

企业级软件的组件化和动态化开发实践

池建强--@sagacity

用友集团-瑞友科技-IT应用研究院

www.ufida.com

www.rayootech.com

RayooTech

大纲

- ▶ 软件开发之殇
- ▶ 组件化与动态化
- ▶ 基于WTP Facet、XDoclet、ANT的组件构建
- ▶ OSGi——兄弟，全靠你了
- ▶ 展望：所有的Java应用都是Bundle

现代人类文明运行于软件之上，
而软件的构建过程却隐于黑暗里，多
少青春的日子耗费在了软件的缺陷上

从软件系统诞生之初，程序员们就开始梦想有一天能够像建造桥梁和房屋那样“透明”的构造软件，实现“即插即用”的软件系统



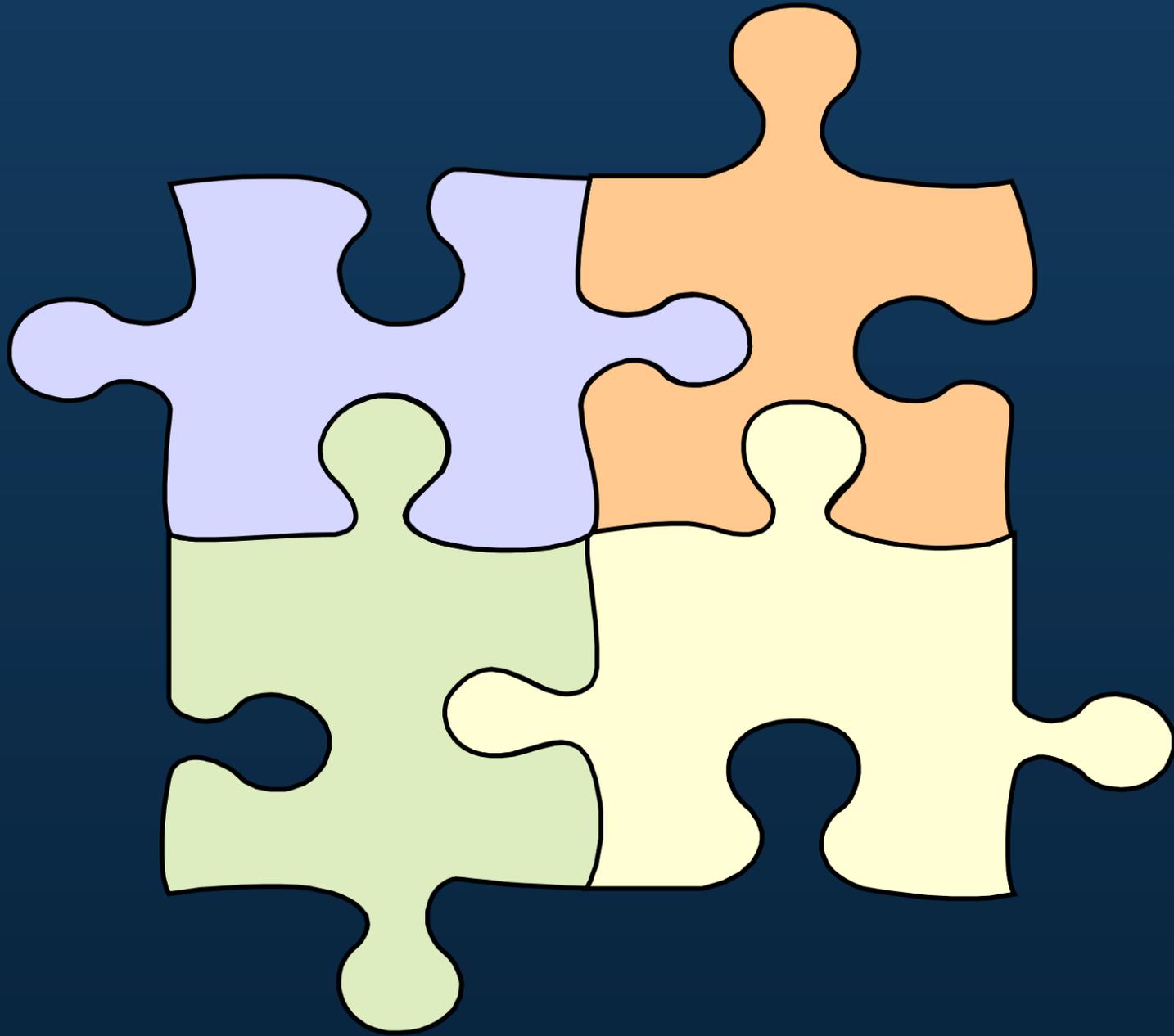
但是到目前为
止，软件的开发依
然让我们倍感挫
折，失败的软件项
目有增无减，我们
还在路上



但是到目前为
止，软件的开发依
然让我们倍感挫
折，失败的软件项
目有增无减，我们
还在路上

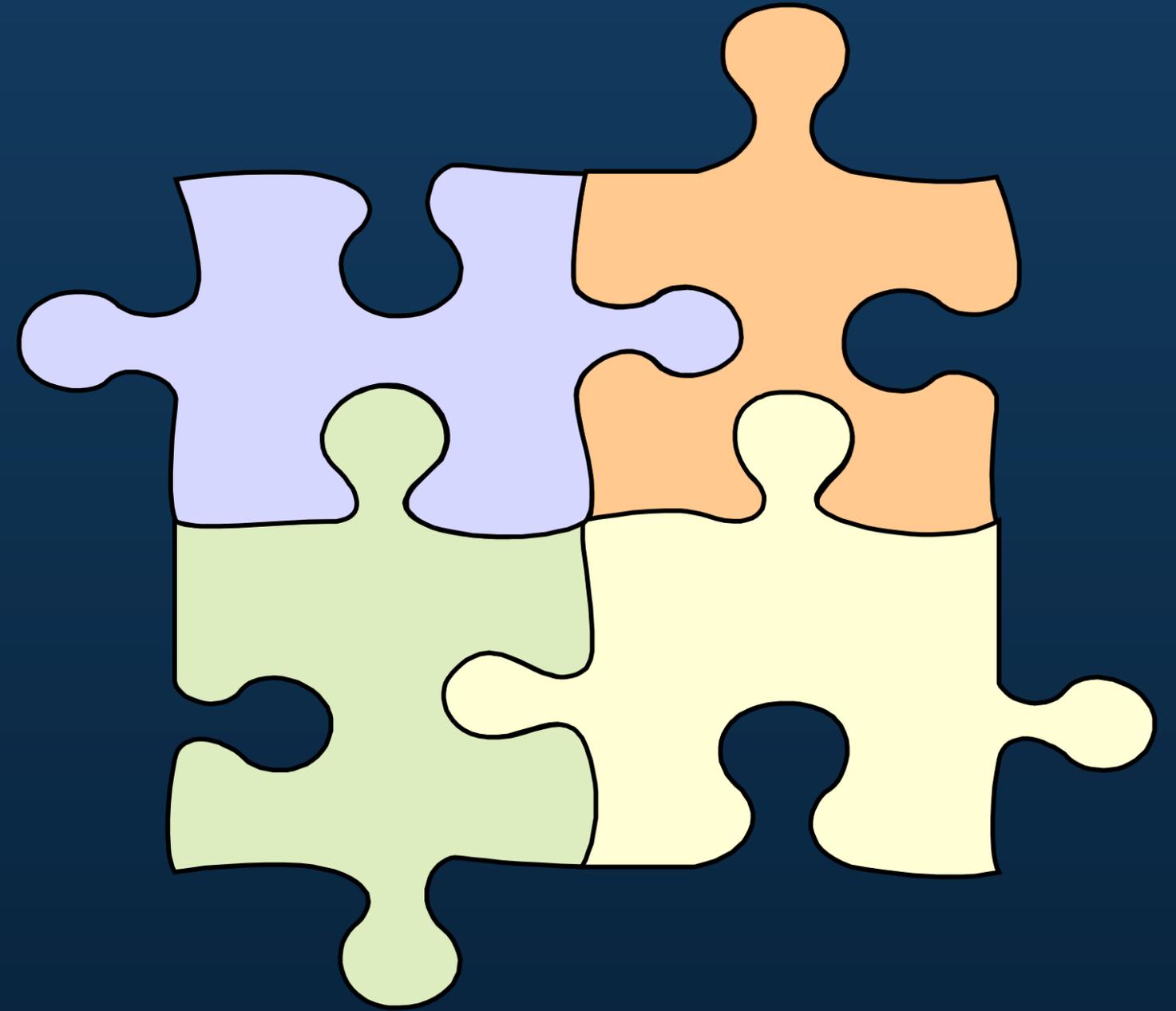


组件化的技术让我们更加接近真理之门



组件化的技术让我们更加接近真理之门

OSGi?



组件化与动态化

▶ 组件化

- ▶ 完整的业务逻辑单元
- ▶ 解决某一方面的问题
- ▶ 每个组件可以独立开发
- ▶ 实现组织级复用
- ▶ 按需分配
- ▶ 动态的安装启动停止卸载

▶ 动态化

- ▶ 动态的改变组件的状态
- ▶ 动态的改变数据和数据结构
- ▶ 动态的修改业务流程
- ▶ 动态的调整业务规则
- ▶ 动态的分配资源
- ▶ 动态的计算

组件化与动态化

Metadata
Workflow
RuleEngine

组件化

- ▶ 完整的业务逻辑单元
- ▶ 解决某一方面的问题
- ▶ 每个组件可以独立开发
- ▶ 实现组织级复用
- ▶ 按需分配
- ▶ 动态的安装启动停止卸载

动态化

- ▶ 动态的改变组件的状态
- ▶ 动态的改变数据和数据结构
- ▶ 动态的修改业务流程
- ▶ 动态的调整业务规则
- ▶ 动态的分配资源
- ▶ 动态的计算

