**Struts2是一套很优秀的Web应用框架，实现优雅、功能强大、使用简洁。能够说是Struts2是一款很成熟的MVC架构。**

**在我们学习Struts2时，最好是先学习它的执行流程、核心概念。从中得到启示。提升自己，而不不过学习怎么怎么使用它。**

**在网上看到这样一句话：**

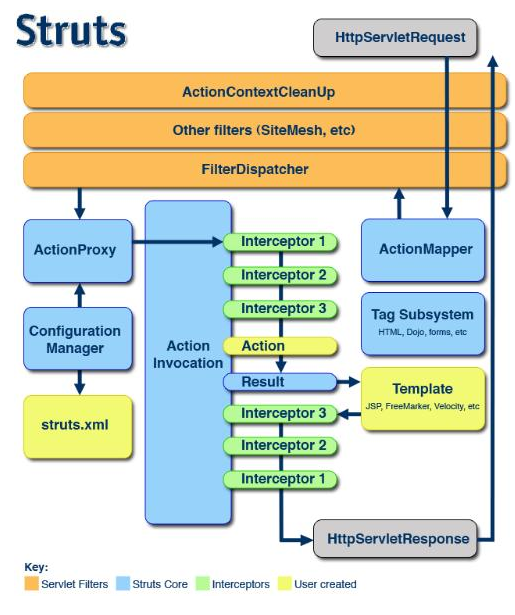
**你千万不要成为一个仅仅会熟练使用框架的程序猿。那样。你会疲于奔命，你或许永远仅仅会使用 Hadoop ，而写不出一个 Hadoop ，你仅仅是一个 Hadoop程序猿，而不是一个分布式project师。**

**你或许永远仅仅会使用 Struts，而忘记了自己写 filter，你仅仅是一个 SSH 程序猿，而不是一个 Web project师。  
  
话不多说，一起走进Struts2**

**一、系统架构**

**Struts2的官方文档附带了Struts2的架构图。**

**从这张图能够非常好的去理解Struts2**

****

**关于图中的Key：**

* **Servlet Filters：过滤器链，client的全部请求都要经过Filter链的处理。**
* **Struts Core：Struts2的核心部分，可是Struts2已经帮我们做好了，我们不须要去做这个**
* **Interceptors。Struts2的拦截器。Struts2提供了非常多默认的拦截器。能够完毕日常开发的绝大部分工作；而我们自己定义的拦截器，用来实现实际的客户业务须要的功能。**
* **User Created，由开发者创建的。包含struts.xml、Action、Template。这些是每一个使用Struts2来进行开发的人员都必须会的。**
* **1.FilterDispatcher是整个Struts2的调度中心。也就是MVC中的C（控制中心），依据ActionMapper的结果来决定是否处理请求，假设ActionMapper指出该URL应该被Struts2处理。那么它将会运行Action处理，并停止过滤器链上还没有运行的过滤器。**
* **2.ActionMapper 会推断这个请求是否应该被Struts2处理，假设须要Struts2处理。ActionMapper会返回一个对象来描写叙述请求相应的ActionInvocation的信息。**
* **3.ActionProxy。它会创建一个ActionInvocation实例，位于Action和xwork之间，使得我们在将来有机会引入很多其它的实现方式。比方通过WebService来实现等。**
* **4.ConfigurationManager是xwork配置的管理中心，能够把它看做struts.xml这个配置文件在内存中的相应。**
* **5.struts.xml，是开发人员必须光顾的地方。是Stuts2的应用配置文件，负责诸如URL与Action之间映射关系的配置、以及运行后页面跳转的Result配置等。**
* **6.ActionInvocation：真正调用并运行Action，它拥有一个Action实例和这个Action所依赖的拦截器实例。ActionInvocation会依照指定的顺序去运行这些拦截器、Action以及相应的Result。**
* **Interceptor(拦截器)：是Struts2的基石。类似于JavaWeb的Filter，拦截器是一些无状态的类。拦截器能够自己主动拦截Action，它们给开发人员提供了在Action运行之前或Result运行之后来运行一些功能代码的机会。**
* **7.Action：用来处理请求，封装数据。**
* **二、执行流程**

**1.当用户的发出请求。比方http:localhost:8080/Struts2/helloworld/helloworldAction.action,请求会被Tomcat接收到，Tomcatserver来选择处理这个请求的Web应用，那就是由helloworld这个webproject来处理这个请求。**

