EAS Integration Project Presentation

Perspectives and Facts on How to Design and Implement EAS Integration Project

吴继鹏

Software Institute
Nanjing University

Integration, 2014

Outline

1. 展示我们的功能实现情况。
2. 展示我们的设计思路。
3. 展示其他抽象层次的观点。

我们做了哪些？

1. 实现了 3 个教务系统服务器和 1 个集成服务器。
2. 每个教务系统服务器都提供一下服务：
    1. 专门监听其他服务器的 HttpRequest 的服务：
        1. CourseProvider：从数据库里提取课程数据，转化为 XML，然后返回这个 XML。
        2. UpdateProvider：更新数据库，加入新的学生，新的 course_namelist。
    2. 面向用户的服务：CourseSyn, Logout, StudentLogin, StudentOrder。
    3. 集成服务器的服务：Update.java, WatingForSyn.java

我们做了哪些？

1. 实现了 3 个教务系统服务器和 1 个集成服务器。
2. 每个教务系统服务器都提供一下服务：
    1. 专门监听其他服务器的 HttpRequest 的服务：
        1. CourseProvider：从数据库里提取课程数据，转化为 XML，然后返回这个 XML。
        2. UpdateProvider：更新数据库，加入新的学生，新的 course_namelist。
    2. 面向用户的服务：CourseSyn, Logout, StudentLogin, StudentOrder。
    3. 集成服务器的服务：Update.java, WatingForSyn.java

我们做了哪些？

1. 实现了 3 个教务系统服务器和 1 个集成服务器。
2. 每个教务系统服务器都提供一下服务：
    1. 专门监听其他服务器的 HttpRequest 的服务：
        1. CourseProvider：从数据库里提取课程数据，转化为 XML，然后返回这个 XML。
        2. UpdateProvider：更新数据库，加入新的学生，新的 course_namelist。
    2. 面向用户的服务：CourseSyn, Logout, StudentLogin, StudentOrder。
    3. 集成服务器的服务：Update.java, WatingForSyn.java

我们做了哪些？

1. 实现了 3 个教务系统服务器和 1 个集成服务器。
2. 每个教务系统服务器都提供一下服务：
   1. 专门监听其他服务器的 HttpRequest 的服务：
      1. CourseProvider：从数据库里提取课程数据，转化为 XML，然后返回这个 XML。
      2. UpdateProvider：更新数据库，加入新的学生，新的 course_namelist。
   2. 面向用户的服务：CourseSyn, Logout, StudentLogin, StudentOrder。
   3. 集成服务器的服务：Update.java, WatingForSyn.java

我们做了哪些？

1. 实现了 3 个教务系统服务器和 1 个集成服务器。
2. 每个教务系统服务器都提供一下服务：
    1. 专门监听其他服务器的 HttpRequest 的服务：
        1. CourseProvider: 从数据库里提取课程数据，转化为 XML，然后返回这个 XML。
        2. UpdateProvider: 更新数据库，加入新的学生，新的 course_namelist。
    2. 面向用户的服务：CourseSyn, Logout, StudentLogin, StudentOrder。
    3. 集成服务器的服务: Update.java, WatingForSyn.java

我们做了哪些？

1. 实现了 3 个教务系统服务器和 1 个集成服务器。
2. 每个教务系统服务器都提供一下服务：
   1. 专门监听其他服务器的 HttpRequest 的服务：
      1. CourseProvider: 从数据库里提取课程数据，转化为 XML，然后返回这个 XML。
      2. UpdateProvider: 更新数据库，加入新的学生，新的 course_namelist。
   2. 面向用户的服务：CourseSyn, Logout, StudentLogin, StudentOrder。
   3. 集成服务器的服务: Update.java, WatingForSyn.java

我们做了哪些？

1. 实现了 3 个教务系统服务器和 1 个集成服务器。
2. 每个教务系统服务器都提供一下服务：
   1. 专门监听其他服务器的 HttpRequest 的服务：
      1. CourseProvider：从数据库里提取课程数据，转化为 XML，然后返回这个 XML。
      2. UpdateProvider：更新数据库，加入新的学生，新的 course_namelist。
   2. 面向用户的服务：CourseSyn, Logout, StudentLogin, StudentOrder。
   3. 集成服务器的服务：Update.java, WatingForSyn.java