Component Building

企业级软件开发遇到的挑战

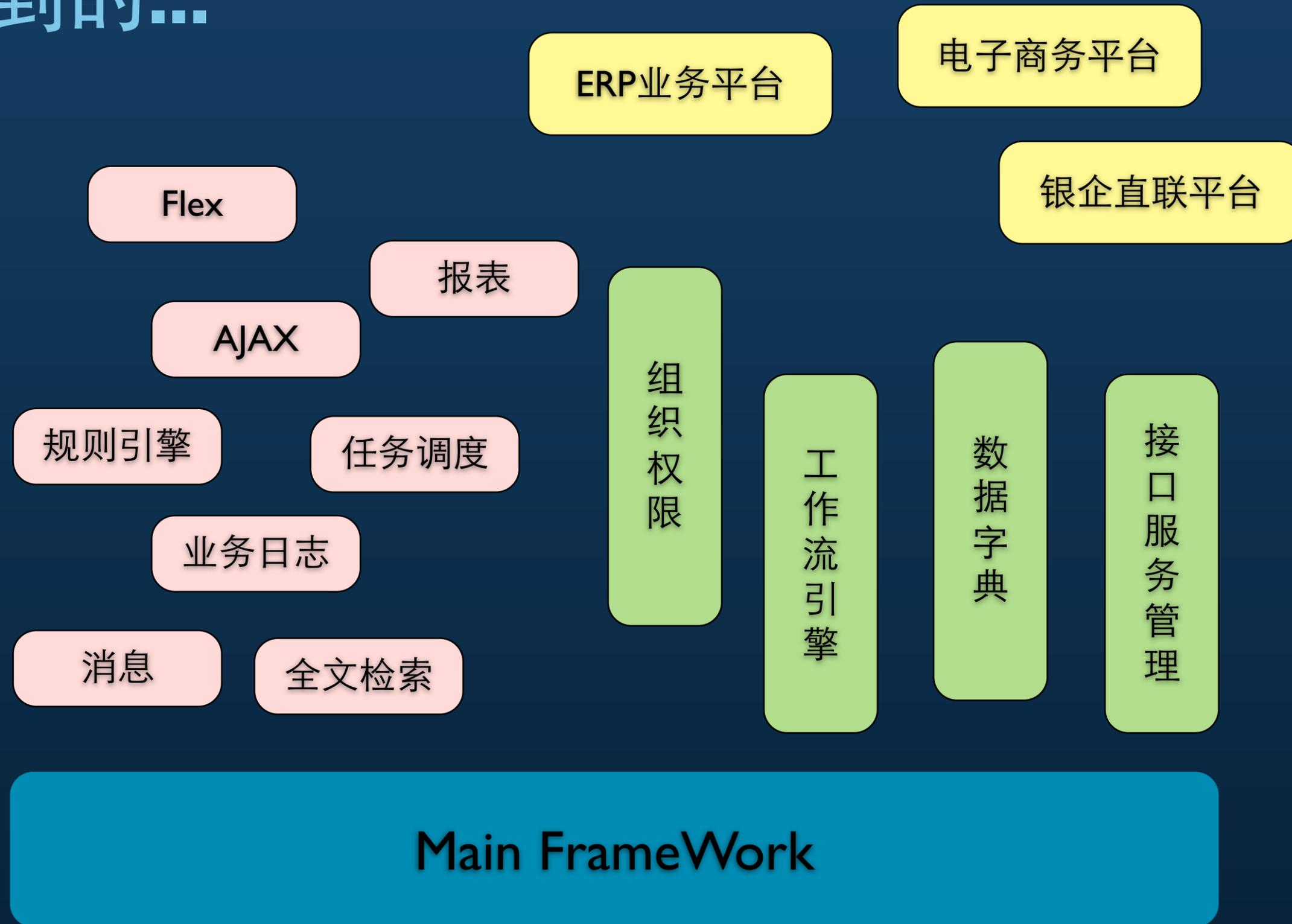
组织级复用

- ▶ 不是复制粘贴，不是简单的代码、接口或算法的复用
- ▶ 没人愿意编写重复代码
- ▶ 复用是永远的痛——200个表的故事

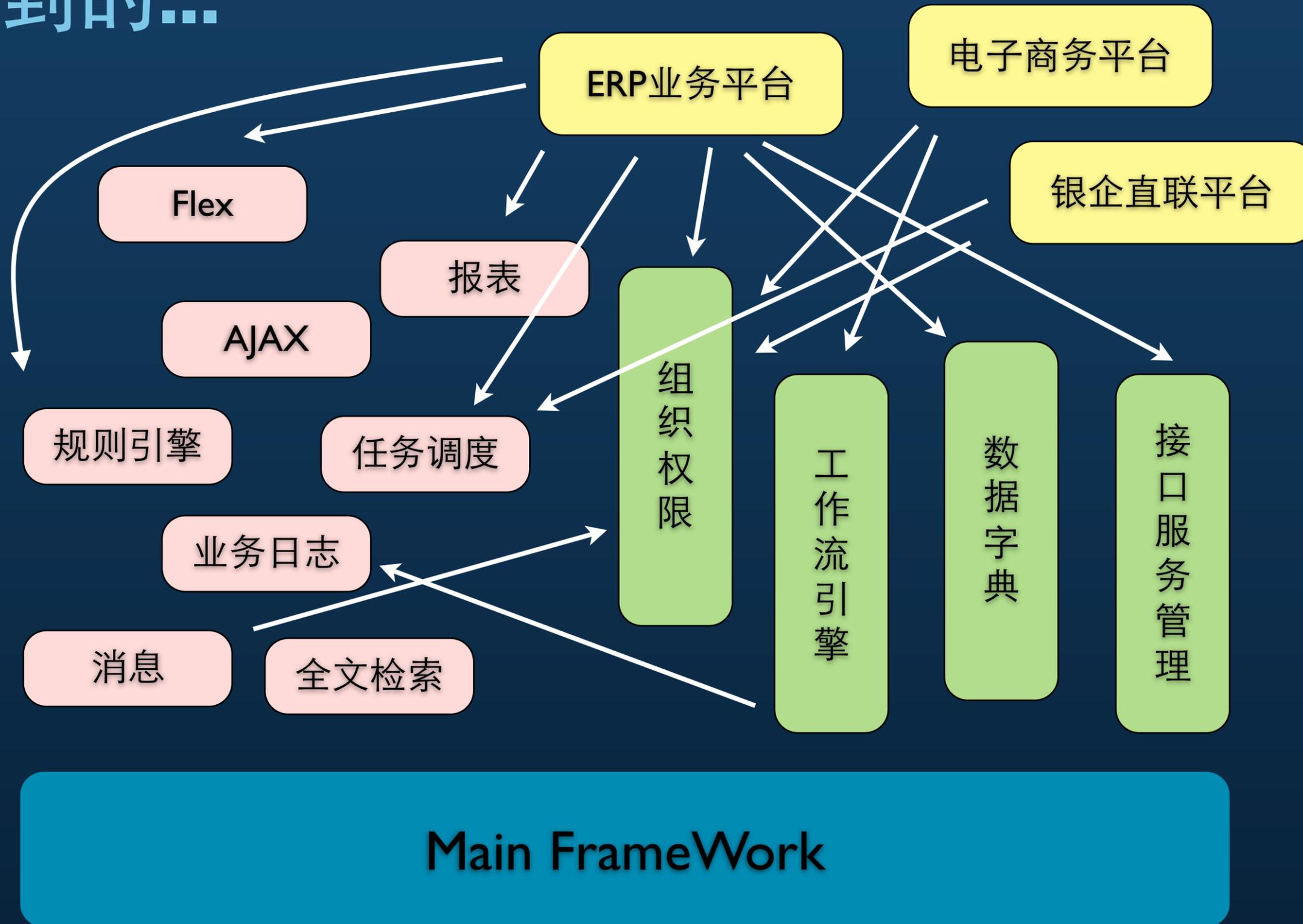
组件的管理

- ▶ 组件的独立开发
- ▶ 组件的依赖关系
- ▶ 自动构建——机器的活得让机器干

我们遇到的...

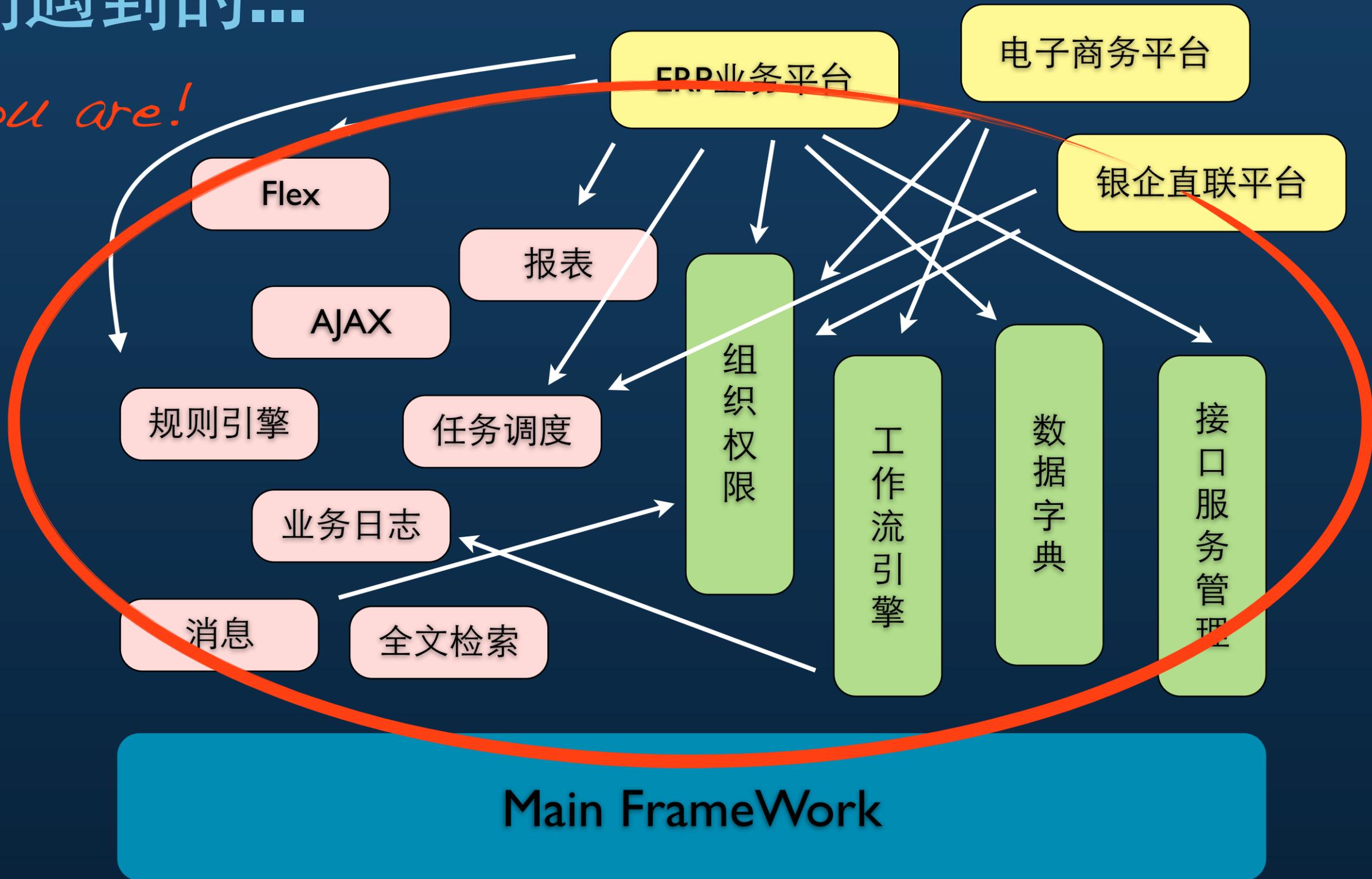


我们遇到的...



