**副本 Tomcat 生产环境应用配置**

**课程概要：**

* Tomcat启动与部署方式
* Tomcat server.xml 配置详解
* Tomcat集群与会话复制

## Tomcat启动与部署方式

提问：

**我们是如何在Tomcat上部署项目上线的？**

1. 打好WAR包
2. 停止Tomcat
3. 上传并替换 原程序Context目录
4. 删除原来的WAR包
5. 删除原来的Context 目录
6. 进行 WEB-INF/classes/app.propertites config.propertites 目录 找到应的配置文件并修改
7. 启动Tomcat

这么做的弊端是什么？

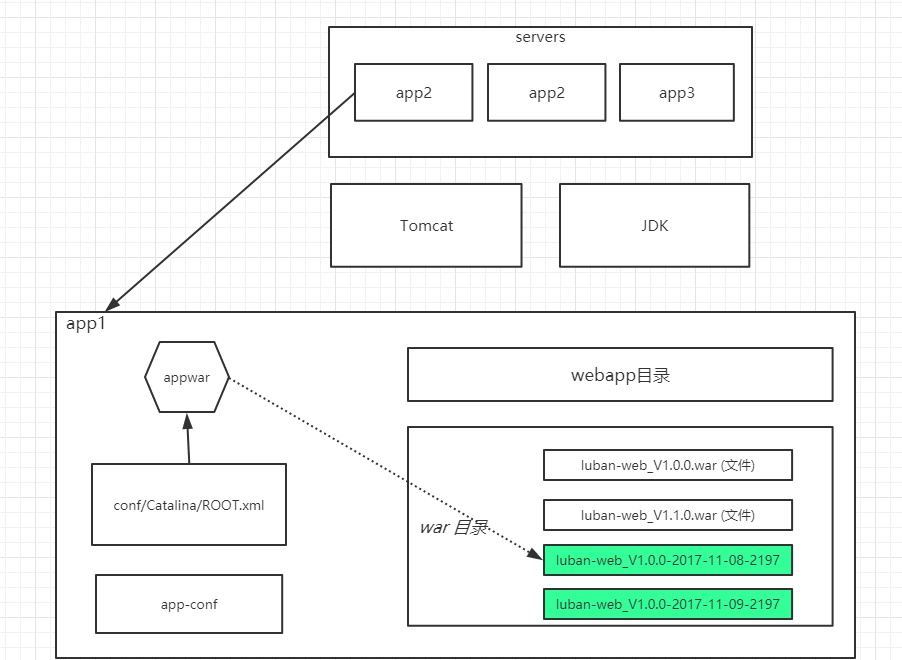
如果发布失败需要快速回滚怎么办？

Tomcat升级怎么弄？

**基于shell 编写自定义启动脚本实现一键发布。**如要完成已下功能。

1. Tomcat 执行文件与程序目录分离。（便于后续升级Tomcat或统一配置Tomcat）(OK)
2. 一键部署发布应用 (OK)
3. 可快速回滚应用和配置 (OK)
4. 自定义配置应用 (OK)

**部署规化：**



**实现步骤**

1. 下载并解压Tomcat
2. 创建并拷贝应用目录
3. 创建Tomcat.sh
4. 编写Tomcat.sh
5. 测试Tomcat.sh （启动|停止|重启）
6. 编写deploy.sh 用于版本发布
7. 基于当前时间、应用名称、版本号创建appBase
8. 下载WAR包 并保存至WAR 目录
9. 解压WAR文件
10. 删除软链接
11. 停止Tomcat
12. 创建 并写入Context 文件
13. 执行Tomcat启动脚本

[deploy.sh](https://dn-shimo-attachment.qbox.me/TfpchxraLxY2XHUZ/deploy.sh)

[tomcat.sh](https://dn-shimo-attachment.qbox.me/inZUJT1a7AkbOHdZ/tomcat.sh)

## **Tomcat server.xml 配置详解**

**server.xml 是一个**

server.xml 的主要职责

server.xml 体系结构

<Server>

<Listener /><!-- 监听器 -->

<GlobaNamingResources> <!-- 全局资源 -->

</GlobaNamingResources

<Service> <!-- 服务 用于 绑定 连接器与 Engine -->

<Connector 8080/> <!-- 连接器-->

<Connector 8010 /> <!-- 连接器-->

<Connector 8030/> <!-- 连接器-->

<Engine> <!-- 执行引擎-->

<Logger />

<Realm />

<host "www.tl.com" appBase=""> <!-- 虚拟主机-->

<Logger /> <!-- 日志配置-->

<Context "/luban" path=""/> <!-- 上下文配置-->

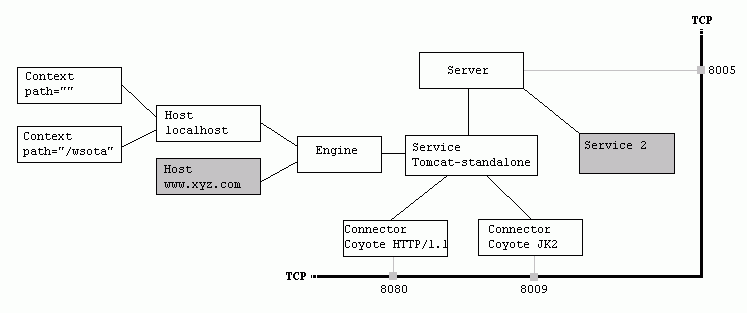
</host>

</Engine>

</Service>

</Server>

**server 体系结构图**



配置文件示：

[server.xml](https://dn-shimo-attachment.qbox.me/HG1MoQpJlA0hPPei/server.xml)

