PERA13.1 WS&WF 性能优化点及思路

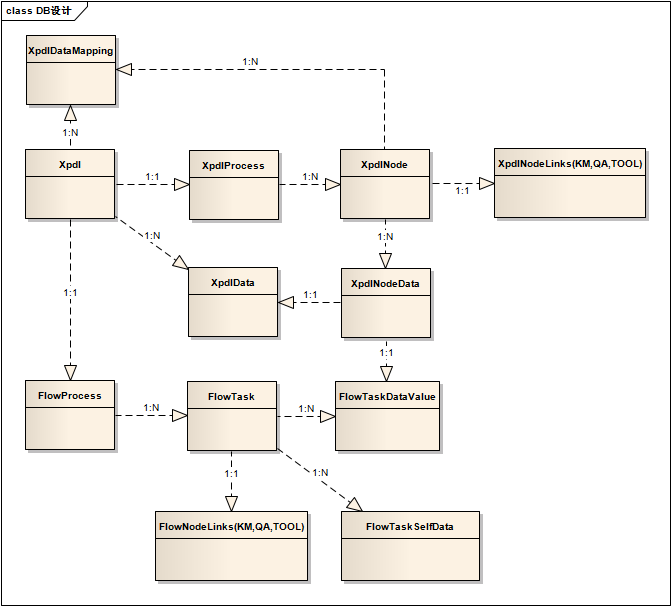
1. 并发流程（611需求）

并发流程主要针对13.1中的功能需求。其特点是活动之间无顺序关系，仅存在数据之间的交互。流程发起时所有活动直接产生。同时也不存在路由和转移线的功能。因此可以考虑重写一个全新的驱动引擎。初步方案如下：

* 1. 数据格式

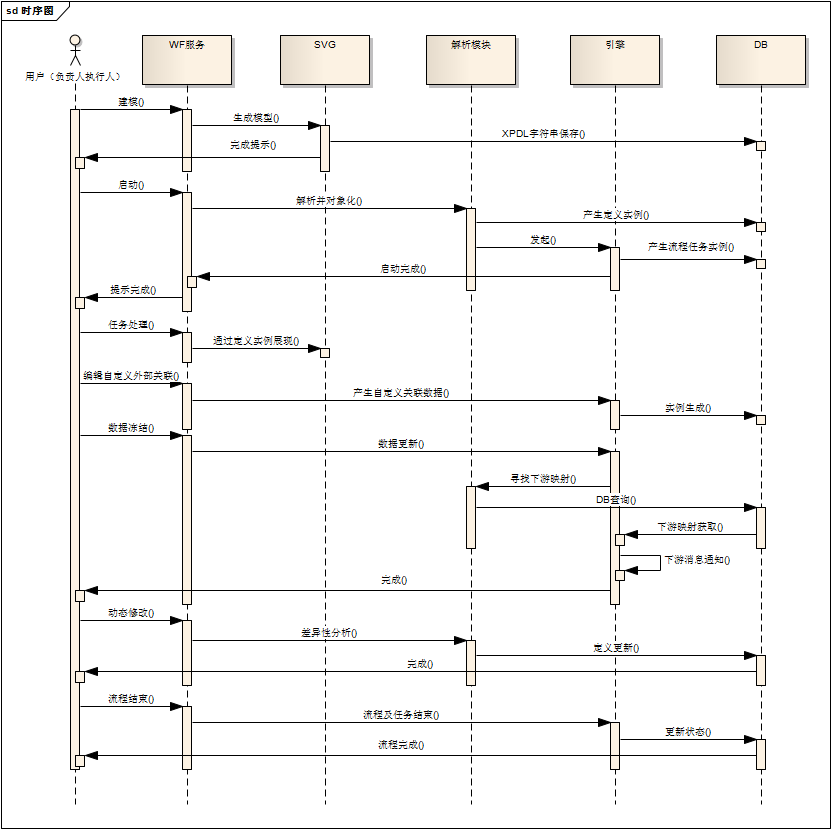
对于流程定义，仍采取既有的XPDL规范。仅对于数据之间的流转线，仍在XPDL中记录，但是引擎不再解析使用，仅作为SVG绘图使用。

