GlassFish 社区奉献的 Java EE 应用服务器

Eduardo Pelegri-Llopart
Yutaka Yoshida
Alexis Moussine-Pouchkine
Sun Microsystems, Inc.

http://blogs.sun.com/theaquarium

最后更新 2007 年 9 月

在 JavaOne 2005 上，Sun 宣布了 *GlassFish 项目*，这是一个关于开源的应用服务器和 Java EE 的参考实现的项目(参看：GlassFish 项目)。这迈出了所有 Java 平台开源的第一步，此外它还有着更多的意义。GlassFish 项目加速了 Java EE5 的采用，为开源社区在具备企业品质的应用服务器方面增添了新的选择，并促使 Sun 在应用服务器产品的开发、测试、提供和改进方面进行方式转变，以提供更优秀的产品。

在项目启动一年半后，*GlassFish 社区*已经发布了它的第一个正式版本，并且正在开发其第二个版本。这篇文章里我们将就 GlassFish 社区的各个方面和 GlassFish 应用服务器进行总体概述。

GlassFish 项目

Sun 于 2005 年 6 月 JavaOne 期间启动了 GlassFish 项目，开源的 Java EE 5 参考实现和 Sun 的应用服务器。在 JavaOne 2006 (2006 年 5 月)，不到一年的时间内就发布了其第一个版本。第二个版本于 2007 年 9 月发布。目前社区已将开发重点放在 GlassFish v3－轻量级的模块化的应用服务器上。

什么是 GlassFish

GlassFish 是一个社区和一个应用服务器产品。GlassFish 社区提交的主要是与 Java EE 5 兼容的应用服务器－GlassFish 应用服务器，以及 Java Persistence API 参考实现－TopLink Essentials。

GlassFish 社区提交了许多符合 Java EE 5 的 JCP 规范的可重用的组件，这其中包括 JAXB, JAX-WS, JAXP, StAX, JSP 和 JSF。GlassFish 还包括了几个流行的 Web.Next 项目比如 jMaki, Phobos 和 dynaFaces，还有许多工具比如持续化的集成工具 Hudson，以及有用的架构比如基于 NIO 服务器框架的 Grizzly。GlassFish 社区还为所有这些组件维护着一个 Maven 存储库(repository)。

Java 平台和应用服务器

Sun 于 1995 年发布 Java 并于次年的第一届 JavaOne 大会上展示了 Java Servlet API。到了 1999 年，Java Server Pages 和 Enterprise JavaBeans 连同 Java Servlets 被组装到第一个 Java 企业平台 J2EE 1.2 - 其版本号是为匹配相应的 J2SE。接下来发布的版本是 2001 年的 J2EE 1.3 和 2003 年的

J2EE 1.4，在2006年，这一平台的名称最后变更为Java EE 5。

Java EE规范被广泛地被包括Sun在内的软件提供商和企业所接受，并在他们的应用服务器中加以实现。多年来，Sun将参考实现(参看：什么是参考实现)和商业的应用服务器标成不同的品牌发布。这两者有着不同的用途：RI专注于开发和传授规范，可以免费开发但不能在真正的生产环境下使用，但商业产品面向市场，并且侧重在部署后的使用。7.0商业版(带有SunOne的商标)第一次包括了J2EE 1.3 SDK的功能，自此两者被统一起来，在其后以发布Sun Java System商标发布的8.x和9.x延续了这一做法（参看：Sun的应用服务器)。

到了J2EE 1.4，其参考实现和SJS AS 8.0 PE是相同的，可自由开发和部署，并且包含在为广泛分发的J2EE 1.4 SDK中。

SJS AS 8.x产品提供3种不同的版本：*Platform Edition* (PE) 和参考实现相同，*Standard Edition* (SE) 包括一些企业品质的特性比如集群Clustering和Failover，*Enterprise Edition* (EE) 提供给要求高可用性(99.999%的可用性)的市场。从9.x版开始，所有特性都被集中到单一的版本，这就是GlassFish社区在开源许可下所提供的版

什么是参考实现

规范的创建是经过Java Community Process (JCP)程序，并由专家组（Expert Group，EG)负责领导完成。专家组由一位被称为EG Lead的人负责组织协调。

专家组的产出包括：

- 规范文档
- 技术兼容包(TCK)
- 参考实现(RI)

参考实现是规范的实现，并通过了TCK验证。参考实现的质量可以从简单的概念验证到产品质量的实现。GlassFish就是一个拥有产品质量，Java EE 5兼容的应用服务器。

严格地说，Java EE 5的RI是一个GlassFish代码的特别快照(snapshot)。

Sun的应用服务器

Sun为其应用服务器打上了多个商标。近期的主要发布如下：

- iPlanet AS 6.0 －源自Netscape (Netscape AS)，兼容J2EE 1.2
- SunOne AppServer 7.0 － J2EE 1.3 参考实现中的第一个版本
- Sun Java System AppServer 8.0 －兼容J2EE 1.4
- Sun Java System AppServer 9.0 －兼容Java EE 5，GlassFish v1
- Sun Java System AppServer 9.1 －兼容Java EE 5， GlassFish v2

本。用户可以自由地设定 GlassFish 所用的 3 个 profile： developer, cluster 和 enterprise。

GlassFish 应用服务器

2005 年 6 月在 JavaOne 2005 上，Sun 公司宣布启动 GlassFish 项目，不到一年的时间，2006 年 5 月即下一个 JavaOne 大会上，GlassFish 的第一个版本面世并且 JavaEE 5 的规范也正式推出。

这个应用服务器产品可以在 Sun 网站通过下载 Sun Java System AS 9.0 PE 获取，也可以在 GlassFish 社区通过下载 GlassFish v1 获取，除了安装程序，两者是相同的。

GlassFish v1 侧重于 Java EE 5 规范的实现，一些企业级的特性并没有包含在这一版本中。此应用服务器在 Sun 发布中被标为 PE，就反映了这一点。GlassFish v2 加上了所有企业级的特性，并且去除了 PE 标签，同一个可执行文件可根据所期望的用途安装成为 developer，enterprise 或者 cluster 的 profile。

开源许可

OSI 组织(Open Source Initiative)承认多种的许可。许可之间通常有着某种继承关系。

ASL 许可出自 BSD 和 MIT 许可，它的使用非常灵活。

GPL, LGPL, GPLv2 和 GPL+CPE 都是 GNU 许可。 GNU 许可强调保留原始代码的“自由性”。GPL 许可是基于项目的。

CDDL 许可派生自 MPL(Mozilla Public License)。不同于 GPL 许可，这些许可是基于文件的。

学习开源许可的好地方是 OSI 组织的网站(opensource.org)。

GlassFish v2 可以在 Sun 网站通过下载 Sun Java System AS 9.1 PE 获取，它发布于 2007 年的 9 月。

Java 平台

Java EE 规范是一个 umbrella 规范，构建在相关 Java SE 规范（即 Java EE5 需要 Java SE 5）和许多其它 JCP 规范之上。GlassFish 社区反映了这点，并且包括了许多子项目作为这些规范的官方参考实现。通过这些子项目，可以更容易被其它组织和项目重用。例如，JAXB 的实现被用在许多项目，这些项目大部分和 Sun 公司无关。