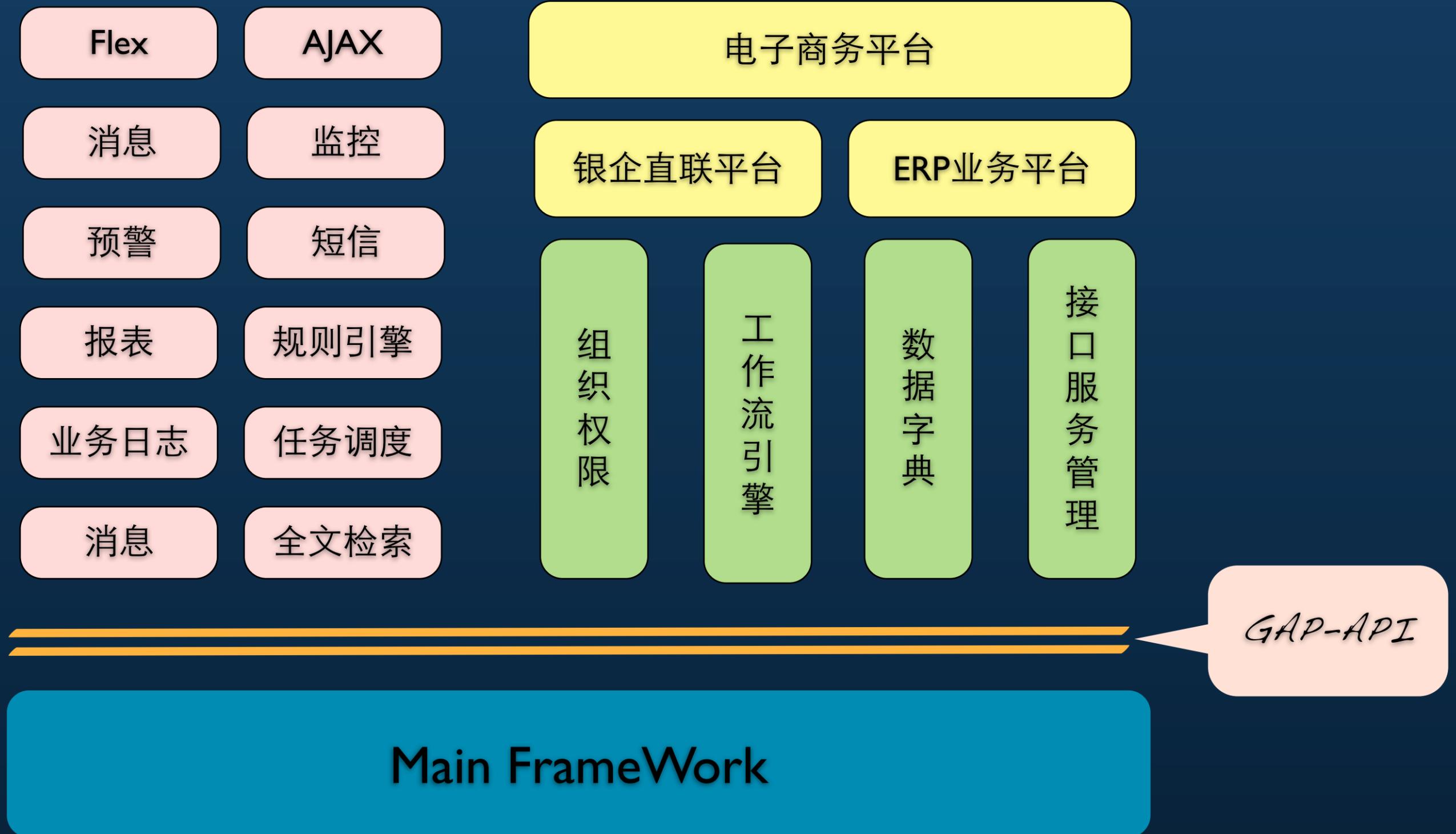
我们遇到的...

Here you are!



Main FrameWork

我们希望的...



开发实践

- ▶ Eclipse WTP Facets
- ▶ XDoclet
- ▶ Ant
- ▶ 为什么当时没有采用OSGi
 - ▶ 资源文件的处理
 - ▶ Servlet容器的支持
 - ▶ 持久化、事务

Eclipse WTP Facets

- ▶ 提供了优秀的扩展机制
- ▶ 对WTP (Web Tools Platform) 平台进行扩展
- ▶ 创建WTP项目时增加自定义的组件
- ▶ 提供组件的依赖关系和多版本管理
- ▶ Facets允许开发人员以组件集成的方式来创建项目，像搭积木一样把各种组件进行组合并完成最终构建，是面向组件开发的一种实现机制
- ▶ Facets不提供运行时组件生命周期的管理

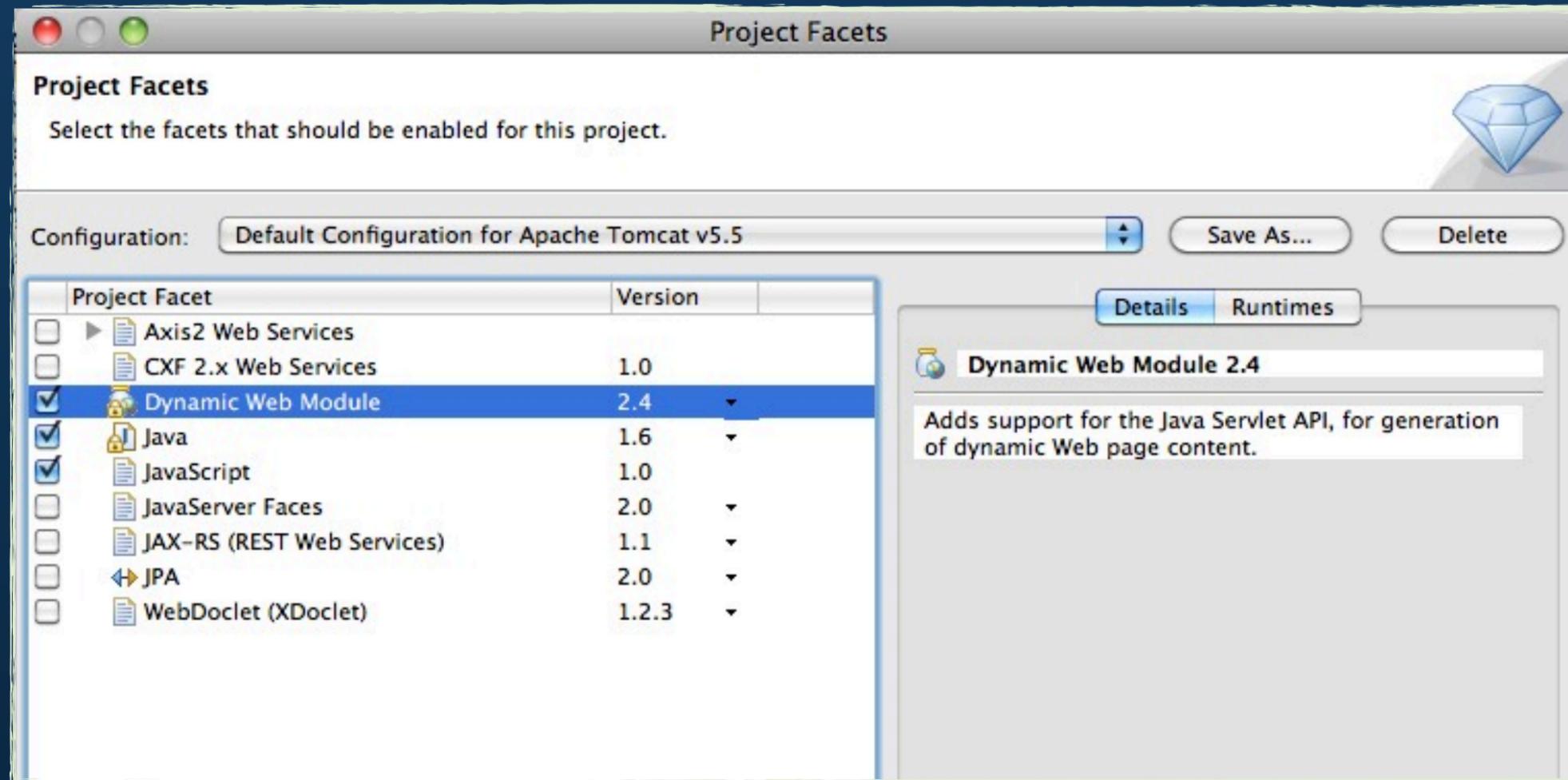
核心的WTP Faceted包括以下内容:

id	versions	requires	conflicts
jst.java	1.3, 1.4, 1.5, 1.6		jst.ear
jst.web	2.2, 2.3, 2.4	jst.java	jst.ear, jst.ejb
jst.ear	1.2, 1.3, 1.4, 5.0, 6.0		jst.java
jst.ejb	1.1, 2.0, 2.1, 3.0, 3.1	jst.java	jst.ear, jst.java

