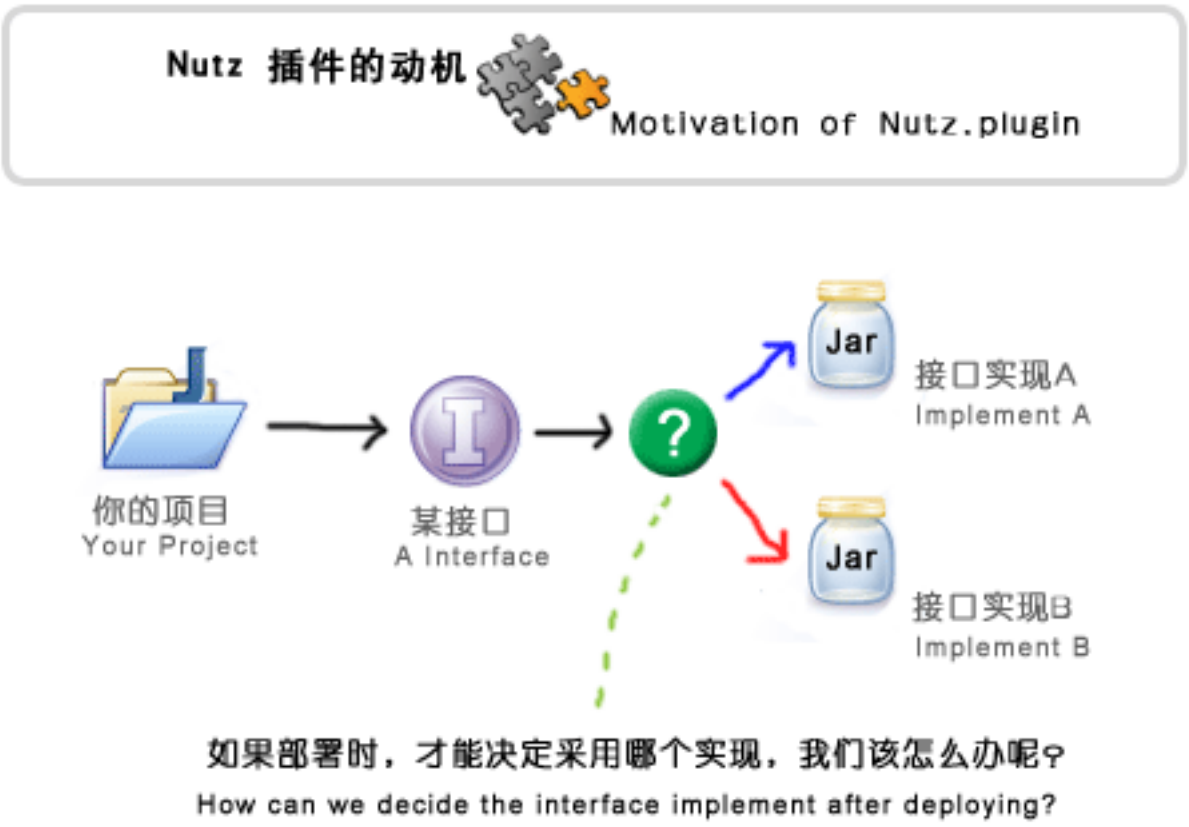
文件池当然可以被任何 Ioc 容器管理，下面我们以 Nutz.Ioc 的 JSON 配置方式来举例:

```
[Json]
{
    pool : {
        type : 'org.nutz.filepool.NutFilePool',
        args : ['~/tmp/files', 2000]
    }
}
```

我想熟悉 Spring 或者 Guice 的朋友，怎么配置自然不用我多嘴了吧

11.6. 部署时决定-插件机制

11.6.1. 什么是插件



如果我们的项目依赖了一个接口，但是我们在开发时，真的没办法确定，部署的时候，到底采用哪个实现。

比如 Nutz 的 Log。 它在运行时，会判读当前运行环境 log4j 是否可用（有 log4j 的 org.apache.log4j.Logger类），如果没有，那么它就把日志信息输出到控制台上。它的实现，就是依靠的方式。

但是，同复杂强大的 OSGI 插件体系不同，这里的插件只是强调，在部署时决定采用什么实现。在运行时，它是没办法更改的。因此 Nutz 虽然在编译时依赖了 Log4j，但是在运行时，没有 log4j 的 jar，依然能够工作的很好。

也正因为，这个插件简单的令人发指。有兴趣的同学可以参看：org.nutz.plugin 里面的源代码，我想几分钟你就会全部看完。

11.6.2. 简单使用插件

比如有一个接口：

```
[Java]
public interface Said{
```

```
    String say();
}
```

你有两个实现类：

* 实现类 A

```
[Java]
public class TomSaid implements Said{
    public String say(){
        return "I am Tom";
    }
}
```

* 实现类 B

```
[Java]
public class PeterSaid implements Said{
    public String say(){
        return "I am Peter";
    }
}
```

这两个实现类分别放在两个 jar 包里，在你的工程部署时，负责部署的工程师很希望：

* 将 tom.jar 放到项目里，整个工程就会使用 TomSaid，
* 将 peter.jar 放到项目，整个工程就会使用 PeterSaid，
* 将两个 jar 都放到项目，PeterSaid 有更高的优先级

怎样做到这一点呢？

首先我们需要在你的工程里为 TomSaid 实现一个插件：

```
[Java]
public class TomSaidPlugin implements Plugin, Said{
    private Said said;
    public boolean canWork(){
```

```
        try {
            said =
(Said)(Class.forName("com.you.app.TomSaid").newInstance());
            return true;
        } catch (Exception e) {}
        return false;
    }
    public String say(){
        return said.say();
    }
}
```

同理，为 PeterSaid 也实现一个插件：

```
[Java]
public class PeterSaidPlugin implements Plugin, Said{
    private Said said;
    public boolean canWork(){
        try {
            said =
(Said)(Class.forName("com.you.app.PeterSaid").newInstance());
            return true;
        } catch (Exception e) {}
        return false;
    }
    public String say(){
        return said.say();
    }
}
```

在调用代码里这样实现：

```
[Java]
PluginManager<Said> plugins = new SimplePluginManager<Said>(
        "com.you.app.PeterSaidPlugin",
        "com.you.app.TomSaidPlugin");
Said said = plugins.get();
```

```
System.out.println(said.say());
```

上面的代码既不依赖 PeterSaid，也不依赖 TomSaid，完全能满足部署工程师的要求。

采用 SimplePluginManager 有几个注意事项：

* 插件实现类必须有一个默认的构造函数
* 插件实现类必须实现目标接口，在上例中就是 Said 接口
* 插件实现类实际上就是一个被适配目标的一个代理（在这里，你可以套套“代理模式”）
* 构造函数参数的顺序，就是插件的优先级，第一个最优先

11.6.3. 与 Ioc 容器一起工作

有些时候，你的 Plugin 实现类需要一些配置信息，某些配置信息可能相当复杂。 我们可以将插件同 Ioc 容器联用（通过 IocPlugManager<T>）。

比如我们修改一下上面的两个插件，让它们都需要被配置一个字段:

```
[Java]
public class PeterSaidPlugin implements Plugin, Said {
    private String prefix;
    private Said said;
    public boolean canWork() {
        try {
            said = (Said)
(Class.forName("com.you.app.PeterSaid").newInstance());
            return true;
        } catch (Exception e) {}
        return false;
    }
    public String say() {
        return prefix + said.say();
    }
}
```

这两个插件，都需要一个 "prefix" 的属性

```
[Java]
```

```
public class TomSaidPlugin implements Plugin, Said {
    private String prefix;
    private Said said;
    public boolean canWork() {
        try {
            said = (Said)
(Class.forName("com.you.app.TomSaid").newInstance());
            return true;
        } catch (Exception e) {}
        return false;
    }
    public String say() {
        return prefix + said.say();
    }
}
```

在 Ioc 容器的 Json 配置文件中:

```
[Json]
// plugins.js
{
    peter : {
        type: 'com.you.app.PeterSaidPlugin',
        fields: {
            prefix : 'Peter: '
        }
    },
    tom : {
        type: 'com.you.app.TomSaidPlugin',
        fields: {
            prefix : 'Tom: '
        }
    }
}
```

调用代码改成：

```
```

```
[CODE]
Ioc ioc = new NutIoc(new JsonLoader("conf/plugins.js"));
PluginManager<Said> plugins = new IocPluginManager<Said>(ioc,
"peter", "tom");
Said said = plugins.get();
System.out.println(said.say());
```

11.6.4. 最后一点说明

我曾一度怀疑这个插件功能很无聊，因为它真的可以用“简陋”二字来形容（我们实现这个插件用的时间还没有我写这篇文档所用时间的一半），但是它毕竟给了你一条很简明的途径，让你的程序可以做到：

部署时才决定某一个接口的实现

你用微小的代价（实现一个接口函数）获得下面两个好处：

* 你的应用很容易做到 **模块化**
* 它几乎 **没有侵入性**

挺值得的，不是吗？ 这也是为什么我们将它放到 Nutz 的核心发布包里原因：（*超值的东西才会被放到 Nutz 的核心包里*）

如果你想做到运行时加载/卸载，这个“简陋”的小插件方案恐怕是帮不上你了。但是你真的需要吗？我注意到一个事实：Eclipse 采用的是 “OSGI”，但是在安装了一个插件之后它还是会建议你重启应用，每次我看到这个对话框，都觉得是对 “OSGI” 的一个讽刺。

11.7. 日志

11.7.1. 为什么要自己写一个 Log?

首先，我们没有从头实现一个日志，我们不过是 **适配** 日志。我们适配了 log4j 。但是为什么呢?为什么不直接依赖 slf4j 呢?

我必须要在这里重申一下 Nutz 框架的目标：

* 尽量让程序员在 **设立开发环境** 以及 **部署应用** 的时候获得最佳的体验

如果你下载了 Nutz 的 jar，满心欢喜的加到你的项目里，然后，运行你的项目，你会发现从 Nutz.jar 里会迅速抛出一个异常。经过一番查阅，你发现你不得不访问 slf4j 的网站，下载它的 jar 包，然后再加入自己的项目里...

你会因此而很高兴吗？

.....

不不，起码我不会高兴。 :(

如果这个功能特别复杂，以至于我们没有能力做到很好，我们肯定会毫不犹豫的依赖其他的 jar 包的。比如 1.a.23之前，我们依赖了 javassist， 1.a.24 我们包括了 ASM。

另外一个原因：随着 Nutz 使用的人越来越多，如果它依赖了过多的 Jar 包，便会导致人们更多的下载行为，从而浪费不必要的带宽，耗费更多的电能，从而让这个地球排放更多的温室气体，厄尔尼诺现象加剧，北极融化，珍稀物种灭绝，农作物减产，粮食供应紧张，不利于喝血社会，不能体现带三个婊的以德治国的精神。

.

..

...

....

.....

.....

好吧，我承认，我上面说的统统是狡辩，根本原因不过是因为手痒痒。而且我揭发：实际上这个模块代码都是 wendal 和 Sunonfire 写的，我其实不过是出来打酱油的 ...

11.7.2. 如何使用日志

Nutz Log的基本用法和常见的log4j等工具没什么不同。比如：

```
[CODE]
public class MyClass {
    private static final Log log = Logs.getLog(MyClass.class);
    public void myFuncion() {
        if (log.isDebugEnabled())
            log.debug("I am debug message");
    }
}
```

Nutz.Log 使用了插件技术，其中优先级依次为：

1. Log4j --> 通过log4j-over-slf4j桥,你可以将其拓展到其他log框架,如logback
2. System.out | System.err

它们可以使用的条件是：

* Log4j
    1. 如果能够加载类 **org.apache.log4j.Logger**，则认为可用;
* System.out | System.err
    1. 默认永远可用

11.7.3. 让 Nutz 输出日志

以 log4j 为例（因为我就对这个熟），你需要：

* 在你的项目里部署某一个 Log4j 的 jar 包，推荐 Log4j 1.2.12 或更新版本
    > Log4j 1.2.11及之前的版本不支持 Trace,自动转为使用Debug级别来记录.
* 在项目的 CLASSPATH 根目录部署 log4j.properties
* 将 log4j.category.org.nutz 基本设成 DEBUG，你就能看到 Nutz 框架详细的 Log

```
[CODE]
log4j.category.org.nutz=DEBUG, A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c
- %m%n
```

* 你可以分别控制 Dao, Ioc 和 Mvc:
    > category.org.nutz.dao=DEBUG
    > category.org.nutz.ioc=DEBUG
    > category.org.nutz.mvc=DEBUG

11.7.4. 什么都不配置呢?Nutz在你没有配置任何Log时,一样工作得很好. 下面是默认行为:

* 输出到System.err/out
* 默认日志等级为 DEBUG

12. zMole

12.1. zMole 快速入门

12.1.1. 现有实现

已经有一个独立项目来完成这个功能 -- nutzam.nutzmole, 现阶段仅仅是一个简单实现,已经可以完成DB-->项目的生成,但CRUD页面还没完成,一对一/一对多/多对多关系未完成.

你可以通过svn来获取当前的代码,期待的建议与想法.

12.1.2. 后续进展

敬请期待!!

13. zDoc 手册

13.1. zDoc 概述

13.1.1. 我为什么要建立 zDoc 项目?

开源项目最需要的（同时也是最缺乏的）就是丰富全面的文档。在建立开源项目 Nutz 的时候，我深刻的体会到了这一点。为此，经过思考，我认为只要做到下面两件事：

1. 让每篇文档很容易写
2. 让每篇文档可以同时被多人写

那么，对在开源项目的文档写作方面的工作，会有很大帮助的。

根据多年惨痛的文档写作经验我得出了如下两个结论：

* 类似 Word 的桌面文档工具并不适合多人协同工作
    > 除非你给MS纳钱，架个 SharePoint 类的服务器
    > 文档的改动历史追踪很困难
    > 跨平台性不太好（因为兼容性的问题）
* 在线文档应用并不适写庞大的文档
    > Google Doc 之类的，当文档大的时候，编辑起来会很慢
    > 总有意外的小Bug，而且通常很恼人

所以，我必须构建自己的文档写作方式。我注意到两个事实：

1. 对于文档变动的管理，我想现在的版本控制系统已经做的足够好了（比如 SVN, Git, Mercurial）。
2. 市面上所有的操作系统都会对纯文本文件（尤其是 UTF-8）支持的很好。（比如 Windows的Nodepad++，Editplus,Ultra-Edit 以及 Linux 下的 Vim 等）

所以，我打算将这两个事实充分利用起来。尤其幸运的是我本人就是一名程序员，于是我便是建立了一个小小的文本文件解析渲染器 -- zDoc。

13.1.2. zDoc 要达到的目标

* 我将使用任何一款我喜欢的**纯文本编辑器**来编辑我的文档
    > 所以我可以利用 SVN 等工具来管理版本，协作编写文档
* 我的文档将包含文字，以及 B/I/S 等格式信息
* 利用缩进来划分文档的结构
* 可以生成文档目录索引
* 支持超链接，包括文档内部和外部
* 支持图片
    > 流行的图片

    > 支持 Icon
    > 可自定义大小
* 支持输出的格式
    > PDF
    > HTML (browser from local)
    > HTML (browser from web)
    > Google Wikie
* 文档可以 include 另外一个文档

13.1.3. 如何使用 zDoc

13.1.3.1. 设置运行环境

zDoc 提供了命令行程序，使用它之前你需要做如下设定

1. 在发布包里面找到 run 目录
2. 打开其中的 win 目录或者 linux 目录(依你自己的操作系统而定)
3. 修改脚本文件 zdoc (Windows 下为 zdoc.bat) -- 你需要修改三个变量
    1) JAVA_HOME 指向你的 Java 安装的根目录 (这个目录下有 lib 子目录，其内有 tools.jar, dt.jar 以及 rt.jar)
    2) NUTZ_HOME 指向 Nutz 的核心 jar 包: 可从此处下载=> http://nutz.googlecode.com
    3) ZDOC_HOME 指向 Nutz.Doc 的运行 jar 包: 可从此处下载=> http://nutzdoc.googlecode.com
4. 修改运行路径
    * **Windows** 用户可以将 run 目录加入到你的系统环境变量 **%PATH%** 中去
    * **Linux** 用户，可以将 zdoc 脚本文件链接至 ~/bin 目录下。
        > 请确保此脚本文件权限为可执行

```bash
chmod +x ./zdoc
```

13.1.3.2. 将 zDoc 转换成 HTML

命令格式

```
zdoc html [源目录] [目标目录]
```

说明

* 这三个参数缺一不可
* 第一个参数表示要转换成 HTML
* 第二个参数声明了zDoc源目录
    > 该目录下如果有 index.xml，将能更为精确的控制生成的 html 目录索引
    > 请看 zDoc 目录转换中的索引文件--index.xml的语法
    > 你的 zdoc 文件必须以 .man 或者 .zdoc 为后缀名，大小写不敏感
    > 更多的关于目录转换的细节请参看 目录转换细则
* 第三个参数为目标输出的目录

比如

```
[CODE]
zdoc html /folder/abc /output/abc
```

会将目录 /folder/abc 所有的 zdoc 文档转换成 HTML 文档，并输出到 /output/abc 目录下

13.1.3.3. 将 zDoc 转换成 Goolge wiki

Google code 是很多开源项目理想的开发平台。 它提供了 wiki 功能。 如果你建立一个 wiki 页面，在它的版本控制服务的目录内，会出现一个 wiki 的目录，通常，是在 /trunk/wiki 下。

Google code 的 wiki 格式非常简单，易于理解，Google 为其提供了较强大的页面渲染支持。所以将 zDoc 直接渲染成 Google wiki 对大多数在 Google code 上的开源项目是很有用的，你需要执行如下格式的命令

命令格式

```
[CODE]
zdoc gwiki [源目录] [目标目录] [索引文件名] [图片地址前缀]
```

说明

* 第一个参数表示要转换成 Goolge Wiki
* 第二个参数声明了zDoc源目录
    > 该目录下如果有 index.xml，将能更为精确的控制生成的目录索引
    > 请看 zDoc 目录转换中的索引文件--index.xml的语法
* 第三个参数指明 wiki 目录， zDoc 将会在这个目录下生成所有的 .wiki 文件
* 第四个参数为索引文件名，请参看 Google wiki 关于目录索引的描述
    > zDoc 会替你生成这个索引文件

> 如果你写 null，zDoc 将不会生成这个文件

* 第五个参数有些奇怪，因为 Google wiki 并不支持图片存储，它只支持图片引用。所以你的文档中如果有图片，必须要存放在某一个可被互联网访问到的地址上
  > 这个参数也不是必须的
  > 如果你声明了这个参数，所有的 zDoc 文档中引用的本地图片
    - 将会被目标目录根录下一个叫 **wiki_imgs** 的目录下，你可以用 ftp 工具，将图片上传到你指定的地址
    - 生成后的 wiki 文件，图片地址将指向你声明的地址
    - 如果你的图片重名，zDoc 将自动为你改名（按顺序增加数字后缀）

比如

```
[CODE]
zdoc gwiki /folder/abc /output/abc wiki_index_page
http://www.my.com/wiki/img
```

会将目录 /folder/abc 所有的 zdoc 文档转换成 Google wiki 文档，并输出到 /output/abc 目录下。并会根据 index.xml（如果没有，根据文件）生成一个索引 wiki 页，文件名为 wiki_index_page.wiki。所有的本地图片都会被打入压缩包 /output/abc/wiki_images.zip，并且所有的本地图片地址都将变成 http://www.my.com/wiki/img/xxxx.xxx

13.1.3.4. 将 zDoc 转换成 PDF 文件

PDF 是一个非常通用的文档格式。如果你通过 zDoc 建立一组文件，你就可以用如下命令生成 PDF 文件

```
[CODE]
zdoc pdf [source dir] [target pdf path]
```

说明

* 第一个参数表示要转换成 PDF 文档
* 第二个参数声明了zDoc源目录
  > 该目录下如果有 index.xml，将能更为精确的控制生成的目录索引
  > 请看 zDoc 目录转换中的索引文件--index.xml的语法
* 第三个参数指明一个 PDF 的文件，如果该文件不存在，会被自动创建，否则会被覆盖。

这里还需要注意的是，因为 PDF 输出，zDoc 依赖的是 iText 项目，你需要

1. iText-2.X.X.jar
2. iTextAsian.jar - 用来支持中文
3. 在你的 CLASSPATH 内，放置一个字体文件，名字为 **pdf_font.ttf** 或者 **pdf_font.ttc**
    * 这个文件必须放置在 CLASSPATH 的根，即，同 Log4j.properties 同级，比如：

```
[CODE]
 [My Class Folder]
   [com]
     [my]
        Abc.class
        Main.class
   log4j.properties
   pdf_font.ttf
```

    * **.ttf** 字体文件更优先

比如

```
[CODE]
zdoc pdf /home/peter/myDoc /home/peter/myPdf.pdf
```

会将目录 /home/peter/myDoc 所有的 zDoc 文档转换成一个大的 PDF 文件 /home/peter/myPdf.pdf

13.1.4. 其他

* 关于 zdoc 文件详细的语法，请参看 zDoc 的语法说明
* 关于目录转换的细则，请参看 目录转换细则 以及 使用 index.xml 作为索引文件
* zDoc 已经被我应用到具体的工作中，比如写作界面设计文档，以后如果我打算写小说，也会使用 zDoc。

总之，一个文本编辑器，一个 SVN 服务器就可以多人协作，写出漂亮的文档，这种感觉会很美妙的。

13.2. zDoc语法

13.2.1. 什么是zDoc文档

* zDoc 是纯文本的文档
* zDoc 必须是 UTF-8 编码的
* 它以"行"为单位进行文档解析

13.2.2. 标题与段落

```
[zDoc]
这里是标题
    这里是内容
    这里是二级标题
        这里是内容
    二级标题可
    以折行
        紧跟着一个缩进内容就可以
```

* 缩进的行是它上一行的**子**
* 拥有**子**的行就是标题行
* zDoc 理论上拥有无限层次的标题行

13.2.3. 无序列表

```
[zDoc]
 * 项目 A
    * 项目 A1
    * 项目 A2
 * 项目 B
```

* 格式段落以 * (星号+空格) 或者 (空格+星号+空格) 开头
* 连续的这样的段落为一个列表

13.2.4. 有序列表

```
[zDoc]
 # 项目 A
    # 项目 A1
    # 项目 A2
 # 项目 B
```

* 格式段落以 # (星号+空格) 或者 (空格+星号+空格) 开头
* 连续的这样的段落为一个有序列表，生成器会自动为你生成序号

13.2.5. 超链接

* 在行内 被方括弧[和]包裹起来的就是超链接
* [http://www.google.com] 将显示成 http://www.google.com
* [http://www.google.com Google] 将显示成 Google
* [http://www.google.com text with whitespace] 将显示成 text with whitespace
* [http://nutz.googlecode.com <http://code.google.com/p/nutz/logo?logo_id=1239077401>] 将显示成

13.2.6. 逃逸字符

* 被字符 包括起来的内容将不做格式转换
* 连续两个 将表示一个

13.2.7. 文字及样式

zDoc 支持如下字体样式，暂时不支持下划线，以及字体颜色

名称	zDoc 语法	文本
粗体	{*some text}	**some text**
斜体	{/some text}	*some text*
下划线	{_some text}	some text
穿越线	{~some text}	~~some text~~
红色斜体	{#F00;_some text}	some text
穿越线斜体	{*/~some text}	***~~some text~~***
标注	txt{^sup}	txtsup
底注	txt{,sub}	txtsub