Java 标准接口是专家组通过 Java 社区流程(Java Community Process)开发

的，许多专家组的领导本身就是GlassFish社区的成员。GlassFish社区实现了这些接口，并在JCP的审核过程中提供反馈并对将来的版本提出需求。

关于Java EE 6的工作已经通过JSR 316展开，这是个umbrella项目，包括比如EJB 3.1，JPA2.0等规范。GlassFish v3 将实现Java EE 6规范。

开源和GlassFish

Tomcat是被广泛采用的Java服务器端项目。Tomcat由Apache通过包括Sun和JServ开发人员构成的工作组发起，并且成为Java Servlets 和Java Server Pages规范早期版本的参考实现(GlassFish现在是这些规范最新的参考实现)。Tomcat对于早期服务器端的Java技术被认可和采用起到了关键的作用，因为它在开源许可下提供，它也为开源软件在企业间的流行作出了贡献。

有许多不同的许可（参看“开源许可”），这里并不打算详细地描述它们。各类有着不同特性的许可吸引着不同的社区：许可中被某社区认为是优点的特性可能被另一个社区认为是缺点，例如，Apache社区使用 ASL许可而GNU社区使用GPL许可。

当GlassFish发布之初，仅使用Common Development and Distribution License(CDDL)，但是为了配合Java SE 和Java ME 在2006年11月所开始的开源， Sun重新评估了来自不同社区的请求并决定对Java ME采用GPLv2许可，对Java SE 采用GPLv2 加ClassPath异常 (GPLv2+CPE)的许可。使用GPL许可增强了GNU 的ClassPath 和Kaffe 社区，而CPE允许链接非GPL的构件。这样，GlassFish 社区在CDDL的基础上添加了GPLv2+CPE 许可，成为双许可。双许可意味着可以使用其中任一个许可，这样可以在为GlassFish的商业伙伴保留CDDL的优点的同时，增强和GNU、Java SE和Java Me社区的合作。在GlassFish所分发的软件中不存在二进制许可。

采用GlassFish 和Java EE 5

开源的参与、产品级的质量和Java EE 5的参考实现使得Java EE 5的需求和供给都得到增加，从而加速了这个平台的采用。此文写作之时，GlassFish已经有超过 1百万次的下载，这些从事开发和部署的人员感受到了Java EE 5所带来的好处并要求增强它所支持的特性。GlassFish应用服务器满足了其中一些要求，同时它还在帮助其它的Java应用开发商使用GlassFish的具有产品品质的组件来更快且花销更小地完成一个Java EE 5 兼容的实现。

使用GlassFish的应用开发商包括：TmaxSoft，Oracle，BEA，Jboss，Jetty和Geronimo。TmaxSoft是韩国最早获取Java EE许可的公司，他们的最新产品JEUS 6使用了JAXB，JAX-WS，Fast Infoset，JSP，JSF和TopLink Essentials。通过参与GlassFish社区，TmaxSoft提前获取了许多组件，并确保这些组件可以很好地满足他们的要求。这使得TmaxSoft的JEUS 6成为在Sun之后第一个通过Java EE 5 认证的应用服务器。

Oracle尚没发布完全支持Java EE 5的应用服务器，但他们为GlassFish社

Java EE 5中的WS Endpoints

下面的代示了一个WS end point:

```
import javax.jws.WebService;
@WebService
public class MySimpleWS {
    public String sayHello(String s) {
      return “Hello” +s;
    }
}
```

 里能如下面的代所示一个(transactional) Stateless Session Bean：

```
import javax.jws.WebService;
import javax.ejb.Stateless;
@WebService
@Stateless
public class MySimpleWS {
   public String sayHello(String s) {
       return “Hello” +s;
   }
}
```

资源注射

J2EE 1.4 需要相当多的template code。下面是典型的 EJB引用：

```
Context initial = new
InitialContext();
Context myEnv = (Context)initial....
Object objref =myEnv.lookup(“...”);
ConverterHome home = (ConverterHome)
PortableRemoteObject.narrow(...);
Converter currentConverter =
home.create();
```

在Java EE 5，通过约定和资源注射，我们只要说：

```
@EJB Converter currencyConverter;
```

区贡献了 TopLink Essentials 项目，并将其应用在自己的产品中。在上面提到的提供商中，BEA 使用了 JAXB 和 JAX-WS 技术，JBoss 使用了 JAXB 和 JSF 技术，Jetty 使用了 JSP 和 Grizzly 技术，Geronimo 使用了 JAXB 技术。

Java EE 5 概览

简化开发是 Java EE 5 的主旋律，而主要实现这一目标的方法是借助 Java SE 5 的 annotations 技术使得可以基于 POJO(Plain Old Java Objects)进行编程。Annotation 用于许多方面包括方法和类的属性(参看：Java EE 5 中的 WS Endpoints)，资源注射(参看：资源注射)，以及可移植的行为描述(参看：缺省的 annotation)。

主要的 Java EE 5 规范如下：

- JAX-WS 2.0 和 JAXB 2.0
- EJB 3.0 和 Java Persistence API
- JSF 1.2 和 JSP 2.1
- StAX

其他一些规范 (比如 JAXP 和 Servlet)也有变化相对较小的维护和发布。这些变化在开发体验上所带来的总的效果是非常显著的。

Java EE 5 和 GlassFish 紧密关联：Java EE 5 来自规范的力量增加了 GlassFish 的价值，同时获取 GlassFish 可帮助验证 Java EE 5。我们希望将来能够在 Java EE 平台中保持二者的这种关系，为规范的不断改进提供更有效的反馈，通过早期获取参考实现以促进社区更早地尝试规范。

缺省的 **annotation**

下面是 JAXB 2.0 生成的典型代码。

```
@XmlAccessorType(FIELD)
@XmlType(name="",propOrder={"x","y"})
@XmlRootElement(name="point")
public class Point {
  protected float x, y;
  public float getX() {return x; }
  public setX(float value){ this.x = value; }
  public float getY() {return y; }
  public void setY(float value) { this.y = value;}
}
```

我们正在尝试如何使用 annotation。例如，JAXB 2.0 规范由两部分构成。一部分是描述如何映射 XML Schema 到 annotation，另一部分是描述这些 annotation 的语义。这意味着通过修改生成的代码，仍可以将应用移植到兼容的应用服务器上运行。特

殊情况下，需要插入和增加代码以调用 setters 和 getters 方法，使之可以移植。

GlassFish 发布

GlassFish 有 3 个处在不同开发阶段的版本，它们是：V1，V2，V3。目前，关于 GlassFish 发布的几个周期性阶段如下所示(目前仍是社区的早期阶段，相关的定义仍将不断完善)：

- 概念创建(Concept Creation) - 收集关键特性，粗略的时间表，原型。
- 活跃开发(Active Development) - 为里程碑和发布正式版本进行实现。
- 维护(Maintenance) - 为正式版本进行错误修订，创建更新版本。

GlassFish v1 目前处于维护阶段，GlassFish v2 刚刚发布并进入维护阶段，而 GlassFish v3 正从概念创建转向活跃开发阶段。它将可能在实现其模块化特性后发布。

GlassFish v1

GlassFish v1 目前处于维护阶段。其最后一个版本发布于 2006 年五月，正值 2006 年 JavaOne 大会之前，之后又发布于 2006 年 10 月发布了错误修订的 v1 UR1 并于 2006 年 12 月发布了第 2 个改动更小的 v1 UR1p1 。目前 v1 没有更多的维护版本发布的计划，因为主要的开发已经转移到 v2 上来。