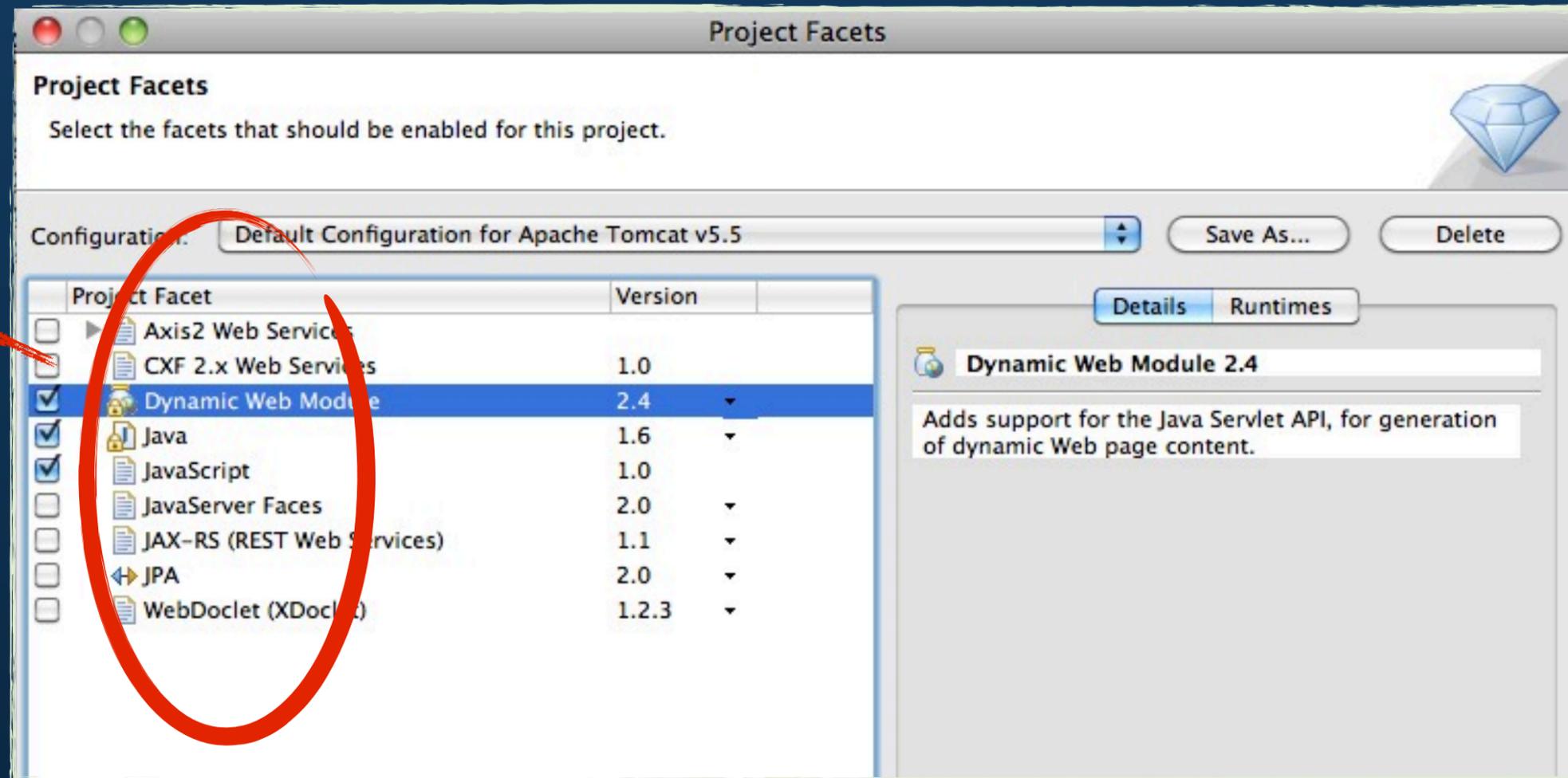
核心的WTP Faceted包括以下内容:

id	versions	requires	conflicts
jst.java	1.3, 1.4, 1.5, 1.6		jst.ear
jst.web	2.2, 2.3, 2.4	jst.java	jst.ear, jst.ejb
jst.ear	1.2, 1.3, 1.4, 5.0, 6.0		jst.java
jst.ejb	1.1, 2.0, 2.1, 3.0, 3.1	jst.java	jst.ear, jst.java

创建一个标准的Dynamic Web Project



创建一个标准的Dynamic Web Project



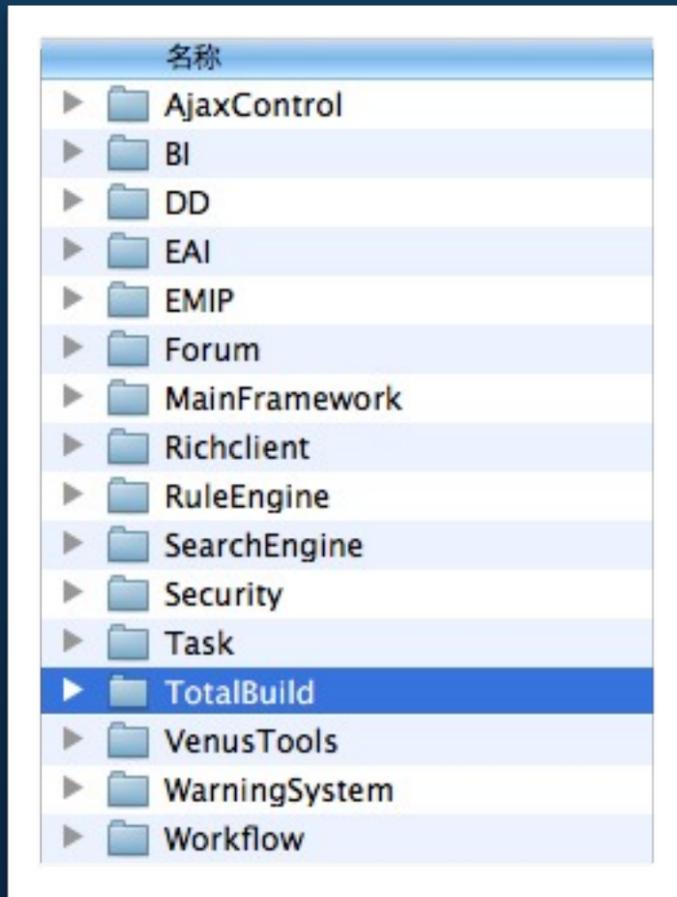
XDoclet & Ant

- ▶ XDoclet: 优秀的文本处理和构建工具
- ▶ 资源文件处理、Merge Point
- ▶ Ant: 历史悠久的构建脚本工具
- ▶ 独立组件构建
- ▶ 整体构建
- ▶ Facets → Ant → XDoclet

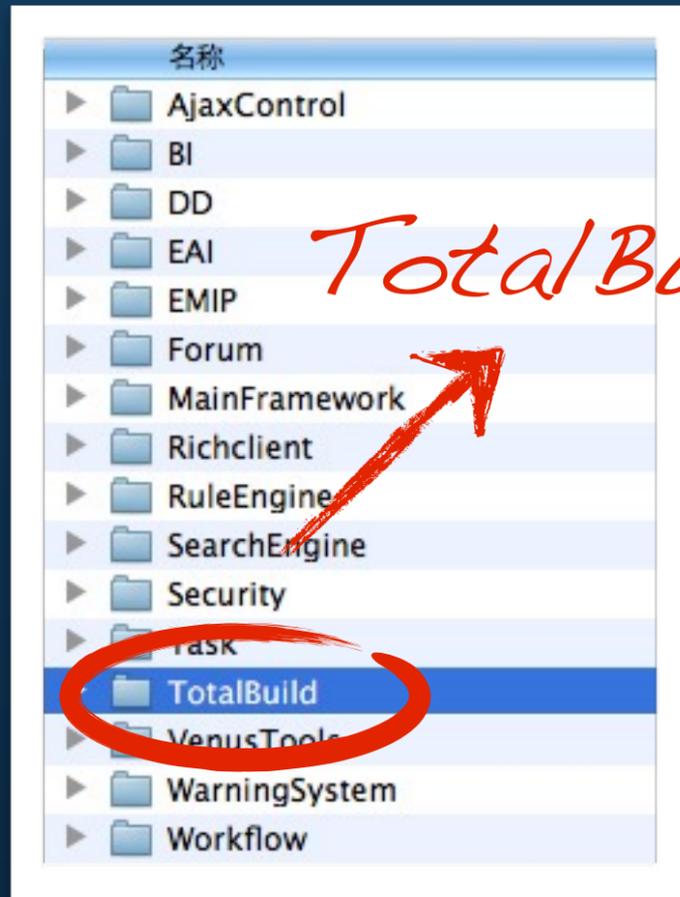
组件化开发实践

- ▶ 各组件独立开发，独立构建 (build.xml)
- ▶ 公共资源文件处理-MergePoint (i18n, oid, spring, include)
- ▶ 整体构建 (TotalBuild)
- ▶ 基于Facet机制装配组件

实际效果

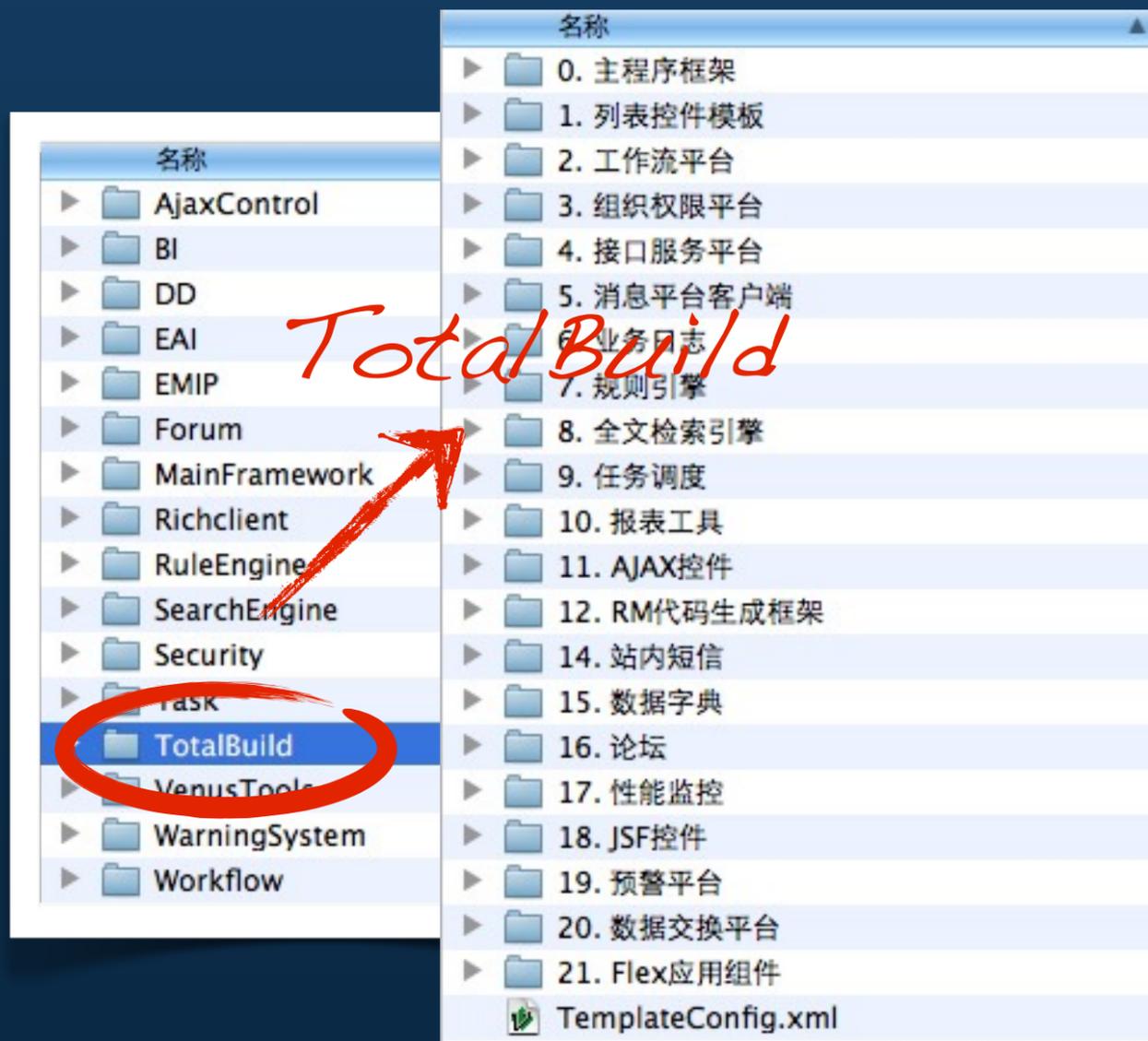


实际效果



TotalBuild

实际效果



实际效果

The image shows a screenshot of an IDE's configuration window for project facets. On the left, a file explorer shows a tree of folders, with 'TotalBuild' circled in red. A red arrow points from this folder to the 'GAP平台' facet in the configuration window, which is also circled in red. Another red arrow points from the 'TotalBuild' folder to the word 'TotalBuild' written in red above it. The configuration window shows a list of facets with their versions. The 'GAP平台' facet is expanded, showing a sub-list of facets including '0. 主程序框架', '1. 列表控件模板', '10. 报表工具', '11. AJAX控件', '12. RM代码生成框架', '14. 站内短信', '15. 数据字典', '16. 论坛', '17. 性能监控', '18. JSF控件', '19. 预警平台', '20. 数据交换平台', '21. Flex应用组件', and '9. GA平台构建'. The word 'Facets' is written in red next to the 'GAP平台' facet.

