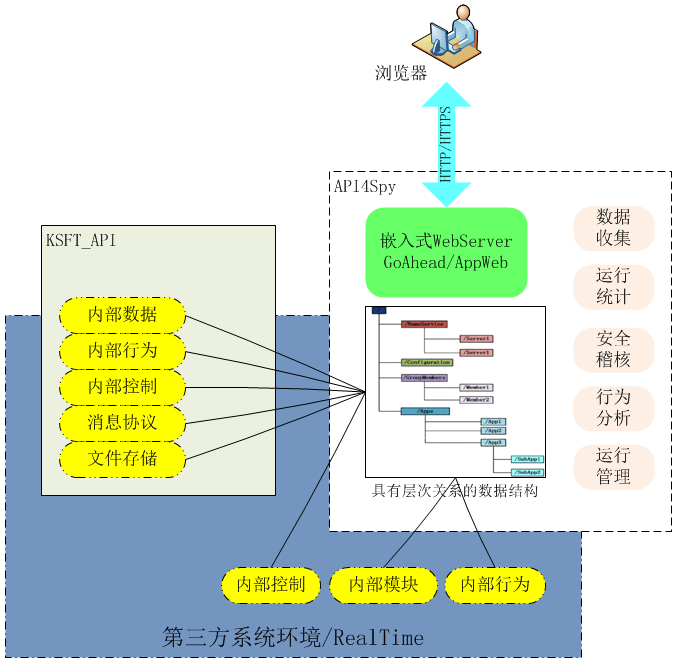
文档修订历史

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **日期** | **描述** | **文档所有者** |
| 1.0 | 2013-08-13 | 创建 | 夏之春 |
| 1.0 | 2013-08-15 | 修改API接口定义和结构图 | 夏之春 |
| 1.1 | 2013-11-20 | 增加[Watcher](#_关于Watcher的考虑)的接口 | 夏之春 |
|  |  |  |  |
|  |  |  |  |

# 系统思路



**API可调用Spy服务接口**

该API的对外接口，允许外部任何一方调用，不论是KSFT\_API还是第三方产品。

该新API的外部接口（C-style）定义为：

// 初始化该API，必须先调用，失败返回false，允许重复调用；

bool SpyInit();

// 销毁释放该API，可以最后调用，允许重复调用；

void SpyFinalize();

// 服务启动，调用时，触发Spy的外部web服务，默认情况下Spy服务不启动 ，启动失败返回false；

bool SpyStartWebServer(char \*ip=“127.0.0.1”, int port = 8080);

// 服务关闭，调用时，关闭Spy的web服务，关闭失败返回false；

bool SpyStopWebServer();

// 获取当前Spy操作失败时的错误代码和错误信息，如果没有则返回false，每条错误信息获取后即不再保留，且当前只保留最新的一条信息；

bool SpyGetErrorInfo(int \*err\_code, char \*info, int size);

// 获取Spy数据值，根据key，如果没有对应key，则返回false

bool SpyGet(char \*key, char \*value);

// 设置Spy数据值，根据key，如果设置失败，则返回false

bool SpyPut(char \*key, char \*value);

// 删除Spy数据，根据key，如果没有对应key，则返回false, recursive为真时，同时清除key下属的成员key数据；

bool SpyDelete(char \*key, bool recursive=false);

// 清除所有Spy数据，并不触发或处理其他关联功能，如果清除失败，则返回false

bool SpyClearAll();

// 导出所有Spy数据值，格式为(key,value)集合，如果导出失败，则返回false

bool SpyDumpAll(char \*file\_name=”ksft\_api4spy\_dump.txt”);