我们做了哪些？

1. 实现了 3 个教务系统服务器和 1 个集成服务器。
2. 每个教务系统服务器都提供一下服务：
   1. 专门监听其他服务器的 HttpRequest 的服务：
      1. CourseProvider：从数据库里提取课程数据，转化为 XML，然后返回这个 XML。
      2. UpdateProvider：更新数据库，加入新的学生，新的 course_namelist。
   2. 面向用户的服务：CourseSyn, Logout, StudentLogin, StudentOrder。
   3. 集成服务器的服务：Update.java, WatingForSyn.java

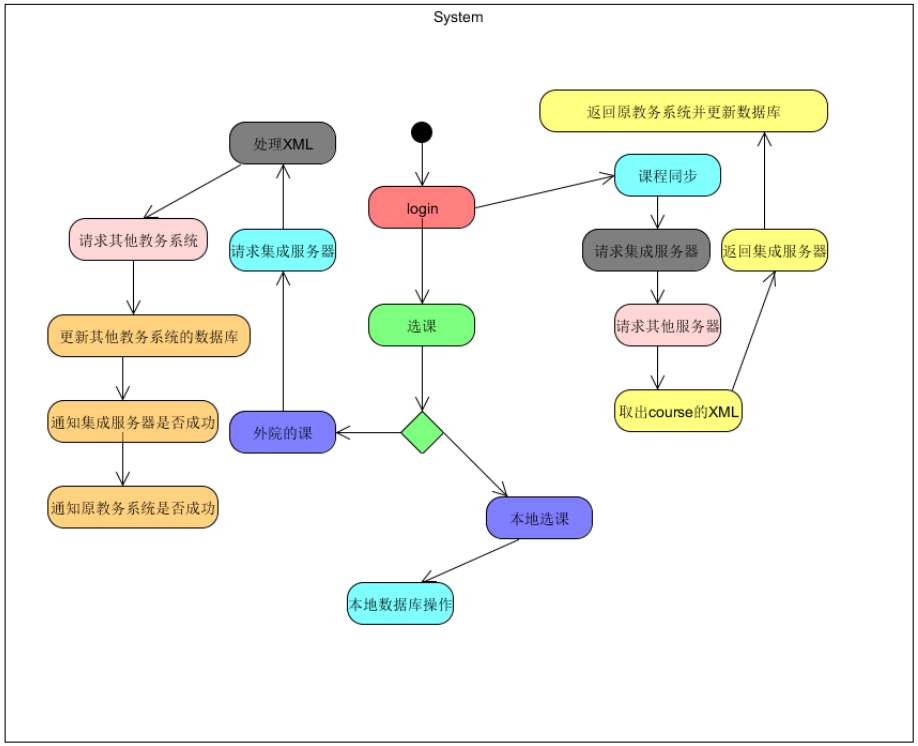

Figure : Activity Diagram of Integration Homework 1

本次作业的 3 个关注点：

1. 可构建性
2. 统一接口
3. 通用组件

本次作业的 3 个关注点：

1. 可构建性
2. 统一接口
3. 通用组件

本次作业的 3 个关注点：

1. 可构建性
2. 统一接口
3. 通用组件

本次作业的 3 个关注点：

1. 可构建性
2. 统一接口
3. 通用组件

Outline

1. 目标：写完 Server_A 后，复制这个项目，然后更改 1 行代码即可完成 Server_B, Server_C 和 Server_I 的部署。
2. 不同项目的区别在于 url 不同。需要把这种代码上的变量进行集中控制。

1. 目标：写完 Server_A 后，复制这个项目，然后更改 1 行代码即可完成 Server_B, Server_C 和 Server_I 的部署。
2. 不同项目的区别在于 url 不同。需要把这种代码上的变量进行集中控制。

1. 目标：写完 Server_A 后，复制这个项目，然后更改 1 行代码即可完成 Server_B, Server_C 和 Server_I 的部署。
2. 不同项目的区别在于 url 不同。需要把这种代码上的变量进行集中控制。

```java
public class Project {
    public static String A="Server_A";
    public static String B="Server_B";
    public static String C="Server_C";
    public static String I="Server_I";
    public static String integration_server_name="Server_I";

    //配置项
    public static String project_name=A;
    public static void echo(String content)
    {
        System.out.println(project_name +">>>: "+content);
    }
    public static void panic(String content)
    {
        System.err.println(project_name +">>>: "+content);
    }
    public static String xml_dir()
    {
        return "D:/tmp/"+project_name+"/";
    }
    public static String split_reg="wjpjw";
}
```