**2.Web容器会去读取helloworld这个project的web.xml。在web.xml中进行匹配，但发现，由struts2这个过滤器来进行处理（也就是**

**StrutsPrepareAndExecuteFilter）。依据Filter的配置，找到FilterDispatcher（Struts2的调度中心）**

**3.然后会获取FilterDispatcher实例，然后回调doFilter方法，进行真正的处理**

**PS：FilterDispatcher是不论什么一个Struts2应用都须要配置的，通常情况下。web.xml文件里还有其它过滤器时，FilterDispatcher是放在滤器链的最后；假设在FilterDispatcher前出现了如SiteMesh这样的特殊的过滤器，还必须在SiteMesh前引用Struts2的ActionContextCleanUp过滤器**

**相应Struts2的架构图例如以下**

****

**4.这时FilterDispatcher会将请求转发给ActionMapper。ActionMapper负责识别当前的请求是否须要Struts2做出处理。ActionMapper就类似于公司的保安。来识别是不是当前客户是不是我公司的人**

**相应Struts2的架构图例如以下**

****

**5.假设须要Struts2处理。ActionMapper会通知FilterDispatcher。须要处理这个请求。FilterDispatcher会停止过滤器链以后的部分，（这也就是为什么，FilterDispatcher应该出如今过滤器链的最后的原因）。然后建立一个ActionProxy实例。这个对象作为Action与xwork之间的中间层。会代理Action的执行过程。**

**相应Struts2的架构图例如以下**

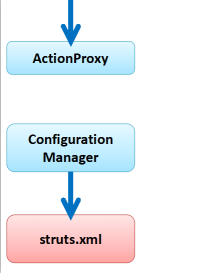
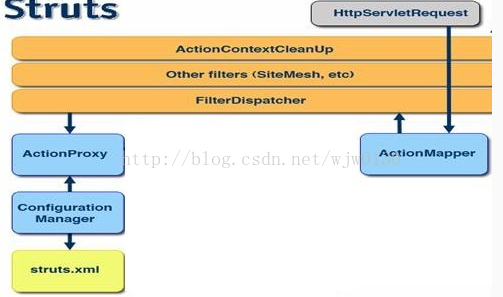
****

**6.ActionProxy对象在被创建出来的时候，并不知道要执行哪个Action，它手里仅仅有从FilterDispatcher中拿到的请求的URL。**

**而真正知道要执行哪个Action的是ConfigurationManager。由于仅仅有它才干读取我们的strtus.xml**

**（在server启动的时候，ConfigurationManager就会把struts.xml中的全部信息读到内存里，并缓存。当ActionProxy带着URL向他询问要执行哪个Action的时候，就能够直接匹配、查找并回答了）**

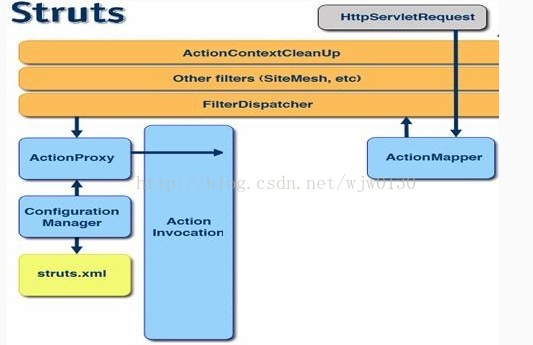
**相应Struts2的架构图例如以下**

**  -> **

**7.ActionProxy知道自己该干什么事之后（执行哪个Action、相关的拦截器以及全部可能使用的result信息），然后立即建立ActionInvocation对象了，ActionInvocation对象描写叙述了Action执行的整个过程。**

**注意：Action完整的调用过程都是由ActionInvocation对象负责**

**相应Struts2的架构图例如以下**

****

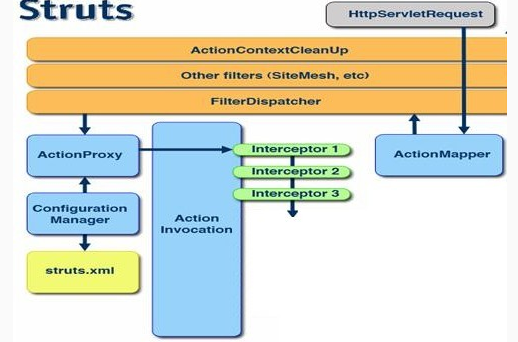
**8.在execute方法之前，好像URL请求中的參数已经赋值到了Action的属性上。这就是我们的"雷锋"—拦截器。**

