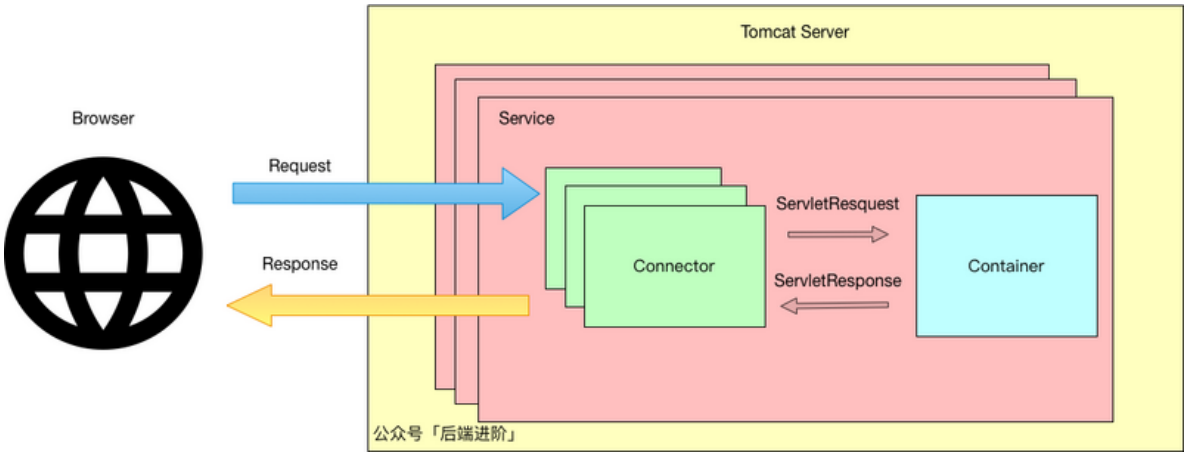
Tomcat体系结构

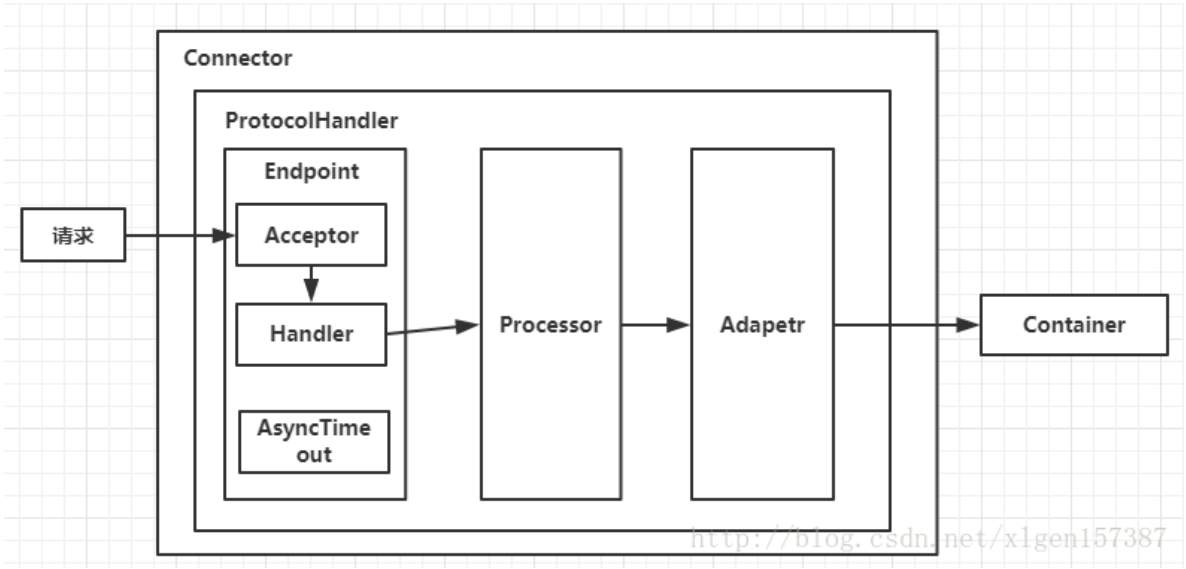
1.整体架构

如图所示，Tomcat本质上是一个**HTTP服务器+Servlet容器**



一个 Tomcat 代表一个 Server 服务器，一个 Server 服务器可以包含多个 Service 服务。在Service中可以看到，主要有两部分组成，即**Connector和Container**，Connector用于接受请求并将请求封装成Request和Response来具体处理，而Container用于封装和管理Servlet，以及具体处理request请求。

2.Connector连接器



> connector官方定义：A Connector handles communications with the client.即处理与客户端的交互。

当客户端发来一个请求时，connector会将它封装成request对象，并适配成container看得懂的模样交给container处理，在container处理完成后再将返回结果封装成response对象，响应给客户端。

connector的存在使得container容器（容器组件）与具体的请求协议及IO操作方式完全解耦，容器只要专心的处理请求即可。

2.1 Connector的组成结构

Connector内部使用ProtocolHandler来处理请求，ProtocolHandler 是一个接口，不同协议去实现该接口来完成相应协议的请求处理。ProtocolHandler 采用组件模式的设计，将请求到来后的一连串处理动作封装成了三大组件。