GlassFish v1 也被作为官方正式的 Java EE 5 的参考实现。

同时，Sun 公司将 GlassFish v1 以 Sun Java System AS 9.0 PE 的名义分发，并将它包含在其他一些 Sun 公司所分发的包装中，这其中包括 Java EE SDK, Java Application Platform SDK 和 NetBeans 5.5 Tools Bundle。GlassFish v1 和 SJS AS 9.0 PE 仅有的不同就是安装程序。Sun 提供不同层次的商业支持给 GlassFish v1 (SJS AS 9.0 PE)的最后版本，并由社区来全力提供支持良好的支持。

GlassFish v2

GlassFish v2 的正式版本于 2007 年 9 月发布。这个版本中最主要并且是最重要的特性是 clustering(包括 grouping, loadbalancing,data replication)。

GlassFish v2 继承了 SJS AS 8.2 SE/EE 所有企业级的特性并经过 Java

Enterprise System 中所有中间件产品的测试。GlassFish v2 支持 profile 的概念并支持将同一个可执行应用配置成 developer, enterprise 或是 cluster 的 profile。enterprise 的 profile 也可以被配置成使用 HADB (High Availability Data Base) 来达到高可用性 (99.999%)。

Sun 将 GlassFish v2 标以 Sun Java System AS 9.1 对外发布并发布在一系列的软件包中(比如 Java ES 5.1)，并为其提供商业的和社区的支持。

GlassFish v3

关于 GlassFish v3 的演示在 JavaOne 2007 上得到广泛关注。它缺省采用模块化(modular)架构，内核特别小（小于 100Kb，这使得它也适用于桌面甚至移动应用)，它的启动时间少于 1 秒。容器被多个模块化的内核所支持，目前已集成有 Java Web, PHP (通过 Caucho 的 Quercus), jRuby on Rails, Phobos JavaScript。更多的模块将会推出，也欢迎你来开发自己的模块。

GlassFish v3 的预览版本位于 http://glassfish.java.net ，相关文档位于 http://wiki.glassfish.java.net 和 http://hk2.dev.java.net。

GlassFish v3 上的开发正在进行。相关计划尚在制定中，其开发的模块化特性很可能使之发布成一个在 Java EE 6 上完整提供特性的产品。

GlassFish v2 的特性

这一节描述了 GlassFish V2 的特性。对许多特性只是作了简要的概述，更多的信息将提供在网上。

Web 层是许多企业应用的重点，GlassFish 提供了丰富的 Web 层支持。

Web 层的 Java 规范

GlassFish 支持最新的 JCP 规范: JSP 2.1, JSF 1.2, Servlet 2.5 和 JSTL 1.2。

Java Server Faces(简称 JSF, 下一代基于标准的 MVC 架构) 同样被加入到 Java EE 5 平台中，JSF 在 Java 表现层提供了一个组件模型。JSF 能够和 JavaServer Pages (JSP)或其他的如 facelets 一起使用。JSF 1.2 对早期版本做了很多改进，尤其是在统一表达式语言(unified expression language）方面。统一表达式语言目前已经可以被 JSF 和 JSP 所共用。JSF 1.2 还包括了一些其他方面的改进，比如请求的生命周期(request cycle)以及对 AJAX 的支持。

JSP 2.1 主要的改进是在统一表达式语言方面，Servlet 2.5 和 JSTL 同样有新的变化但变化并不大。

GlassFish 还包括了不少在规范之外的重要改进。

GlassFish 的 JSF 实现在性能上有了显著的提高并且解决了一些关键问题。其 JSF 的实现方法被 JBoss 和其他一些组织采用。Servlet 容器的实现源自 Tomcat，但现在由于稳定性和性能方面的原因，GlassFish 在独立地维护它。

GlassFish 另一个显著的改进发生在 Jasper(JSP 编译器)上，它能够利用 Java SE 6 中的编译器接口 (JSR-199)来避免文件的读写并且编译得更快(据称 10 倍以上)。Jasper 能够被配置成使用 Eclipse 的 JDT 编译器，虽然不如使用 JSR-199 时快 。JSF 的性能已经得到充分提高，虽然目前还没有其 benchmark 的结果。

GlassFish 开始使用 Woodstock 项目中的高品质的 JSF 组件，许多开发人员对此感兴趣。这些组件已于 http://woodstock.dev.java.net 开源。

性能提高

GlassFish v2 的性能在很多方面得到了提高。这其中包括 Web 层的 JSF, JSP 和 Grizzly，Web 服务方面的 JAXB 和 JAX-WS，以及 CORBA 和 EJB。

所有这些创造了 GlassFish v2 在 SPECjAppServer 2004 上的世界记录。2007 年 7 月，Sun 宣布了在 T2000 服务器上创造的 883.66 JOPS@Standard 的位居第一的处理能力。这个结果比 GlassFish V1/SJSAS 9.0 快了 60%，比 BEA WebLogic 快了 10%，比 IBM WebSphere 6.1 快了 30%！另一项 benchmark 的结果(使用 PostgreSQL 数据库获得 813.73 JOPS@Standard 处理能力)显示出 3 倍于 Oracle 和 HP 分数的 rice/perf ration。

很明显，**你不须被迫在开源软件和企业级特性中间作出选择：你可以同时拥有二者**。

说明：SPEC 和命名为 SPECjAppServer 2004 的 benchmark 被注册为 Standard Performance Evaluation 公司的商标。Sun Fire T2000 (1 芯 8 核) 1.4ghz 883.66 SPECjAppServer2004 JOPS@Standard。上述 benchmark 比较结果于 07/10/07 发布于 www.spec.org 。要获得最新 SPECjAppServer 2004 benchmark 结果，请访问 http://www.spec.org/.

提高 GlassFish 的启动性能被予以特别的重视。通过简化启动和关闭的设计，分析启动时的依赖关系，将非必须的服务的初始化时间被减少到最小。启动和关闭的时间无论是对单个实例还是整个集群都被显著地减少。

和流行架构的兼容

多年来 Java 社区的核心力量之一就是它的多样化，并且这种多样化特别明显地体现在 Web 层各种形式的框架上。这些框架有效地提高了开发效率。GlassFish 社区一个直接目标就是确保这些框架在 GlassFish 的不同版本上工作良好。类似的，一些流行的应用也被确保和 GlassFish 兼容。

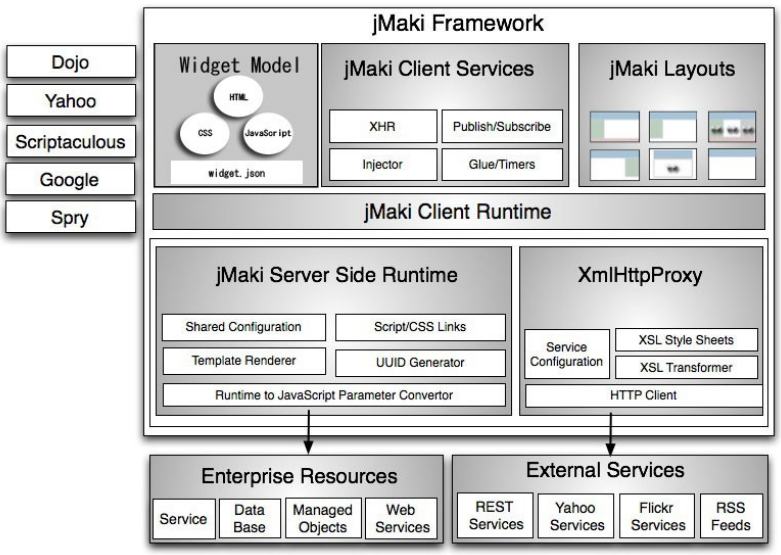
所兼容的框架的清单非常庞大且在不断增长，以下只是一个小的子集：

AppFuse, DWR, Facelets, IBATIS, JBoss Seam, Shale, Spring, Struts, Tapestry,WebWork, Wicket 等。