各个格式的组合，顺序没有限制

13.2.8. 图片

格式为

```
[CODE]
<[宽x高:]图片地址>
```

* 在行内，被 < 和 > 包括起来文字将表示一个图片地址
* 这个地址如果以 http:// 或者 https:// 开头，将作为远程图片地址
* 否则将作为本地图片
    > 本地图片必须存在
    > 如果本地图片存在，将被**相对拷贝**
    > 所谓**相对拷贝**指
        - 如果图片地址为绝对地址，则保持不变
        - 如果图片地址为相对地址，则相对于输出位置，拷贝图片，以使图片链接生效

比如

```
[CODE]
图片：<../logo.png>
```

将显示成:

图片： 

当然，我们可以控制图片的大小

```
[CODE]
图片：<60x15:../logo.png>
```

将显示成:

图片： 

注意，你必须同时写出图片的宽度和高度。

13.2.9. 分隔线

* 任何行内内容全部为减号 - ，并且数量不少于5个，将被认为是分隔线
* 比如如果一行内容为

```
[zDoc]
-----
```

```
[HTML]
<div class="hr"></div>
```

13.2.10. 代码

* 以 {{{ 开始的行，下面所有的行都被认为是代码，一直到 }}} 开始的行为止
* {{{<java> 表示代码的类型为java源代码
    > 你可以随便输入你的代码类型，比如 C#, Python, SQL, HTML 等等

比如

{{{<Javascript>

function abc(msg){

alert(msg);

}

}}}

将显示为:

```
[Javascript]
function abc(msg){
alert(msg);
}
```

13.2.11. 表格

* 以 || 开始，并以 || 结束的行，被认为是表格行
* 连续的表格行被认为是一个表格
* 符号 || 之间的内容被认为是一个单元格

> *[zDoc]*
> ||A||{*B}||[http://www.google.com Google]||
> ||X||{*YYY}||[http://www.apache.org Apache]||

13.2.12. 目录索引

* 在文档任何行写 #index:3 将会在该位置生成目录
    > 其中 3 表示目录级别到第3层
* **层数是以 0 开始的**
* 声明 #index:0,3 表示目录级别为第0层到第3层

13.2.13. 文档属性

zDoc 语法	说明
#title:文档标题	声明文档标题，如 #title:今天是个好日子
#author:文档作者	声明文档作者，格式为：作者名(电子邮件) 其中 **(电子邮件)** 可选，如 #author:zozoh<zozoh@mail.com>

13.3. 使用 index.xml 作为索引文件

13.3.1. 关于 index.xml

13.3.1.1. 为什么需要索引文件

通常，你会将你的很多 zDoc 文件分门别类的放在某个目录下。然后你有可能：

1. 将这个目录全部输出到另外一个目录下
    * 比较多的情况是输出成 HTML 格式的文档
2. 将这个目录所有的 zDoc 文件合并到一个大的文档中去。
    * 可以是个大的 HTML
    * 更多的是个大的 PDF

13.3.1.1.1. 那么在 zDoc 转换程序组织这些文件时会碰到类似如下的问题：

* 哪些文件需要转换？
* 如果是 HTML 格式的目录转换，索引是什么？
* 如果是合并成一个大文件，大文件都有哪些章节，顺序如何？

所以在根目录放置一个 **index.xml**，zDoc 转换程序就有办法获得充足的信息.

13.3.1.2. 为什么要用 XML 格式

* 我不假定使用 zDoc 的人都是程序员
* 即使是程序员也不一定喜欢 JSON，非程序肯定不喜欢 JSON，因为他们不喜欢很多的 { 和 }
* XML 有足够的灵活性和扩展空间
* XML 的结构化表现力很强
* 索引文件只有一份，XML 虽然麻烦一点，但是也不会很累

13.3.2. index.xml 语法

该文件的语法简单的令人发指

* 它只有一个元素 <doc>，
* <doc> 只有两个常用属性和两个不常用属性
* <doc> 可以嵌套

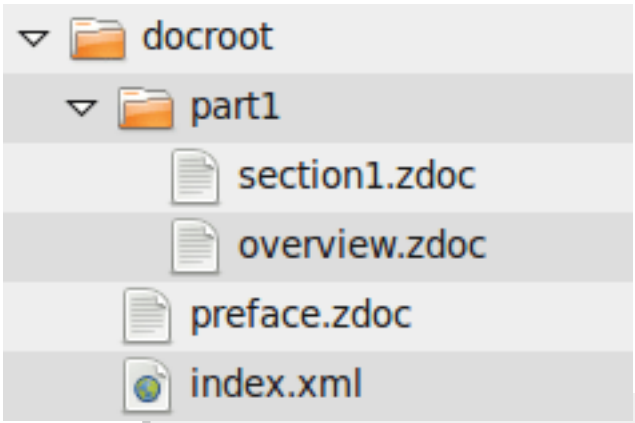
index.xml 的格式如下:

```
[index.xml]
<?xml version="1.0"?>
<doc title="你的 Doc 文档名" author="作者名" index="目录级别">
   <doc path="根目录下的文档名称.zdoc"title="这个文档的标题"/>
   <doc path="目录名称" title="目录标题">
      <doc path="文档名称.zdoc"/>
   </doc>
</doc>
```

举一个更详细的例子，比如你有一个如下的目录结构

那么在 index.xml 的内容应该为

```
[index.xml]
```




```
<?xml version="1.0"?>
<doc title="文档示例" author="zozoh(zozohtnt@gmail.com)"
index="0,1">
   <doc path="preface.zdoc" title="前言"/>
   <doc path="part1" title="第一部分">
      <doc path="overview.zdoc"/>
      <doc path="section1.zdoc"/>
   </doc>
</doc>
```

* 根元素 <doc> 对应到根目录 docroot，即，**index.xml 所在的目录。**
* 在跟元素你可以用 @author 属性声明这篇文档的默认作者，。如果目录下的 zDoc 文档没有声明 #author，那么便被认为是默认作者所写。
* 属性 @index 的值符合 zDoc语法 中的 **目录索引** 中的定义
* @path 属性表示当前对应的目录或者文件
    > 如果是目录，需要有标题 (通过属性@title声明)，如果没有声明 @title，则使用目录名称来代替
    > 如果是文件，如果没有 @title 属性，则采用文档的 #title，详情惨参看 zDoc语法 **文档属性** 这一节。

13.4. zDoc 目录转换细则

13.4.1. 将目录转换成另外一组 HTML 文档

* 将 index.xml 的内容生成一组 HTML 元素
* 根据根目录下 index.html 文件，将生成的 HTML 元素插入 index.html 占位符 ${html} 处
* 如果根目录下有 zdoc.css ，则将其链入所有的生成的 HTML 中
* 如果根目录下有其他的 css 文件以及 js 文件，将其拷贝到目标目录下。

13.4.2. 将目录转换成一个大文件

* 无论转换成 PDF 或者 HTML，均无特别需要注意的。
* 根据 index.xml 生成章节顺序以及结构

* 将各个子目录下的zDoc文档作为章节内容

13.5. 在Ant中使用

13.5.1. 配置Taskdef

使用Resource

```
[CODE]
<taskdef resource="zdoc_ant.properties">
  <classpath path="nutz-1.a.32.jar"></classpath>
  <classpath path="nutzdoc-1.13.jar"></classpath>
</taskdef>
```

直接指定类

```
[CODE]
<taskdef name="zdoc" classname="org.nutz.doc.ant.ZDocTask">
  <classpath path="nutz-1.a.32.jar"></classpath>
  <classpath path="nutzdoc-1.13.jar"></classpath>
</taskdef>
```

13.5.2. 使用zdoc Task

通过定义不同的suffix,生成不同类型的文档

```
[CODE]
<zdoc suffix="html" src="doc/manual" dest="manual-dir"/>
<zdoc suffix="pdf" src="doc/manual" dest="nutz.manual.pdf"/>
<zdoc suffix="rtf" src="doc/manual" dest="nutz.manual.rtf"/>
```

* html/htm 在目标文件夹生成网页
* pdf 生成PDF文件
* rtf 生成RTF文件

14. 版本历史

14.1. 1.b.47 发行注记

14.1.1. 1.b.47 发行注记

前一版 1.b.46 的时候有一个小问题，就是 JSON 文件的解析更严格了，如果对象尾部多了一个逗号（","）（当然，JSON 语法上，这个的确是错的），JSON 解析会报错。详细请参看 Issue 349

考虑到大家都是粗心的人（多打一个逗号很平常），并且之前 Nutz 的 JSON 解析检查并没有这么严格，所以我们想最好还是紧急发布一版，这个版本和 1.b.46 区别就是能容忍更多一点 JSON 的语法错误。

这样，升级到最新的 **1.b.46** 的同学如果 JSON 解析遇到问题，愤然质问我们的时候，我们可以蛋定的答道： “同学，你 out 了，你应该用 1.b.47”

---------- Nutz 的下载地址的分隔线 ----------

* 稳定版下载地址 : http://code.google.com/p/nutz/downloads/list
* 日编译下载地址 : http://build.sunfarms.net/nutz/
* Nutz 的主页： http://nutzam.com

14.1.2. 问题修复

* Issue 348 当一个IocBean的字段注入失败时, 第二次获取这个bean,会得到一个不完整的对象(部分字段为注入) by **wendal**
* Issue 349 1.b.46 加载json配置文件时尾部多个“,”会出错 by **zwt**

14.1.3. 质量

共通过了 826 个单元测试用例,代码覆盖率达到 70%(按line计算)

Nutz.Dao 经测试在如下数据库上可以工作正常

* H2
* SQLite -- 仅有限支持事务操作
* hsqldb
* MySql
* Oracle
* Postgresql
* SqlServer2005
* SqlServer2000

* DB2
* Derby

14.1.4. 文档

-- 木有任何改动 --

14.1.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 NutzDemo 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献列表，我已经写了一个小程序，根据 Issue 列表来自动统计...

贡献者	问题	博客	支持	代码	示例	文档	测试
wendal	O	O	O	O	O	-	O
zwt	O	-	-	-	-	-	-

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz & XBlink （58444676 超级群）
* Nutz在微笑 （60504323）超级群
* Nutz ② 群 （68428921）超级群
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*
* Nutz的新浪微群
* Nutz的豆瓣小组
* Nutz的聊天室

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^!

欢迎访问官网,以获取 最新的快照版 和用户手册

14.2. 1.b.46 发行注记

14.2.1. 1.b.46 发行注记

人有旦夕祸福，月有阴晴圆缺，万万想不到，3个月前，认为铁板钉钉的事还是阴差阳错的没搞完。就像你信心满满的准备打一个喷嚏的时候，你万万木有料到旁边会有人捅你的咯吱窝。吹牛死全家，看来以后没做出东西来不能随便说。X个月后，我们会有啥，bulabulabula ... 这个话以后我再也不说了，再吹牛就让那个大再开一次！同时，突然觉得老罗微胖在我心目中毅然决然的高大了起来，跳票还能说的那么理直气壮，这得是什么境界才能做得到啊。

当然，想不到的事情还有很多，譬如木有想到 GoogleDoc 又不能用了，也木有想到，都完球了，还不能用，靠，骏马跑在戈壁上！

不管怎么样，Nutz 这个小 jar 包这 3 个月来一直在被持续的微量的改进着，无敌的 wendal 懒洋洋的催道：“该写注记了吧!”然后统计了一下，靠，又 fix 了 30 多个 Issue，接受的 pull request 也更多了 ...

这个版就是个 bug fix 版，没啥大改动，很多微小的地方出现的错误一一被各位同学揪了出来，大家可以安心升级，享受各位同学这 3 个月来用心的成果 ^_^

---------- Nutz 的下载地址的分隔线 ----------

* 稳定版下载地址 : http://code.google.com/p/nutz/downloads/list
* 日编译下载地址 : http://build.sunfarms.net/nutz/
* Nutz 的主页： http://nutzam.com

14.2.2. 问题修复

* Issue 278 使用nutz获取clob字段,并将转换为String后,出现乱码问题 Dao by **xiaxiaofeng**
* Issue 271 将 Mock 移动到新的源码包里去 需求 by **zozoh**
* Issue 265 让适配器所产生的错误,可捕捉,可处理 Mvc by **wendal**
* Issue 264 DB2 9.x下对boolean属性为空的实体更新时报错 Dao by **wendal**
* Issue 263 Mirror类的evalGetterSetter方法无法正确获取getter/setter Lang 重要 by **wendal**
* Issue 260 Blob字段处理 Dao 重要 by **aodixiaoqiang**
* Issue 255 glassfish-3.1.2-web下使用nutz上传文件出现file为null Mvc 重要 by **wendal**
* Issue 254 允许通过Ioc加载MessageLoader实现 by **enzozhong**
* Issue 250 编译时，也应该支持 Servlet 3.0 by **zozoh**
* Issue 249 在glassfish-3.1.2-web下项目启动失败 项目维护 Lang 重要 by **ichaly**
* Issue 247 文件池文档中的在池中创建一个文件描述错误 项目维护 Lang FAQ by **hzl7652**
* Issue 245 查询返回List的方法，Record中的key都变成了小写 by **zhuer0632**
* Issue 244 文档的MVC部分没有关于@Attr的信息 项目维护 Mvc by **shevawen**

* Issue 243 Mapl.cell的Path规则 by **shevawen**
* Issue 240 filter中执行跳转时,obj未传进去,报空指针异常 项目维护 Mvc by **lziilzii**
* Issue 239 Mapl添加"增删改" Json 需求 by **juqkai**
* Issue 238 国际化动态切换失效 项目维护 Mvc by **mamacmm**
* Issue 180 Eclipse debug 模式启动时，AOP 导致 AjaxModule 弹出错误提示 FAQ AOP by **zozoh**
* Issue 82 文档 Ioc，需要一篇介绍 Ioc 原理的文档，解释加载对象的过程 Ioc 文档 by **zozoh**
* Issue 49 探讨“插入前后的设置” Dao 需求 by **conanca**

14.2.3. 质量

共通过了 826 个单元测试用例,代码覆盖率达到 70%(按line计算)

Nutz.Dao 经测试在如下数据库上可以工作正常

* H2
* SQLite -- 仅有限支持事务操作
* hsqldb
* MySql
* Oracle
* Postgresql
* SqlServer2005
* SqlServer2000
* DB2
* Derby

14.2.4. 文档

* 少量修改 文件池
* 少量修改 我的Nutz的源码是乱码怎么办？

14.2.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 NutzDemo 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献列表，我已经写了一个小程序，根据 Issue 列表来自动统计...

贡献者	问题	博客	支持	代码	示例	文档	测试
aodixiaoqiang	O	-	-	-	-	-	-
biggates	-	-	-	O	-	-	-
conanca	O	-	-	-	-	-	-
enzozhong	O	-	-	O	-	-	-
hzl7652	O	-	-	-	-	-	-
ichaly	O	-	-	-	-	-	-
juqkai	O	O	O	O	-	-	O
lziilzii	O	-	-	-	-	-	-
lzxz1234	-	-	-	O	-	-	-
mamacmm	O	-	-	-	-	-	-
nutblog	-	-	-	O	-	-	-
shevawen	O	-	-	-	-	-	-

wendal	O	O	O	O	O	O	O
xiaxiaofeng	O	-	-	-	-	-	-
xing-kenny	-	-	-	O	-	-	-
zhuer0632	O	-	-	-	-	-	-
zozoh	O	-	O	O	-	-	-

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz & XBlink （ 58444676 超级群 ）
* Nutz在微笑 （ 60504323 ）超级群
* Nutz ② 群 （ 68428921 ）超级群
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*
* Nutz的新浪微群
* Nutz的豆瓣小组
* Nutz的聊天室

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^!

欢迎访问官网,以获取 最新的快照版 和用户手册

14.3. 1.b.45 发行注记

14.3.1. 1.b.45 发行注记

Hi 大家好，从六月初发布了 Nutz.1.b.44 以来，已经过去将近两个月了，期间我们颇收集了一些小 bug，于是就发这个 bug fix 版吧。 使用 1.b.4x 的同学可以安心升级。 还在用 1.b.3x 的同学建议你快点升级 -_-!。

另外说一下，临到发布前，我们还是决定将 "mock" 包从 nutz.jar 里移走，它的内容现在被移动到 "test" 目录下了，这个包可能不会有太多人用到，如果个别人想用，可以从 test 目录里 copy 源代码自行编译。这个算是 nutz瘦身活动的开始。我们之后会努力让这个小 jar 包更稳定和小巧。

这两个多月，无敌的Wendal 开始录制 Nutz 系列教程，现已经录了两季了：

* 视频: Nutz 入门教程第一季 - 基础框架的搭建
* 视频: Nutz入门教程第二季 - Ioc 漫谈

这两个教程的下载地址都在 Nutz 下载列表里能找到。内容对初学者是很有帮助的，高级用户也推荐看一看：视频录制期间，Wendal 出现了几次可爱的输出错误，看着他修复时果断且狐疑的样子，我都被萌到了，啊哈哈哈哈 ^_^。

如果想更及时关注教程更新信息的同学，可以到新浪微博关注：

* @WendalMe
* 或者 @zozoh
* 或者 @胖五86

这三个微博都会随时推 Nutz 的相关动态。

5月底由于参加了 我们的开源项目 组织的活动，我们录了几段视频：

* 视频:Nutz一次刻意的意外！
* 视频: nutz_宣传片第0季

有兴趣的朋友大家可以看看。

======= 特别预告 ===========

一个月前，我曾经因为好玩，假模假样的采访过 @胖五86:

* 视频: Nutz [任何人采任何人] 之 zozoh采胖五

但是这个谈话我自己回头看的时候觉得很有意思。尤其是最后，胖五透露了一点我们未来的规划 ...

由于很多不可预测的原因，我们年初计划的 "Nutz 的应用" 一拖再拖。不过，如果不出意外，我可以比较负责任的说，下次我们再见面，大家会得到一个基于 Nutz 的应用的。一个绝对值得你试着用用，而且肯定有用的应用（ *注：不是 CMS 类的* ）。我都有点等不急了，2个月，快点过去吧 ... 阿门 ~~~

---------- Nutz 的下载地址的分隔线 ----------

* 稳定版下载地址 : http://code.google.com/p/nutz/downloads/list
* 日编译下载地址 : http://build.sunfarms.net/nutz/

* Nutz 的主页： http://nutzam.com

14.3.2. 问题修复

* Issue 278 使用nutz获取clob字段,并将转换为String后,出现乱码问题 Dao by **xiaxiaofeng**
* Issue 271 将 Mock 移动到新的源码包里去 需求 by **zozoh**
* Issue 265 让适配器所产生的错误,可捕捉,可处理 Mvc by **wendal**
* Issue 264 DB2 9.x下对boolean属性为空的实体更新时报错 Dao by **wendal**
* Issue 263 Mirror类的evalGetterSetter方法无法正确获取getter/setter Lang 重要 by **wendal**
* Issue 260 Blob字段处理 Dao 重要 by **aodixiaoqiang**
* Issue 255 glassfish-3.1.2-web下使用nutz上传文件出现file为null Mvc 重要 by **wendal**
* Issue 254 允许通过Ioc加载MessageLoader实现 by **enzozhong**
* Issue 250 编译时，也应该支持 Servlet 3.0 by **zozoh**
* Issue 249 在glassfish-3.1.2-web下项目启动失败 项目维护 Lang 重要 by **ichaly**
* Issue 247 文件池文档中的在池中创建一个文件描述错误 项目维护 Lang FAQ by **hzl7652**
* Issue 245 查询返回List的方法，Record中的key都变成了小写 by **zhuer0632**
* Issue 244 文档的MVC部分没有关于@Attr的信息 项目维护 Mvc by **shevawen**
* Issue 243 Mapl.cell的Path规则 by **shevawen**
* Issue 240 filter中执行跳转时,obj未传进去,报空指针异常 项目维护 Mvc by **lziilzii**
* Issue 239 Mapl添加"增删改" Json 需求 by **juqkai**
* Issue 238 国际化动态切换失效 项目维护 Mvc by **mamacmm**
* Issue 180 Eclipse debug 模式启动时，AOP 导致 AjaxModule 弹出错误提示 FAQ AOP by **zozoh**
* Issue 82 文档 Ioc，需要一篇介绍 Ioc 原理的文档，解释加载对象的过程 Ioc 文档 by **zozoh**
* Issue 49 探讨“插入前后的设置” Dao 需求 by **conanca**

14.3.3. 质量

共通过了 781 个单元测试用例,代码覆盖率达到 70%(按line计算)

Nutz.Dao 经测试在如下数据库上可以工作正常

* H2
* SQLite -- 仅有限支持事务操作
* hsqldb
* MySql
* Oracle
* Postgresql
* SqlServer2005
* SqlServer2000
* DB2

* Derby

14.3.4. 文档

* 少量修改 文件池
* 少量修改 我的Nutz的源码是乱码怎么办？

14.3.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 NutzDemo 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献列表，我已经写了一个小程序，根据 Issue 列表来自动统计...

贡献者	问题	博客	支持	代码	示例	文档	测试
aodixiaoqiang	O	-	-	-	-	-	-
conanca	O	-	-	-	-	-	-
enzozhong	O	-	-	-	-	-	-
hzl7652	O	-	-	O	-	-	-
ichaly	O	-	-	-	-	-	-
juqkai	O	-	O	O	-	-	O
lziilzii	O	-	-	-	-	-	-

mamacmm	O	-	-	-	-	-	-
shevawen	O	-	-	-	-	-	-
wendal	O	O	O	O	O	O	O
xiaxiaofeng	O	-	-	-	-	-	-
ywjno	-	-	-	O	-	O	-
zhuer0632	O	-	-	-	-	-	-
zozoh	O	-	O	O	-	-	O

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz & XBlink （58444676 超级群）
* Nutz在微笑 （60504323）超级群
* Nutz ② 群 （68428921）超级群
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*
* Nutz的新浪微群
* Nutz的豆瓣小组
* Nutz的聊天室

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^!