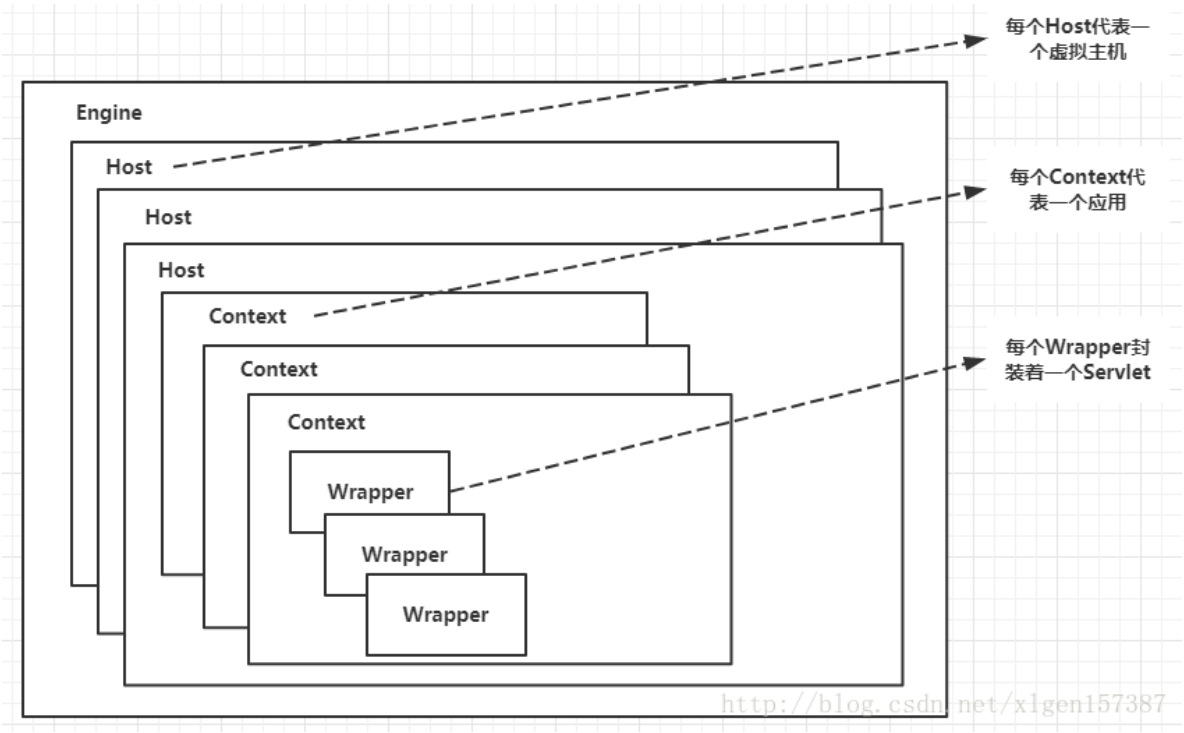
1. Endpoint：用来处理底层的 Socket 网络连接

   Endpoint 是connector的通信端点，是具体Socket接收和发送处理器。endpoint是对传输层的抽象，因此它是用来实现TCP/IP协议的。

2. Processor：用于将Endpoint接收到的Socket封装成Request，是connector的协议处理接口，用来实现HTTP协议的。

3. Adapter：用于将Request交给Container进行具体的处理，需要将Tomcat封装的请求对象Tomcat Request转换成标准的ServletRequest。

3.container 容器



容器主要是指Servlet容器，负责加载和管理Servlet，以及具体处理Request请求。在 Tomcat 中一共设计了 4 种容器，它们分别为 Engine、Host、Context、Wrapper。

1. Engine：表示一个虚拟主机的引擎，**一个 Tomcat Server 只有一个引擎**，连接器所有的请求都交给引擎处理，而引擎则会交给相应的虚拟主机去处理请求；
2. Host：表示虚拟主机，**一个容器可以有多个虚拟主机**，每个主机都有对应的域名（如 localhost），在 Tomcat 中，一个 webapps 就代表一个虚拟主机，当然 webapps 可以配置多个；
3. Context：表示一个应用容器，**一个虚拟主机可以拥有多个应用**，webapps 中每个目录都代表一个 Context，每个应用可以配置多个 Servlet。
4. Wrapper：是对具体Servlet的封装。

上述四种容器都实现了Container接口：

```
public interface Container extends Lifecycle {
  Container getParent();
  void setParent(Container container);
  void addChild(Container child);
  Container findChild(String name);
  Container[] findChildren();
  void removeChild(Container child);
}
```