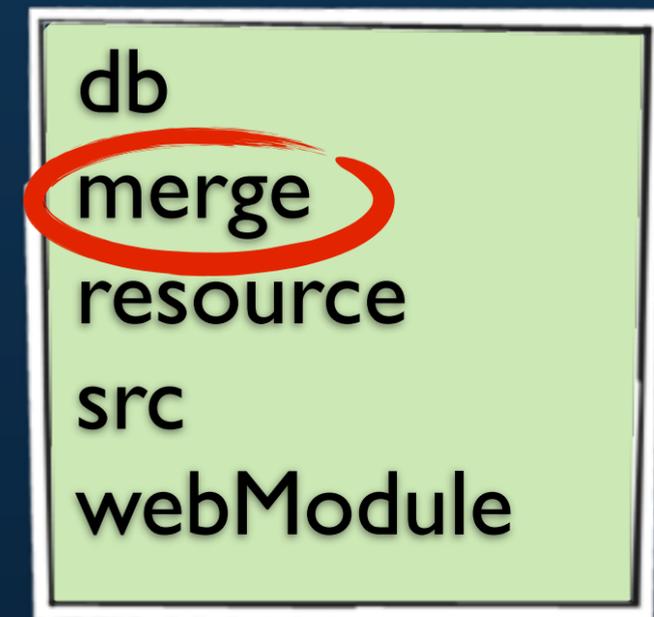
Project Facet	Version
Axis2 Web Services	
Dynamic Web Module	2.4
GAP平台	
0. 主程序框架	3.7
1. 列表控件模板	3.7
10. 报表工具	1.1
11. AJAX控件	1.3
12. RM代码生成框架	1.4
14. 站内短信	1.0
15. 数据字典	2.5
16. 论坛	1.0
17. 性能监控	1.0
18. JSF控件	1.0
19. 预警平台	1.0
2. workflow平台	3.0
20. 数据交换平台	1.0
21. Flex应用组件	1.0
3. 组织权限平台	3.0
4. 接口服务平台	1.1
5. 消息平台客户端	1.0
6. 业务日志	3.0
7. 规则引擎	2.0
8. 全文搜索引擎	1.0
9. 任务调度	1.0
9. GA平台构建	1.0

开始实践之旅

- ▶ 各组件独立研发，依赖框架，依赖接口
- ▶ 编写自己的build.xml，实现clean compile jar distribution

```
<?xml version="1.0"?>
<project name="GAP-WarningSystem" basedir="." default="all">
  <property file="build.properties" />
  <target name="all" depends="clean,start,dist" />
  <target name="start">...</target>
  <target name="clean">...</target>
  <target name="compile">...</target>
  <target name="jar" depends="compile">...</target>
  <target name="dist" depends="start">...</target>
</project>
```

- ▶ 在组件项目的根目录下执行ant
- ▶ 构建出统一目录结构的Distribution



公共资源文件处理 (Merge)

- ▶ 建立公共模板文件，文件后缀设置为xdt

- ▶ applicationContext.xdt-----spring
- ▶ global.xdt-----include global.jsp
- ▶ web.xdt-----web.xml
- ▶

- ▶ 设置Merge Point

```
<XDtMerge:merge file="workflow-hbm.data"></XDtMerge:merge>
```

Merge文件的内容

```
<value>gap/wf/persistent/db/bo/GapwfProcessinstance.hbm.xml</value>  
<value>gap/wf/persistent/db/bo/GapwfActivityinstance.hbm.xml</value>  
<value>gap/wf/persistent/db/bo/GapwfActivityinstancepre.hbm.xml</value>  
<value>gap/wf/persistent/db/bo/GapwfRevelantdata.hbm.xml</value>  
<value>gap/wf/persistent/db/bo/GapwfWorkitem.hbm.xml</value>  
<value>gap/wf/persistent/db/bo/GapwfWorkitemdata.hbm.xml</value>  
<value>gap/wf/persistent/db/bo/GapwfDynprocessrelation.hbm.xml</value>
```



```
db  
merge  
resource  
src  
webModule
```

XDoclet构建

- ▶ 设置XDoclet的classpath
- ▶ 设置Ant的taskdef
- ▶ 定义target

```
<path id="xdoclet.classpath">  
  <fileset dir="${lib.dir}">  
    <include name="*.jar"/>  
  </fileset>  
  <pathelement location="classes"/>  
</path>
```

```
<taskdef name="xdoclet" classname="xdoclet.DocletTask"  
  classpathref="xdoclet.classpath" />
```

```
<target name="gen-applicationContext.xml">  
  <xdoclet destdir="${webmodule.dir}/WEB-INF/conf/applicationContext" force='true' mergedir="${merge.dir}">  
    <fileset dir="${source.dir}" includes="**/*.java" />  
    <template templateFile="template/applicationContext.xdt" destinationfile="applicationContext.xml" />  
  </xdoclet>  
</target>
```

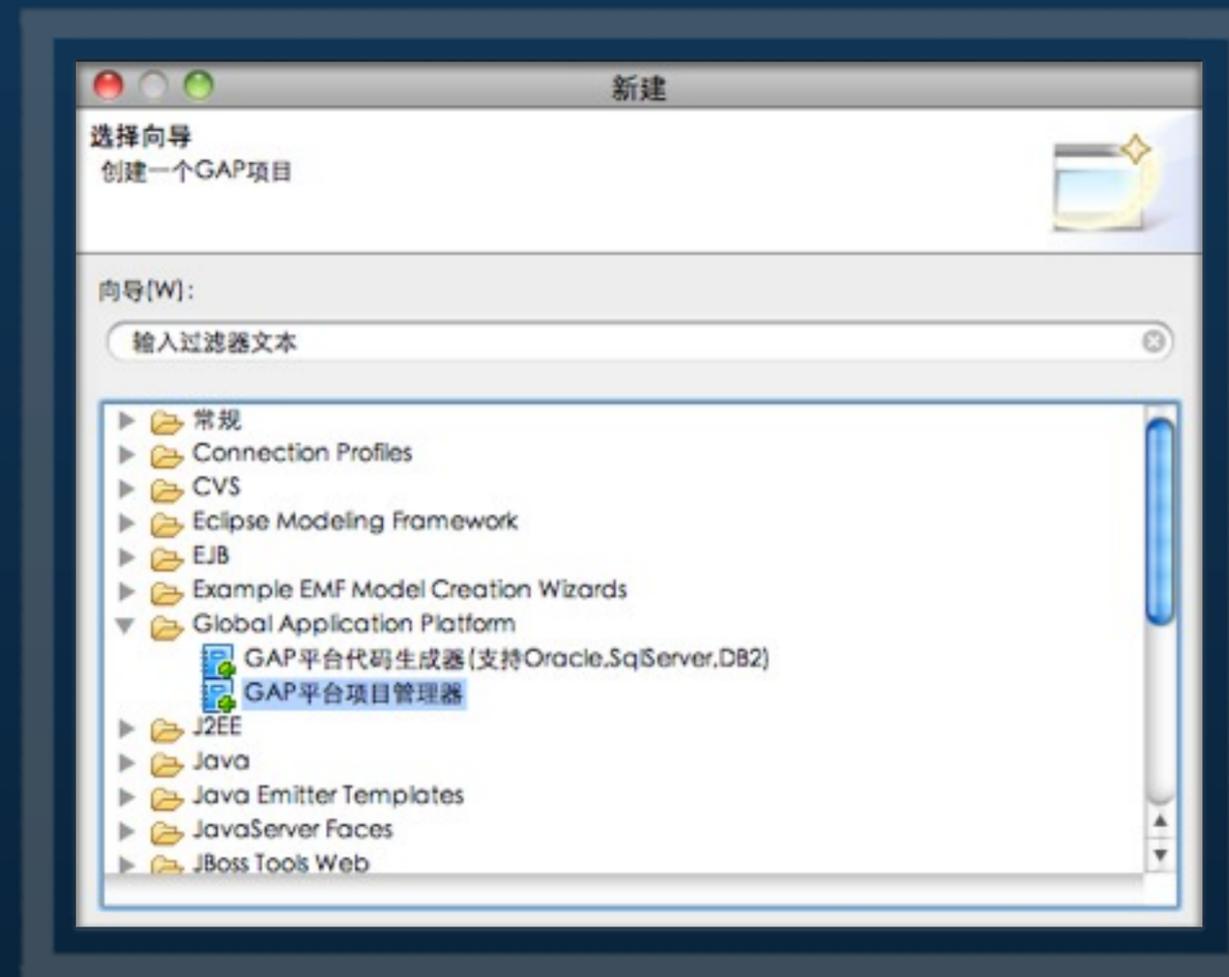
TotalBuild

构建Facets项目所需要的组件内容

```
<!--调用各个组件的build.xml-->
<target name="build">
  <ant dir="{gap-api}" antfile="build.xml" target="all" />
  <ant dir="{venus-frame}" antfile="build.xml" target="all" />
  <ant dir="{venus-page}" antfile="build.xml" target="all" />
  <ant dir="{venus-log}" antfile="build.xml" target="all" />
  .....
</target>
<!--重命名-->
<target name="plugin">
  <available file="Templates" type="dir" property="Templates.present" />
  <antcall target="clean" />
  <move todir="Templates/0. 主程序框架">
    <fileset dir="{gap-api}/dist" />
    <fileset dir="{venus-frame}/dist" />
  </move>
  .....
</target>
```

构建我们自己的Facet

- ▶ 创建基于org.eclipse.ui.newWizards的扩展点
- ▶ 设置category和wizard，实现独立的项目向导



核心扩展点

- ▶ `org.eclipse.wst.common.project.facet.core.facets`
 - ▶ `<category />` —— Facet的类别
 - ▶ `<project-facet />` —— Facet的属性描述
 - ▶ `<project-facet-version />` —— Facet的版本处理
 - ▶ `<template />` —— 设置必选的Facet
 - ▶ `<preset />` —— 预置Facet集合