AJAX 支持

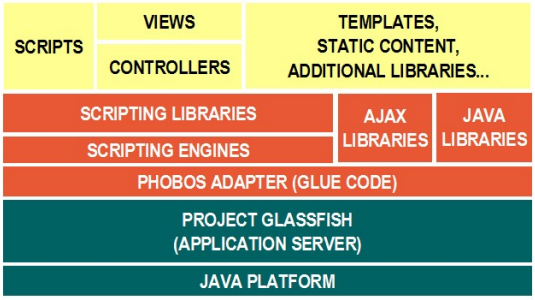
GlassFish 社区里有许多项目是关于 AJAX 的，其中多数项目并不仅限于在 GlassFish 上使用，也有一些是依赖于 GlassFish 特有的扩展。

jMaki(ajax.dev.java.net)是一个提供了轻量级模型的框架，它支持使用 Java、PHP 和 Phobos 语言来创建以 JavaScript 为核心的支持 AJAX 的 web 应用。JMake 的模型基于 JavaScript、HTML 和 CSS，并且可以被方便地修改。jMaki 和许多 AJAX 的组件库一起工作，包括 Dojo，Yahoo，Scriptaculous，Google 和 Spry 并可以方便地添加新的组件库。jMaki 不太依赖服务器端的模型，容易被采纳和集成。



Phobos (phobos.dev.java.net)是一个轻量级的适合脚本的运行在 Java 平台

上的 web 应用环境，致力于满足新兴(emerging)开发者的需求。Phobos 的出发点是方便 JavaScript 的使用但设计时同样考虑了其它动态语言的支持。比起 jMaki，Phobos 是一个更加雄心勃勃的项目，虽然只是早期的开发阶段但它已引起广泛的关注。通过服务器端和客户端的 JavaScript 脚本借助 JSON 和代码重用来简化内容传输。其 Java 脚本引擎 Rhino 非常坚实，并提供有 E4X 扩展和 bytecode 兼容性。



Phobos Architecture

GlassFish 还拥有一些和 AJAX 相关的 JSF 项目，包括 Woodstock(一组包括了 AJAX 组件的 JSF 组件)和 Dynamic Faces(高质量将 AJAX 和 JSF 集成)。此外，还有象 DWR 和 AJAX4JSF 这样的项目支持 AJAX。

Grizzly

GlassFish 的 Web 层的底层实现采用了 Grizzly 架构(grizzly.dev.java.net)。这个架构用 Java 写成，利用了 NIO 接口(可扩展的网络和文件 I/O)来提供扩展性，它高度可定制而且相当通用。Grizzly HTTP 连接器的角色类似 Tomcat 中的基于 Java 的 Coyote 连接器和基于 C 的 Apache Portable Runtime (APR)。一些初步的非正式的测试已经显示出其具有很好的扩展性。

Grizzly 受到许多人的推崇因为它通过利用 NIO 使得自身的使用非常简单，比如， AsyncWeb 和 jetty 都在原型中集成了它。Grizzly 正在被改造得更加通用，以提供进一步的功能来满足 GlassFish 社区内外各类组织的要求。

Grizzly 能够被剥离出 GlassFish，单独使用，例如它被 NetBeans 的 Phobos and Ruby 的 plugins 所使用的方式。Grizzly 的可扩展性能够被用来为“长时间持续的 HTTP 连接(long-lasting HTTP connections)”提供有效的支持，这被称作是 Comet。Comet 技术可使用在比如聊天、在线日历或文档共享

之类的应用中，持续地完成内容更新。

表现层的替代

GlassFish 社区一个基本的旋律就是拥抱多样性。这样做的理由有许多，从“竞争出质量” 到 “一个尺寸无法适用所有” 到“创新随处发生”。GlassFish 对于表现层上 Java 或其新的替代平台上的典型架构所作的支持突出地体现出这一点。在某些情况下，这些替代平台直接驻留在 GlassFish 应用服务器上，但另一些情况下，它们和应用服务器的关系不那么紧密。

如前所述，服务器端的 JavaScript 是由 Phobos 项目来支持的。在这种情况下，服务器端的 JavaScript 代码通过 Rhino 解释器在 GlassFish 上执行。

基于 Ruby 的应用，比如，最著名的 Rails, 能够通过两个不同的方式执行。Rails 在 jRuby (jruby.codehaus.org)上执行，是基于 Java 平台的方式。另一种方式，Rails 通过 native 的 Ruby 解释器来执行， 并通过 CGI 接口来和 GlassFish 通讯。在 GlassFish 上运行 Rails(Rails on GlassFish)是特别有吸引力的想法而且得到了积极的探讨，请继续关注这方面的发展。

PHP 也能够作为外部的 native 解释器在 GlassFish 执行。Quercus(由 Caucho 开发)是基于 Java 平台的 PHP 5 实现，Caucho 和 GlassFish 社区正在一起致力于使得 Quercus 能在 GlassFish 上工作。

Web 服务 - Metro

GlassFish 的 Web 服务栈被称作“Metro”。GlassFish v1 通过新的 JAX-WS 2.0 API(jax-ws.dev.java.net)在简化 Web 服务开发方面作出巨大的改进。GlassFish v2 继续这方面的改进，并支持 JAX-WS 2.1 规范 ，但这些改进主要来自多年来的演变和几个重新的设计。Metro 的实现是经得起考验的，其完整功能在基于 AXIS 2 stack (见下图)的 benchmarks 测试中有 30%-100%的性能提高。我们相信它是业界领先的技术。JAX-WS 2.0 同样在 Sun 的 Java SE 6 参考实现中提供并且能够升级到 JAX-WS 2.1。

SJS AS 8.2 有着对 JAX-RPC 1.1 的实现。JAX-WS 2.0 的初步实现在 GlassFish v1 中被使用，它松散地基于 JAX-RPC 1.1 之上，但有着两方面的变化，这两方面的变化是为了应对两个主要的挑战。第一个挑战是由新的规范带来的：不同于 JAX-RPC 1.1，JAX-WS 2.0 将所有的数据绑定交由 JAXB 2.0 处理并且严格地支持非 HTTP 的协议。另一方面，Sun 意识到 XML 文本编码上所带来的性能损耗并且开始参与制定高效 XML 编码的各类标准 (参看:

Fast Infoset)。

结果是 JAX-WS 的实现被重构，它被清楚地分离成传送(Transport),编码(Encoding)和数据绑定(Data Binding)。这一新的 Metro 架构被 GlassFish v2 集成并且支持以下特性：