可以看到，Container继承了Lifecycle 接口：

```
public interface Lifecycle {
  public static final String INIT_EVENT = "init";
  public static final String START_EVENT = "start";
  public static final String BEFORE_START_EVENT = "before_start";
  public static final String AFTER_START_EVENT = "after_start";
  public static final String STOP_EVENT = "stop";
  public static final String BEFORE_STOP_EVENT = "before_stop";
  public static final String AFTER_STOP_EVENT = "after_stop";
  public static final String DESTROY_EVENT = "destroy";

  public void addLifecycleListener(LifecycleListener listener);
  public LifecycleListener[] findLifecycleListeners();
  public void removeLifecycleListener(LifecycleListener listener);
  public void start() throws LifecycleException;
  public void stop() throws LifecycleException;
}
```

实际上，Tomcat很多组件都直接或间接实现了Lifecycle 接口，从而可以对众多组件进行相对统一、规范的生命周期维护。

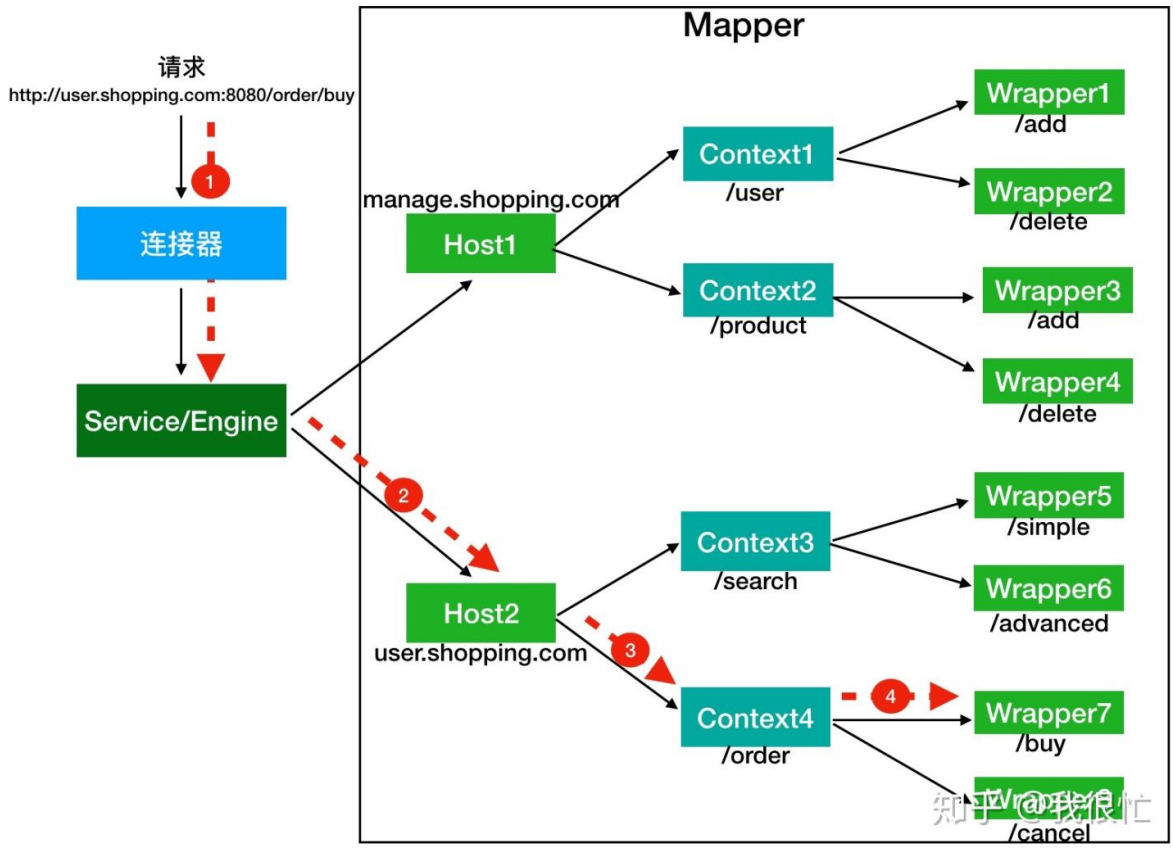
3.1 Mapper组件

上述容器中，Host、Context和Wrapper，他们相互之间的关系以及请求来临时逐级定位Servlet的操作都是由Mapper组件来完成和维护的。

在容器初始化的时候，Tomcat通过解析配置文件，会对容器逐级进行注册，将注册好的容器的依赖关系保存在Mapper组建中。这样当一个请求地址来临时，便可以在Mapper中一层一层的寻找到对应的Servlet。

Mapper 的构建过程是在 **MapperListener** 中完成的， MapperListener 实现了 Tomcat 的生命周期接口，在 StandardService 启动的时候会调用 MapperListener 的 start 方法

附一张知乎上的请求定位图：


Mapper
请求
http://user.shopping.com:8080/order/buy
1
连接器
Service/Engine
manage.shopping.com
Host1
Context1
/user
Wrapper1
/add
Wrapper2
/delete
Context2
/product
Wrapper3
/add
Wrapper4
/delete
2
Host2
user.shopping.com
Context3
/search
Wrapper5
/simple
Wrapper6
/advanced
3
Context4
/order
4
Wrapper7
/buy
/cancel
知乎 @我很忙