辅助扩展点 —— 增加图标和向导页面

- ▶ `org.eclipse.wst.common.project.facet.ui.images`
- ▶ `org.eclipse.wst.common.project.facet.ui.wizard`

Facet的核心配置

```
<!--Facet的基础信息配置-->
<project-facet id="gap.mainframe">
  <label>0. 主程序框架</label>
  <description>GAP平台的基础框架</description>
  <category>gap.category</category>
</project-facet>

<!--设置Facet的某个版本的相关信息（安装代理类，约束等）-->
<project-facet-version facet="gap.mainframe" version="3.7">
  <action id="gap.mainframe.install" type="INSTALL">
    <delegate class="venus.tools.ide.facet.GAPFacetInstallDelegate"/>
  </action>
  <constraint>
    <requires facet="jst.web" version="[2.4"/>
  </constraint>
</project-facet-version>
```

Facet的两个代理类

- ▶ 实现接口：`org.eclipse.wst.common.project.facet.core.IDelegate`
- ▶ `GAPFacetInstallDelegate`：负责负责所有实体Facet的安装
- ▶ `BuildFacetsDelegate`：
 - ▶ 虚拟Facet，在所有实体Facet安装完成后调用
 - ▶ 负责Facet的装配策略
 - ▶ 创建配置文件、驱动XDoclet进行文件合并、设置classpath
 - ▶ 创建project的nature（.tomcatplugin）
 - ▶ 删除不需要的文件

解决了哪些问题

- ▶ 组件的独立开发
- ▶ 组件的组合与依赖关系
- ▶ 组件版本管理
- ▶ 按需分配
- ▶ 最终实现了组织级别的复用

遗留问题

- ▶ 能装不能拆，无法实现物理隔离
- ▶ 独立jar包的版本冲突
- ▶ 运行时动态的生命周期管理 (Install、Update、Remove)
- ▶ **Time For:**

遗留问题

- ▶ 能装不能拆，无法实现物理隔离
- ▶ 独立jar包的版本冲突
- ▶ 运行时动态的生命周期管理 (Install、Update、Remove)
- ▶ **Time For:**



OSGi

- ▶ 天将降大任
- ▶ 2005，R4.0发布——最好的年代，也是最坏的年代
- ▶ 2010年3月，OSGi企业级规范4.2正式发布
- ▶ 世界是你的，也是我的，归根结底应该是由Bundle构建的
- ▶ 完美的解决方案

OSGi 企业级特性

- ▶ 声明式服务 (Declarative Services)
- ▶ 框架加载 (Framework Launching)
- ▶ Blueprint服务 (Blueprint Service)
- ▶ Web应用 (Web Application Bundle)
- ▶ 远程服务 日志服务 JPA JTA JDBC.....

OSGi 企业级特性

- ▶ 声明式服务 (Declarative Services)
- ▶ 框架加载 (Framework Launching)
- ▶ Blueprint服务 (Blueprint Service)
- ▶ Web应用 (Web Application Bundle)
- ▶ 远程服务 日志服务 JPA JTA JDBC.....

声明式服务

- ▶ Service Component Model
- ▶ 通过声明的方式实现OSGi服务的发布、查找和绑定
- ▶ 通过xml的方式定义Component和Service
- ▶ 引用服务和提供服务都以接口的方式定义
- ▶ 允许以纯Java POJO的方式开发组件，不依赖OSGi框架
- ▶ 更容易测试和模拟

声明式服务——提供服务 and 消费服务

- 在Bundle的Manifest中定义一个或多个组件文件

```
Bundle-ManifestVersion: 2
...
Service-Component: OSGI-INF/component.xml [, ...]*
```

- 在OSGI-INF/目录下编写组件文件

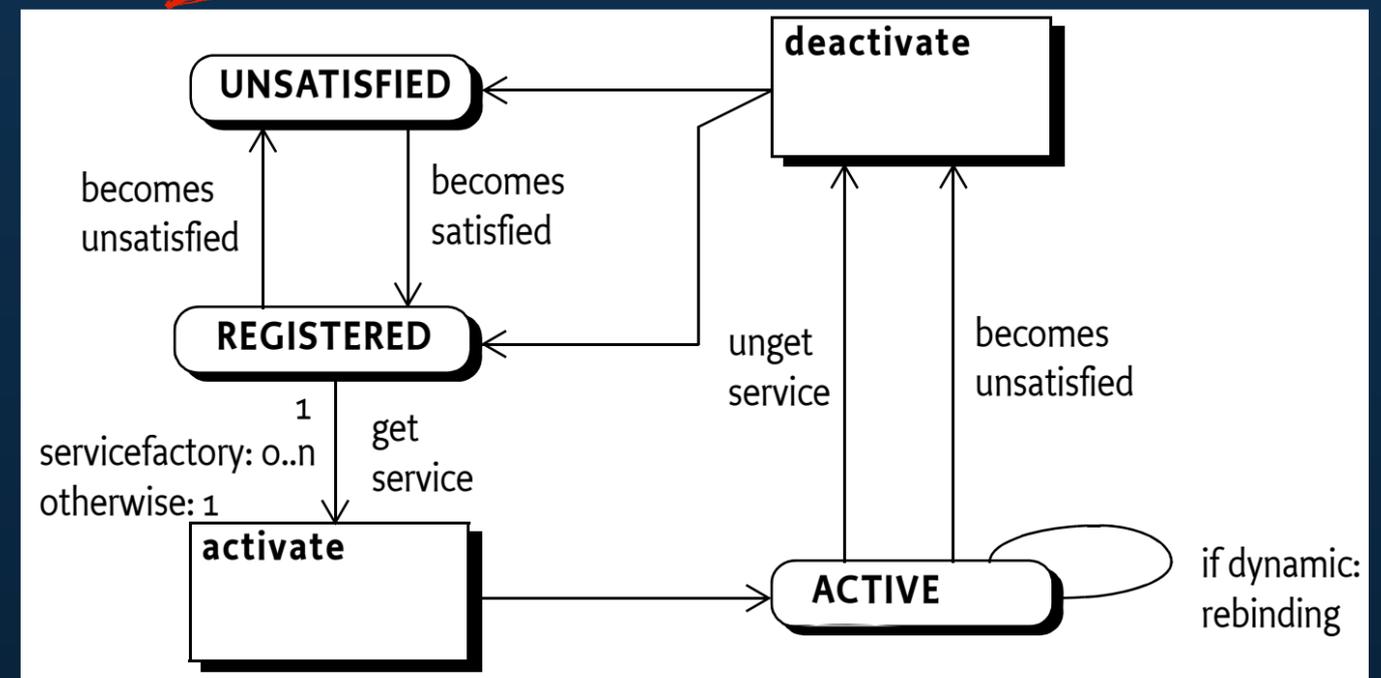
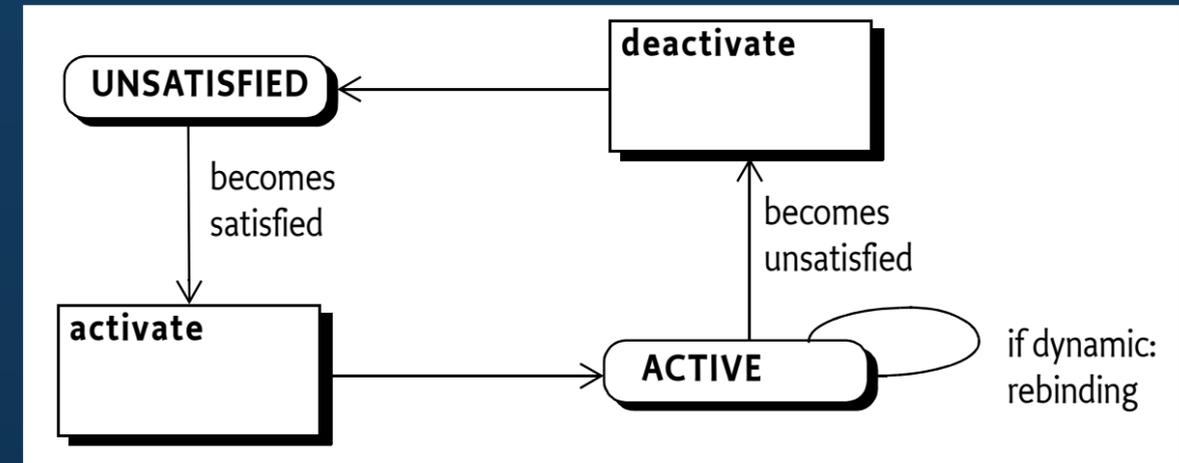
```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component name="bookstore-order" xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0">
  <!--该组件的实现类-->
  <implementation class="com.osgi.ds.OrderService"/>
  <!--提供的服务-->
  <service><provide interface="com.osgi.ds.IOrder"/></service>
  <!--消费的服务-->
  <reference interface="com.osgi.ds.ICart" bind="setCart" unbind="unsetCart"
    policy="dynamic" cardinality="1..1"/>
</scr:component>
```

声明式服务——组件的生命周期

- ▶ Enabled——组件的启动
- ▶ Satisfied——满足组件的配置信息
- ▶ Immediate Component——立即激活组件
- ▶ Delayed Component——延迟加载组件
- ▶ Factory Component——工厂组件

声明式服务——组件的生命周期

- ▶ Enabled——组件的启动
- ▶ Satisfied——满足组件的配置信息
- ▶ Immediate Component——立即激活组件
- ▶ Delayed Component——延迟加载组件
- ▶ Factory Component——工厂组件



框架加载

- ▶ R4.2提供了一套标准的API用来启动OSGi容器
- ▶ 各OSGi框架都需要实现——org.osgi.framework.launch.FrameworkFactory
- ▶ 允许在容器外启停OSGi框架，例如在web容器中通过listener启动OSGi
- ▶ 示例代码如下：

```
Class<FrameworkFactory> factoryClass = ServiceLoader.load(FrameworkFactory.class);
FrameworkFactory factory = factoryClass.newInstance();
Map<Object, Object> conf;
.....//初始化配置信息
Framework framework=factory.newFramework(conf);
framework.init();
BundleContext bundleContext = framework.getBundleContext();
.....//安装bundle
framework.start();
```