欢迎访问官网,以获取 最新的快照版 和用户手册

14.4. 1.b.44 发行注记

14.4.1. 1.b.44 发行注记

又到了 Nutz 发布的日子，本次发布不像之前的几次，仅仅是一点微调，这次我们还是稍微的用了一点心思

* 我们支持了 Derby 数据库
* 为了能更灵活的支持 JSON 类的数据转换，将 JSON 包中的某些方法抽象成 Mapl 结构
* 之前一直被抱怨的 Nutz.Mvc 多国语言字符串，切换本地字符串时，那几个脑残的接口也被改的不那么脑残
* 最重大的修改是 Wendal 重构了 Scans 包，我想，这应该能成为我们更好支持 Maven 运行环境的基础

总之这个版，是很值得升级的。

说点新闻。前几天，我们的开源项目 在北京弄了一个沙龙一样的聚会，征集演讲主题，并投票决定是否给你分配演讲的时间。由于自知是一个小众的项目，因此如果事先知道还要被投票，我们可能就不会参加了。不过另我们意外的是，Nutz 水军力量似乎比较强大，我们获得了排名第二的投票。所以阴差阳错的去了。说起来，这似乎是第一次，我们在公开场合肉身谈论 Nutz 这个项目。记得当天早上，我神奇发了烧，嗓子发炎的厉害。以致被 胖五 坚决的认为我心理素质有问题，遇见大场面就需要尿不湿。

无论怎样，我们是去参加了这个活动，并且认真的准备了一组演讲稿:Nutz:一次刻意的意外 de 演讲稿

当天 胖五 录了一段未剪辑过的演讲视频。有兴趣的朋友请看这里:Nutz - 一次刻意的意外

这个活动是一个系列的活动，我想关注国内开源的朋友或多或少都听说过，有的还参加过。这里，我们希望活动的主创人员能够坚持下去，也祝愿这个活动越办越好。

因为要肉身现场和上百个人谈论 Nutz， Wendal 也就决定肉身过来听听。因此我心里上很感谢我们的开源项目，要不是他们，我可能到现在还没有见过 Wendal 的真人。 我们在一起度过一个愉快的周末，当然我们也录了很多 DV，因为太多，不能全放上来，我做了一点剪辑，后来发现加几个字配个背景音乐，可以完全当作一个 Nutz的宣传视频，有兴趣的朋友可以看这里 nutz_宣传片第0季

jeffsui 在 OSChain 上也投递了一篇相关的新闻IT范儿是什么范儿-nutz_宣传片第0季

期间，我就 Nutz 这个项目采访了过 胖五同学， 稍候我会做整理，把视频传上来。 胖五同学 对 Nutz 未来的一些设想，未来的几个月内将变成现实，我可以负责任的说: "虽然还需要继续做一些工作，但是基本已经没有什么悬念了。 "

---------- Nutz 的下载地址的分隔线 ----------

* 稳定版下载地址 : http://code.google.com/p/nutz/downloads/list
* 日编译下载地址 : http://build.sunfarms.net/nutz/
* Nutz 的主页： http://nutzam.com

14.4.2. 问题修复

* Issue 231 引入MapList, 取消JsonFilter 项目维护 Json Lang 文档 需求 by **juqkai**
* Issue 229 org.nutz.dao.util.cri.Exps.create方法NOT LKIE写错 项目维护 Dao by **hzl7652**
* Issue 228 ColType 字段类型太少 Dao 需求 重要 by **gnoloahs**
* Issue 224 Json.toJson 如果对象有枚举字段，将输出 null Json by **zozoh**
* Issue 222 maven项目引用其他子项目时，class文件不能被扫描 需求 by **for5million**
* Issue 218 NutzDao建表时 ，Blob转译问题(sqlserver2000/2005) Dao 重要 by **naily**
* Issue 216 Mvc 的本地化 问题 Mvc by **enzozhong**
* Issue 214 1.44对象扫描异常 项目维护 Lang by **albinhdk**
* Issue 209 nutz dao 能不能实现 update table set col=col+10 where id=1 这样的语句? Dao FAQ by **ming300**
* Issue 208 添加对SQLServer2012的支持 Dao 需求 重要 by **wendal**
* Issue 207 添加对Derby数据库的支持 by **wendal**
* Issue 206 Dao.execute如果传入null,会报Not implement yet! Dao 重要 by **wendal**
* Issue 205 JSON结构转换 by **juqkai**
* Issue 204 dao.query和动态sql中 分页 在sql2000中无效 by **j9dai**
* Issue 203 Scans 总是抛异常 项目维护 Lang 重要 by **zozoh**
* Issue 201 DAO对于pgsql的date类型问题 Dao 重要 by **enzozhong**
* Issue 200 重新实现Resouce包 项目维护 Lang 重要 by **wendal**
* Issue 198 Nutz Ioc找不到jar包内的类 Lang 需求 by **fnet123**
* Issue 195 Json中对entities缓存不能清空 Json 需求 by **weirhp**
* Issue 194 在多对多的映射关系中,NutzDao不支持SET类型！ Dao 需求 by **elkan1788**
* Issue 159 视图的填充占位符只能获取到req作用域以下的值。 Mvc 需求 by **godson741**
* Issue 83 Criteria对between and没有支持 不接纳 by **mengqiang81**
* Issue 34 国际化的一个小BUG Mvc 需求 by **zwt**

14.4.3. 质量

共通过了 781 个单元测试用例,代码覆盖率达到 70%(按line计算)

Nutz.Dao 经测试在如下数据库上可以工作正常

* H2
* SQLite -- 仅有限支持事务操作
* hsqldb
* MySql
* Oracle
* Postgresql
* SqlServer2005
* SqlServer2000

* DB2
* Derby

14.4.4. 文档

* 增加 Mapl 结构

14.4.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 NutzDemo 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献列表，我已经写了一个小程序，根据 Issue 列表来自动统计...

贡献者	问题	博客	支持	代码	示例	文档	测试
albinhdk	O	-	-	-	-	-	-
elkan1788	O	-	-	-	-	-	-
enzozhong	O	-	-	-	-	-	O
fnet123	O	-	-	-	-	-	-
for5million	O	-	-	-	-	-	O
gnoloahs	O	-	-	-	-	-	-
godson741	O	-	-	-	-	-	-

hzl7652	O	-	-	-	-	-	-
j9dai	O	-	-	-	-	-	-
jeffsui	-	O	-	-	-	-	-
juqkai	O	-	O	O	-	O	O
mengqiang81	O	-	-	-	-	-	-
ming300	O	-	-	-	-	-	-
naily	O	-	-	-	-	-	-
weirhp	O	-	-	-	-	-	-
wendal	O	O	O	O	O	O	O
ywjno	-	-	O	O	-	-	-
zozoh	O	-	O	O	-	-	O
zwt	O	-	-	-	-	-	-

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz & XBlink （58444676 超级群）
* Nutz在微笑 （60504323）超级群
* Nutz ② 群 （68428921）超级群
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*
* Nutz的新浪微群
* Nutz的豆瓣小组

* Nutz的聊天室

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^!

欢迎访问官网,以获取 最新的快照版 和用户手册

14.5. 1.b.43 发行注记

14.5.1. 1.b.43 发行注记

大家好，今天是一个值得纪念的日子。为了国荣哥哥，韩寒都开微博 ...

我们这次按计划发布，修了一批小 bug，这些 bug 基本都是小修改，再次感谢提交了这些 bug 的 Nutzer 们。并且颇有几个问题，是 Nutzer 通过 pull request 提交的修改。喜欢 Nutz 就 Fork 它。

14.5.1.1. 做一点呼吁

如果你在使用 Nutz 的过程中碰到了问题，最好从公开渠道提出你的问题。这样更多的人会从你的提问中受益，比如你可以发到:

* Nutz 的豆瓣小组
* Nutz 的新浪微群 128323
* Nutz 的问题列表

这样，我们可以更容易的追踪和分享你的问题。虽然你可能更偏爱 IM 工具，以为它可能会让你更快得到你要的答案，其实是一样的。

着急得到 Nutz 最新版，且懒得从源代码编译的同学，你们可以从 Nutz 的 Daily Build 随时获取最新的 jar，其实大多数时候，这个 jar 和每次发行版一样稳定。

关注 Nutz 时间规划的同学，可以关注 Nutz 的里程碑，不过请注意，我们定的里程碑表示 "我们希望"，而不是 "我们会确保"。因为这是一个 "自由且随意" 的项目，我们决定里程碑的日期，以及里程碑包含的特性也基本是 "自由且随意" 的。并且，我们认为只有这样，我们才能在品质上做到 "极致"

这几天连微博都禁评了，这里，我也少说几句吧，祝大家节日快乐 ^_^

---------- Nutz 的下载地址的分隔线 ----------

* 稳定版下载地址 : http://code.google.com/p/nutz/downloads/list
* 日编译下载地址 : http://build.sunfarms.net/nutz/
* Nutz 的主页 : http://nutzam.com

14.5.2. 问题修复

* Issue 188 HttpResponse针对有压缩的情况的处理Code Attached by **superhanliu**
* Issue 187 jar嵌套调用报错 需求 by **blueram**
* Issue 184 Objs对象转换bug Json by **juqkai**
* Issue 181 Mvcs.hasLocaleName方法的逻辑错误？ by **kevin0571**
* Issue 179 合并Files中isEquals(File f1, File f2) 和equals(File f1, File f2)两个方法 Code Attached by **gevinhjy**
* Issue 176 判断“a@b.c”这样的值的时候Strings.isEmail的返回值 by **ywjno**
* Issue 174 不知道什么原因Nutz就挂掉了 by **enzozhong**
* Issue 173 PropertiesProxy只加载了最后一个配置文件!! 重要 Lang by **wendal**
* Issue 172 nutz + oracl 10.1.0.2.0 FetchSize设值问题 Dao by **godson741**
* Issue 171 dao set,list,map对于String类型的支持 Dao 需求 by **enzozhong**
* Issue 170 当数据库连接池的连接数耗尽,NutDaoRunner会抛NPE 重要 Dao by **wendal**
* Issue 169 日志在OSGi环境下无法正常使用 Lang by **vurt**
* Issue 162 nutz json 循环引用，可以输出，但是不能还原 by **zozoh**
* Issue 158 dao对于set的支持 Dao 需求 by **enzozhong**
* Issue 155 页面String为空 映射 date问题 Lang by **ysenysen**
* Issue 147 SimpleCriteria类的toString方法未实现 Dao by **ywjno**
* Issue 136 是否需要更新Files类findFile方法的javadoc以及对应的wiki文档？ 文档 by **ywjno**
* Issue 130 自定义的 Sql 设置 Pager Dao by **gongrui**
* Issue 120 JSON循环引用的序列化与反序列化, 以及EL支持 Json 需求 by **juqkai**
* Issue 75 bug,实体解析报错 Dao 需求 by **cqyunqin**

14.5.3. 质量

共通过了 **781** 个单元测试用例,代码覆盖率达到 70%(按line计算)

Nutz.Dao 经测试在如下数据库上可以工作正常

* H2
* SQLite -- 仅有限支持事务操作
* hsqldb
* MySql
* Oracle
* Postgresql
* SqlServer2005
* SqlServer2000
* DB2

14.5.4. 文档

... 少量改动 ...

14.5.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 NutzDemo 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献列表，我已经写了一个小程序，根据 Issue 列表来自动统计...

贡献者	问题	博客	支持	代码	示例	文档	测试
Jay 蓝色幽默	-	-	O	-	-	-	-
Gevin	O	-	-	-	-	-	-
bluera m	O	-	-	-	-	-	-
conanc a	-	-	-	O	-	-	-
cqyunq in	O	-	-	O	-	-	-
enzozh ong	O	-	-	-	-	-	-
gevinhj y	-	-	-	O	-	-	-
godson 741	O	-	-	-	-	-	-
gongru i	O	-	-	-	-	-	-

juqkai	O	-	-	-	-	-	-
kevin0571	O	-	-	-	-	-	-
superhanliu	O	-	-	O	-	-	-
vurt	O	-	-	-	-	-	-
wendal	O	O	O	O	-	O	O
ysenysen	O	-	-	-	-	-	-
ywjno	O	-	-	-	-	-	-
zozoh	O	-	O	O	-	O	O

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz & XBlink （58444676 超级群）
* Nutz在微笑 （60504323）超级群
* Nutz ② 群 （68428921）超级群
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*
* Nutz的新浪微群
* Nutz的豆瓣小组
* Nutz的聊天室

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^!

欢迎访问官网,以获取 最新的快照版 和用户手册

14.6. 1.b.42 发行注记

14.6.1. 1.b.42 发行注记

大家好，情人节快乐 （或许你看到这段文字，已经是情人节之后了，那么就是情人节之后，快乐）

本版没啥好说的，我们修复了几个错误，决定在 1.b.43 前，发一版，因为 1.b.43 可能要等到3月底了。所以它是一个非常安全升级版

这期间，罗马de温泉 同学非常抢眼，给 Nutz 的GitHub 代码库 提交了 很多 Pull Request。啥都别说了，感动啊，眼泪 Hua Hua 的 ...

鉴于很多入门级朋友抱怨的 demo 不给力问题，我们会开始有重点的整理一下 Nutz Demo的一些文档和注释，当然也会慢慢 review 代码，让 demo 程序变得更好

关注 Nutz 的老朋友都知道，为了支持我们把 Web 程序做得更好，我们发布 Nutz 的同时也一直在维护下面这些项目:

* zDoc - 一个更友好的 wiki
* zMole - Nutz 的项目脚手架代码生成器
* NutzMore - Nutz 的扩展包
* NutzDiff - 序列比较算法

这些都是一些辅助程序包，或者简易工具 ...

但是今年刚刚开始 ...

* Howe Chiang 同学 召集了一个 问答类项目， 无敌的Wendal 以及 罗马de温泉 为这个项目贡献了主要的代码。
* 无敌的Wendal 创建了 一个基于 Nutz 的 Issue 管理系统，但还未完工 ...
* zozoh 同学 创建的 zTask 也接近完工。
* ...

是的，没错，**我们开始逐渐转向应用层面了**

虽然上面那些项目我们并不承诺一定要做的惊世骇俗，甚至我们都不承诺一定要做完，但是我们的确会不断的，快速得，试验性的写一些应用出来。如果这个应用不错，我们希望能持续投入，做的更好。 我们也将会在 nutzam.com 上有选择的部署几个成熟一点的应用 ...

如果有兴趣，你可以回头看看 这个页面，它写于 3 年前的某个寒冷的晚上， 那个时候这个小项目刚刚诞生，这篇文章说出了它存在的意义:

"在力所能及的情况下，最大限度的提高Web开发人员的生产力。"

现在，为了这句话，我们终于开始制作一些完整的应用了。 下面这句话，我忍了1年多，现在终于可是迫不及待的说出来了：

"Nutz 并不仅仅意味着一个 jar 包 ..."

到底它还意味着什么，今年，我们将拭目以待 ^_^

最后， 大家情人节快乐哦

---------- Nutz 的下载地址的分隔线 ----------

* 稳定版下载地址 : http://code.google.com/p/nutz/downloads/list
* 日编译下载地址 : http://build.sunfarms.net/nutz/
* Nutz 的主页： http://nutzam.com

14.6.2. 问题修复

* Issue 153 Mvc Setup 的 destroy 方法，读取不到 Ioc by **zozoh**
* Issue 146 给Cnd中添加limit方法 Code Attached by **ywjno**
* Issue 138 HttpStatusView返回403时,不会使用web.xml中的errorPage配置 文档 Mvc by **wendal**
* Issue 135 Images.java无法处理源文件是URL link的图片 FAQ by **ywjno**
* Issue 134 1.41没法找到At()中的路径 by **enzozhong**
* Issue 133 修改两个javadoc错误 Code Attached by **ywjno**
* Issue 131 无法回滚fastInsert后的数据 Dao 重要 by **fjay**
* Issue 60 希望能支持全局设置请求范围，这样不需要的时候可以不用写ioc Mvc by **cqyunqin**

14.6.3. 质量

共通过了 781 个单元测试用例,代码覆盖率达到 70%(按line计算)

Nutz.Dao 经测试在如下数据库上可以工作正常

* H2
* SQLite -- 仅有限支持事务操作
* hsqldb
* MySql
* Oracle
* Postgresql
* SqlServer2005
* SqlServer2000
* DB2

14.6.4. 文档

... 少量改动 ...

14.6.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 NutzDemo 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献列表，我已经写了一个小程序，根据 Issue 列表来自动统计...

贡献者	问题	博客	支持	代码	示例	文档	测试
Jay 蓝色幽默	O	-	-	-	-	-	-
cqyunqin	O	-	-	-	-	-	-
enzozhong	O	-	-	-	-	-	-
wendal	O	O	O	O	O	-	O
ywjno	O	-	-	O	-	O	O
zozoh	O	-	O	O	-	-	O

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz & XBlink （ 58444676 超级群 ）
* Nutz在微笑 （ 60504323 ）超级群
* Nutz ② 群 （ 68428921 ）超级群
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*
* Nutz的新浪微群

* Nutz的豆瓣小组
* Nutz的聊天室

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^!

欢迎访问官网,以获取 最新的快照版 和用户手册

14.7. 1.b.41 发行注记

14.7.1. 1.b.41 发行注记

大家好，经过2个多月，我们又见面了。兔年岁末这2个月，贺岁档很是让人失望，但是网络上却热闹的要命。方舟子和老罗陪我度过了很多有趣的夜晚，现在连韩寒都加入了。鉴于网调一边倒的局面，我就没啥好说的，龙年让我们搬个板凳继续看某斗士的嘴脸吧。

说到 Nutz 的这个版，我们主要进行了一些微小的调整，我相信大多数使用者都能平滑的从 1.b.40 升级到 1.b.41 -- 换一个 Jar 而已

本版，我们还不得不隆重介绍一下 Nutz 另外一位提交者 -- 胖五， 经过百般忽悠，终于诱使胖五 同学为 Nutz 做了好几个比较关键的提交，尤其是完善了 Dao 的 @Index 注解。这下声明一个 POJO，创建一个数据表就会更加轻松，以前 @Index 注解没有完全实现，所以大家必须的手工建立索引。 当然，对于高性能要求的应用，这个特性可能没啥用，因为索引的建立和调整，通常是需要 DBA 来做的，我们程序员定好表结构和主键就是了。但是一些中小型项目，程序员往往得兼 DBA，所以没啥好说得，@Index 注解会很帮你得忙的。

总之，龙年就要到了，一不小心又过了一年。从 09 年发起的这个小项目已经进入了它第三个年头。我记得当初发这个项目的时候，有一类很大的质疑声音就是：

“个人的项目不能保证长期维护，不敢用呀。”

为此，我们花了2年多的时间将它变成一个非个人项目，同时也证明了：

“嘿嘿，它会是一个长期的项目”

今年我们也关注了一下其他的语言，在自己的工作中也分别用了用 Python 之类的语言，有很多感悟，比如 wendal 同学的 这篇文章。 而且现在似乎 Java 过气了 的这种说法慢慢淡下去了，所以似乎这个小项目还会走上很长的一段时间。

通常来说，一个孩子，过了3岁，就不太容易夭折，一个公司过了3年，就不太容易倒闭。 因此这一年对这个小项目似乎很是关键，如果到了明年的这个时候，大家还能读到 1.x.xxx 的发行注记，那么我们可以负责的说，它真的会是一个长期的项目了。这一点也与国内其他做开源的同行一起共勉吧。喊的多响都木有用，踏踏实实的活三年下来，并且每周都有点进步，那么三年后，你的项目总是不会差的，而且多半还是很好的 ^_^

最后，这里代表 Nutz 所有的提交者（抱歉，来不及沟通了，你们暂时被戴表一下吧），祝大家

* 龙年行大运
    > 更少的 bug
    > 更多的技术积累
    > 更少的加班
    > 更多的奖金
    > 耶~~~ ^0^

14.7.2. 问题修复

* Issue 125 el中使用函数调用有问题 EL by **hujun82589167**
* Issue 124 动态实体功能不能正常使用 重要 by **goulin**
* Issue 119 为何从数据库里面取出二进制数据时会被转成String类型？ by **elkan1788**
* Issue 118 fastInsert存在问题 by **fjayblue**
* Issue 111 文档中"文件操作"有纰漏 文档 by **conanca**
* Issue 107 Nutz在JDK1.5下编译不过了 Dao by **pangwu86**
* Issue 106 Oracle下使用出现ora-01780问题 Dao by **pangwu86**
* Issue 101 智能判断字段值为空的时候生成is null语句 Dao by **sjbwylbs**
* Issue 100 JSON合并输出 by **juqkai**
* Issue 94 NutzMVC多主模块部署中Ioc冲突问题 Mvc by **netstarry**
* Issue 92 能不能搞个@comment注解 Dao by **pangwu86**
* Issue 90 @Index注解没有实现 Dao by **pangwu86**
* Issue 88 扩展Each接口 by **fjay**
* Issue 87 nutz项目有没有像Apache 的commons-confiugrations一样的工具类？ by **wanghaipeng789**
* Issue 86 希望EL能支持精度 EL FAQ by **qilicn**
* Issue 85 让Nutz Json支持多泛型 by **fjay**
* Issue 84 updateIgnoreNull隐藏的BUG by **JefWang**
* Issue 30 表单收集不支持List字段 Mvc 需求 by **fjay**

14.7.3. 质量

共通过了 738 个单元测试用例,代码覆盖率达到 64.5%(按line计算)

Nutz.Dao 经测试在如下数据库上可以工作正常

* H2
* SQLite -- 仅有限支持事务操作
* hsqldb
* MySql
* Oracle
* Postgresql

* SqlServer2005
* SqlServer2000
* DB2

14.7.4. 文档

* 修改: dao/customized_sql.man : 增加 SQL 逃逸字符的描述
* 修改: lang/lang.man : 增加文件/目录拷贝的描述
* 新增: ioc/ioc_properties.man 让Ioc容器帮你规划配置文件

14.7.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 NutzDemo 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献列表，我已经写了一个小程序，根据 Issue 列表来自动统计...

贡献者	问题	博客	支持	代码	示例	文档	测试
Jay 蓝色幽默	O	-	O	-	-	-	O
JefWang	O	-	-	-	-	-	-
conanca	O	-	-	-	-	-	-
elkan1788	O	-	-	-	-	-	-
goulin	O	-	-	-	-	-	-
hujun8	O	-	-	-	-	-	-

2589167							
juqkai	O	-	O	O	-	-	O
netstarry	O	-	-	-	-	-	-
pangwu86	O	-	O	O	-	-	O
qilicn	O	-	-	-	-	-	-
sjbwylbs	-	-	-	-	-	O	-
wanghaipeng789	O	-	-	-	-	-	-
wendal	O	O	O	O	O	-	O
zozoh	O	-	O	O	O	O	O
花米®	O	-	O	-	-	-	-

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz & XBlink （ 58444676 超级群 ）
* Nutz在微笑 （ 60504323 ）超级群
* Nutz ② 群 （ 68428921 ）超级群
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*
* Nutz的新浪微群
* Nutz的豆瓣小组
* Nutz的聊天室

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^!