● 多种传送(Multiple Transports)－实现基于 HTTP，JMS 和 TCP/IP 协议上的传送并且对 SMTP 协议进行了设计考虑。

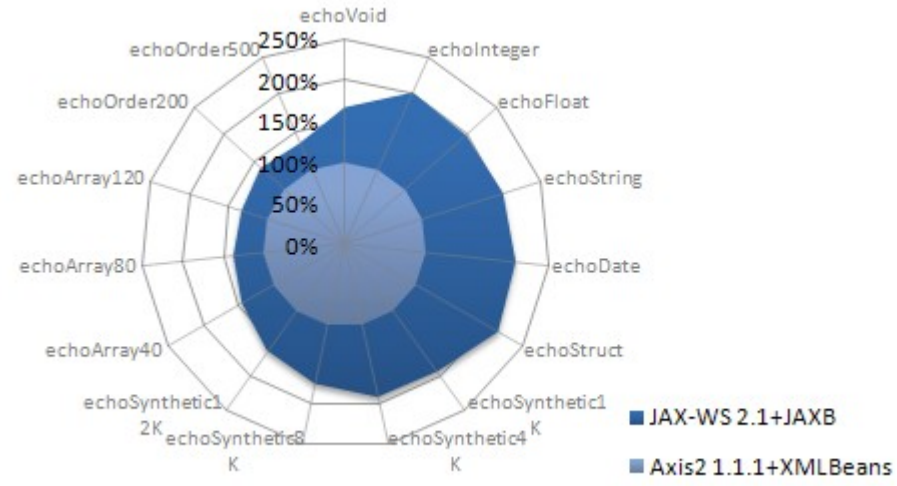
● 多种编码(Multiple Encodings)－支持文本(Textual) XML，MTOM 和 Fast Infoset 等多种方式编码

● 数据绑定(Data Binding)－由 JAXB 控制数据绑定，小心地处理数据移交(handover)以避免数据拷贝。

这个架构被设计成支持多个可叠放(stackable)的处理实体，每个实体实现一个 WS-*规范，而不会导致数据拷贝。

其架构见下图：

性能也是一个目标，JAX-WS 2.0 的实现性能比原先的 JAX-RPC 1.1 栈(曾被认为具有高性能和高竞争性)高出一倍。关于最近的 JAX-WS 2.1 栈和 Axis 2 性能测试的比较结果显示 GlassFish 团队提供了业界最好的栈。



然而性能并不是考量生产力的唯一要素，在某种程度上扩展性更加重要，业务集成将要求能够处理成千上万的事务，其中一些是长时间激活的(long-lived)。通过任务共享执行线程池，只需很少的线程就可完成大量的事务处理。

WSIT 和项目 Tango

Web 服务的需求之一就是搭建桥梁，实现 Java 平台和 Microsoft 平台之间互操作。Tango 项目是 GlassFish 社区发起的关于这类互操作的方案，其正式名称是 WSIT(Web Services Interoperability Technologies，wsit.dev.java.net)。它是 Metro (metro.dev.java.net)的子集。

项目 Tango 的详细描述，可以参见文章：http://wsit.dev.java.net/docs/tango-overview.pdf

WSIT 利用了 GlassFish v2 中新的 WS 架构，通过 WS 管道(pipeline)中的元素去处理不同的 WS-* 标准。WSIT 透明地工作，开发人员可以使用标准的 JAX-WS 开发模型并通过 NetBeans 增加 WS-*属性(要包含一个 NetBeans 的 plugin)。WSIT 保证了和其它提供商的互操作性，主要是 Microsoft，这一互操作方案已在许多 WS Plugfests 上测试过。

下列标准目前被支持：

Boostrapping: WS-MetadataExchange, WS-MetadataExchange WSDL, WS-Transfer

Reliable Messaging: WS-ReliableMessaging, WSReliableMessaging Policy

Atomic Transactions: WS-Coordination and WSAtomic Transaction

安全：WS-Security, WS-SecurityPolicy, WS-Trust, WS-SecureConversation

策略: WS-Policy, WS-PolicyAttachment

XML ·理

GlassFish 对关于 XML 处理的 JAXP 和 StAX 接口提供支持。

The Streaming API for XML(StAX) 由 BEA 领导是新的 XML 解析接口，并且它已成为 Java 平台的一个新的部分，包含在 Java EE 5 和 Java SE 6 平台中。StAX 能够被用来替代 SAX 和 DOM，并且有着不同的性能表现和接口特性。DOM 是一个驻留在内存中以文档为核心的开发接口。它将文档解析成一组能够以任何次序访问的对象，但这种灵活性的获取是以性能受到显著影响为代价的，这种影响包括了内存的消耗和解析的性能。SAX 是一种事件驱动的开发接口，可以在 XML 解析到新元素时进行回调（callbacks）。 SAX 比 DOM 有着更好的性能特性，但当事件响应要求状态进行更新时，它会导致控制反转（an inversion of control）。

Fast Infoset

在某些情况下 XML 文档相当冗长以致于妨碍了对它的应用，不仅是因为文档大小，而且因为传输文档的花费。Gzip 编码压缩了大小但需花费更多的时间在编码上，Fast Infoset ANSI/ISO 标准在这方面提供了更好的平衡。

目前有 4 种不同的 Fast Infoset 商业实现 ，但其它标准也许会主导这个方向 (例如 W3C EXI 工作组也在关注这个领域)，所以 GlassFish Web Services stack 被设计成支持多种编码。它目前支持 MTOM 和 Fast Infoset 编码（除了文本编码），但当其它标准出现时它也会支持。

Fast Infoset 能够和 WSIT 一起使用，Windows Communication Foundation (WCF) 提供了编码隔离(encoding isolation)，一些厂家对其扩展并开发了对 FI 的支持。

拉式解析器(pull parser)具有了SAX的性能优势，同时主控代码保留了对需要时事件的控制。近来GlassFish社区在和Woodstox 社区(Codehaus.org，提供有高度优化的StAX实现)进行合作开发。

最新的JAXP规范是JAXP 1.4。JAXP 1.4是JAXP 1.3以来的一个维护版本。JAXP 1.4包括了一系列的改进，最主要的变化是和StAX 的集成。 JAXP 1.4被包含在Java SE 6中，J2SE 5.0包含的只是JAXP 1.3。 GlassFish 可以和J2SE 5 或Java SE 6一起工作。

JAXP 1.4的实现包括了JAXP 1.4 和StAX，但StAX 的实现还可以被单独提供。所有这些实现由GlassFish社区完成，并通过不同的发布分发。

XML数据绑定

在Java EE中XML Data Binding是通过JAXB API实现的，Java EE 5 中的版本是JAXB 2.0 (jaxb.dev.java.net) ，类似与JAX-WS，JAXB相比于J2EE 1.4中的规范同样提供了显著的改进。这一改进得到Java社区的认可并且很快将成为一个新的规范。

JAXB 2.0的实现是高质量的快速的且功能完整。JAXB为其它社区的开发人员接纳，并因此被许多组织采用，这其中包括JBoss, Apache Axis, Apache XFire, TmaxSoft, ServiceMix和ActiveSOAP。

Java EE 5之后

Java EE 5规范要求的是JAXB 2.0和JAX-WS 2.0，并且这些规范由GlassFish v1 和 Java SE 6提供了实现。经过在多种情况下的使用，包括与Microsoft的Windows Communication Framework所尝试的Web服务的互操作，导致了JAXB 2.1和 JAX-WS 2.1.x.规范的形成，以改善兼容性，而这些规范的实现是GlassFish v2所要完成的。