**拦截器的执行被分成两部分，一部分在Action之前执行，一部分在Result之后执行，并且顺序是刚好反过来的。也就是在Action执行前的顺序，比方是拦截器1、拦截器2、拦截器3，那么执行Result之后，再次执行拦截器的时候，顺序就变成拦截器3、拦截器2、拦截器1了。**

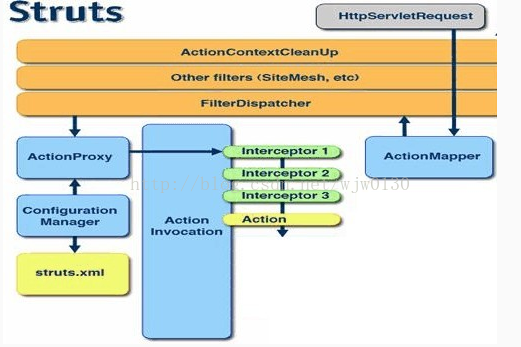
**这就好比，你要去奶奶家。需要通过 水泊梁山->盘丝洞 -> 索马里，到了奶奶家。看奶奶回来的时候，就必需要通过 索马里 -> 盘丝洞 -> 水泊梁山。**

**所以ActionInvocation对象运行的时候须要通过非常多复杂的过程，依照指定拦截器的顺序依次运行。**

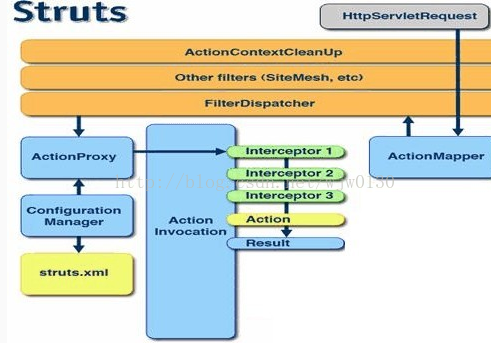
**相应Struts2的架构图例如以下**

****

**9.到了奶奶家，然后运行Action的execute方法**

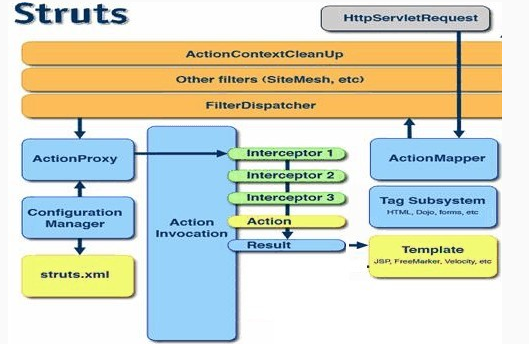
****

**10.然后依据execute方法返回的结果（Result），去struts.xml中匹配选择下一个页面**

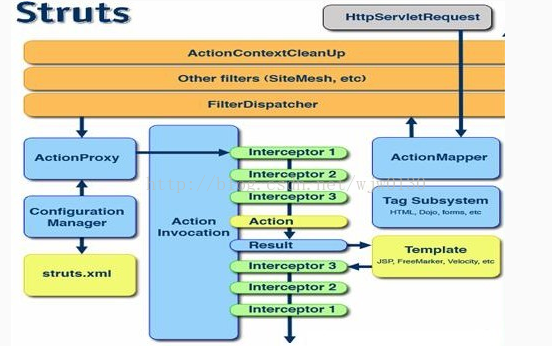
****

**11.依据结果(Result)找到页面后，在页面上(有非常多Struts2提供的模板)，能够通过Struts2自带的标签库来訪问须要的数据，并生成终于页面**

**注意：这时还没有给client应答，仅仅是生成了页面**

****

**12.最后，ActionInvocation对象倒序运行拦截器,从奶奶家回来**

****

**13.ActionInvocation对象运行完成后，已经得到响应对象（HttpServletResponse）了，最后按与过滤器（Filter）配置定义相反的顺序依次经过过滤器，向client展示出响应的结果**