欢迎访问官网,以获取 最新的快照版 和用户手册

14.8. 1.b.40 发行注记

14.8.1. 1.b.40 发行注记

Hi 大家好，好久不见，^_^

又到了30天一度的 Nutz 发布的日子了。 当然，上一个 30天我们木有发布，为啥涅？ 因为看看 Issue 列表，想再攒攒嘛。不过可以看出1.b.39 这个版 顶了2个多月，看起来工作的还可以。

我记得 1.b.39 这个版发布后 和 Wendal 私下小声嘀咕过，看起来再过几个版，Nutz 似乎可以 RC 了。 说起 Wendal 同学，如果你仔细阅读他的博客，你会发现，这个博客很有料，文章的品质很高，而且更新的频率也不低，这里推荐几篇文章给大家:

* 《初试Jetty使用Mongodb作为Session管理器》
* 《Nutz事务模板值得注意的细节》
* 《捐点钱给自己喜欢的开源软件》
* 《JGit初试牛刀》
* 《实现Rk2918的System分区可写》

技术博不少，深度广度兼具的技博难找，Wendal 同学 的博客已经初具摸样了，作为它的老订户，不得不说:"此博值得一订呀"。这里是 Wendal随笔的RSS订阅地址。

回过头来我们再说说 Nutz:

日子过的很快，子在川曰：“逝者如斯夫”，看看 Nutz 在 Google code 的首页，上面有一句: "它没有过去，只有未来"。 现在再说这话似乎有点不好意思的，都2年了，你还跟这儿装嫩，你好意思吗？ 但是现在到也实在不想把这句话删掉，因为它可以做一个时间的铭，告诉我们这个小开源项目是无所谓过去的，它不关心自己所谓的用户量，也不关心自己所谓的影响力，更不关心和现在的所有流行代码库之间的比较。它不过是一些喜欢写程序的人凑在一起，写一些自认为很有用的代码，并且希望更多的人使用自己的代码，然后和朋友吃饭喝酒的时候，可以吹个牛X，就非常满足了。

我们的策略很简单，就是，希望大家来这里报Bug，然后我们修复。当然我们欢迎任何喜欢这个游戏的朋友们参与进来，你可以在github 上 fork Nutz 然后，给出你的修改，给我们发 pull request，我们接受了，贡献者名单就一定有你。

喜欢 Nutz，就 fork 它 ^_^

在 Nutz 正式版发布的时候，我们也打算把所有版本的贡献者，放到代码里，做成一个常量，算是一个永久的纪念吧。

为此，特地感谢 ywjno 同学，现在似乎只有他提了不少 pull request

一件神奇的事情

此外，还有一件神奇的事情，ClarenceAu 同学在某年某月静悄悄的提交了 Nut 在 Netbeans 上的插件，经过数小时调查，有证据显示该插件系由 **华南农业大学人机交互工作室** 集体创作，试用过的用户分分表示"有点意思"，据相关人士透露，该工作室由一伙不明身份的大学生构成，数目未知，据一份不完全名单透露，下面这些帐号与该工作室有非常紧密的联系，甚至很可能是其中一员：

* ClarenceAu
* steven0lisa
* Niuza
* old_oU
* 华农金中菊
* 钟泽明
* KevinO凯

其中，我们可以确切的知道， KevinO凯 是 迷你志的作者，它是一个基于Nutz的网站，并且作者曾经表示过要开源。想要 迷你志 源代码的同学们，冤有头债有主，本家儿我终于给你们找到了!

推介

如果有对数据转换有需求的同学，XBlink 项目是非常值得关注的。它的切入点非常犀利，就是追求一个字 --- **快**，我们期待着他们能搞出一些更酷的代码，大家拿来用用哈 :D

最后 ...

* Nutz 的首页 : http://nutzam.com/
* Nutz 的下载地址 : http://code.google.com/p/nutz/downloads/list

Enjoy it ^_^

14.8.2. 问题修复

* Issue 73 Bug,org.nutz.dao.FieldMatcher类38行 by **cqyunqin**
* Issue 71 应用启动后,客户端用Chrome登录,同时请求某个Module类的两个入口方法的URL时,其中一个方法中抛空指针异常 Ioc Mvc by **conanca**
* Issue 70 NUTZ DAO @DEFAULT注解存在一定问题 by **JefWang**
* Issue 65 AOP调试信息不够 Ioc by **qianshan**
* Issue 62 源代码放入web项目报2个js错误 Mvc by **cqyunqin**

* Issue 59 WIKI中中说多对多关系，插入使dao.insertLinks()有所误导。 by **liuyxit**
* Issue 52 Json.toJson 可不可以对某些字段特殊处理 Json 需求 by **xbl**
* Issue 50 建议在IOC手册文档里新增注解列表 Ioc 文档 by **JefWang**
* Issue 47 nutzmole-1.2生成的代码。添加时不能自动封装对象 Mvc by **godson741**
* Issue 46 获取当前会话的 Locale 名称 不支持EL表达式？ Mvc 文档 需求 by **zwtlong**
* Issue 45 39中于dao的问题，当条件为null 或空字符串时出错 Dao by **yunhaifeiwu**
* Issue 44 Json.fromJson(null)报错 Json by **yunhaifeiwu**
* Issue 42 国际化配置文件注释前面有空格会报空异常？ Mvc by **zwtlong**
* Issue 41 nutz-1.b.39在glassfish v2.1环境出现类初始化失败 Dao 重要 by **shinwell**
* Issue 36 表达式引擎的使用报错 重要 by **dengqi100**
* Issue 35 DAO多对多查询出错 by **liuyxit**
* Issue 33 实现Setup接口类里面不能使用注入方式？希望可以改进 Mvc 需求 by **zwt**
* Issue 32 mvc JsonAdaptor遇对象集合适配有问题 Mvc 重要 by **hujun82589167**
* Issue 29 实体用继承Pojo的方式生成字段，生成的ID排到字段的最后面去了 by **crab041**
* Issue 26 url匹配的问题 Mvc by **hszdz**
* Issue 22 在Setup的init方法中，通过Mvcs.getAtMap(config.getServletContext())得到的AtMap的size为0 Mvc 需求 by **conanca**
* In google code
* Issue 513 关于类org.nutz.ioc.meta.IocEventSet的描述错误 by **ronggen...@gmail.com**
* Issue 514 上传文件 获取本地文件名 tempFile.getMeta().getFileLocalName()可能不正确的问题 by **maiger...@gmail.com**
* Issue 519 1.b.39的manual文档中关于El的例子 by **lanlon.zen**
* Issue 521 RecurArrayRandom 在多线程环境下会出错 by **feiyan35...@gmail.com**
* Issue 522 关于dao数据删除接口dao.clear()调用返回值的问题 by **hjingfen...@gmail.com**
* Issue 524 Nutz表达式引擎的问题 by **chaoshi...@gmail.com**
* Issue 528 使用dao进行数据插入，Chain条件报错 by **wak...@gmail.com**
* Issue 531 关于nutz1.b.39中po属性类型为clob无法查询 by **shao0...@163.com**

14.8.3. 质量

共通过了 738 个单元测试用例,代码覆盖率达到 64.5%(按line计算)

Nutz.Dao 经测试在如下数据库上可以工作正常

* H2
* SQLite -- 仅有限支持事务操作
* hsqldb
* MySql
* Oracle
* Postgresql
* SqlServer2005

* SqlServer2000
* DB2

14.8.4. 文档

* 修改: 文档过滤器
* 添加: Ioc注解列表

14.8.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 NutzDemo 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献列表，我已经写了一个小程序，根据 Issue 列表来自动统计...

贡献者	问题	博客	支持	代码	示例	文档	测试
conanca	O	-	-	-	-	-	-
cqyunqin	O	-	-	-	-	-	-
crab041	O	-	-	-	-	-	-
dengqi100	O	-	-	-	-	-	-
godson741	O	-	-	-	-	-	-
hszdz	O	-	-	-	-	-	-
hujun8258916	O	-	-	-	-	-	-

7							
JefWang	O	-	-	-	-	-	-
liuyxit	O	-	-	-	-	-	-
qianshan	O	-	-	-	-	-	-
shinwell	O	-	-	-	-	-	-
wendal	-	O	O	O	O	O	O
xbl	O	-	-	-	-	-	-
yunhaifeiwu	O	-	-	-	-	-	-
ywjno	-	-	-	O	-	-	-
zozoh	O	-	O	O	-	O	O
zwt	O	-	-	-	-	-	-
zwtlong	O	-	-	-	-	-	-

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz & XBlink （58444676 超级群）
* Nutz在微笑 （60504323）超级群
* Nutz ② 群 （68428921）超级群
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*
* Nutz的新浪微群
* Nutz的豆瓣小组
* Nutz的聊天室

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^!

欢迎访问官网,以获取 最新的快照版 和用户手册

14.9. 1.b.39 发行注记

14.9.1. 1.b.39 发行注记

又发布了？ 是的，因为这是一个 1.b.38 的 Bug Fix 版。Bug Fix 版发布就要快嘛，嘿嘿，你看人家 Firefox ^_^

当然这么快发布还有一个原因 =》 Issue 23

1.b.38 的 Dao 部分进行了重大的调整，这几周来，似乎没有被大家抱怨特别的不适，Dao 的变动带来的影响比我想想的小多了，因此强烈推荐你升级到这个版本

* 这里是 Nutz 的下载列表 : http://code.google.com/p/nutz/downloads/list
* 木有听说过 Nutz 的同学请看 : http://nutz.googlecode.com

本版涉及到的问题，有兴趣的同学请看下面的问题列表，这里就不废话了。

还有，天气热，气压低，国度神奇，希望8月份的人民注意防暑，胖五同学做动车时不要轻易睡着，我会持之以恒的为你祈祷的 :P

14.9.2. 问题修复

* Issue 28 NutzQuickStart 木有与时俱进 by **fjay**
* Issue 25 Mvc：使用UploadAdaptor时，ArrayInjector和NameInjector注入的参数值错误 Mvc by **lAndRaxeE**
* Issue 24 主模块声明应用的子模块时，能够直接指定包的相对路径，来搜索子模块 Mvc 需求 by **conanca**
* Issue 23 一对多关联，调用insertWith方法关联新增后，发现关联属性字段为空 Dao by **fnet123**
* Issue 22 在Setup的init方法中，通过Mvcs.getAtMap(config.getServletContext())得到的AtMap的size为0 Mvc 需求 5 comments by **conanca**
* Issue 21 DB2环境上NutzDao的fastInsert失败 by **nneverwei**
* Issue 17 dao.delete 需要支持传入一个集合或者数组 Dao by **zozoh**
* Issue 15 Mirror类无法获取对象中的静态变量 by **happyday517**
* Issue 14 Nutz.dao获取实体的回调，实体类中的非数据表字段未被赋值 Dao FAQ by **conanca**
* Issue 13 1.38执行sql语句有问题. Dao FAQ by **wangyingdong**
* Issue 12 发布 Nutz1.b.39 项目维护 by **zozoh**
* Issue 11 1.b.38Model类中@Readonly注解失效 Dao 重要 by **gevinhjy**

* Issue 10 测试与JDK7的兼容性 项目维护 by **wendal**

14.9.3. 质量

共通过了 710 个单元测试用例,代码覆盖率达到 64.5%(按line计算)

Nutz.Dao 经测试在如下数据库上可以工作正常

* H2
* SQLite
* hsqldb
* MySql
* Oracle
* Postgresql
* SqlServer2005
* SqlServer2000
* DB2

14.9.4. 文档

没有更新

14.9.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 NutzDemo 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献列表，我已经写了一个小程序，根据 Issue 列表来自动统计...

贡献者	问题	博客	支持	代码	示例	文档	测试
Gevin	O	O	-	-	-	-	-
Jay 蓝	O	-	-	-	-	-	-

色幽默							
conanca	O	-	-	-	-	-	-
fnet123	O	-	-	-	-	-	-
lAndRaxeE	O	-	-	-	-	-	-
nneverwei	O	-	-	-	-	-	-
wangyingdong	O	-	-	-	-	-	-
wendal	O	O	O	O	O	-	O
zozoh	O	-	O	O	-	-	O
幸福的旁边	O	-	-	-	-	-	-
胖五	-	O	-	-	-	-	-

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz & XBlink （ 58444676 超级群 ）
* Nutz在微笑 （ 60504323 ）超级群
* Nutz ② 群 （ 68428921 ）超级群
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*
* Nutz的新浪微群
* Nutz的豆瓣小组
* Nutz的聊天室

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^!

欢迎访问官网,以获取 最新的快照版 和用户手册

14.10. 1.b.38 发行注记

14.10.1. 1.b.38 发行注记

自 Nutz 1.b.37 发布以来，时隔近3个月。为什么间隔这么时间涅？ 因为我们做了很多重大的调整：

1. 重构了 Dao -- 兑现了我们之前的承诺
2. 重构了 EL
3. Nutz 的源码管理迁移到了 Github 上

以后，希望大家报 Issue 到 Nutz Github 的问题列表 中。Google Code 的问题列表还有66个 Issue 没有处理，我们会在后续的版本中尽快修复。如果都修复了，我们就会关闭 ~~Google Code 的问题列表~~，只维护 Github 的问题列表。

另外，如果你在 Github 上有帐号，欢迎随时 fork Nutz，请记住我们的口号就是： **喜欢 Nutz，就 Fork 它**

当然，Nutz 的下载地址仍然一直会是 Google Code 下载列表，并且 Nutz 在 Google Code 的项目主页 也会一直维护。

同时，我想提醒大家注意一下 Nutz 的官网，我们会不断的充实它的内容，比如最近做的

* Nutz 的文档
* Nutz 的小白测试

说到小白测试，不得不提一下 Dao 重构。我们这次重构 Dao 后总是不放心，其实我们在今年 5 月初就重构完了，然后我们测试呀，测试，毕竟是重构 Dao 了嘛，但是我们还是不放心，于是又测试呀测试... 但是我们还是不放心...

于是 Juqkai 同学在我和 Wendal 的撺掇下痛下决心，写了这个应用:

Nutz 小白测试计划

我们希望这个计划，能让更多人方便的参与测试，提交测试结果，这样我们就能为更多的人提供更稳定的 Jar 包。同时，我们也能更准确的统计贡献者名单。

当然如果没有人参与这个测试计划，最差的结果就是维持现状。令人欣慰的是，还是有4位同学(包括我)参加了1.b.38测试的小白测试

那么，让我们看看，以后的版本参加的人是会越来越多，还是越来越少 ^_^!

14.10.2. Dao 兼容性问题

作为一篇发行注记，这次我们要上点干货了

1.b.38 的 Dao 与之前的 Dao 使用上的主要区别

* 更快的批量操作 -- 比如大数据量的插入
* 提供了 Criteria 接口，扩展了 Condition 接口，这样查询的时候，可以用 PreparedStatement 参数
* 同时 Criteria 接口也非常方便你组织更复杂的 SQL 条件
* 自定义 SQL 可以设置 fetchSize，但是依然不能支持 pager，你还得自己用 SQL 方言来翻页
* 重新设计的 Entity ，可以方便的扩展，这样有些偏爱 JPA 或者配置文件的同学，可以扩展自己的实体配置方式
* 重新设计的 LOG，可以让 SQL 打印的更清晰
* 你甚至可以 dao.insert 或者 dao.update 一个 Map
* 支持 dao.create/dao.drop 方式来建表和删表
* org.nutz.dao.Dao 原有的接口函数统统保持不变，从而保证了兼容性不会有太大问题。

为了上述的优点，我们放弃了一点点兼容性，你的项目如果用 Nutz.1.b.38 可能需要少量修改几行代码。

Cnd 类的兼容问题

当然，如果你直接实现 Condition 接口也不会有问题。

* Cnd.exp 的返回值类型变成了 SqlExpression
* Cnd.exps 的返回值类型变成了 SqlExpressionGroup

Entity 接口的兼容问题

极个别很有 Hacking 精神的同学使用 Nutz 的时候，使用了 Entity 类，但是不幸的是现在 Entity 已经变成一个接口并且某些方法已经变了名字，但是原来 Entity 类所有的功能，现在 Entity 接口都能提供。如果你发现某些方法找不到了，耐心看看新的 Entity 接口的定义，我想你很快就能找到你要找的方法。

最重要的一个改变，就是原来的 entity.fields() 方法，变成了 entity.getMappingFields()。这个是有同学向我抱怨过，因此觉得有必要在这里特别提一下。

org.nutz.dao.tool 包的兼容问题

某些很有探索精神的同学可能偶然发现了 Nutz 还隐藏了一个 Dao 工具类的包，里面的方法可以跨数据库的建表，所以很有可能在自己的项目里也这么应用了。但是不幸的是，1.b.38 之后，这个包没了。因为我们不再需要它了。你的建表可以用:

```
[CODE]
dao.create(Pet.class, true);  // true 表示如果存在，先 DROP 掉再建,
false 表示如果存在就不建了
```

来完成。当然，删表，可以用:

```
[CODE]
dao.drop(Pet.class);
```

除此之外，Nutz.Dao 不会有和之前不兼容的地方了。当然如果你发现了，请随时告诉我们，应该都不是大问题。

14.10.3. EL 兼容性问题

EL 作为比较新的一个功能，可能用的人比较少，我们之后会再各个模块里逐渐发掘 EL 的潜力。本次发布 EL 经过了重构，效率提升了将近1倍。当然，作为一个用反射实现的东东，它的还是很慢，但是我们设计 EL 的时候假定它的使用场景是配置文件，后台进程等一些不是非常需要效率的地方。它可以让你的程序更有弹性。如果非常需要效率的地方，恐怕它不是一个很好的选择。

重构后的 EL，eval 的结果不再是 ElValue，而是普通的 Object，这样，你使用的时候会更方便一些。

并且EL的预编译，不再是

```
[CODE]
BinObj exp = El.compile("3+4");
```

而是

```
[CODE]
El exp = new El("3+4");
```

看起来更清爽一些不是吗？ Juqkai 同学的设计 ^_^

14.10.4. 问题修复

* Issue 34 dao: support JPA Annotation by **zozoh**
* Issue 85 NutDao 中需要提供批量更新操作的方法. by **ming300**
* Issue 121 Nutz是否有根据实体上的注解来自动创建数据库表等对象的计划？ by **hzzdong**
* Issue 137 数据库操作是否可以增加对Blob类型的支持 by **Toni.xutao**
* Issue 155 Dao 的 @Column注解可否标注在getter函数上 by **jinghui70**
* Issue 192 NutDao性能问题：关于批量操作，使用PreparedStatement的Batch功能 by **hzzdong**
* Issue 230 关于nutz dao能否动态根据tables.dod 更新表结构 by **shao0707**
* Issue 267 NutzDao能否提供设置fetchSize的方法 by **superxlm1985**
* Issue 294 Nutz Dao 大量数据插入效率问题 by **fjayblue**
* Issue 332 sqlserver2005 nutz@google Junit测试 by **haoyoushuai1986**
* Issue 416 建议nutz提供一个拼接复杂的查询条件的方法 by **liuxiaogang1987**
* Issue 420 Sqls.create语句中包含'@'会出错。 by **wangyingdong**
* Issue 426 Condition Cnd.where + orderBy by **muyushi85**
* Issue 433 MVC的代码覆盖率实在太低,很多都没测试用例! by **wendal**
* Issue 437 SqlLiteral中的@过滤 by **zhuyingxi**
* Issue 457 Json:增加字段映射方式 by **wendal**
* Issue 458 ActionChainMakerConfiguration类名拼写错误 by **jentrees2008**
* Issue 466 使用MySQL数据库，使用Nutz.dao的自定义SQL报错 by **mamacmm**
* Issue 469 Json.toJson 对char类型没有加引号 by **lwk0571**
* Issue 471 Dao: 添加对Hsql数据库的支持 by **wendal**
* Issue 473 使nutz兼容OSGi by **windywany**
* Issue 478 1.b.38 的新 Dao 应该忽略 Column 不存在的情况 -- by Jay by **zozoh**
* Issue 479 38 update 问题 by **fjayblue**
* Issue 482 将Nutz打成jar， 2 个 Scans 的测试过不了 by **zozoh**
* Issue 485 EL表达式引擎在某些情况下会出现错误 by **ywjno.dev**
* Issue 487 ioc中某个属性的值是Map时,如果是null偶尔报：String can not cast to java.util.Map by **superhanliu**
* Issue 489 AnnotationIocLoader类对set方法处理@Inject 有bug by **feiyan**
* Issue 490 38的Dao,在DB2下,使用@Column('abc')的话,insert会报错 by **wendal**
* Issue 493 Lang的str2number方法在某些情况下会出错 by **ywjno.dev**
* Issue 494 自定义SQL wiki by **superhanliu**
* Issue 497 Nztz.Dao 注释没写全，和注释重写 by **hongchongyuan**
* Issue 500 nutz.Json 应扩展一下@JsonField注解 by **conanca**

14.10.5. 质量

共通过了 697 个单元测试用例,代码覆盖率达到

Nutz.Dao 经测试在如下数据库上可以工作正常

* H2
* SQLite
* MySql
* Oracle
* Postgresql
* SqlServer2005
* SqlServer2000
* DB2

14.10.6. 文档

修改了很多文档 ...

14.10.7. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 NutzDemo 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献列表，我已经写了一个小程序，根据 Issue 列表来自动统计...

贡献者	问题	博客	支持	代码	示例	文档	测试
Toni.xutao	O	-	-	-	-	-	-
caji.net	O	-	-	-	-	-	-
conanca	O	-	-	-	-	-	-
feiyan	O	-	-	-	-	-	-

fjayblue	O	-	-	-	-	-	-
haoyoushuai1986	O	-	-	-	-	-	-
hongchongyuan	O	-	-	-	-	-	-
hzzdong	O	-	-	-	-	-	-
jentrees2008	O	-	-	-	-	-	-
jinghui70	O	-	-	-	-	-	-
juqkai	O	-	O	O	-	O	O
liuxiaogang1987	O	-	-	-	-	-	-
lwk0571	O	-	-	-	-	-	-
mamacmm	O	O	-	-	-	-	O
ming300	O	-	-	-	-	-	-
muyushi85	O	-	-	-	-	-	-
shao0707	O	-	-	-	-	-	-
shine	-	-	-	-	O	-	-

superhanliu	O	-	-	-	-	-	-
superxlm1985	O	-	-	-	-	-	-
wangyingdong	O	-	-	-	-	-	-
wendal	O	O	O	O	O	O	O
windywany	O	-	-	-	-	-	-
ywjno.dev	O	-	-	-	-	-	-
zhuyingxi	O	-	-	-	-	-	-
zozoh	O	-	O	O	O	O	O
Jay 蓝色幽默	O	-	-	O	-	-	O

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz & XBlink （58444676 超级群）
* Nutz在微笑 （60504323）超级群
* Nutz ② 群 （68428921）超级群
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*
* Nutz的新浪微群
* Nutz的豆瓣小组
* Nutz的聊天室

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^!

欢迎访问官网,以获取 最新的快照版 和用户手册

14.11. 1.b.37 发行注记

14.11.1. 1.b.37 发行注记

春暖花开，万物复苏，那啥的那啥在地震后那啥了；那谁谁因为用飞架炸了自己的那谁谁而被那谁谁给那啥了；当然还有那谁因为不减肥而且还要做飞机去做展会，在机场被伟大的那啥给那啥了...我们这个世界悲惨的丰富多彩着。和这个世界的多彩相比，我想 Nutz 这个小项目的动静简直是不值一提的。尤其是这次更新 ...

我们这次发布的是一个 bug fix 版，没有重大改动

这个版主要解决了前一个版本 1.b.36 的某些问题。前一个版本因为重写了 Mvc 的核心逻辑，因此带来了一点点不太严重的小问题。在这个版（1.b.37）基本都被我们修了，总之，我可以负责任的告诉你，这个版，比 1.b.36 要好那么一点点 ^_^

还有一件事: Wendal 同学 录制了两款视频教程，分别在这里和这里，无论怎样，你在 Nutz 的下载列表 就能找到这两个文件，强烈推荐刚接触 Nutz 框架 的同学看一下

另外需要推介的是 QinerG 同学写的两篇文章

* 《让Nutz支持最快的模板引擎Smarty4j
* 《在Nutz框架中提供Pojo校验（验证）功能的支持》

我们已经收录在 Nutz 的推荐文章列表中了。看到这样的扩展很欣慰，因为我们的确希望大家这么来扩展和使用 Nutz 的，这个小框架就是为了让你很方便的扩展而设计的。它可以很容易的让你进行高度的订制化，并且它还有很多很方便的扩展点在等着你去挖掘 ^_^

最后，和大家预告一件事情，下个版（1.b.38），我们会提供一个全新的 Dao，当然，主要接口不变，你原来使用的

* Dao
* Condition
* Sql & SqlCallback

这三个接口的用法会维持不变，其中 Dao 和 Sql 接口会有所增强。新的 Dao 会解决批量插入，Blob 等问题。同时会带来一些更酷的功能，到时候我们会提供一些相关的说明文档

如果不出意外下个版会在6月的某一天发布 :)

下面是本版（1.b.37）的一些细节...