GlassFish 集成技术

TopLink Essentials 和 JPA

Java EE 5 规范中一个巨大的功能提升是 Java Persistence API (JPA) 和新的 Enterprise JavaBeans 规范－EJB 3.0。这些开发接口源自早期的 EJB 2.1 ，但利用了 Java 语言的新特性比如 annotation，并吸纳了来自社区的经验比如 Hibernate, Oracle TopLink 和 JDO。

两套 APIs 能够被单独使用。JPA 为实体的持久化提供了简单的编程模型，它可脱离 Java EE 5 提供，用在 J2EE 1.4 甚至 Java SE 中。它还包括了一套 API 来插入开发商的不同的持久化方案。EJB 规范因为受益于使用 JPA 而变得简单。JPA 和 EJB 3.0 都已被很好地采纳和接受。

TopLink Essentials (TLE) 是 GlassFish 社区所采用的 JPA 实现。它是 Oracle 公司给与社区的巨大贡献，TLE 基于 Oracle 在 Top Link 产品上的长期经验，其现已成为 Oracle 和 Sun 的联合开发的项目。TLE 的初始实现比如 GlassFish v1 所包含的，只是为了和规范兼容，但 TLE 所试图成为的是一个完整的实现，Oracle 的计划是将持续地增强 TopLink 的特性并将其贡献给更广泛的社区。TopLink Essentials 被用在许多 Oracle 的商业产品上，包括其应用服务器，此外还包括 Sun 公司的 GlassFish 和 NetBeans，Apache Tomcat, TmaxSoft's JEUS, Spring 2.0 和 EasyBeans。

JPA 的可插拔性意味着可以非常容易地替换不同的 JPA 实现并放置到 GlassFish 应用服务器上。比如，替换成 Apache 的 OpenJPA 和 Hibernate 的 JPA 方案，而且这样的方案非常受欢迎。

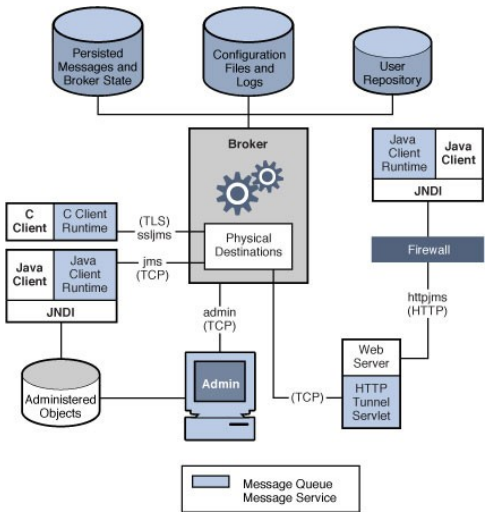
CORBA

GlassFish 带有全面地 CORBA 实现。这种实现已经被持续地改进了多年，例如 GlassFish v2 现在可以动态生成 stubs 和 skeletons，并应用 NIO 架构。GlassFish v2 还将包括许多关键性能上的提高。

OpenMQ Messaging

GlassFish 包括了产品品质的 Message Queue 实现 (mq.dev.java.net) ，它提供以下特性：

- Message buffering between enterprise system components
- 可扩展的 message servers (broker clustering)分布性
- 集成的 SOAP / HTTP 消息
- Java 和 C 的客户端 API
- 可扩展的 JCA 1.5 兼容资源适配器
- 企业 Administration 特性
- Extensive JMX 支持



GlassFish 还包括了 JMS 通用资源适配器(Generic Adaptor for JMS) ，它能够被用在许多不同的 Messaging 的解决方案中，包括： Tibco, WebSphere MQ 6, Active MQ 和 JBoss Messaging。

Java Business Integration

GlassFish v2 还包括对 Java Business Integration (JBI) 接口的支持。JCP 规范(JSR-208)定义了面向服务的集成总线和组件架构的核心。JBI 标准

化了通用消息路由(routing)架构，服务引擎和绑定的插入接口，以及组合多个服务到单个可执行和审计的工作单元的机制。

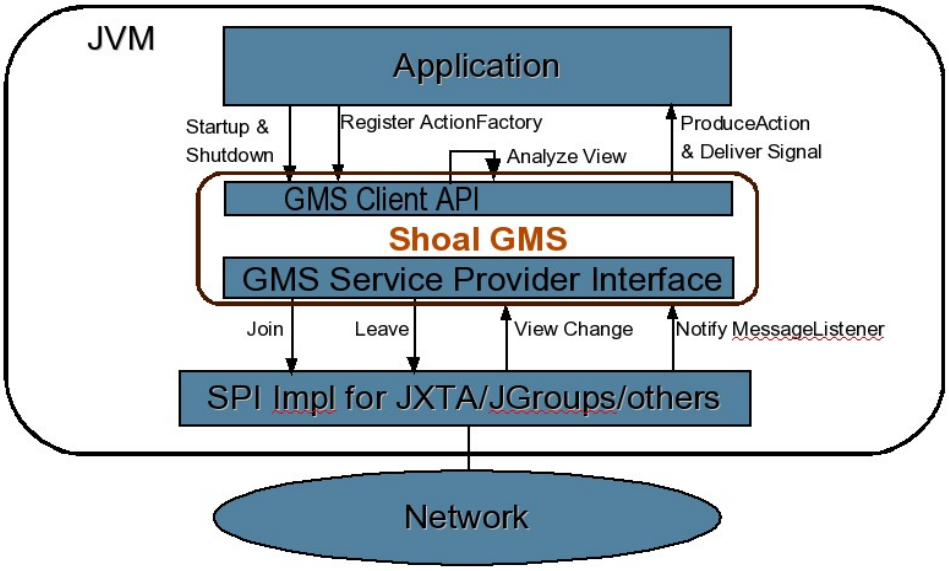
GlassFish v2 所使用的 JBI 实现来自项目 OpenESB ，这种实现也被用来在 Sun 公司其它的企业级别的产品中。 OpenESB 的实现通过 Java EE 服务引擎(JavaEE Service Engine)被集成进 GlassFish，相同接口也可用于集成其它的实现，比如 ServiceMix。Enterprise Pack of NetBeans 为此提供了一流的开发工具，包括 Service Assembly 编辑器，图形化的 WSDL 和 XSLT 编辑器，和智能消息处理器(an Intelligent Event Processor)。

Clustering 和状态复制(State Replication)

GlassFish v2 支持集群(clustering)，并支持动态集群(dynamic clustering)和内存复制(memory replication)。动态集群的特性由项目 Shoal (shoal.dev.java.net)提供，该项目利用了 JXTA 技术的通用框架，使得一个应用程序可以动态地成为一个预定义的集群的成员，并同样订阅到集群的事件，比如：

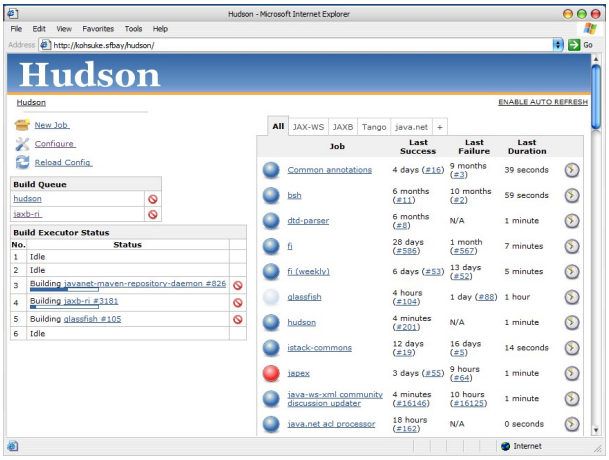
- 成员加入(Member join)，计划关闭(planned shutdown)，失败发生(failures)
- 选择恢复成员
- 自动委派恢复(delegated recovery initiation)