**元素说明：**

一个 server 可对应多个 service

<service> 元素的主要作用是将 一到多个Connector 与一个 Engine 关联。当Connector 接收到请求后分发给 Engine 进行处理。

*演示配置多个Connector(10分钟)*

Host

host 表示一个虚拟主机,默认使用localhost ，一个Engine 中可配置多个host

***演示配置 建立多个虚拟站点 即Host (10分钟)***

Context

表示应用加载目录 通过 path 属性指定。其相对路径为 catalina\_base 目录。可配置多个 Context。另外也可以在 $catalina\_base/conf/$host\_name/XXX.xml 中添加 Context 元素。

演示添加 Context

|  |  |  |
| --- | --- | --- |
| 元素名 | 属性 | 解释 |
| server | port | 指定一个端口，这个端口负责监听关闭tomcat的请求 |
| shutdown | 指定向端口发送的命令字符串 |  |
| service | name | 指定service的名字 |
| Connector(表示客户端和service之间的连接) | port | 指定服务器端要创建的端口号，并在这个断口监听来自客户端的请求 |
| minThread | 服务器启动时创建的处理请求的线程数 |  |
| maxThread | 最大可以创建的处理请求的线程数 |  |
| enableLookups | 如果为true，则可以通过调用request.getRemoteHost()进行DNS查询来得到远程客户端的实际主机名，若为false则不进行DNS查询，而是返回其ip地址 |  |
| redirectPort | 指定服务器正在处理http请求时收到了一个SSL传输请求后重定向的端口号 |  |
| acceptCount | 指定当所有可以使用的处理请求的线程数都被使用时，可以放到处理队列中的请求数，超过这个数的请求将不予处理 |  |
| connectionTimeout | 指定超时的时间数(以毫秒为单位) |  |
| Engine(表示指定service中的请求处理机，接收和处理来自Connector的请求) | defaultHost | 指定缺省的处理请求的主机名，它至少与其中的一个host元素的name属性值是一样的 |
| Context(表示一个web应用程序，通常为WAR文件，关于WAR的具体信息见servlet规范) | docBase | 应用程序的路径或者是WAR文件存放的路径 |
| path | 表示此web应用程序的url的前缀，这样请求的url为http://localhost:8080/path/\*\*\*\* |  |
| reloadable | 这个属性非常重要，如果为true，则tomcat会自动检测应用程序的/WEB-INF/lib 和/WEB-INF/classes目录的变化，自动装载新的应用程序，我们可以在不重起tomcat的情况下改变应用程序 |  |
| host(表示一个虚拟主机) | name | 指定主机名 |
| appBase | 应用程序基本目录，即存放应用程序的目录 |  |
| unpackWARs | 如果为true，则tomcat会自动将WAR文件解压，否则不解压，直接从WAR文件中运行应用程序 |  |
| Logger(表示日志，调试和错误信息) | className | 指定logger使用的类名，此类必须实现org.apache.catalina.Logger 接口 |
| prefix | 指定log文件的前缀 |  |
| suffix | 指定log文件的后缀 |  |
| timestamp | 如果为true，则log文件名中要加入时间，如下例:localhost\_log.001-10-04.txt |  |
| Realm(表示存放用户名，密码及role的 数据库 ) | className | 指定Realm使用的类名，此类必须实现org.apache.catalina.Realm接口 |
| Valve(功能与Logger差不多，其prefix和suffix属性解释和Logger 中的一样) | className | 指定Valve使用的类名，如用org.apache.catalina.valves.AccessLogValve类可以记录应用程序的访问信息 |
| directory | 指定log文件存放的位置 |  |
| pattern | 有两个值，common方式记录远程主机名或ip地址，用户名，日期，第一行请求的字符串，HTTP响应代码，发送的字节数。combined方式比common方式记录的值更多 |  |

## **Tomcat集群与会话复制**

**Tomcat 集群**

演示基于Nginx 集群配置Tomcat (10分钟)

**Tomcat 会话管理器**

**(1).StandardManager**

Tomcat6的默认会话管理器，用于非集群环境中对单个处于运行状态的Tomcat实例会话进行管理。当Tomcat关闭时，这些会话相关的数据会被写入磁盘上的一个名叫SESSION.ser的文件，并在Tomcat下次启动时读取此文件。

**(2).PersistentManager**

当一个会话长时间处于空闲状态时会被写入到swap会话对象，这对于内存资源比较吃紧的应用环境来说比较有用。

**(3).DeltaManager**

**用于Tomcat集群的会话管理器，**它通过将改变了会话数据同步给集群中的其它节点实现会话复制。这种实现会将所有会话的改变同步给集群中的每一个节点，也是在集群环境中用得最多的一种实现方式。

**(4).BackupManager**

用于Tomcat集群的会话管理器，与DeltaManager不同的是，某节点会话的改变只会同步给集群中的另一个而非所有节点。