14.11.2. 问题修复

* Issue 298 文档的覆盖不够 by **zozoh**
* Issue 323 建议对REST的改进 by **wenglonghui**
* Issue 346 Json.fromJson 对Long类型有问题 by **lvjinhua**
* Issue 347 应该设计一个PostActionFilter或者改造现在的ActionFilter by **moreztea**
* Issue 369 mvc:PairAdaptor在处理REST风格的URL时对参数的解析不够智能 by **Landraxee**
* Issue 383 1.b.35Rollback SQLException by **幸福的旁边**
* Issue 410 关于EL中的StringElValue by **juqkai**
* Issue 412 UploadAdaptor不支持路径参数 by **天行健**
* Issue 417 入口方法中包含数组参数时,启动报错 by **幸福的旁边**
* Issue 418 redirect时null参数失效 by **jdomyth**
* Issue 419 重构后的 Mvc 创建 Module 的效率 by **zozoh**
* Issue 425 "传人" 应该改为"传入" by **wczhang.china**
* Issue 427 3.6版，@Fail不自动把Exception放进request的obj了！！ by **zvin.strategy**
* Issue 429 当直接请求一个jsp页面时，null取不到 by **caji.net**
* Issue 430 org.nutz.lang.meta.Pair by **goodyorkye**
* Issue 432 文件池的 默认构造方法中 文件池的大小 不妥 by **feiyan**
* Issue 434 升级到nutz1.b.36后, json解析出错. by **wenzhihong2003**
* Issue 435 使用37rc的jar包的时候出现的无法更新数据库的情况 by **ywjno.dev**
* Issue 438 当URL中使用EL表达式时,每次都扫描OPT类,效率低下 by **Wendal**
* Issue 439 ioc:XmlIocLoader的parseField对内部对象的解析错误 by **Landraxee**
* Issue 440 ioc:XmlIocLoader的parseMap解析错误 by **Landraxee**
* Issue 450 FreeMarker视图下，在注解@OK里使用EL表达式报错 by **mamacmm**

14.11.3. 质量

共通过了 **697** 个单元测试用例,代码覆盖率达到

Nutz.Dao 经测试在如下数据库上可以工作正常

* H2
* SQLite
* MySql
* Oracle
* Postgresql
* SqlServer2005
* SqlServer2000
* DB2

14.11.4. 文档

* 增加 REST 的支持
* 修改 适配器>路径参数

* 修改 Nutz.Mvc 注解一览表

14.11.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 Demo Site 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献列表，我已经写了一个小程序，根据 Issue 列表来自动统计...

贡献者	问题	博客	支持	代码	示例	文档	测试
caji.net	O	-	-	-	-	-	-
feiyan	O	-	-	-	-	-	-
goodyorkye	O	-	-	-	-	-	-
jdomyth	O	-	-	-	-	-	-
landraxee	O	-	-	-	-	-	-
juqkai	O	-	-	O	-	-	O
lvjinhua	O	-	-	-	-	-	-
mamacmm	O	-	-	-	-	-	-

morezt ea	O	-	-	-	-	-	-
QinerG	-	O	-	-	-	-	-
wczhan g.china	O	-	-	-	-	-	-
wendal	O	O	O	O	O	O	O
wenglo nghui	O	-	-	-	-	-	-
wenzhi hong20 03	O	-	-	-	-	-	-
ywjno. dev	O	-	-	-	-	-	-
zozoh	O	-	O	O	-	O	-
zvin.str ategy	O	-	-	-	-	-	-
天行健	O	-	-	-	-	-	-
幸福的 旁边	O	-	-	-	-	-	-

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz & XBlink （58444676 超级群）
* Nutz在微笑 （60504323）超级群
* Nutz ② 群 （68428921）超级群
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^!

欢迎访问官网,以获取 最新的快照版 和用户手册

14.12. 1.b.36 发行注记

14.12.1. 1.b.36 发行注记

Hi, 大家好，又是我。

接着本次发布的机会，我们要 highlight 一位开发者 -- **juqkai**。

在他和 Wendal同学的前后鼓动下，我们终于毅然的重写了Mvc的核心加载逻辑。

现在的 Mvc 更加灵活，至于如何灵活，大家可以读读 这篇文档。

同时，我发现 **juqkai** 同学也是一位对代码的品质有执着追求的同学，比如他毅然的重构了 Nutz 的 JsonParsing，记得当时我写这个类时，就是一个大函数，一个长长的 switch...case，因为我想:"JSON 这点简单的东东，还用弄个什么结构吗？"。 但是后来我发现代码越来越长，也犹豫过："要不要重构一下，起码弄几个私有函数嘛。"。 但是后来想:"算了，反正长也不过 400行。我还要留点时间看火影的好 ^_^"

但是，我这点懒散被目光如炬的 **juqkai** 的同学发现了，他挺身而出，完成了我一直想做但可耻的没有做的事情。实话说，这种精神已经感动我好几个礼拜了。 喂神马！喂神马！又出现一个执着于代码品质的提交者涅？

这版 Nutz 经过这次重构，解决了很多 Mvc 方面潜在的问题。 这时，似乎大家又把目光投向了 Dao。 恩，是的，我们打算重构一下它。彻底的重构。

有些人可能会担心，你们这么乱搞，代码的正确性如何保证啊？ 我们基本不太担心，我们的代码质量现在基本由 600 多个 JUnit 来保证。每当有人报 Issue，我们会尽可能添加 JUnit 来重现。 我记得在早些时候，我认为如果项目的 JUnit 到了 1000 个，那么代码的质量应该是坚若磐石的。现在我们可以说，Nutz 差不多是一块 65% 的磐石。 即使我们肆无忌惮的重构，它的质量也不太会发生什么大的变化。所以，很有可能在之后的1 - 2个版本，我们会重构 Dao.

14.12.2. 问题修复

* Issue 204 Mirror表达式（嵌套属性的支持） by **jiongs753**
* Issue 350 实现更优雅的URL by **wesnow**
* Issue 354 Mvc:是否可以在执行ActionFilter之前进行参数适配，并将适配后的参数数组作为传入ActionFilter by **landraxee**
* Issue 381 关于上传文件正则表达式匹配可以支持的文件名报错的问题 by **axhack**
* Issue 382 能否考虑将DefaultEntityMaker中的哪些私有的方法改为protected的? by **a357857613@qq.com**
* Issue 384 换成 JSON 字符串时，\u转义有误 by **fjayblue**
* Issue 388 Json: 多线程的时候，转换时间类型到字符串会出现错误 by **liaojiaohe**

* Issue 389 Cnd拼条件时使用not in，生成的SQL文不正确 by **pangwu86**
* Issue 392 配置正确的url路径却进不到对应的Action by **pangwu86**
* Issue 393 Feature: Please support variant in the locale message by **袁青云**
* Issue 394 路径参数最好定义成包装类型 by **wesnow**
* Issue 396 处理异常信息的时候出现异常 by **fjayblue**
* Issue 397 El.eval在某些算式中出现的问题 by **ywjno.dev**
* Issue 399 WebResourceScan.list NullPointerException by **jiongs753**
* Issue 404 FailProcessor 异常时不会在控制台打印信息 by **juqkai**
* Issue 411 WebResourceScan 不能扫描 classpath 下面的 jar 包. by **Json.Shen**

14.12.3. 质量

共通过了 664 个单元测试用例,代码覆盖率达到

Nutz.Dao 经测试在如下数据库上可以工作正常

* H2
* SQLite
* MySql
* Oracle
* Postgresql
* SqlServer2005
* SqlServer2000
* DB2

14.12.4. 文档

* 增加 动作链

14.12.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 Demo Site 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献列表，我已经写了一个小程序，根据 Issue 列表来自动统计...

贡献者	问题	博客	支持	代码	示例	文档	测试
A357857613	O	-	-	-	-	-	-
Axhack	O	-	-	-	-	-	-
Fjayblue	O	-	-	-	-	-	-
Json.Shen	O	-	-	-	-	-	-
Juqkai	O	-	-	O	-	-	O
Landraxee	O	-	-	-	-	-	-
Liaojiaohe	O	-	-	-	-	-	-
Pangwu86	O	-	-	-	-	-	-
Wendal	O	O	O	O	-	-	O
Wesnow	O	-	-	-	-	-	-
Ywjno.dev	O	-	-	-	-	-	-
zozoh	O	O	O	O	-	O	O
天行健	O	-	-	-	-	-	-
袁青云	O	-	-	-	-	-	-

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz & XBlink （58444676 超级群）
* Nutz在微笑 （60504323）超级群
* Nutz ② 群 （68428921）超级群
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^!

欢迎访问官网,以获取 最新的快照版 和用户手册

14.13. 1.b.35 发行注记

14.13.1. 1.b.35 发行注记

对于春节，很多朋友的总结是 -- **没劲**

对于情人节，以前也有朋友反映 -- **纠结**

因此，基本上伴随着 **没劲** 和 **纠结**， Nutz 兔年发布了第一版 -- 1.b.35

这个版本膨胀了 110K， 整个 Jar 文件达到了可观的 898K。 为什么呢？ 因为我们在 Nutz 里内置了表达式引擎我们简单的整理了一下文档，因为几个月以来，将某些在社区内比较集中的问题写成文档。希望能让新手阅读文档时，少一些疑惑

同时，这两周来让我也有点小感触：

> *[CODE]*
> 元旦过后，人心惶惶是个常态。
> 我当时觉得可能一直到正月十五，这个项目是不会怎么活跃的。
> 但是竟然没想到，还是收获了很多提交和 Issue。
> 讨论区还是很多人在玩 "提问回答" 以及 "扯淡" 这两种游戏。
> 这让我觉得很意外：
> 　原来嘴里说讨厌，但是很多人心里还是都在琢磨着编程哪?!
> 这也让我想起了我原来和别人吹牛时说的话：
> 　世界上只有两种人：程序员，其他人
> 于是突然心中冒出了一种迫切的想为同道中人抛头颅洒热血，肋扇上多插几把刀 的冲动。
> 当然，我是个理智的人，能活到今天就证明了这一点
> 冲动是魔鬼
> 头就不抛，刀也就不插了
> 下个月多为项目修几个 Issue 吧 ^_^

另外， **juqkai** 同学这段时间为项目做了很多工作，提出了好意见，尤其是得到了提交权限后，更加一发不可收也。 我们也期待着想下个月的版本会有他更多的贡献。

下个版本我们会根据 Wendal 以及 **juqkai** 提出的方案，重新修改 Mvc 的一段核心逻辑

同时我们也会修复几个重要的 Issue

另，祝大家情人节快乐。 单身且不快乐的同学，祝你不那么不快乐一点 ^_^

14.13.2. 问题修复

* Issue 343 @ToJson作用在非public 的类上就无效的问题 by **lvjingjie**
* Issue 349 Ioc:注入Java调用的静态属性未实现 by **landraxee**
* Issue 351 LinkedArrayIterator的hasNext方法判断的问题??? by **天边的流星**
* Issue 352 LinkedArray类的构造方法 by **weirhp**
* Issue 353 Ioc:建议改进XmlIocLoader by **landraxee**
* Issue 354 Mvc:是否可以在执行ActionFilter之前进行参数适配，并将适配后的参数数组作为传入ActionFilter by **landraxee**
* Issue 357 JsonParsing类parseFromCurrentLocation方法中按位与和按位或的用法 by **天边的流星**
* Issue 358 Maven配置文件的放置错误 by **sjbwylbs**
* Issue 359 Maven编译的版本设置 by **sjbwylbs**
* Issue 365 mvc:Mvcs.getRequestPathObject对参数中含有小数点的情况处理有误 by **landraxee**
* Issue 366 MVC: @At处理有误,可能映射为//methodName by **wendal1985**
* Issue 369 mvc:PairAdaptor在处理REST风格的URL时对参数的解析不够智能 by **landraxee**
* Issue 371 ObjectPairInjector 不能多层注入 by **juqkai**
* Issue 373 蛋疼的forward跳转 by **juqkai**
* Issue 375 upload Adapter加上cancel功能 by **amosleaf**
* Issue 376 at中使用多路径，如有 “” ，会导致URL截流 by **juqkai**
* Issue 377 注释上有错误算不算问题？ by **hisenseme**
* Issue 378 DAO:当自定义复杂的关联查询SQL时,如何方便返回一个实体List by **liuyxit**
* Issue 380 DAO:mysql通过select last_insert_id()无法获取到正确的值 by **a357857613**

14.13.3. 质量

共通过了 610 个单元测试用例,代码覆盖率达到

Nutz.Dao 经测试在如下数据库上可以工作正常

* H2
* SQLite
* MySql

* Oracle
* Postgresql
* SqlServer2005
* SqlServer2000
* DB2

14.13.4. 文档

* 重写 Ioc - Annotation加载器
* 增加 Ioc - 复合加载器
* 增加 Ioc - 文件池
* 增加 表达式引擎
* 修改 文件上传
* 修改 视图

14.13.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 Demo Site 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献列表，我已经写了一个小程序，根据 Issue 列表来自动统计...

贡献者	问题	博客	支持	代码	示例	文档	测试
a357857613	O	-	-	-	-	-	-
amosleaf	O	-	O	-	-	-	-

E-Hunter	-	-	O	-	-	-	-
juqkai	-	-	-	-	-	O	O
hisenseme	O	-	-	-	-	-	-
landraxee	O	-	-	-	-	-	-
liuyxit	O	-	-	-	-	-	-
lvjingjie	O	-	-	-	-	-	-
sluggard	-	-	-	O	-	-	-
shìne	-	-	-	-	O	-	-
Wendal	O	O	O	O	-	O	O

weirhp	O	-	-	-	-	-	-
zozoh	O	O	O	O	-	O	O
花米	O	-	-	-	-	-	-
天边的流星	O	-	-	O	-	-	-
云海飞舞	-	-	O	-	O	-	-

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz & XBlink （58444676 超级群）
* Nutz在微笑 （60504323）超级群
* Nutz ② 群 （68428921）超级群
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^!

欢迎访问官网,以获取 最新的快照版 和用户手册

14.14. 1.b.34 发行注记

14.14.1. 1.b.34 发行注记

从 09 年初至今， 将近 20 个月的时间， 这个小项目前前后后发布了 33 个版本，今天我们可以负责任的说：

“呃 ... 差不多可以 beta 了吧 ^_^”

请允许我带上三块高仿劳力士，向近 2 年来不断试用这个项目的 **“小白”** 们致敬:

> *[CODE]*
> 如果没有你们
> 这个项目将注定成为一片浮云
> 有如一个 P
> 淡淡滴消散在喧闹的苍穹

Beta 意味着神马涅？

* Beta 意味着，我们还会不断改进，但是你每次替换新的 jar 包， 你的代码不需要改动
* 事实上，今年下半年的版本，即使出现新 jar 的编译错误，大家也是很容易修改的，基本上都是分分钟搞定
* 以后，Nutz 的版本号都会是 1.b.xxx

今年的最大的憾事就是：**我们还是没有完成官网的制作**， Wendal 同学耐不住寂寞，开了自己的博客，我订阅了一下，发现他的文章还是灰常有营养的， 这里祝愿他坚持一年成名博 ^_^

Nutz 的 QQ 群，几乎成为了这个项目主要的交流方式，但是，同学们请注意，我们的确还有以下的交流方式：

* Nutzam 讨论区 (*需轻功*)
* **GTalk 聊天群** - GTalk 添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令
* Nutz 的新浪微群

虽然他们很冷清，但是的确是有。 我们还有致力于提供更好的交流方式

今年这个项目被指责的有：

1. Demo 不给力！
2. 文档还是不够细致！
3. 对于混外企的我，没有英文文档怎么推？
4. 没有快速生成工具！
5. 没有成功案例不敢用啊！

解决上面的问题，将成为 2011 年我们的目标

另，虽然没有快速生成工具，但是我们也提供了一个 Nutz快速启动模板可以暂时顶个先

对于英文翻译，显然我们没有足够的人力物力，不过某天我一时兴起，用了下 Google 翻译，

* 请看： http://translate.google.com/translate?js=n&prev=_t&hl=en&ie=UTF-8&layout=2&eotf=1&sl=zh-CN&tl=en&u=http://nutz.googlecode.com

效果虽然不如人工的翻译，但是，已经让我很惊艳了。 所以 2011 年，我们还是不会把精力放在英文文档翻译上。Google 很给力

成功案例的问题，我们提供了一个列表：

* http://code.google.com/p/nutz/wiki/appendix_case

它现在还很短，据我所知，主要的原因是

* 已经使用 Nutz 的同学没有动力告诉我们 TA 项目的情况。
* 而且在这个具备神奇小气候的国度里，很多企业级项目，都是粉低调的滴~~~

所以我们的策略是准备好了这个列表，然后等啊等，看看事情会不会起变化

最后，祝愿所有看到上面这段文字的同学们 ---- **圣诞快乐 ^_^**

14.14.2. 重大改动

我们去掉了默认对 JDK Log 的支持，详细请参看 关于日志部分的文档

14.14.3. 问题修复

* Issue 320 使用@Ok("raw:XXXX")时,会产生异常 by **goulin.home**
* Issue 322 Nutz对LogBack的支持 by **yeahyf**
* Issue 324 lang: SimpleNode 添加节点的问题 by **wei li**
* Issue 325 Chain不能处理中文字段 by **sd6733531**
* Issue 326 在某种情况下用Mirror给一个javabean赋值的时候出现org.nutz.lang.FailToSetValueException的问题 by **ywjno.dev**
* Issue 328 mvc: 文件上传时，在 Mac 系统上的 Chrome 总是不成功 by **zozoth**
* Issue 329 NutConfig接口应该给出定义getInitParameterNames() 方法 by **amosleaf**
* Issue 331 IocBean注解的fetch,depose事件没有执行 by **kinglong408**
* Issue 334 1.a.33 使用@Ok("json")视图，AJAX提交，处理xhr时，浏览器报js错 by **conanca2006**
* Issue 336 为顶级package补全package-info.java by **wendal**
* Issue 338 nutz过滤器中 得IOC容器不理想 by **yunhaifeiwu**
* Issue 339 Record的toPojo方法,不能为已有类型的某些属性赋值 by **goulin.home**

14.14.4. 质量

共通过了 610 个单元测试用例,代码覆盖率达到

Nutz.Dao 经测试在如下数据库上可以工作正常

* H2
* SQLite
* MySql
* Oracle
* Postgresql
* SqlServer2005
* SqlServer2000
* DB2

14.14.5. 文档

只是简单的修改了一些小问题，不值一提

14.14.6. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 Demo Site 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献列表，我已经写了一个小程序，根据 Issue 列表来自动统计...

贡献者	问题	博客	支持	代码	示例	文档	测试
Amosleaf	O	-	-	-	-	-	-
Conanc	O	-	-	-	-	-	-

a							
Jay	-	-	O	-	-	-	-
Kinglong408	O	-	-	-	-	-	-
Sd6733531	O	-	-	-	-	-	-
Wei li	O	-	-	-	-	-	-
Wendal	O	O	O	O	-	O	-
Yeahyf	O	-	-	-	-	-	-
Ywjno.dev	O	-	-	-	-	-	-
Zozoth	O	-	O	O	-	O	O
前冲	O	-	-	-	-	-	-

幸福的旁边	-	-	O	-	-	-	-
云海飞舞	O	O	O	-	-	-	O

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz交流超级群(58444676)
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^!

欢迎访问官网,以获取 最新的快照版 和用户手册

14.15. 1.a.33 发行注记

14.15.1. 1.a.33 发行注记

大家好，又到了一月一度的 Nutz 新版本发行的日子了 ^_^

前几天 360 和腾讯刚干完架，特地记录一下。作为一名不明真相且兴致勃勃的围观群众，我表示："**不过瘾，不给力，希望他们能早日再战**"。

关于 Nutz，最大的新闻就是，从这个版起，无敌的Wendal同学终于把 Nutz 提交给 Maven 库了，之后，据说他要看亚运会，让我们在这里祝福他。

这个版我们修复了一些小问题，关闭了一些老问题。其中重点有两个

* Issue 233 - Dao 关于映射的操作可以接受集合，数组，以及 Map 了
* Issue 303 - Mvc 中，Ioc 容器内的对象可以支持注入 ServletContext 对象
    > 我想对于打算用 Freemarker 或者其他视图模板引擎的同学来是，是个好消息

八卦

前几天一个朋友告诉我，他面试一个实习生，对方在介绍自己的编程经历时提到了 Nutz，说很是喜欢这个框架。我问他：此人水平如何?他说挺好的。对此我表示很欣慰。同时我也深刻的意识到喜欢 Nutz 的人，水平都不会太菜，即使现在菜，以后也能迅速变得不菜。理由如下：

1. 喜欢小众东东的多半是发烧友
2. 发烧友通常很快都能成为该领域的专家
3. Nutz 是个小众的东东
4. Nutz 是个编程框架
5. 因此，喜欢 Nutz 的同学多半都会很快成为编程专家

另外，在这里也推荐一下另外一个小众框架 -- Guzz。这个项目的文档非常不错，作者这一年来做了难以想象的工作，前段时间，Wendal 同学为 Guzz 贡献了一个 QQ 群(36429094)。我简单的浏览过 Guzz 的代码，写的很漂亮，我想在这个项目会帮助到不少人的，希望喜欢 Guzz 的同学，踊跃加入 QQ 群(36429094) ^_^

同时也向其他战斗在开源第一线的的同学们致敬： **不挣钱都这么认真? 你真牛 ^_^!**

预告:

下个版，我们有可能会再做一次小小的膨胀:

* 增加功能更全面的 Condition 实现类
* 为 Mvc 再增加个默认视图实现
* 并打算内置一个文件比较工具类

下面是这个版本的更多细节...