Figure : Project Config

1. 我们的项目跨越了多个生态系统。语言上包括 Html，JS，JSP，Java，SQL，部署环境包括数据库，服务器端程序，Web 客户端。
2. 我们遇到的一个重要的可构建性问题就是如何进行不同子系统的交互。
3. Google 有一个 protocol buffer 就是设计来解决这个问题的，非常不好用。因为 Google 没有办法让它的标准成为一切的标准。
4. 我们需要一个普世的标准化的可扩展的数据类型能够作为不同系统交互的接口。
5. 目前软件行业里没有一个保有数据结构的标准数据类型，仅有的通用接口是文本流。
6. 所以，我们用文本流。
7. 至于 XML，只是文本流的一种协议与规范，在各个平台下都能勉强解析出来，可以用作数据交互。更何况项目中传递的文本流不一定是 XML，也可以包括其他信息，所以说我们使用的接口是文本流，而非 XML。

1. 我们的项目跨越了多个生态系统。语言上包括 Html，JS，JSP，Java，SQL，部署环境包括数据库，服务器端程序，Web 客户端。
2. 我们遇到的一个重要的可构建性问题就是如何进行不同子系统的交互。
3. Google 有一个 protocol buffer 就是设计来解决这个问题的，非常不好用。因为 Google 没有办法让它的标准成为一切的标准。
4. 我们需要一个普世的标准化的可扩展的数据类型能够作为不同系统交互的接口。
5. 目前软件行业里没有一个保有数据结构的标准数据类型，仅有的通用接口是文本流。
6. 所以，我们用文本流。
7. 至于 XML，只是文本流的一种协议与规范，在各个平台下都能勉强解析出来，可以用作数据交互。更何况项目中传递的文本流不一定是 XML，也可以包括其他信息，所以说我们使用的接口是文本流，而非 XML。

1. 我们的项目跨越了多个生态系统。语言上包括 Html，JS，JSP，Java，SQL，部署环境包括数据库，服务器端程序，Web 客户端。
2. 我们遇到的一个重要的可构建性问题就是如何进行不同子系统的交互。
3. Google 有一个 protocol buffer 就是设计来解决这个问题的，非常不好用。因为 Google 没有办法让它的标准成为一切的标准。
4. 我们需要一个普世的标准化的可扩展的数据类型能够作为不同系统交互的接口。
5. 目前软件行业里没有一个保有数据结构的标准数据类型，仅有的通用接口是文本流。
6. 所以，我们用文本流。
7. 至于 XML，只是文本流的一种协议与规范，在各个平台下都能勉强解析出来，可以用作数据交互。更何况项目中传递的文本流不一定是 XML，也可以包括其他信息，所以说我们使用的接口是文本流，而非 XML。

1. 我们的项目跨越了多个生态系统。语言上包括 Html，JS，JSP，Java，SQL，部署环境包括数据库，服务器端程序，Web 客户端。
2. 我们遇到的一个重要的可构建性问题就是如何进行不同子系统的交互。
3. Google 有一个 protocol buffer 就是设计来解决这个问题的，非常不好用。因为 Google 没有办法让它的标准成为一切的标准。
4. 我们需要一个普世的标准化的可扩展的数据类型能够作为不同系统交互的接口。
5. 目前软件行业里没有一个保有数据结构的标准数据类型，仅有的通用接口是文本流。
6. 所以，我们用文本流。
7. 至于 XML，只是文本流的一种协议与规范，在各个平台下都能勉强解析出来，可以用作数据交互。更何况项目中传递的文本流不一定是 XML，也可以包括其他信息，所以说我们使用的接口是文本流，而非 XML。

1. 我们的项目跨越了多个生态系统。语言上包括 Html，JS，JSP，Java，SQL，部署环境包括数据库，服务器端程序，Web 客户端。
2. 我们遇到的一个重要的可构建性问题就是如何进行不同子系统的交互。
3. Google 有一个 protocol buffer 就是设计来解决这个问题的，非常不好用。因为 Google 没有办法让它的标准成为一切的标准。
4. 我们需要一个普世的标准化的可扩展的数据类型能够作为不同系统交互的接口。
5. 目前软件行业里没有一个保有数据结构的标准数据类型，仅有的通用接口是文本流。
6. 所以，我们用文本流。
7. 至于 XML，只是文本流的一种协议与规范，在各个平台下都能勉强解析出来，可以用作数据交互。更何况项目中传递的文本流不一定是 XML，也可以包括其他信息，所以说我们使用的接口是文本流，而非 XML。