GlassFish v2 还支持 In-Memory Replication，这是一个为 HTTP 会话持久化和 Stateful Session Bean 持久化提供的轻量级的低成本的方案。此方案对于大多数程序已经足够，但对于那些需要达到 99.999% 可提供性的应用，仍可能需要使用 High Availability Data Base (HADB) 的持久化方案。



工具

GlassFish 和 NetBeans 社区合作来开发和实现 Java EE 5 的规范，NetBeans 是第一个支持 GlassFish 的集成开发环境(IDE)。



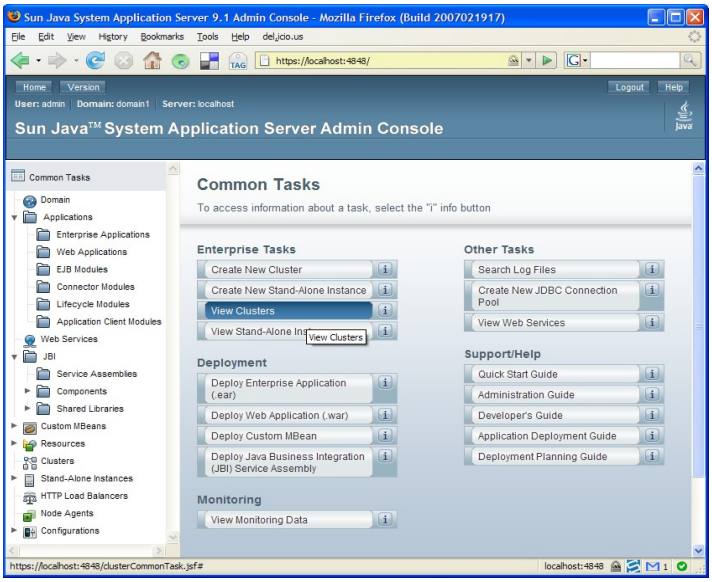
之后，IDEA、JBuilder 和 MyEclipse 也增加了对 GlassFish 的支持。

从另一个角度看，GlassFish 社区自身也包括了许多工具，其中一些在其所属领域已经相当流行。

值得一提的是 Hudson，一个可持续进行集成的工具，被用来完成构建、执行测试和其它任务。Hudson 的构建非常简单，支持多个从属(slave)机器并拥有可插拔的架构来进行扩展。

管理

管理是 GlassFish v2 的强项之一。GlassFish 包括一个图形化管理界面，它功能全面且由任务驱动，另外还包括一个命令行界面，它支持 TCL（Tool Command Language）解释器，并应用“do what I mean”机制来修正拼写错误。GlassFish 还支持基于 JMX 的管理，并且支持对集群和负载均衡(Load Balancing)的管理。



安装体验

所前所述，GlassFish v2 有一个单一的安装程序，它可用来安装下面 3 种用户 profile 中的任一种： developer, enterprise 或 cluster。使用网站

http://glassfish.java.net 所得到的二进制文件进行安装过程，分为两步：安装和配置。注意，可以从 *developer 的* profile 升级到 *cluster* 的 profile。

GlassFish v2 安装包(大约 55MB)通过借助 Pack200 技术大大压缩了大小(这一技术同样被用在 Java SE 6 的安装中)。

Sun Java System Application Server 的安装程序将略为高级和图形化。

Ubuntu 上，安装可以通过 *apt-get* 提供。

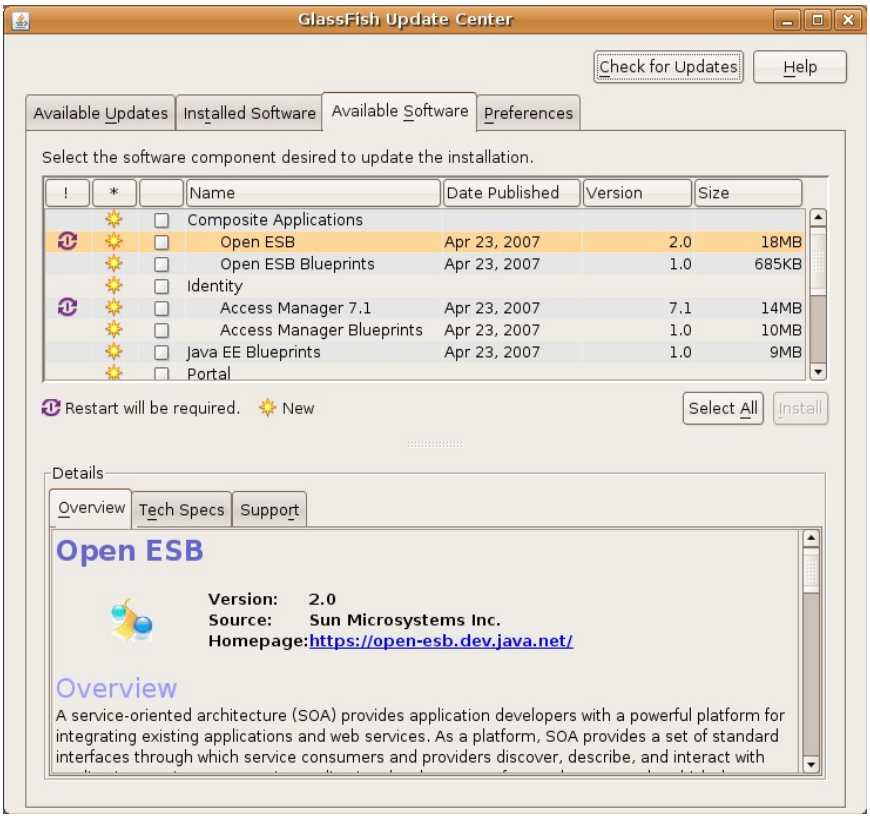
Sun Java System Application Server 9.1 同样被 Java Enterprise System 5 Update 1 提供和支持。

升级中心

升级中心负责发布 GlassFish 附加的组件和更新并提供一个机制来给社区发布贡献。现在各种附加的组件都可以通过升级中心发布。

GlassFish 升级中心是个开源项目 http://updatecenter.dev.java.net。GlassFish V2 上升级中心的初始版本注重给 GlassFish 的附加应用提供简单的下载和安装体验。

GlassFish 升级中心的功能被在两层实现。在服务器端，有更新中心模块和相应的 XML 目录文件。在客户端，包含在 GlassFish V2 的安装中。当启动时，更新中心的客户端(位于 **GlassFish 目录/updatecenter**) 会连接到一个预先定义的更新中心服务器的 URL 地址，然后下载目录文件，将其和本地注册的已安装组件比较，然后显示可安装或升级的组件。 最后，根据用户的选择，进行装组件的下载和安装。