14.15.2. 问题修复

* Issue 29 Dao在执行SQL时应提供处理数组类型数据的功能。 by **satellite**
* Issue 122 不支持设置单PK by **Chen.Bao.Yi**
* Issue 181 问个弱弱的问题,怎么实现内部跳转?forward方式 by **zkgale**
* Issue 231 关于dao多表查询 by **shao0707@163.com**
* Issue 233 Dao: 接口混乱问题-relation 相关操作支持了集合 by **jiongs753**
* Issue 241 PairAdaptor应支持POJO中的数组类型的变量注入 by **landraxee**
* Issue 252 mvc: HTTP 重名参数 by **zozohtnt**
* Issue 279 希望能使用Record进行数据库操作 by **ywjno.dev**
* Issue 297 文件上传增加种方式，直接在内存操作文件，不存入临时目录 by **conanca2006**
* Issue 301 web.xml配置问题 by **happyday0517**
* Issue 303 实现ServletContext更优雅的绑定到ioc容器以便可以方便的访问它 by **nightmarelin**
* Issue 305 在使用Mirror.setValue给bean的属性赋值的时候出错 by **ywjno.dev**
* Issue 306 @Views注入的viewmaker是直接new出来而不是ioc容器注入，这个设计是否有bug by **nightmarelin**

* Issue 308 castor: 日期转换成 java.util.Date 失败 by **zozohtnt**
* Issue 311 updateIgnoreNull在更新集合时，没有忽略null by **happyday0517**
* Issue 313 mvc: 文件上传应该支持 InputStream 和 Reader by **zozohtnt**
* Issue 314 JsonAdaptor不能将键值对转化为对象。 by **satellite**
* Issue 315 MVC文档编辑错误 by **sjbwylbs**
* Issue 316 以json方式传递数组,入口函数无法正常接收转换为对象数组 by **goulin.home**
* Issue 317 FileSqlManager重复加载文件,导致获取sql时出错 by **wendal1985**

14.15.3. 质量

1. 共通过了 597 个单元测试用例,代码覆盖率达到**65%**(按line计算)
2. Nutz.Dao 经测试在如下数据库上可以工作正常
    * H2
    * SQLite
    * MySql
    * Oracle
    * Postgresql
    * SqlServer2005
    * SqlServer2000
    * DB2

14.15.4. 文档

* 视图 : 添加两种新视图的描述(内部重定向视图/Raw视图)
* 修正文档中的一些字眼
* 在Ant中使用zDoc : 演示如何在Ant中使用zDoc
* 案例 : 展示使用Nutz构建的网站/应用
* 同Ioc一起工作 : 在容器对象里获得 ServletContext
* ../appendix/work_with_maven.man : 在Maven中使用Nutz

14.15.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 Demo Site 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献者	问题	博客	支持	代码	示例	文档	测试

Conanca	O	-	-	-	-	-	-
Landraxee	O	-	-	-	-	-	-
Nightmare	O	-	-	-	-	-	-
Satellite	O	-	-	-	-	-	-
Shao0707	O	-	-	-	-	-	-
Wendal	O	O	O	O	-	O	O
YanKunCheng	-	-	-	O	-	-	-
Ywjno.dev	O	-	-	-	-	-	-
Zkgale	O	-	-	-	-	-	-
Zozoh	O	O	O	O	-	O	O
宝	O	-	-	-	-	-	-
冬天温泉	-	-	O	-	-	-	-
花米®	-	-	-	-	-	O	-
前冲	O	-	-	-	-	-	-
天行健	O	-	-	-	-	-	-
幸福的旁边	O	-	-	-	-	-	-

云海飞舞	-	-	O	-	-	-	-

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz交流超级群(58444676)
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^

欢迎访问官网,以获取 最新的快照版 和用户手册

14.16. 1.a.32 发行注记

14.16.1. 关于 1.a.32

本版除了修复问题，主要增加如下特性

1. Ioc: 字段注入的循环依赖，不能正常工作
2. Mvc: 上传时，支持文件名过滤，以及大小限制

14.16.2. 问题修复

* Issue 94 Ioc中有相互依赖时会出现空指针异常 by **Blue Rain**
* Issue 200 MVC: UploadAdapter文件大小限额 by **天行健**
* Issue 249 mvc: UploadAdaptor 应该支持一个选项，是否记录空文件（默认应该为 "不记录"） by **zozohtnt**
* Issue 266 通过注解方式使用Ioc文档太少,也没有Demo,建议尽快补充 by **LQ10001**
* Issue 274 lang: Files.write 应该支持 InputStream by **zozohtnt**
* Issue 278 入口函数返回View时，ActionInvokerImpl处理不对 by **茶几**
* Issue 283 Http#encode by **caji.net**
* Issue 284 关于项目打成JAR包,IOC,MSG,SQL目录无法找到. by **shao0707@163.com**
* Issue 288 希望多增加几个内置转换器 by **LQ10001**
* Issue 289 更新20100916快照版的jar包后，JsonLoader无法找到js文件所在的文件夹. by **qi.yang.cn**
* Issue 290 NutDao主键类型转换错误 by **yonlist**
* Issue 291 Castors 初始化错误 by **caji.net**
* Issue 293 关于dao查询问题,查询email的一个bug by **shao0707@163.com**
* Issue 296 在gae上使用nutz ioc报错， nutz版本1.a.31及以后版本都报 by **yuansicau**

14.16.3. 质量

1. 共通过了 XXX 个单元测试用例,代码覆盖率达到62%(按line计算)
2. Nutz.Dao 经测试在如下数据库上可以工作正常
    * H2
    * SQLite
    * MySql
    * Oracle
    * Postgresql
    * SqlServer2005
    * SqlServer2000
    * DB2

14.16.4. 文档

* 本地化字符串 : 修改描述
* zDoc 概述 : 添加生成PDF的描述
* 可用插件 : 添加插件中心的描述

14.16.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 Demo Site 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献者	问题	博客	支持	代码	Demo	文档	测试
O(∩_7∩)O	-	-	-	-	O	-	O
Blue Rain	O	-	-	-	-	-	-
caji.net	O	-	-	-	-	-	-
Jay	-	-	O	-	-	-	-
LQ10001	O	-	-	-	-	-	-

qi.yang.cn	O	-	-	-	-	-	-
shao0707	O	-	-	-	-	-	-
yonlist	O	-	-	O	-	-	-
yuansicau	O	-	-	-	-	-	-
wendal	O	O	O	O	O	O	O
zozohtnt	O	-	O	O	-	O	O
茶几	O	-	-	-	-	-	-
老袁	-	-	-	O	-	-	-
水蓝	O	-	O	-	-	-	-
天行健	O	-	-	-	-	-	-
云海飞舞	-	O	O	-	O	-	-
冬天温泉	-	-	-	-	-	-	O

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz交流超级群(58444676)
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^

欢迎访问官网,以获取最新的快照版和用户手册

14.17. 1.a.31 发行注记

14.17.1. 关于 1.a.31

这个版本中，Nutz.Mvc 将 Filter 方式作为主要挂载方式,新增对SQLite的支持

14.17.2. 问题修复

* Issue 47 支持 SQLite by **zozohtnt**
* Issue 184 json: JsonFormat 应该支持选项，输出的字符串，是用单引号还是双引号 by **zozohtnt**
* Issue 251 MVC: 改善NutServlet对静态文件处理的缺陷，使REST风格更完美 by **ToFishes**
* Issue 256 nutz dao 在读取int类型列时不能区分数据库到底是存放0, 还是为null by **wenzhihong2003**
* Issue 259 MVC: 不能从mvc框架中拿到完整的PathInfo by **天行健**
* Issue 260 dao.query 出现 Number of input values does not match number of question marks 异常问题 by **清风徐来**
* Issue 262 ComboSql的vars不能用, 出现 nullPoint by **wenzhihong2003**
* Issue 264 缺少 how to build with ant 的文档说明, 建议加入wiki中 by **wenzhihong2003**
* Issue 265 @Param("..")获取checkbox形式的内容时,只能取到第一个选中的checkbox值 by **幸福的旁边**
* Issue 268 NutDao类的错误信息格式化错误 by **茶几**
* Issue 270 Http#encode方法不对 by **茶几**
* Issue 272 String转Number问题 by **Jay**
* Issue 273 注解的aop拦截器和json配置的拦截器冲突 by **幸福的旁边**

其中 **ToFishes** 报的问题 Issue 251 对我们很重要，特此感谢

14.17.3. 质量

1. 共通过了 573 个单元测试用例,代码覆盖率达到62%(按line计算)
2. Nutz.Dao 经测试在如下数据库上可以工作正常
    * H2
    * SQLite
    * MySql
    * Oracle
    * Postgresql
    * SqlServer2005
    * SqlServer2000
    * DB2 (未详细测试)

14.17.4. 文档

* 修正部分文字错误

14.17.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 代码: 提交过至少一个修订
* Demo: 为 Demo Site 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献者	问题	博客	支持	代码	Demo	文档	测试
9yong8yuan	O	-	-	-	-	-	-
Bony Fish	-	O	-	O	-	-	-
Jay	O	-	O	-	-	-	-
LQ10001	O	-	-	-	-	-	-
ToFishes	O	-	O	-	-	-	-
wenzhihong2003	O	-	-	-	-	-	-
E-Hunter	-	-	O	-	-	-	-
wendal	O	-	O	O	-	O	O
zozohtnt	O	-	O	O	-	O	O
茶几	O	-	-	-	-	-	-
幸福的旁边	O	-	-	-	-	-	-
清风徐来	O	-	-	-	-	-	-

天行健	O	-	-	-	-	-	-

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz交流超级群(58444676)
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*

回答新手的问题，我们现在只能根据印象草草统计，贡献列表非常不完善。我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^

欢迎访问官网,以获取最新的快照版和用户手册

14.18. 1.a.30 发行注记

14.18.1. 关于 1.a.30

这是一个Bug Fix版本

14.18.2. 问题修复

* Issue 161 ClassLoaderUtil 应该同 Resources 合并 by **zozohtnt**
* Issue 193 mvc: DefaultLoading 和 UrlMap 的逻辑混乱 by **zkgale**
* Issue 229 mvc: UploadAdaptor处理中文文件名、路径名乱码 by **landraxee**
* Issue 236 mvc: Mock 的 CharsetInputings 应该作为一个可实例化的对象 by **zozohtnt**
* Issue 237 默认字符集应该该回到 UTF-8 by **zozohtnt**
* Issue 238 关于主模块的@Modules注解 by **conanca2006**
* Issue 240 castors: 在 weblogic 下不工作 by **zozohtnt**
* Issue 244 考虑放弃Resources.scanClass(Class)对GAE的支持 by **wendal**
* Issue 254 从ioc里获取当前rquest对象 by **happyday0517**
* Issue 255 dao: 实现 DB2Pager by **zero args**

14.18.3. 质量

1. 共通过了 527 个单元测试用例,代码覆盖率达到62%(按line计算)
2. Nutz.Dao 经测试在如下数据库上可以工作正常
    * H2
    * MySql
    * Oracle
    * Postgresql
    * SqlServer2005
    * SqlServer2000

14.18.4. 文档

* 修正部分文字错误

14.18.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 支持: 总结至少出 FAQ 文档
    > 关于 FAQ 文档的写作规范请先阅读： Nutz 的虎年计划.5提供更好的社区支持
* 代码: 提交过至少一个修订
* Demo: 为 Demo Site 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周，发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献者	问题	博客	支持	代码	Demo	文档	测试
zero args	-	-	-	O	-	-	-
E-Hunter	-	-	O	-	-	-	-
conanca2006	O	-	-	-	-	-	-
landraxee	O	-	-	O	-	-	-
wendal	O	-	O	O	-	O	O
zkgale	O	-	-	-	-	-	-
zozohtnt	O	-	O	O	-	O	O

另外，很多朋友都在：

* Nutzam 讨论区
* Nutz交流超级群(58444676)
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*

回答新手的问题，我们现在没法一一统计，我们正在想办法，争取在不远的将来，能记录下来大家每一点一滴的付出 ^_^

14.19. 1.a.29 发行注记

14.19.1. 关于 1.a.29

这是一个Bug Fix版本

14.19.2. 问题修复

* Issue 211 更换1.28后的问题
* Issue 214 aop: 有些时候，用 AOP 生成的字节码抛 UnsupportedClassVersion
* Issue 216 Aop: 声明式切片 难以配置
* Issue 217 Json toJson()当传入的是HttpRequest时出错
* Issue 218 org.nutz.lang.String 510行，注释字写错了
* Issue 219 上传文件出错
* Issue 221 Ioc 如果DataSource是在Web容器里面定义的,就无法获取
* Issue 223 Ioc @Inject应当允许标注在set方法上
* Issue 225 ioc: 自定义注入的特殊类型不能正确解析
* Issue 227 mvc: ServerRedirectView.render函数类型转换bug
* Issue 228 ioc: 注入容器上下文貌似没有实现

14.19.3. 质量

1. 共通过了 503 个单元测试用例,代码覆盖率达到62%(按line计算)
2. Nutz.Dao 经测试在如下数据库上可以工作正常
    * H2
    * MySql
    * Oracle
    * Postgresql
    * SqlServer2005
    * SqlServer2000

14.19.4. 文档

* 修正部分文字错误

14.19.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 支持: 总结至少出 FAQ 文档
    > 关于 FAQ 文档的写作规范请先阅读： Nutz 的虎年计划.5提供更好的社区支持

* 代码: 提交过至少一个修订
* Demo: 为 Demo Site 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周，发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献者	问题	博客	支持	代码	Demo	文档	测试
happyday0517	O	-	-	-	-	-	-
landraxee	O	-	-	O	-	-	-
wendal	O	-	O	O	-	O	O
winstars	O	-	-	-	-	-	-
zozohtnt	O	-	O	O	-	O	O

另外，非常感谢

* Nutzam 讨论区
* Nutz QQ 群(75818186)
* Nutz QQ 2群(68315571)
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*

的朋友们，你们提出的任何问题，发表的任何言论，实际上都对这个小框架 **产生着潜移默化的影响**。我们也正在不断的努力，希望每一个新版本都能让大家获得更好的编程体验。

14.20. 1.a.28 发行注记

14.20.1. 关于 1.a.28

Nutz 1.a.27 发布之后的近两个月，社区发布了 **1.a.28** 版。它进行了较重大的改进：

1. 对 Ioc 接口关于 Ioc 做了一个稍微重要的修改--增加了一个方法

> *[CODE]*
> <T> T get(Class<T> type) throws IocException;

    * 同时将注解 @InjectName 从 mvc 包移动到 ioc 包
2. Ioc 支持注解方式的配置
3. 重写了 AOP 的字节码模型
4. 文件上传适配器默认采用新写的 FastUploading，上传速度提高一倍

14.20.2. 问题修复

* Issue112通过直接声明一个包来声明包中所有的类为Module by **xjf1986518**
* Issue162Nutz DAO的事物是否能参考Spring的声明式事物管理方式比较好，不知道是否有计划? by **hzzdong**
* Issue166dao: EntityField中的如下方法是不是有点重复啊? by **jiongs753**
* Issue168无法在Mvc中使用xml作为ioc的配置文件 by **wendal1985**
* Issue171DAO: Cnd 生成的 in 语句不对 by **jiongs753**
* Issue172DAO: Dao.delete Dao.clear 等方法无返回值 by **jiongs753**
* Issue176考虑为Ioc提供注解式的配置 by **wendal**
* Issue177MVC默认适配器问题 by **zkgale**
* Issue178首页的字体太难看了 by **nowindLee**
* Issue182mvc: @Ok 的默认值问题 by **zozohtnt**
* Issue183json: 当 Map 的 Key 为非字符串时，渲染不出结果 by **zozohtnt**
* Issue186@Prev出错 by **happyday0517**
* Issue187demo hello Mvc中/demoredirect/byid和byobj，不能传递参数 by **for5million**
* Issue190静态方法注入时,提示不够 by **zkgale**
* Issue194事务拦截器报错 by **happyday0517**
* Issue199MVC: 为UploadAdaptor添加一个空参构造方法 by **jiongs753**

14.20.3. 质量

1. 共通过了 484 个单元测试用例,代码覆盖率达到62%(按line计算)
2. Nutz.Dao 经测试在如下数据库上可以工作正常
    * H2
    * MySql
    * Oracle
    * Postgresql
    * SqlServer2005
    * SqlServer2000

14.20.4. 文档

* 添加 使用注解配置Ioc
* 添加 NutAop模型
* 更新 Aop -声明式切面
* 更新了一对一、一对多、多对多映射部分

14.20.5. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 支持: 总结至少出 FAQ 文档
    > 关于 FAQ 文档的写作规范请先阅读： Nutz 的虎年计划.5提供更好的社区支持
* 代码: 提交过至少一个修订
* Demo: 为 Demo Site 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题
* 测试: 发布前，参与测试周，发布人给出的任务

如有遗漏,请提醒我们 ^_^

贡献者	问题	博客	支持	代码	Demo	文档	测试
Bird.Wyatt	-	-	O	-	O	-	-
for5million	O	-	-	-	-	-	-
E-Hunter	O	-	-	-	-	-	-
Peter Tung	-	-	-	-	-	O	-
hzzdong	O	-	-	-	-	-	-
jiongs753	O	-	-	-	-	-	-
zkgale	O	-	-	-	-	-	-
happyday0517	O	-	-	-	-	-	-
wendal	O	-	O	O	O	O	O
zozohtnt	O	-	O	O	-	O	O

另外，非常感谢

* Nutzam 讨论区
* Nutz QQ 群(75818186)
* Nutz QQ 2群(68315571)
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*

的朋友们，你们提出的任何问题，发表的任何言论，实际上都对这个小框架 **产生着潜移默化的影响**。我们也正在不断的努力，希望每一个新版本都能让大家获得更好的编程体验。

14.21. 1.a.27 发行注记

14.21.1. 前言

今天, Nutz社区自豪地宣布,Nutz 1.a.27版正式发布了! 本版有很多新功能,并修复了近30个issue,拥有改进的文档和更丰富的Demo.

* 由于更多人试图通过更多的方法定制和使用 Nutz，所以遇到了更多的问题
* 这些问题都是对 Nutz 小幅的修正和调整
* 本次发布修复了其中绝大多数问题
* 所以我们强烈推荐你更新到这个版本.马上下载

曾有朋友建议将 Nutz 的级别调至 **b** (*beta*，因为现有的功能已经很稳定了。经过半年的不断修正，它在生产环境中的表现无论是开发效率还是灵活性上，完全不比很多 RC 框架差，甚至更好。

但是我们还是坚持它是 alpha。

* 因为我们并不认为把一个发布包的名字由 alpha 改为 beta，它就是 beta 品质的了。
* 我们认为只有它达到了我们心中的 beta 品质，即，我们实在不认为有调整接口的必要了，我们才会将其升级成 beta。
* 起码一段时间内，我们还会继续保留修改 Nutz 主要接口和使用方法的权利。

1.a.27 从另外一个角度来说，也是一个里程碑。从 09 年初的个人框架，到现在社区已经成为这个框架发展不可或缺的力量。

* 新人可以很容易的加入到社区，并得到帮助
* 使用者提出的问题会得到及时的响应
* 每一个意见和建议都会得到讨论
* 被关注的问题很快可以得到修复。

因此我们可以说，从这个版开始，它已经成为一个真正的开源框架了。

14.21.2. 概述

这个版本是在 1.a.26 基础上的重要改进

* 为 IdEntityService 和 IdNameEntityService 增加了 exists 函数，根据主键判断一条记录是否存在将更加轻松

```
[CODE]
IdNameEntityService<Pet> pets = new
IdNameEntityService<Pet>(dao){};
pets.exists(34);
pets.exists("XiaoBai);
```

* 为 IdNameEntityService 增加了 smartFetch 函数，可以自动判断是根据 @Id 还是 @Name 来获取对象
* mvc: Uploading速度提高一倍
* ioc: 注入支持新的 Value 类型： sys
* ioc: 支持使用XML作为配置文件
* ioc: 通过配置文件添加AOP
* mvc: 通过直接声明一个包来声明包中所有的类为Module

14.21.3. 问题修复

* 68通过配置文件添加AOPby **Chen.Bao.Yi**
* 112通过直接声明一个包来声明包中所有的类为Moduleby **Toni.xutao**
* 119对@Id字段是否能在没有设置@Next的情况下，insert后自动把数据库中的id值回填到Entity中？by **hzzdong**
* 126Dao: 执行 Trans.exec for informix 出现 Castors 错误by **jiongs753**
* 127dao: NutTransaction 的实现代码有错误，在 commit 的时候 clear list 以及 close connby **jiongs753**
* 128无法将自定义的EntityMaker注入到NutDao中by **caiceclb**
* 129Link的构造函数不方便自己实现的JPAEntityMakerby **jiongs753**
* 132DAO使用proxool做数据源会出警告by **happyday0517**
* 134log: 使用Nutz的log工具在eclipse控制台打印的日志无法通过点击链接后跟踪到调用的代码行by **zozohtnt**
* 135Daos.java在jdk5编译通不过，因为其中使用了java.sql.Statement.isClosed()方法。 造成Nutz不兼容JDK5。by **jiongs753**
* 136fastInsert 支持多一个参数，是否执行 @prev by **happyday0517**
* 139dao: DefaultEntityMaker代码逻辑错误by **wendal1985**
* 140DAO: 关于Entity和EntityField get\set Attributeby **zozohtnt**
* 142Disk.absolute(String, ClassLoader, String),当上层文件夹中包含非英文字符时,返回的path可能不正确by **wendal1985**
* 143org.nutz.ioc.impl.NutIoc无法设置MirrorFactoryby **wendal1985**
* 144ioc: 注入应该支持新的 Value 类型： propertyby **zozohtnt**
* 147Nut.Aop没有提供示例by **happyday0517**
* 148Nut.Aop 当方法的返回值为基本数据类型,而拦截器的beforeInvoke返回false时,会出现NPEby **wendal1985**

* 149空json配置文件启动时会报错by **wendal1985**
* 150ActionFilter中能否多增加一个方法或在现有方法上加个参数，使实现类可以获得@At所注解的方法名by **hzzdong**
* 151关于upload File时候的性能问题
* 153当类文件在中文路径下,org.nutz.lang.util.Resources.scanClasses()无法得到任何class by **wendal1985**
* 154新的代码在JDK5，mysql下测试发现一个单元测试没有通过 by **hzzdong**
* 157 MVC部分《请求范围模块》request 注入类型为空 by **axhack**
* 160 换1.a.27后找不到自定义sql by **happyday0517**
* 163 json: 获取属性值时，应该调用 getter by **zozohtnt**
* 164 json: 字段类型为 Map<String, List<String>> 会抛错 by **zozohtnt**

14.21.4. 质量

1. 共通过了 476 个单元测试用例,代码覆盖率达到62%(按line计算)
2. Nutz.Dao 经测试在如下数据库上可以工作正常
    * H2
    * MySql
    * Oracle
    * Postgresql
    * SqlServer2005
    * SqlServer2000

14.21.5. 文档

* 更新英文文档 by **mutang**
* 添加Nutz插件页 by **wendal**
* 整理 FAQ 文档

14.21.6. 博客

* 《NUTZ与SQL SERVER2000兼容性问题》@Ge.PH /hl
* 《Nutz DAO懒加载实体关联对象》 @天行健 **推荐**

14.21.7. 主要贡献者名单

贡献的种类:

* 问题: 给项目的问题列表汇报一个上的问题，并且该问题被本次发布包括
* 博客: 在本版本开发期间，写过关于 Nutz 的文章，并被 推荐列表收录
* 支持: 总结至少出 FAQ 文档
    > 关于 FAQ 文档的写作规范请先阅读： Nutz 的虎年计划.5提供更好的社区支持
* 代码: 提交过至少一个修订
* Demo: 为 Demo Site 提交过代码
* 文档: 提交过文档，在讨论区发帖或者通过文档上的留言指出现有文档存在的问题