1. 我们的项目跨越了多个生态系统。语言上包括 Html，JS，JSP，Java，SQL，部署环境包括数据库，服务器端程序，Web 客户端。
2. 我们遇到的一个重要的可构建性问题就是如何进行不同子系统的交互。
3. Google 有一个 protocol buffer 就是设计来解决这个问题的，非常不好用。因为 Google 没有办法让它的标准成为一切的标准。
4. 我们需要一个普世的标准化的可扩展的数据类型能够作为不同系统交互的接口。
5. 目前软件行业里没有一个保有数据结构的标准数据类型，仅有的通用接口是文本流。
6. 所以，我们用文本流。
7. 至于 XML，只是文本流的一种协议与规范，在各个平台下都能勉强解析出来，可以用作数据交互。更何况项目中传递的文本流不一定是 XML，也可以包括其他信息，所以说我们使用的接口是文本流，而非 XML。

1. 我们的项目跨越了多个生态系统。语言上包括 Html，JS，JSP，Java，SQL，部署环境包括数据库，服务器端程序，Web 客户端。
2. 我们遇到的一个重要的可构建性问题就是如何进行不同子系统的交互。
3. Google 有一个 protocol buffer 就是设计来解决这个问题的，非常不好用。因为 Google 没有办法让它的标准成为一切的标准。
4. 我们需要一个普世的标准化的可扩展的数据类型能够作为不同系统交互的接口。
5. 目前软件行业里没有一个保有数据结构的标准数据类型，仅有的通用接口是文本流。
6. 所以，我们用文本流。
7. 至于 XML，只是文本流的一种协议与规范，在各个平台下都能勉强解析出来，可以用作数据交互。更何况项目中传递的文本流不一定是 XML，也可以包括其他信息，所以说我们使用的接口是文本流，而非 XML。

Outline

1. 文本流是通用接口，至今没有什么可行的替代方案。
2. 它有很多缺点，比如说它的耦合是隐式的，这对编程来说带来了挑战，如果多人合作项目的规格没写完善，可能导致很难消除的 bug。不过这对于我们来说不是问题，涉及到隐式耦合的地方都直接完成了编码，没有拖延。
3. 文本流的优点是计算机系统的任意子集都必须支持它，唯一比较麻烦的是编码，不过如果数据全是英文的，那么可以不去管编码。

1. 文本流是通用接口，至今没有什么可行的替代方案。
2. 它有很多缺点，比如说它的耦合是隐式的，这对编程来说带来了挑战，如果多人合作项目的规格没写完善，可能导致很难消除的 bug。不过这对于我们来说不是问题，涉及到隐式耦合的地方都直接完成了编码，没有拖延。
3. 文本流的优点是计算机系统的任意子集都必须支持它，唯一比较麻烦的是编码，不过如果数据全是英文的，那么可以不去管编码。

1. 文本流是通用接口，至今没有什么可行的替代方案。
2. 它有很多缺点，比如说它的耦合是隐式的，这对编程来说带来了挑战，如果多人合作项目的规格没写完善，可能导致很难消除的 bug。不过这对于我们来说不是问题，涉及到隐式耦合的地方都直接完成了编码，没有拖延。
3. 文本流的优点是计算机系统的任意子集都必须支持它，唯一比较麻烦的是编码，不过如果数据全是英文的，那么可以不去管编码。

1. 文本流是通用接口，至今没有什么可行的替代方案。
2. 它有很多缺点，比如说它的耦合是隐式的，这对编程来说带来了挑战，如果多人合作项目的规格没写完善，可能导致很难消除的 bug。不过这对于我们来说不是问题，涉及到隐式耦合的地方都直接完成了编码，没有拖延。
3. 文本流的优点是计算机系统的任意子集都必须支持它，唯一比较麻烦的是编码，不过如果数据全是英文的，那么可以不去管编码。

1. 一个使用文本流作为通用接口的例子就是 SaxParseService 这个类。
2. 这个类用的时候传入的参数全是字符串或者字符串数组。可以做到通用的 XML 解析，不依赖于 model 里面的任何模型。

Outline

为了更好地完成这个项目，我完成了 3 个独立的模块，与项目本身没有任何耦合。

1. wjp.xml
2. wjp.fileio
3. wjp.http

Thanks