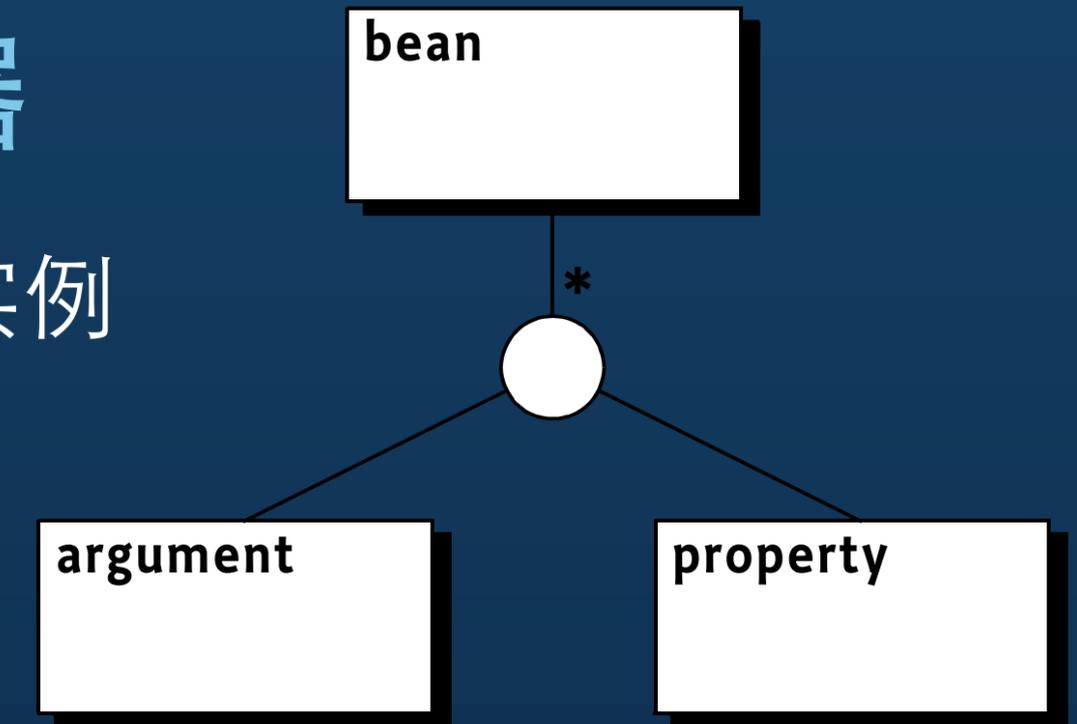
Blueprint服务

- ▶ Blueprint的基础——Spring Dynamic Modules-->Eclipse Gemini
- ▶ 为OSGi定义了依赖注入（dependency injection）框架
- ▶ 处理OSGi服务的动态性
- ▶ 与POJO协同工作
- ▶ OSGI-INF/blueprint/

```
<?xml version="1.0" encoding="UTF-8"?>  
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">  
    ...  
</blueprint>
```

BluePrint服务——Bean管理器

- ▶ 创建具有给定参数和属性的 Java 对象的实例
- ▶ 管理对象的生命周期
- ▶ argument, property, scope
- ▶ 构建方式: className, factoryMethod, factoryComponent



```
<bean id="order" class="com.osgi.bs.OrderService" init-method="init"
  destroy-method="destroy">
  <argument value="1"/>
  <argument value="2"/>
  <property name="Description" value="This is the order"/>
</bean>
```

Blueprint服务——服务注册

`<!--服务注册-->`

```
<service id="orderService" interface="com.osgi.bs.IOrderService" ref="order">  
  <service-properties>  
    <entry key="order.discount" value="0.8"/>  
  </service-properties>  
</service>
```

`<!--实际代码-->`

```
OrderService os = new OrderService(1, 2);  
os.setDescription( "This is the order" );  
Hashtable props = new Hashtable();  
props.put("order.discount", "0.8");  
bundleContext.registerService(com.osgi.bs.OrderService.class.getName(), os,  
props);
```

Blueprint服务——服务引用

<!--reference方式-->

```
<reference id="orderRef" interface="com.osgi.bs.IOrderService" timeout="1000"
availability="optional"/>
<bean id="orderClient" class="...">
  <property name="order" ref="orderRef"/>
</bean>
```

<!--reference-list方式-->

```
<reference-list id="orderRefs" interface="com.osgi.bs.IOrderService"
availability="optional">
  <reference-listener bind-method="bind" unbind-method="unbind">
    <bean class="com.osgi.bs.ReferenceListener"/>
  </reference-listener>
</reference-list>
```

Blueprint服务——服务引用

<!--reference方式-->

```
<reference id="orderRef" interface="com.osgi.bs.IOrderService" timeout="1000"
availability="optional"/>
<bean id="orderClient" class="...">
  <property name="order" ref="orderRef"/>
</bean>
```

List中的对象是动态的，会随着匹配到的服务动态增减。
List没有超时属性，而且是只读的。

<!--reference-list方式-->

```
<reference-list id="orderRefs" interface="com.osgi.bs.IOrderService"
availability="optional">
  <reference-listener bind-method="bind" unbind-method="unbind">
    <bean class="com.osgi.bs.ReferenceListener"/>
  </reference-listener>
</reference-list>
```

BluePrint服务 VS 声明式服务

- ▶ 通过不同的方式实现组件和服务的管理
- ▶ 声明式服务重点关注快速构建轻量级的组件模型
- ▶ BluePrint服务则面向现有的Java开发者构建富组件模型
- ▶ R4.0发布DS，R4.2发布BS
- ▶ 目前DS应用更为广泛，BS则前景广阔

OSGi与Web应用

- ▶ 在Web应用中引入OSGi的好处：
 - ✓ Modularity——改进物理和逻辑结构，易于部署和维护
 - ✓ Lifecycle——动态安装|使用|更新|卸载
 - ✓ Services——动态解偶和依赖

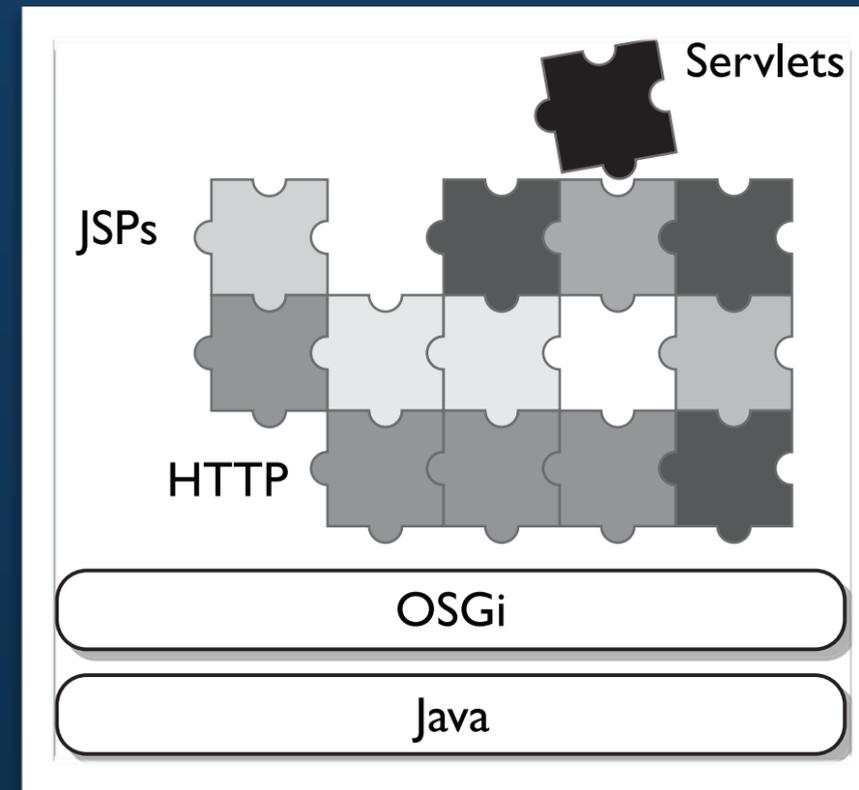
OSGi与Web应用

- ▶ 集成的两种方式：
 - ✓ HttpService
 - ✓ Bridged

OSGi与Web应用

集成的两种方式:

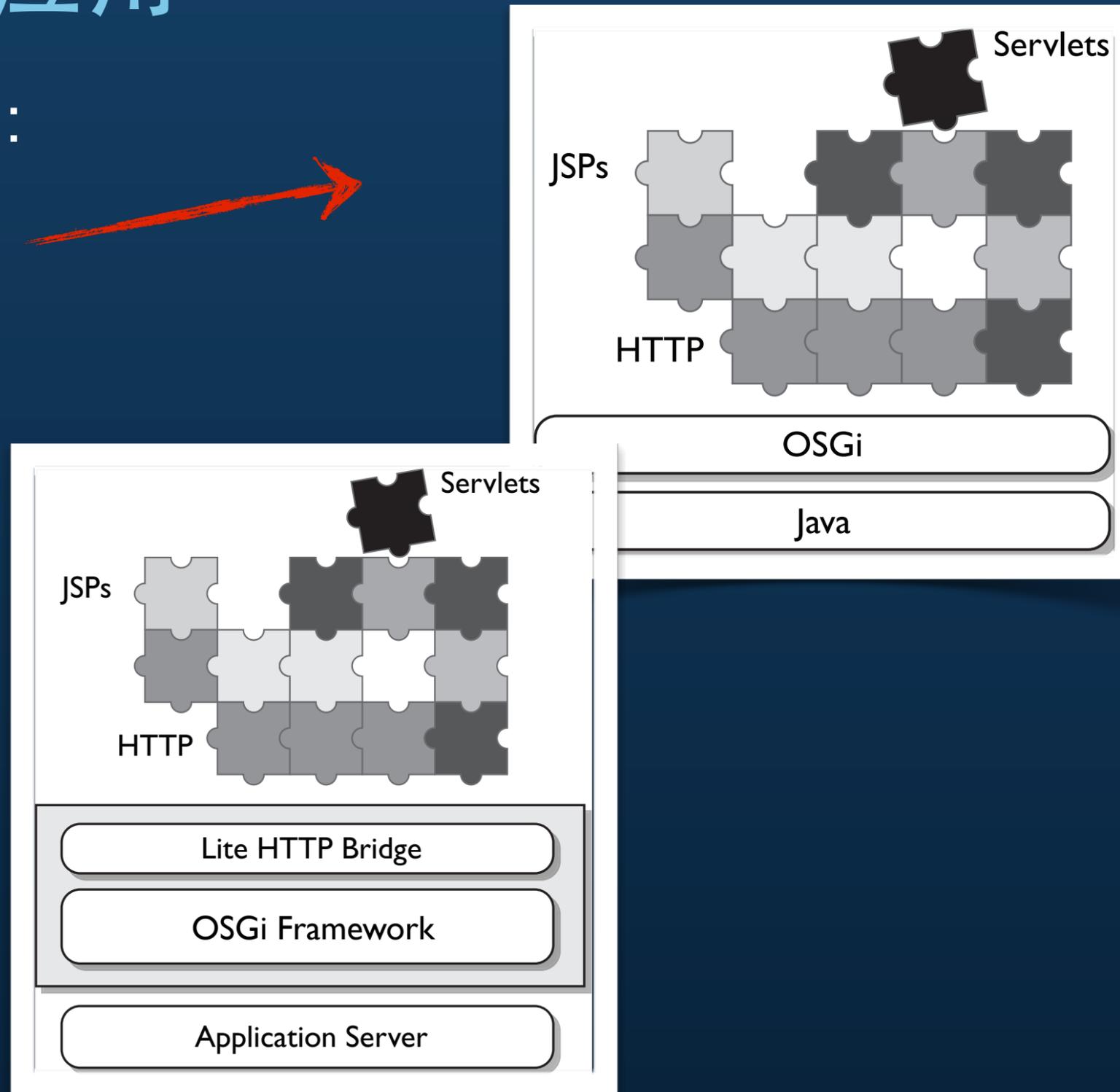
- ✓ HttpService
- ✓ Bridged



OSGi与Web应用

集成的两种方式:

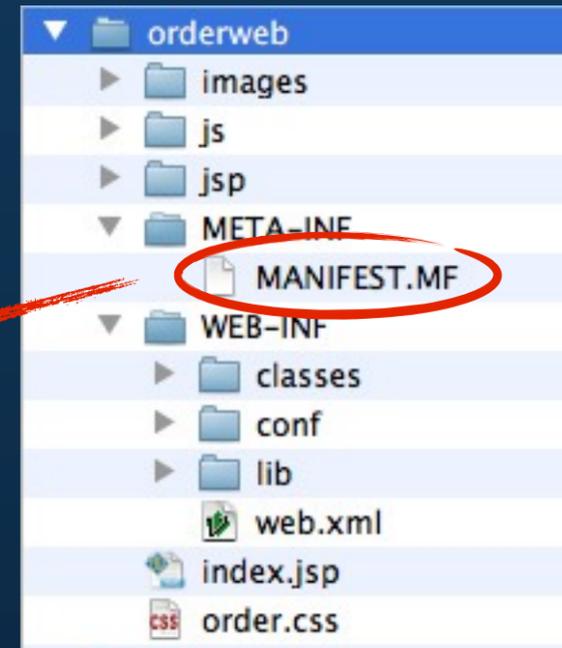
- ✓ HttpService
- ✓ Bridged



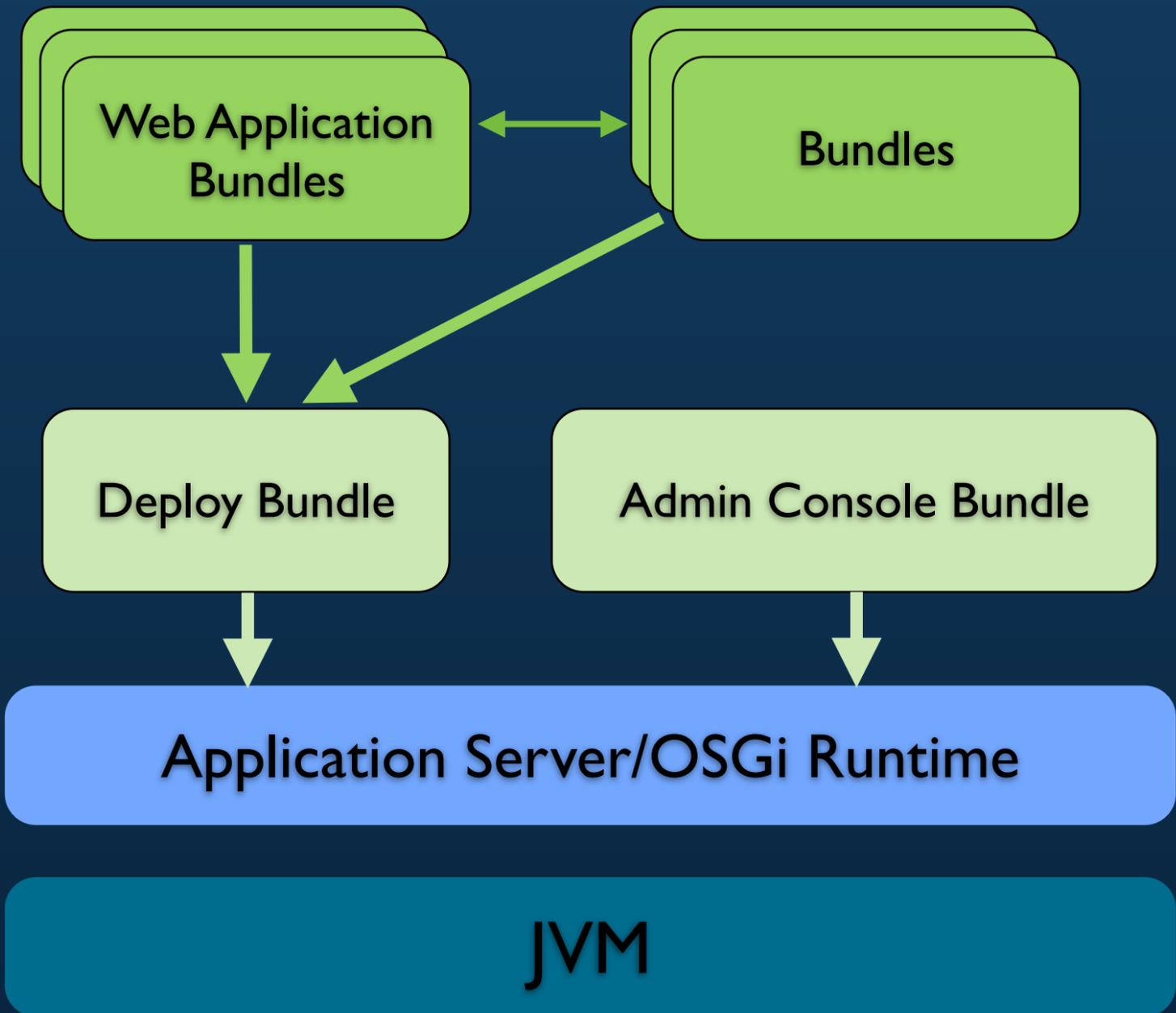
第三种武器——WAB

- ▶ WAB——Web Application Bundle 1.0 (R4.2)
- ▶ 以Bundle形式部署在OSGi框架中，提供Web应用的功能
- ▶ 支持Servlet 2.5和JSP 2.1

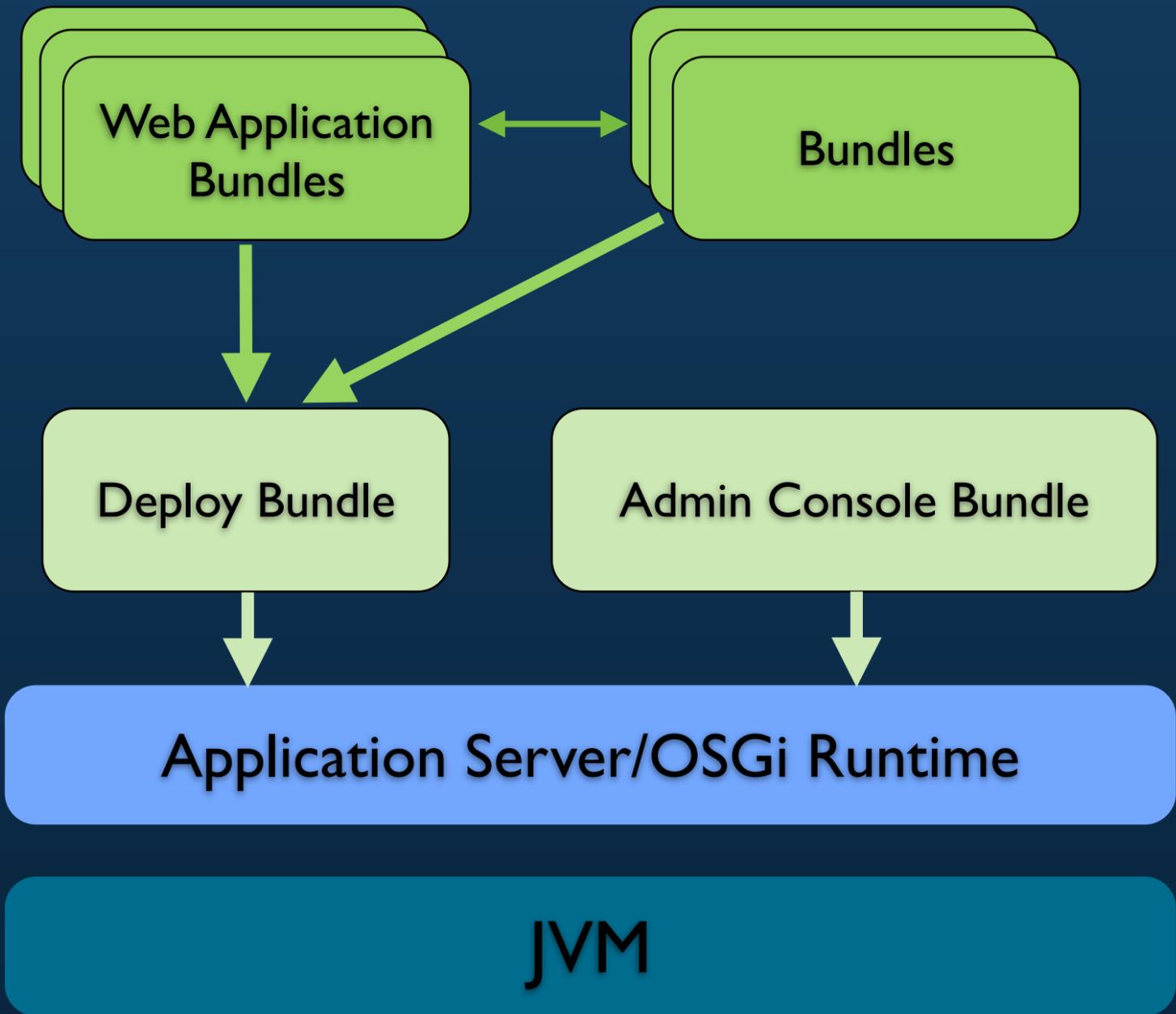
```
Manifest-Version: 1.0
Import-Package: ...
...
Bundle-Version: 1.0
Bundle-Name: orderManager
Bundle-ClassPath: WEB-INF/lib/commons-logging.jar,WEB-INF/classes
Web-ContextPath: /order
Bundle-ManifestVersion: 2
Bundle-SymbolicName: com.osgi.bs.order
```



Web Bundle Architecture



Web Bundle Architecture



开源实践

- ▶ Equinox——最成熟的OSGi开源实现，Eclipse的插件基础
- ▶ Gemini (Spring DM)——实现了R4.2中的企业级规范
- ▶ Virgo (Spring DM Server)——基于OSGi的Application Server
- ▶ Felix——另一个成熟的OSGi开源实现
- ▶ Nuxeo——企业级CMS，实现了部分OSGi规范

Questions?

Follow me at
twitter.com/sagacity
t.sina.com.cn/n/池建强

RayooTech