* 测试: 发布前，参与测试周，发布人给出的任务

如有遗漏,请提醒我们 ^_^

http://docs.google.com/View?id=dc5jpf24_160gshjh436

贡献者	问题	博客	支持	代码	Demo	文档	测试
amosleaf	O	-	-	O	-	-	-
axhack	O	-	-	-	-	-	-
BirdWyatt	-	-	-	O	-	-	-
caicecl b	O	-	-	-	-	-	-
Chen.Bao.Yi	O	-	-	-	-	-	-
deadeye2k	O	-	-	-	-	-	-
E-Hunter	O	-	O	-	-	-	-
hzzdong	O	-	-	-	-	-	-
happyday0517	O	-	-	-	-	-	-
mastung	-	-	-	-	-	O	-
ming300	-	-	-	O	-	-	-
jiongs753(天行健)	O	-	-	-	-	-	-

Toni.xutao	O	-	-	-	-	-	-
wendal	O	-	O	O	O	O	O
zozoh	O	-	O	O	-	O	O

另外，非常感谢

* Nutzam 讨论区
* Nutz QQ 群(75818186)
* Nutz QQ 2群(68315571)
* GTalk 聊天群 *添加帐号 nutzam@chatterous.com 为好友，然后发送 @join 指令*

的朋友们，你们提出的任何问题，发表的任何言论，实际上都对这个小框架 **产生着潜移默化的影响**。我们也正在不断的努力，希望每一个新版本都能让大家获得更好的编程体验。

14.22. 1.a.26

14.22.1. 概述

这个版本是在 1.a.25 基础上修复了一些 BUG，并让 Dao 支持 ActiveRecord

14.22.2. 问题修复

* Issue 95 : Ioc - 没有检测循环依赖,且不支持多次继承
* Issue 97: 从 Dao 里查出来的对象是一个 Map，或者是List<Map<String, Object>> by **Gdunser**
* Issue 100: MVC手册在jsp视图处描述有误 by **ToFishes**
* Issue 101: Dumps.matcher NullpointerException by **for5million**
* Issue 102: 在update时，如果http参数有Date类型为null，会出错 by **Gdunser**
* Issue 103: 移除JsonIocLoader中对$import,$vars的支持 by **Wendal**
* Issue 104: 当server启动后,死一次访问界面无法显示国际化的message by **Bird.Wyatt**
* Issue 106: 在 NutIoc 中缓存 IocObject by **Wendal**
* Issue 108: 添加Maven支持文件-- pom.xml by **Wendal**
* Issue 109: 继承Module类，如果是@OK("json")，会出错 by **Gdunser**
* Issue 111: IoC配置能扫描Package内的Class的注解直接获取到 by **nwangwei**
* Issue 114: context.js的正確格式是? by **wsc0918**
* Issue 115: JSON格式不标准 by **deadeye2k**
* Issue 120: dao: @Next 和 @Prev 变量 $view.NEXTVAL 解析失败 by **hzzdong**

14.22.3. 质量

1. 共通过了 414 个单元测试用例
2. Nutz.Dao 经测试在如下数据库上可以工作正常
    * H2

* MySql
* Oracle
* Postgresql
* SqlServer2005
* SqlServer2000

14.22.4. 改进

1. 小幅修正了一些 Bug 和文档错误
2. Dao : 提供了新的接口函数： List<Record> query(String tableName, Condition condition, Pager pager)
3. Ioc : 引入 IocObject 的缓存，提升了非 Singleton 对象构建速度

14.22.5. 改进

1. 小幅修正了一些 Bug 和文档错误

* Issue 103: 移除JsonIocLoader中对$import,$vars的支持 by **Wendal**
* Issue 106: Ioc: 在 NutIoc 中缓存 IocObject by **Wendal**

14.22.6. 文档

1. 修改了 Mvc 视图部分
2. 增加了 同 Ioc 容器一起工作.需要注意的问题

14.22.7. 博客

* 《我的第一个Nutz程序》@Bird.Wyatt
* 《NUTZ与SQL SERVER2000兼容性问题》@Ge.PH /hl
* 《借于Nutz快速实现对表的增删改操作》 @会跑的蜗牛
* 《使用注解做ORM》@amosleaf

14.22.8. 主要贡献者名单

贡献者	问题	博客	讨论	代码	Demo	文档	测试
amosleaf	X	X	-	-	-	-	X
bird.wyatt	X	X	X	-	X	-	-
deadeye2k	X	-	-	-	-	-	-

E-Hunter	X	-	-	-	-	-	-
for5million	X	-	-	-	-	-	-
mastung	-	-	-	-	-	X	-
mawm	-	X	-	X	-	-	-
rocy03	X	-	-	-	-	-	-
Sega	X	-	-	-	-	-	-
ToFishes	X	-	-	-	-	-	-
yangchunhai2005	-	-	X	-	-	-	-
wendal	X	X	X	X	-	-	X
zozoh	X	X	X	X	-	X	X
会跑的蜗牛	X	-	-	-	-	-	-
知足常乐	X	-	-	-	-	-	X

另外，非常感谢Nutz QQ 群(75818186)以及Nutzam 讨论区的朋友们，你们提出的任何问题，发表的任何言论，实际上都对这个小框架 **产生着潜移默化的影响**。我们也正在不断的努力，希望每一个新版本都能让大家获得更好的编程体验。

另，由于众所周知的原因，建议大家通过别的服务访问讨论区因为非中文版的 Google Group 是没有被 QIAng 的。

14.23. 1.a.25 及 更早版本

14.23.1. 1.a.25

这是个里程碑式的版本，从这个版本开始， Nutz 将不再依赖 Javassist，而依赖 ASM(内嵌Asm 3.2 ,无需额外jar包)所以，编译时，你将不再需要 Javassist.jar （在此，特地感谢无敌的 Wendal）。

同时，为了更好的输出日志，Nutz 在编译时依赖了 Log4j， 但是运行时，没有 Log4j 它会选用 Java 的 Logging 或者是控制台输出。

由于已经有小部分人开始试验性的在自己的项目里应用 Nutz，从本版开始，所有的接口和使用方式**变动会非常慎重**，没有重大理由和经过社区充分讨论，它将保持不变。

14.23.1.1. 问题修复

* Issue 12: Should support one config file can import other config files function. by **satellite168**
* Issue 70: 简化json配置(在ioc配置的是时候能把包名用变量代替？) by **lhasasky**
* Issue 71: aop: 修改拦截器接口,改为使用asm实现 by **wendal**
* Issue 72: 客户端提交表单服务器端自动绑定pojo的建议 by **lhasasky**
* Issue 78: 出现@Id或者@name时 还需要@column by **amosleaf**
* Issue 80: mvc: 支持 @Attr 注解，可以从 request, session, ServletContext 获取属性 by **zozoh**
* Issue 86: NameInjector不能处理一个name有多个值的情况 by **blue_rain**
* Issue 91: delete(Object)操作无法删除@PK对象问题 by **harkoo**
* Issue 93: dao update 接口建议 by **Bird.Wyatt**
* Issue 96: 将HttpServletRequest所接收的参数转为Map by **会跑的蜗牛**

14.23.1.2. 质量

1. 共通过了 397 个单元测试用例
2. 可以 GAE SDK 1.3.0 上工作正常（除了 Dao 部分）
3. Nutz.Dao 经测试在如下数据库上可以工作正常
    * H2
    * MySql
    * Oracle
    * Postgresql
    * SqlServer2005
    * SqlServer2000

14.23.1.3. 改进

1. 重新实现Log模块，支持 Log4j, Jdk Log
2. 新增Plugin模块， 支持“部署时决定插件的实现”
3. 修改了测试用例 JsonCharsetTest.test_zh_CN_from_file
4. Castors 去掉了大多数静态方法
5. JsonFormat 支持设置自定义 Castors
6. Log 部分的结构和实现修改的更加简洁

14.23.1.4. 文档

1. 添加了 Nutz.Lang（Wiki《甜Java》）下全部文档
2. 完成了 Mirror 类的所有 Java Doc

14.23.1.5. 博客

* 《Nutz:重新发明轮子:自己动手,用字节码工具做一个Aop拦截器》@Wendal
* 《Nutz:基于ASM的Nut.Aop实现》@Wendal
* 《给 nutz 添加 freemarker 视图》 @Axhack
* 《回复fireflyc：Nutz 的设计以及提高程序员生产力》 @zozoh
* 《Nutz的 数据库事务》@amosleaf
* 《不用写代码 你也能为开源作出贡献》@hilliate

14.23.1.6. 主要贡献者名单

贡献者	问题	博客	讨论	代码	Demo	文档	测试
amosleaf	X	X	X	-	X	-	X
axhack	-	X	X	-	-	-	-
bird.wyatt	X	-	X	-	X	-	X
blue_rain	X	-	X	-	-	-	X
bonyfish	X	-	-	-	-	-	-
hilliate	-	X	X	-	-	X	X
mastung	-	-	X	-	-	X	X
mawm	X	-	X	X	-	-	-
Roy Tonhee	X	-	-	-	-	-	-
satellite168	X	-	-	-	-	-	-
lhasaskky	X	-	X	-	-	-	-
sunonfire	-	-	X	X	-	X	-

Tony	-	-	-	-	-	-	X
wendal	X	X	X	X	-	-	X
wsc0918	X	-	-	-	-	-	-
yuansicau	X	-	X	-	-	-	-
zozoh	X	X	X	X	X	X	X
密林仙踪	X	-	-	-	-	-	-
农大天狼	-	-	X	-	X	-	-
会跑的蜗牛	X	-	X	-	-	-	X
知足常乐	-	-	X	-	-	-	X

另外，非常感谢Nutz QQ 群(75818186)以及Nutzam 讨论区的朋友们，你们提出的任何问题，发表的任何言论，实际上都对这个小框架 **产生着潜移默化的影响**。我们也正在不断的努力，希望每一个新版本都能让大家获得更好的编程体验。

另，由于众所周知的原因，建议大家通过通过 https 方式访问讨论区有时候是可以上的。但是能跳墙还是跳墙的比较好。

14.23.1.7. 下一版目标

将主要集中在

* Issue 37 ioc: 自动甄别对象的 Ioc 字段
* Issue 68 通过配置文件添加AOP
* Issue 76 Ioc中增加类似Spring的Autoweave的特性
* Issue 83 castor: 用 Method 替换实现类
* Issue 87 重新实现一个更友好的字节码工具

14.23.2. 1.a.22

请参看 JavaEye 的 新闻:《Nutz1.a.22 发布-Mvc,Ioc 文档完成》

14.23.3. 1.a.15

请参看 JavaEye 的 新闻:《 Nutz 框架 1.a.15 发布，全部用例通过 Oracle/Psql/MySql 测试》

15. FAQ

15.1. Nutz 该如何发音

问题来自 **不知名网友**

读作 “纳特Z（Z发重音）”

另外，Nutz 的 Nut 是因为霍金的《果壳中的宇宙》是 zozoh 最喜欢的一本书之一。Z 是 zozoh 小时，动画片《佐罗》给他很深的印象，尤其是每次转场的中间动画都是佐罗的剑在黑色的空中 唰唰唰 三下划出锋利的 Z 字，好像三道闪电，酷的要命。同时 zozoh 本人姓张，所以他很高兴在 Nut 后面 唰唰唰 的来一个 Z

15.2. Nutz需要依赖第三方jar吗?Log如何配置?

15.2.1. 一个老生常谈的问题

1. Nutz需要第三方jar包才能运行吗? 答案是,不需要.但如果有的话更好.
2. 一般有几种疑问:
    1) 刚刚接触Nutz,直接把nutz.jar扔进工程,发现正常运行起来了
    2) 读源码,发现源码其实是依赖Log4j的几个类的,但为什么没有log4j的jar,Nutz也正常运行了呢?
    3) 想把日志输出到log4j,加上log4j的jar,发现nutz的日志就按log4j的设置输出了
    4) 不想使用log4j,想使用更高效的Logback或者传统的JDK Logging,不知道行不行?

15.2.2. 解答

1. Nutz的源码中,依赖两个外部jar -- servlet-api.jar 和 log4j.jar
    1) 前者在任何标准J2EE Web应用中都有, Nutz仅支持Servlet 2.4或以上.
    2) 后者是极其常见的Log4j, 编译器依赖于1.2.14版,但在运行时仅要求是1.2.x. 不过,我推荐最新的log4j 1.2.16 .
2. 我们使用Nutz.Plugin,通过检测是否存在Log4j的核心接口 org.apache.log4j.Log是否存在来判断是否将日志输出到Log4j
    1) 由于仅检测是否存在org.apache.log4j.Log接口,这样就导致两个情况:
        1] 仅仅把Log4j的jar放进classpath,而没有配置之,那么输出日志时,log4j会警告你. 我们认为这是你的失误.
        2] 这样也提供了一个自由度,因为slf4j的log4j-over-slf4j桥也提供这个接口.

15.2.3. 搭配组合

1. 最精简搭配,适合尝试阶段的新手,默认输出的就是Debug信息,完全无鸭梨!!
    * nutz.jar
2. 经典搭配,适合大部分人的需要
    * nutz.jar + log4j-1.2.14.jar
3. 推荐搭配,搭配最新的log4j
    * nutz.jar + log4j-1.2.16.jar

4. 性能至上,使用Logback
    * nutz.jar + log4j-over-slf4j.jar + slf4j-api.jar + logback.jar
5. 传统选择,使用JDK Loggging
    * nutz.jar + log4j-over-slf4j.jar + slf4j-api.jar + slfj4-jdk.jar
6. 另类之选,使用Apache Common Logging
    * nutz.jar + log4j-over-slf4j.jar + slf4j-api.jar + slfj4-jcl.jar

15.3. Nutz.Aop能做什么,不能做什么?

15.3.1. Nutz.Aop能做什么?

为实例方法添加 前置(before)/后置(after)/一般异常处理(Exception)/错误(Throwable)

15.3.2. Nutz.Aop不能做什么?

对构造方法进行拦截,对静态方法进行拦截,对字段进行拦截,对final类/final方法进行拦截,对私有方法进行拦截

15.3.3. Nutz.Aop是如何做到拦截一个方法的

1. 继承需要拦截的类. 这个类必须的非abstract非final的顶层类,不能是内部类和本地类(这个很少见)
2. override需要拦截的方法. 这个方法必须是非abstract非final非static非native
3. 嵌入拦截器逻辑到方法中. 看网页: Nutz.Aop模型

15.3.4. Nutz.Aop实现的细节问题:

1. 使用的是内嵌的精简过的ASM
    1) 虽然说是精简过,但实际代码使用率约50%,还有精简的余地
    2) 位于org.nutz.repo.org.objectweb.asm , 除非非常非常必要,请不要引用这里的类, 如果你要使用Asm,请使用官方原版
2. 通过AbstractClassAgent过滤不可能被拦截的类和方法
    1) 这个类与具体实现无关
    2) 没有引用Asm的类
    3) 如果觉得我实现得太烂,我想继承这个类仍是不错的选择

15.3.5. Nutz.Aop能怎么用:

1. 内置拦截器
    1) Log拦截器 为方法添加Log日志,记录方法进入/返回/异常等情况
    2) 事务拦截器 为方法添加事务
2. 拦截器接口MethodInterceptor
    1) 仅需要实现一个方法void filter(InterceptorChain chain)
    2) InterceptorChain实例包含方法执行的全部信息--对象/参数/方法/返回值
        1] InterceptorChain对象最重要的方法doChain(),用于继续执行过滤链
        2] InterceptorChain对象的方法invoke()用于直接执行原方法,一般不直接调用

3) 大部分情况下,继承AbstractMethodInterceptor是不错的选择,可以清晰处理方法执行的不同阶段

3. 在Ioc中添加Aop支持
    1) @Aop注解
    2) 声明式Aop
        1] json方式
        2] xml方式,暂未资料

15.3.6. Nutz.Aop代码中的注释

1. 在org.nutz.aop/org.nutz.aop.interceptor/org.nutz.aop.matcher下的类,基本上都是注释了的
2. 在org.nutz.aop.asm下的类,基本上是无注释的,在可预见的将来我也不打算添加
    1) 基本上都是Asm操作字节码,苦涩难懂
    2) 即使添加注释,我相信对你的理解不会有太多帮助
    3) 如果你能看懂这部分的代码,那你也不需要注释

15.4. 我的Nutz的源码是乱码怎么办？

15.4.1. 问题的提出

采用的是 nutz-source.jar 的方式存放源码

15.4.2. Ecipse 用户

打开

```
[CODE]
window > Preferebces > General > Workspace
```

看看"Text file encoding" 项是不是设成 "UTF-8" 了还是不行? 请重启你的Eclipse

15.4.3. Netbean 用户

1. 找到你的Netbeans安装目录下的etc文件夹,如C:\Program Files\NetBeans 6.0 M9\etc
2. 用记事本打开netbeans.conf
3. 找到netbeans_default_options这一句(没带#号的,带#号的是注释)
4. 在最后面加上一个空格,再加入-J-Dfile.encoding=UTF-8

15.4.3.1. 比如:

-J-DAM_CONFIG_FILE=\"C:\Sun\AppServer\domains\domain1\config\AMConfig.properties\" -J-Dcom.sun.aas.installRoot=\"C:\Sun\AppServer\" -J-Xms128m -J-Xmx512m -J-XX:PermSize=32m -J-XX:MaxPermSize=160m -J-Dapple.laf.useScreenMenuBar=true -J-

XX:+UseConcMarkSweepGC -J-XX:+CMSClassUnloadingEnabled -J-XX:+CMSPermGenSweepingEnabled -J-Dfile.encoding=UTF-8

"}}}

15.5. 社区常见问答 Part 1

15.5.1. 问: 救命啊!! XXX报错了!!

答: 请先浏览提问的智慧

15.5.2. 问: NutDao支持XXX连接池吗?

答: 实现了DataSource接口的连接池都支持

15.5.3. 问: 我的是遗留系统,用自己的方法来获取连接,能用NutDao不?

答: 能,用DataSource接口封装一下即可

15.5.4. 问: @Fail视图什么走?

答: 抛异常的时候. 有其他方式吗? 没有

15.5.5. 问: 日志中有"Table doesn't exist!"的提示,啥情况?

答: 你所连接的数据源,不存在所请求的表

15.5.6. 问: 启动报错,"folder or file like '^(.+[.])(js|json)$' no found in ioc"

答: 把conf文件夹设置为源文件夹, 右键conf文件夹,Build Path, As Source folder

15.5.7. 问: Sql.create后,为何sql.getList(XXX.class)返回null?

答: 执行dao.execute(sql)后才可能有结果

15.5.8. 问: 执行dao.execute(sql)后, sql.getList(XXX.class)返回null?

答: 确定一下你已经设置了callback

15.5.9. 问: nutz 支持 servlet3 的新特性吗?

答: Servlet3没啥新特性

15.5.10. 问: GoogleCode上的Nutz的svn,还更新吗?

答: 新版本发布的时候会更新一次

15.5.11. 问: 为何Mvcs.getIoc()返回null?

答: NutFilter/NutServlet的作用范围内,才可能返回Ioc容器

15.5.12. 问: 为何json视图的响应,字符串都被""包起来了?

答: @Ok("json")的入口方法,请不要返回String,返回Pojo/Map/List都能自动转为标准的json字符串. 如果你确实需要自己拼接json字符串,那么,请使用@Ok("raw")

15.5.13. 问: Cnd.where("name","in",?)能传入map或collection对象不?

答: 能传collection/数组,但不能传map(但可以用map.values())

15.5.14. 问: NutDao有些方法得用匿名内部类,咋传值呢?

答: 看wendal的博客文章: Java匿名内部类的传值

15.5.15. 问: Nutz的最稳定版本是?

答: 最新版就是最稳定版,因为我们自己就在用最新的代码

15.5.16. 问: 日志打印"find mapping null for path [/test]",访问啥页面都是404?

答: 查看日志,看看模块是不是已经加载了.很多时候,你改成@Modules(scanPackage=true)就解决这种问题

15.5.17. 问: Scans.me() 类都是在classPath 路径下搜索文件么?

答: 基本上是的. 还会在当前文件夹找一下,但web下的当前文件夹往往不一样!

15.5.18. 问: Images类是干啥呢?

答: zozoh偷偷加上去的,处理图片的一些小方法.

15.5.19. 问: jsp页面一定要放在WEB-INF里面嘛?放在外面怎么不行?

答: 看看jsp视图的说明吧,都是可以的,但@Ok里面的值不一样,以"/"开头

15.5.20. 问: web.xml必须要有个mainModule 并且名字是这个不能改?

答: 如果是NutMVC,那么必须有MainModule,但名字随便,不就一个类名嘛!

15.5.21. 问: nutz不做web开发,怎样使用dao呢?

答: 自己new一个NutDao呗,注意哦,单例就好了,NutDao是线程安全的

15.5.22. 问: 用c3p0老有这个错误"EBUG -- CLOSE BY CLIENT STACK TRACE"?

答: ioc中的dataSource的bean,添加close event

15.5.23. 问: 把nutz部署到虚拟主机,报错'~/.nutz/tmp/dao/'should be a directory!

答: 这是NutDao处理Blob/Clob数据的临时文件夹,定义在org/nutz/dao/jdbc/nutz_jdbc_experts.js,你可以拷贝一份出来,改一下里面的pool-home. 改成一个合法的路径,例如你可以通过log信息找到WEB-INF的路径

15.5.24. 问: 写了@Id/@Name的属性,还需要写@Column吗?

答: 不需要.

15.5.25. 问: nutz能开发java web项目吗?

答: 踢馆子?

15.5.26. 问: 自定义SQL能分页不?

答: 1.b.43及之后的版本才支持.

15.5.27. 问: 如果我Image类本身已经做了@Table("image")那下面的继承类再写@Table("bigimage")可以不?

答: 只认当前类的@Table

15.5.28. 问: skip-mode是啥?如何使用?

答: 除非你的很旧的版本升级上来,否则请不要使用!

15.5.29. 问: Nutz有计划模块化支持吗?

答: 一直觉得没啥必要性,所以不积极做这个

15.5.30. 问: nutz的事务模板支持嵌套么?

答: 支持,只认最外面一层

15.5.31. 问: nutz的系统包里没有实现对JSONP 的view?

答: 没有,自己扩展一下吧.

15.5.32. 问: 能根据返回值来跳转到特定的jsp吗?

答: 能,@Ok/@Fail都支持EL语法: @Ok("jsp:$

15.5.33. 问: 建议 Nutz 实现JAX-RS，全面拥抱REST?

答: Nutz基本上不会实现XX规范

15.5.34. 问: @Param("token") String token加@Param和不加 有啥区别啊?

答: 1.b.43之后,没区别.之前的版本,如果不是路径参数,会拿不到值.

15.5.35. 问: NutDao所使用的Pojo类,需要继承什么超类吗?

答: 不需要.

15.5.36. 问: 操作数类型冲突: nvarchar 与 image 不兼容，我们在把文件保存到image数据类性是出错

答: Sorry,第一次听这种类型

15.5.37. 问: update的时候怎么样不忽略null值?

答: dao.updateIngoreNull

15.5.38. 问: redirect 也支持表达式么 ?

答: 基本上内置视图都支持

15.5.39. 问: 把POJO都写好后然后像hibernate那样利用pojo来建所有的表,这个能行不?

答: dao.create(XXX.class)

15.5.40. 问: nutz中可不可以执行普通的SQL语句啊?

答: Sql sql = Sqls.create("XXXXX");dao.execute(sql);

15.5.41. 问: 给一个接口比如 UserService绑定一个实现类 UserServiceImpl 是怎么做的 ?