这一机制被用来部署新的 GlassFish 特性、框架库、甚至完整的 Web 应用程序。你可以创建你自己的更新中心附加模块，将其打包，测试，并通过简单

地添加更新中心的定义使它们可以通过更新中心客户端得到。

GlassFish 产品

开源项目普遍的一个益处是用户可以根据对技术的理解，并结合自身所需特性和风险状况作出自己的决定，从而加速了产品采用的周期。在 GlassFish v1 甚至在 GlassFish v2 中，我们正在见到这种加速采用的情况。部署案例处于反馈环(feedback loop)最后的阶段，它对于 GlassFish 的成功至关重要，我们正在将这些案例收集到 http://blogs.sun.com/stories。

一个采用的例子是 Peerflix (peerflix.com)，一个媒体共享社区(media-sharing community)，最近重新设计了他们的架构并从 Microsoft 架构转移到了 GlassFish v1。新站点使用 JSF 的 Facelets 和 Apache Shale 技术来实现表现层，并采用 Kodo JDO 在 MySQL 数据库上完成持久化，并运行在 Solaris 10。网站重构至今的体验是非常正面的。另一个很好的例子是高容量的澳大利亚第三大商业站点 wotif.com，选用了 GlassFish，而不是 JBoss。

开发 GlassFish

GlassFish 是通过一种“透明”的方式来进行社区的参与和其它社区的合作。社区和开发流程对开发结果有着关键的影响，较之早期的用传统方法和开发流程所开发出的产品，它们创造出更加匹配客户需求的产品，并通过更快更精确的循环反馈得到了更高的产品品质，以及更快更敏捷的开发周期。

有些影响是超出期望的，比如，对外部的通讯工具的使用(web 网站, 博客, 邮件列表等)使得能够使用一些工具比如外部的搜索引擎、博客阅读器和邮件列表归档，对用户在 Web 上行为加以利用。其结果是在由各类不同团队、个人、组织构成的 GlassFish 社区上的信息流动，较之同一个公司的同类组织之间的流动更加顺畅，即使这个社区在地理上分布是分散的跨时区的。

未来方向

GlassFish 正在开发之中。我们期望社区和产品能够继续进步。社区正在规范它的管理并将继续加以完善。我们期望社区的构造在多方面得以继续增强，比如，JavaOne 2007 之前的 **GlassFish 技术日**。我们将启动一些 GlassFish 用户组。请注意，如果 Sun 技术日(http://developers.sun.com/events/techdays/)在您附近的城市举行，很可能那也会有一个 GlassFish 技术日。

社区已经提交了高品质的 GlassFish v2 特性，短期目标是增加对其的采用和部署。我们还希望看到更多的应用和架构在 GlassFish 上工作，并且增强与其它组织和社区的关系。

中期目标是 v3，一个高度模块化的容器，代号 HK2。它将启动非常快速且只需少量的资源，只在需要执行需要管理的程序时才激活相应的服务。这样的容器将适用在许多场合，从非常轻量级的使用到高可靠性的应用。GlassFish v3 的原型提供在 http://hk2.dev.java.net。

Jersey - RESTful Web Services

JAX-RS 是项由 JCP 开发 (JSR 311)的新技术，用来为 RESTful Web 服务定义基于 POJO 编程模型。资源被简单地标注成 POJOs，而 HTTP verbs 被映射成 java 方法(参看：JAX-RS annotations 和 POJOs)。其 Consumer 和 producer 标注可以进一步简化 MIME 类型的操作。

相关的规范制定工作位于 http://jsr311.dev.java.net/ (其早期草稿已经提供)。JAX-RS 列举了即将到来的 Java EE 6 规范(JSR 316)的潜在组件。

JAX-RS annotations 和 POJOs

下面是典型的 JAX-RS 资源使用 POJO：

```
@UriTemplate("/helloworld")
public class HelloWorldResource {
 @HttpMethod("GET")
 @ProduceMime("text/plain")
 public String getMessage() {
   return "Hello World";
 }
}
```

资源一旦部署，将可通过 URI 路径"/helloworld" 访问到。

Jersey (http://jersey.dev.java.net)是 JSR 311 的参考实现，并紧密跟随规范的演变。Jersey 运行时可以被部署到各类环境，包括：GlassFish servlet container, Metro JAX-WS endpoint, Grizzly, Java 6 (使用轻量

级 Web 服务器), Tomcat 等等...

Jersey 是一组库，并且可以通过 GlassFish 的更新中心获取其在 GlassFish 上的安装。NetBeans 6.0 支持从 POJO 创建资源以及为测试 JAX-RS 资源而创建基于 AJAX 的客户端。

Sailfin - GlassFish 的 SIP Servlets 技术

项目 SailFin (http://sailfin.dev.java.net) 是基于由 Ericsson 贡献的稳定的可扩展的 SIP Servlets 技术 和 GlassFish v2 应用服务器。

SIP (Session Initiation Protocol) 和 SIP Servlets 在后端支持着许多流行的服务比如 Voice-over-IP (VoIP) 电话服务、即时通讯、参与者和好友列表管理和 web 会议。SIP Servlets 被期望在下一代电信服务中扮演更加重要的角色。

结合 Java EE 平台，SIP Servlets 可被用来添加富媒体(rich media)和企业应用之间的交互。 SIP Servlet 1.1 (JSR 289) 更新了 SIP Servlets API 并且定义了一个标准的应用编程模型来组合 SIP Servlets 和 Java EE 组件。通过利用 Java EE 服务比如 Web 服务、持久化、安全和交易，将可以更快地开发出更智能的通讯应用。 Sailfin 项目的目标是在其第一个版本中支持 JSR 289。

Sailfin 的每日构建和里程碑构建都可以在 SailFin 的网站得到 (http://sailfin.dev.java.net)，它是自包含的(self-contained)，其安装和 GlassFish 的安装类似。其网站还提供包含文档说明的例程。SailFin 的第一个版本计划于 2008 中期发布。

现在开始

跟踪 GlassFish 社区活动的好去处是水族馆 — TheAquarium (blogs.sun.com/theaquarium)。这是一个关于社区日常重要活动的热点报道。另一个相关的 blog 是 Stories (blogs.sun.com/stories)，这里收集了关于采用 GlassFish 的各类案例。

GlassFish 社区的主页是 java.net (glassfish.java.net)；文档、邮件列表、论坛和 wiki 都放置在这里。所有关于 GlassFish 的构建也在这里提供。

GlassFish 以不同的方式提供其构建：Nightly 和 Weekly builds 是从源代码构建常规的构建；Milestone builds 经历了稳定的开发周期并且通常含有新的特性 - 特别的细节由相关的 Highlight 文本描述；final builds 是最稳定的也是我们所推荐用于部署的版本，Sun 为 final builds 提供支持。

保持联系

我们希望这篇文章能够勾勒出 GlassFish 社区和其应用服务器的优势。我们希望你能尝试并发掘它的用途。。。

我们会一直珍视您所提供的任何反馈。希望在 http://glassfish.java.net/ 和 http://blogs.sun.com/theaquarium 上见到你。

关于作者

Eduardo Pelegri-Llopart 是 Sun 公司的 Distinguished Engineer 并且是 GlassFish 社区的总负责人。

Eduardo 为多个开源项目工作并且于 1995 年就参与了 Java 社区。他居住在 San Francisco Bay Area。

Yutaka Yoshida 是 GlassFish 项目的工程师。

Alexis Moussine-Pouchkine 是一位 GlassFish 技术传道士，居住在法国，巴黎。

索引