* 1. 数据对象



* 1. 数据解析

对于XPDL的解析，由内存中的对象化管理改为DB访问。初始时仅保存XPDL字符串，只有当流程发起时才将XPDL解析并写入DB。对于执行时的动态修改，采用对比DB的方式更新对应的数据。



* 1. 性能问题

结合上述设计，由于采用DB更新，因此内存持续增长的问题不存在。但是对于动态修改时，差异性对比更新指定的DB数据需要优化处理，否则全体更新影响效率。可考虑先查询所有，然后通过字符串的形式对比。对于流程启动，不存在复杂的解析，可视为产生多个数据库实例，即使节点数较多也不影响性能。其他存在的额外开销为与PORTAL和DMS集成接口的调用，可采用批量或异步的方式实现。

1. 强制流程（含审批）

强制流程为顺序流程，要求先后关系，同时存在路由，子流程，迭代等功能。性能分析如下：

* 1. 内存管理

由于流程是基于XPDL数据格式，且在使用是结构复杂，需要频繁的获取定义信息。因此采用将定义数据对象化在内存中，这样使用时减少了解析的开销。但是内存没有释放功能，用户使用数据会持续增长，有可能导致内存溢出。将XPDL转为DB管理，因为强制流程结构复杂，所以不太适合。初步解决方案如下：

* + 1. 增加内存释放功能。例如对于某个定义数据，如长时间未使用，则释放。在服务端可定时检查。
    2. 重新检查加载时机。现在为无条件全体加载。是否可特定时机加载后保留而其他时机加载解析完成后直接释放？
    3. 缓存服务器。将XDPL定义的内存管理移到单独的缓存服务器进行管理。对此需要进一步的调研。
  1. 流程启动

对于强制流程，启动时仅发起首个节点。节点的定义信息已在解析时获取，因此可直接生成任务实例（对于13.0的并发流程，因为采取的是在强制流程中分支代码模拟实现全启动，导致需要在启动时整个流程结构全部走完，极大的影响了性能）。对于首个节点较多的情况，可单独测试，分析性能瓶颈。

* 1. 任务提交

任务提交发起后续节点，后台性能主要在于流程引擎流转分析。对于强制流程，性能瓶颈集中在相关业务处理逻辑上，需要具体分析。单独引擎的性能问题不大。

* 1. DB操作

现在采用的HIBERNATE。表设计时存在多个表之间的关联关系。不确定对于大量用户并发操作时是否存在数据库的性能问题。可考虑是否采用其他数据库操作方式。

1. WS流程

此类流程目前是基于强制流程引擎实现。但是对于流转，没有了直接定义转移线，而是全部顺序执行完成。因此可考虑在结构上进行简化。即对于每个节点的流转，仅需要通过顺序线找到下一节点，而无需考虑强制流的那种转移线的事务处理功能。此修改可简化引擎驱动分析的判断处理，提升部分性能。引擎其他部分暂无变化。性能分析如下：

* 1. 内存管理

基本思路同强制流程的内存优化。目前的功能设计为WS流程全体执行而不是强制流程的某个节点提交再发起后续。因此可考虑在流程结束后就立刻释放。

* 1. 流程启动

同2.2

* 1. 任务提交

同2.3

1. SVG性能优化

现有绘图采用SVG插件实现。以下解决方案均为代码层次的优化：

* 1. 解析渲染

所有与svg页面展示无关的数据延迟加载（已实施）。首先去掉节点打开关闭相关事件，减少页面对于javascript的依赖，然后svg后台用java代码生成。

* 1. 减少svg元素数量

每个活动节点的操作尽量用右键菜单，而不在节点上额外添加svg元素

比如取消缩略图，或减少缩略图的复杂度（已实施）。

* 1. 事件响应

避免一切不必要的鼠标拖动或悬浮事件，减少页面渲染频率。

* 1. 更换js引擎

软件环境采用chrome V8 javascript引擎，可以大幅度加快javascript的运行速度(100倍)。页面将采用HTML5标准SVG，脱离Adobe SVG Viewer插件。