答: Ioc中的bean,只认名字. 所以,在UserServiceImpl写@IocBean("userService")

15.5.42. 问: nutz有自己的标签库吗?

答: 没有

15.5.43. 问: nutz 也是在启动的时 扫描类 然后按照注解注入值吗?

答:. NutIoc启动时,仅仅是加载配置信息,并不马上生成对象

15.5.44. 问: @IocBy能对应到spring提供的ioc不?

答: 能. SpringIocProvider

15.5.45. 问: 有没有用nutz和mongodb结合开发的例子?

答: Nutz做的QA系统

15.5.46. 问: nutz如何关闭debug信息?

答: 加入log4j,把org.nutz下的日志级别改为info

15.5.47. 问: 问个问题512内存的vps能拖起来nutz了吗 ?

答: 能,nutz本身用不了多少内存

15.5.48. 问: nutz 是谁研发的 ? 现在的用户量有多少 ?

答: zozoh发起,现在有几个committer, 用户量没有统计.

15.5.49. 问: nutz 有没有长时间的进行测试过啊?

答: 发布前都有测试覆盖保证的

15.5.50. 问: Nutz 和 etmvc 比较，能比吗?

答: 期待你写一篇对比文章

15.5.51. 问: oracle数据库没有自增,在nutz如何使用dao解决?

答: @Id(auto=false);@Prev("select next_val from dual");

16. 附录

16.1. 案例

16.1.1. 互联网应用

* 迷你志
* 富宝咨询
* 雨忆博客
* 电子科技大学-生命科学与技术学院

16.1.2. 软件产品

* DTRI Signage 系统

16.1.3. 企业应用

* 三喜商务礼品电子商务平台

至 Nutz 的使用者： 据我所知, Nutz 有很多企业信息系统方面的应用，如果你**方便透露**你产品的情况，请发邮件至

* nutzam@googlegroups.com

稍微描述一下你的应用。我们将会记录在这篇文中这篇文档就是用来回答类似：“*Nutz 有没有成功的案例啊？*” 一类的问题的。我希望我们一起来的完成它

请看上面，**企业应用**列表屈指可数，所以，可想而知，我们是多～～～么地渴望您的来信啊!

16.2. 如何创建 DataSource

16.2.1. 何为DataSource

16.2.1.1. 先看看JDK中对DataSource的描述:

* 作为 DriverManager 工具的替代项，DataSource 对象是获取连接的首选方法。
* 基本实现 - 生成标准的 Connection 对象
* 连接池实现 - 生成自动参与连接池的 Connection 对象。此实现与中间层连接池管理器一起使用。

简单来说,就是获取数据库连接的一个通用接口, 常见的dbcp,c3p0,druid,bonecp都是DataSource的实现.

NutDao也选用DataSource作为获取数据库连接的方式, 且只调用其无参数的getConnection()方法,也是大部分数据库连接池唯一支持的方法.

16.2.2. 这篇文档该怎么用?

16.2.2.1. 直接书写 Java 代码

* 如果你只是在main方法中尝试一下NutDao的功能,那么请选取Java

16.2.2.2. 通过 Nutz.Ioc 的 JSON 配置文件

* Nutz项目中最常见的配置方式, 由NutIoc来管理DataSource和NutDao实例
* 特别强调, NutDao与NutIoc没有任何依赖关系, NutDao在NutIoc看来,只是普通的bean

16.2.2.3. 通过 Nutz.Ioc 的 XML 配置文件

* 满足XML强迫症的程序猿, 功能与JSON配置文件类似

16.2.2.4. 再特别特别强调

* NutDao几乎不需要任何配置文件(只有一个nutz_jdbc_experts.js 绝大部分时间你不会遇到它!)
* 本文说到的js/xml文件,都是NutIoc的文件,不是NutDao的配置文件!!
* 不要重复创建DataSource,不要重复创建NutDao!!!!!!!

16.2.3. 内置的SimpleDataSource

16.2.3.1. Nutz内置,非常适合新手!!无需额外下载其他连接池,方便尝试NutDao的功能.

* 不要生产环境中使用这个DataSource!!
* 不要用它来测试NutDao的性能!!
* 自动加载NutDao所支持的数据库的驱动(说白了就是我们认识那几款,不就Class.forName一下嘛)
* 无额外依赖,适合新手试用
* 非连接池,配置简单
* 1.b.43开始提供,旧版本的Nutz可通过拷贝源文件的方式添加这个类

16.2.3.2. SimpleDataSource: 直接书写 Java 代码

```
[Java]
import org.nutz.dao.impl.SimpleDataSource;
...
SimpleDataSource ds = new SimpleDataSource();
//ds.setDriverClassName("org.postgresql.Driver"); //默认加载了大部分数据库的驱动!!
ds.setjdbcUrl("jdbc:postgresql://localhost:5432/mydatabase");
ds.setUsername("demo");
ds.setPassword("123456");
...
//ds.close();  // 这个DataSource不是一个连接池,所以关不关都行
```

16.2.3.3. SimpleDataSource: 通过 Nutz.Ioc 的 JSON 配置文件

```
[IOC-JSON 配置]
{
    dataSource : {
        type : "org.nutz.dao.impl.SimpleDataSource",
        fields : {
            jdbcUrl : 'jdbc:postgresql://localhost:5432/mydatabase',
            username : 'demo',
            password : '123456'
        }
    }
}
```

16.2.3.4. SimpleDataSource: 通过 Nutz.Ioc 的 XML 配置文件

```
[IOC-XML 配置]
    <ioc xsi:noNamespaceSchemaLocation="nutz-ioc-0.1.xsd">
        <obj name="dataSource"
type="org.nutz.dao.impl.SimpleDataSource">
            <field
name="jdbcUrl"><str>jdbc:postgresql://localhost:5432/mydatabase
</str></field>
            <field name="username"><str>demo</str></field>
            <field name="password"><str>123456</str></field>
        </obj>
    </ioc>
```

16.2.3.5. 附送一个完整的NutDao配置js文件

```
[IOC-JSON 配置]
var ioc = {
    dao : {
        type : "org.nutz.dao.impl.NutDao",
        args : [{refer:"dataSource"}]
    },
    dataSource : {
        type : "org.nutz.dao.impl.SimpleDataSource",
```

```
        fields : {
            jdbcUrl : 'jdbc:postgresql://localhost:5432/mydatabase',
            username : 'demo',
            password : '123456'
        }
    }
}
```

如何使用这些配置? 请看文章末尾.

16.2.4. Druid

国产精品连接池,淘宝温少诚意出品,带强大的监控功能哦

16.2.4.1. druid : 直接书写 Java 代码

```
[Java]
import com.alibaba.druid.pool.DruidDataSource;
...
DruidDataSource dds = new DruidDataSource();
dds.setDriverClassName("org.postgresql.Driver");
dds.setUrl("jdbc:postgresql://localhost:5432/mydatabase");
dds.setUsername("enzozhong");
dds.setPassword("123");
...
dds.close();  // 关闭池内所有连接
```

16.2.4.2. druid : 通过 Nutz.Ioc 的 JSON 配置文件

```
[IOC-JSON 配置]
{
    dataSource : {
        type : "com.alibaba.druid.pool.DruidDataSource",
        events : {
            depose : 'close'
        },
        fields : {
            driverClassName : "org.postgresql.Driver",
            url : "jdbc:postgresql://localhost:5432/mydatabase",
            username : "enzozhong",
```

```
        password : "123"
      }
   }
}
```

16.2.4.3. druid: 通过 Nutz.Ioc 的 XML 配置文件

```
[IOC-XML 配置]
   <ioc xsi:noNamespaceSchemaLocation="nutz-ioc-0.1.xsd">
      <obj name="dataSource"
type="com.alibaba.druid.pool.DruidDataSource">
         <events>
            <depose>close</depose>
         </events>
         <field
name="driverClassName"><str>org.postgresql.Driver</str></field>
         <field
name="url"><str>jdbc:postgresql://localhost:5432/mydatabase</str
></field>
         <field name="username"><str>enzozhong</str></field>
         <field name="password"><str>123</str></field>
      </obj>
   </ioc>
```

* 注册了 depose 事件，当整个 Ioc 容器注销时，将 **真正** 关闭所有池内连接
* 更多配置

16.2.5. Apache Tomcat 7 连接池

这里使用的是tomcat7新的自带连接,但是,请把其2个jar移到项目的lib中!!

16.2.5.1. 直接书写 Java 代码

```
[Java]
import org.apache.tomcat.jdbc.pool.DataSource;
...
DataSource ds = new DataSource();
ds.setDriverClassName("org.postgresql.Driver");
ds.setUrl("jdbc:postgresql://localhost:5432/mydatabase");
ds.setUsername("demo");
```

```
ds.setPassword("123456");
...
ds.close();  // 关闭池内所有连接
```

16.2.5.2. 通过 Nutz.Ioc 的 JSON 配置文件

```
[IOC-JSON 配置]
{
    dataSource : {
        type : "org.apache.tomcat.jdbc.pool.DataSource",
        events : {
            depose : 'close'
        },
        fields : {
            driverClassName : 'org.postgresql.Driver',
            url : 'jdbc:postgresql://localhost:5432/mydatabase',
            username : 'demo',
            password : '123456'
        }
    }
}
```

16.2.5.3. 通过 Nutz.Ioc 的 XML 配置文件

```
[IOC-XML 配置]
    <ioc xsi:noNamespaceSchemaLocation="nutz-ioc-0.1.xsd">
        <obj name="dataSource"
type="org.apache.tomcat.jdbc.pool.DataSource">
            <events>
                <depose>close</depose>
            </events>
            <field
name="driverClassName"><str>org.postgresql.Driver</str></field>
            <field
name="url"><str>jdbc:postgresql://localhost:5432/mydatabase</str
></field>
            <field name="username"><str>demo</str></field>
            <field name="password"><str>123456</str></field>
        </obj>
```

```
</ioc>
```

* 注册了 depose 事件，当整个 Ioc 容器注销时，将 **真正** 关闭所有池内连接
* 关于 depose 事件，更多详情请参看 事件监听

16.2.6. Apache DBCP

16.2.6.1. dbcp: 直接书写 Java 代码

```
[Java]
import org.apache.commons.dbcp.BasicDataSource;
...
BasicDataSource ds = new BasicDataSource();
ds.setDriverClassName("org.postgresql.Driver");
ds.setUrl("jdbc:postgresql://localhost:5432/mydatabase");
ds.setUsername("demo");
ds.setPassword("123456");
...
ds.close();  // 关闭池内所有连接
```

16.2.6.2. dbcp: 通过 Nutz.Ioc 的 JSON 配置文件

```
[IOC-JSON 配置]
{
    dataSource : {
        type : "org.apache.commons.dbcp.BasicDataSource",
        events : {
            depose : 'close'
        },
        fields : {
            driverClassName : 'org.postgresql.Driver',
            url : 'jdbc:postgresql://localhost:5432/mydatabase',
            username : 'demo',
            password : '123456'
        }
    }
}
```

16.2.6.3. dbcp: 通过 Nutz.Ioc 的 XML 配置文件

```
[IOC-XML 配置]
    <ioc xsi:noNamespaceSchemaLocation="nutz-ioc-0.1.xsd">
        <obj name="dataSource"
type="org.apache.commons.dbcp.BasicDataSource">
            <events>
                <depose>close</depose>
            </events>
            <field
name="driverClassName"><str>org.postgresql.Driver</str></field>
            <field
name="url"><str>jdbc:postgresql://localhost:5432/mydatabase</str
></field>
            <field name="username"><str>demo</str></field>
            <field name="password"><str>123456</str></field>
        </obj>
    </ioc>
```

* 注册了 depose 事件，当整个 Ioc 容器注销时，将 **真正** 关闭所有池内连接
* 关于 depose 事件，更多详情请参看 事件监听

16.2.7. C3P0

16.2.7.1. c3p0: 直接书写 Java 代码

```
[Java]
import com.mchange.v2.c3p0.ComboPooledDataSource;
...
ComboPooledDataSource ds = new ComboPooledDataSource();
ds.setDriverClass("org.postgresql.Driver");
ds.setJdbcUrl("jdbc:postgresql://localhost:5432/mydatabase");
ds.setUser("demo");
ds.setPassword("123456");
...
ds.close();  // 关闭池内所有连接
```

16.2.7.2. c3p0: 通过 Nutz.Ioc 的 JSON 配置文件

```
[IOC-JSON 配置]
{
    dataSource : {
        type : "com.mchange.v2.c3p0.ComboPooledDataSource",
        events : {
            depose : 'close'
        },
        fields : {
            driverClass : 'org.postgresql.Driver',
            jdbcUrl : 'jdbc:postgresql://localhost:5432/mydatabase',
            user : 'demo',
            password : '123456'
        }
    }
}
```

16.2.7.3. c3p0: 通过 Nutz.Ioc 的 XML 配置文件

```
[IOC-XML 配置]
    <ioc xsi:noNamespaceSchemaLocation="nutz-ioc-0.1.xsd">
        <obj name="dataSource"
type="com.mchange.v2.c3p0.ComboPooledDataSource">
            <events>
                <depose>close</depose>
            </events>
            <field
name="driverClass"><str>org.postgresql.Driver</str></field>
            <field
name="jdbcUrl"><str>jdbc:postgresql://localhost:5432/mydatabase
</str></field>
            <field name="user"><str>demo</str></field>
            <field name="password"><str>123456</str></field>
        </obj>
    </ioc>
```

* 注册了 depose 事件，当整个 Ioc 容器注销时，将 **真正** 关闭所有池内连接

16.2.8. Proxool

16.2.8.1. proxool: 直接书写 Java 代码

```
[Java]
import org.logicalcobwebs.proxool.ProxoolDataSource;
...
ProxoolDataSource ds = new ProxoolDataSource();
ds.setDriver("org.postgresql.Driver");
ds.setDriverUrl("jdbc:postgresql://localhost:5432/mydatabase");
ds.setUser("demo");
ds.setPassword("123456");
...
```

16.2.8.2. proxool: 通过 Nutz.Ioc 的 JSON 配置文件

```
[IOC-JSON 配置]
{
    dataSource : {
        type : "org.logicalcobwebs.proxool.ProxoolDataSource",
        fields : {
            driver : 'org.postgresql.Driver',
            driverUrl : 'jdbc:postgresql://localhost:5432/mydatabase',
            user : 'demo',
            password : '123456'
        }
    }
}
```

16.2.8.3. proxool: 通过 Nutz.Ioc 的 XML 配置文件

```
[IOC-XML 配置]
    <ioc xsi:noNamespaceSchemaLocation="nutz-ioc-0.1.xsd">
        <obj name="dataSource"
type="org.logicalcobwebs.proxool.ProxoolDataSource">
            <field
name="driver"><str>org.postgresql.Driver</str></field>
            <field
name="driverUrl"><str>jdbc:postgresql://localhost:5432/mydatabas
e</str></field>
```

```
        <field name="user"><str>demo</str></field>
        <field name="password"><str>123456</str></field>
    </obj>
</ioc>
```

* Proxool 没有提供关闭所有连接的函数，不过你可以参看它的官方文档，自己写一个释放所有连接的类，配置在 Ioc 容器的 depose 事件中
* 关于 depose 事件，更多详情请参看 事件监听 - **通过实现一个触发器**

16.2.9. BoneCP

16.2.9.1. bonecp: 直接书写 Java 代码

```
[Java]
import com.jolbox.bonecp.BoneCPDataSource;
...
BoneCPDataSource ds = new BoneCPDataSource();
ds.setDriver("org.postgresql.Driver");
ds.setJdbcUrl("jdbc:postgresql://localhost:5432/mydatabase");
ds.setUsername("demo");
ds.setPassword("123456");
...
```

16.2.9.2. bonecp: 通过 Nutz.Ioc 的 JSON 配置文件

```
[IOC-JSON 配置]
{
    dataSource : {
        type : "com.jolbox.bonecp.BoneCPDataSource",
        events : {
            depose : 'close'
        },
        fields : {
            driverClass : 'org.postgresql.Driver',
            jdbcUrl : 'jdbc:postgresql://localhost:5432/mydatabase',
            username : 'demo',
            password : '123456'
        }
    }
}
```

16.2.9.3. bonecp: 通过 Nutz.Ioc 的 XML 配置文件

```
[IOC-XML 配置]
    <ioc xsi:noNamespaceSchemaLocation="nutz-ioc-0.1.xsd">
        <obj name="dataSource"
type="com.jolbox.bonecp.BoneCPDataSource">
            <events>
                <depose>close</depose>
            </events>
            <field
name="driverClass"><str>org.postgresql.Driver</str></field>
            <field
name="url"><str>jdbc:postgresql://localhost:5432/mydatabase</str
></field>
            <field name="username"><str>demo</str></field>
            <field name="password"><str>123456</str></field>
        </obj>
    </ioc>
```

* 注册了 depose 事件，当整个 Ioc 容器注销时，将 **真正** 关闭所有池内连接

16.2.10. 容器提供的连接池(JNDI)

16.2.10.1. Java代码方式:

不写了,这个大家都懂,不懂的自己去google查. 别跟我说baidu没查到!!

16.2.10.2. 由于是通过JNDI获取,所以不再是一个Ioc的bean, 我们只需要引用它就可以了,不需要再写dataSource的bean.例如:

```
[js]
{
    dao : {
        type : "org.nutz.dao.impl.NutDao",
        args : [{jndi:"jdbc/dataSource"}]
    }
}
```

16.2.11. 如何使用这些配置

16.2.11.1. Java代码的方式:

```
[java]
//创建dataSource,以DBCP为例
DataSource ds = new DataSource();
ds.setDriverClassName("org.postgresql.Driver");
ds.setUrl("jdbc:postgresql://localhost:5432/mydatabase");
ds.setUsername("demo");
ds.setPassword("123456");
Dao dao = new NutDao(ds);
dao.create(User.class, true);
dao.insert(User.create("wendal","123456"));
//.... ... ...
//所有操作都已经完成,关闭连接池,退出系统
ds.close();
return;
//额外提醒,NutDao是线程安全的,请不要多次创建NutDao,除非你有多个
DataSource
```

16.2.11.2. 通过 Nutz.Ioc 的 JSON 配置文件

```
[js]
//将配置信息保存到dao.js,并存放于src文件夹下
Ioc ioc = new NutIoc(new JsonLoader("dao.js"));
DataSource ds = ioc.get(DataSource.class);
Dao dao = new NutDao(ds); //如果已经定义了dao,那么改成dao =
ioc.get(Dao.class);
dao.create(User.class, true);
dao.insert(User.create("wendal","123456"));
ioc.depose(); //关闭Ioc容器
```

16.2.11.3. 通过 Nutz.Ioc 的 XML 配置文件

```
[CODE]
//将配置信息保存到dao.xml,并存放于src文件夹下
Ioc ioc = new NutIoc(new XmlIocLoader("dao.js"));
DataSource ds = ioc.get(DataSource.class);
Dao dao = new NutDao(ds); //如果已经定义了dao,那么改成dao =
```

```
ioc.get(Dao.class);
dao.create(User.class, true);
dao.insert(User.create("wendal","123456"));
ioc.depose(); //关闭Ioc容器
```

16.3. 在Maven中使用Nutz

16.3.1. Nutz的POM

当前的POM

```
[CODE]
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.nutz</groupId>
    <artifactId>nutz</artifactId>
    <packaging>jar</packaging>
    <name>Nutz</name>
    <version>1.b.44</version>
    <url>http://nutz.googlecode.com</url>
    <!-- other info ..... -->
</project>
```

你可以在 这里找到当前POM

16.3.2. 如何使用

将Nutz加入你的POM

```
[CODE]
<dependency>
    <groupId>org.nutz</groupId>
    <artifactId>nutz</artifactId>
    <version>1.b.44</version>
</dependency>
```

无需添加repository即可使用Nutz.不要惊讶,因为我们已经将Nutz提交到maven中央仓库!

16.3.3. Nutz使用Maven来构建的吗?

答案是否定的,我们现在仍在使用Ant进行构建.

16.4. 推荐文章

16.4.1. 关于这个项目

* 《不编写代码，你可以能为这个项目做很多事情》 @hilliate **推荐**
* 《Nutz 的设计以及提高程序员生产力》 @zozoh

16.4.2. 关于 Dao

* 《使nutz支持类似hibernate的二级缓存,现支持HashtableCache,OSCache,EhCache》@finallygo **推荐**
* 《Nutz DAO与spring集成讨论》@知足常乐 **推荐**
* 《Nutz DAO懒加载实体关联对象》 @天行健 **推荐**
* 《Nutz的 数据库事务》@amosleaf
* 《Nutz的 使用注解做ORM》@amosleaf
* 《借于Nutz快速实现对表的增删改操作》@会跑的蜗牛
* 《NUTZ与SQL SERVER2000兼容性问题》@Ge.PH /hl
* 《用Nutz做点小任务》@小毛
* 《使用Nutz进行简单的增删改查操作》@CH
* 《使用NutzDao进行复杂SQL条件查询》@CH
* 《NutzDao-自定义SQL语句进行复杂查询》@CH

16.4.3. 关于 Mvc

* 《使用Nutz的json视图实现前台密码验证 @mamacmm **新**
* 《让Nutz支持最快的模板引擎Smarty4j @QinerG **推荐**
* 《在Nutz框架中提供Pojo校验（验证）功能的支持》 @QinerG **推荐**
* 《在Nutz MVC中使用Freemarker》 @wendal **推荐**
* 《给 nutz 添加 freemarker 视图》 @axhack **推荐**
* 《nutz 文件上传例子》 @云海飞舞
* 《nutz 过滤器使用例子》 @云海飞舞
* 《我的第一个Nutz程序》 @Bird.Wyatt
* 《nutz初使用之MVC HelloWorld (netbeans html,jquery版)》 @云海飞舞
* 《nutz初使用之MVC HelloWorld (netbeans jsp版）》 @云海飞舞
* 《为NutLab添加一个新项目-Nutz与OpenID集成》 @wendal

16.4.4. 关于 Ioc

* 《在Nutz中使用Ioc-Annotation的入门教程》@Gevin **新**
* 《Nutz 1.a.27 最新变化之一 -- XML配置ioc》@wendal

16.4.5. 关于 Aop

* 《基于ASM的Nut.Aop实现》@wendal
* 《自己动手,用字节码工具做一个Aop拦截器》@wendal

16.4.6. 综合实践

* 《使用Nutz+ExtJS+JBPM4.4实现会签》 @胖五 新
* 《ExtJS+Nutz+JBPM实现一个简单的请假流程》 @胖五 新
* 《基于Nutz与ExtJs的快速开发》 @胖五 新
* 《Nutz+ExtJS示例教程——搭建开发环境》 @胖五 新
* 《nutz on gae 系列： 1.1 gae数据存储》 @feiyan 新
* 《nutz简单综合实例----通过html网页对数据库进行管理操作（MVC,Ioc,Dao)》 @云海飞舞

16.4.7. 其他文章

* 《nutz项目读取properties文件时发现的两端空格问题》@hilliate

16.5. 可用的插件

16.5.1. 插件中心

插件中心

16.5.2. IocProvider

Spring桥

16.5.3. IocLoader

暂无

16.5.4. View

FreeMaker_View

Nutz集成FreeMarker

16.5.5. Dao

暂无

16.5.6. Aop

暂无

16.5.7. Castor

暂无