【key 作为抽象服务点的解释】

静态服务点：为预先约定的服务约定，可以使数据交互也可以是消息交互；

动态服务点：可临时创建交互点，但需要使用结束后及时撤除；

服务点的响应：服务端可针对特定服务点做响应式的处理，如回调等，构造及时灵活的交互场景；

参考REST体系；

**数据区定义DA**

key格式

型如文件系统命名格式，/xxx/yy/zzzz

可按/, /xxx, /xxx/yy等分别访问；

Value数据格式

型如JSON格式，JSON的开发包可采用[jsoncpp](http://sourceforge.net/projects/jsoncpp/)；

对于目标系统来说，可以在任何模块的任何代码段里，调用Spy服务接口，设置/输出当前运行信息，运行信息按照文件系统方式规划，大致要求如下：

|  |  |  |
| --- | --- | --- |
| **级别** | **解释** | **规定** |
| 1 | 全局  实例号等 | /  /s\_nn 某API实例 |
| 2 | 资源 | /thread  /queue  /obj |
| 3 | 对象信息 | 包括编号、数量、内容等 |

**http访问点定义**

spy\_get:

其URL的参数格式为，http://xx.xx.xx.xx:8080/spy\_get?key=KEYSTR，其中参数是key，取值KEYSTR为前面所定义的key值，请求返回数据为key所存储的value，其值为JSON格式；

spy\_list:

其URL的参数格式为，http://xx.xx.xx.xx:8080/spy\_list?key=KEYSTR，其中参数是key，取值KEYSTR为前面所定义的key值，请求返回数据为key和其下递归所存储的value，其值为JSON格式，返回为集合形式，每一个元素为一个value对象数据；

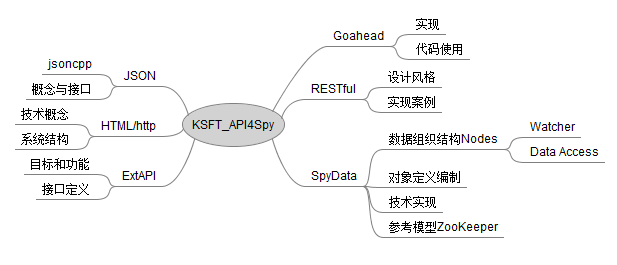
spy\_set: 【暂不实现】

其URL的参数格式为，http://xx.xx.xx.xx:8080/spy\_set?key=KEYSTR&value=VALUESTR，其中参数是key，取值KEYSTR为前面所定义的key值；参数value为取值数据，KEYSTR和VALUESTR 其值类型为JSON格式，该请求为设置功能；考虑VALUESTR的取值复杂性，应该支持POST和GET两种http方式；

**http访问点的安全**

TBD

# 研发次序



方案背景：学习相关技术概念和设计思想；

ExtAPI构造：API接口定义和创建；

JSON编程和对象管理实现：jsoncpp学习和使用；

Goahead使用开发：学习使用

功能接口实现：功能编码和接口测试等；

发布：产品化准备工作；

# 关于Watcher的考虑

**Watcher的功能解释**

观察器类似一种可灵活设置的数据事件观察机制，可以针对任意数据点设置Watcher，然后被观察数据的变更事件都会触发观察器的反馈，这是一种即时的反馈机制。通过此通用性的机制，可以为使用者构造数据变更、通知、回馈等各种基础数据响应以及其他衍生出的丰富模式。

**接口定义**

/\* 设置观察器，watcher为观察器对象指针，recursive标识决定是否观察子节点事件，默认不参与观察

\*/

bool SpySetWatcher(char \*key, CWatcher \*watcher, bool recursive = false);

/\* 观察器类（接口）定义 \*/

class CWatcher{

public:

/\* 观察事件回调，其中DataStatus为数据状态，SpyState为接口访问情况

\*/

virtual void OnWatchEvent(int DataStatus, int SpyState, const char \*key) = 0;

};

其中DataStatus的取值范围是：子节点变更NodeChildrenChanged、节点被创建NodeCreated、节点数据变更NodeDataChanged、节点被删除NodeDeleted、无事件None；

SpyState的取值范围是：连接断开Disconnected、不知道Unknown；该字段为未来保留扩展数值，目前可不处理；

**技术实现**

需要在每个节点中允许设置多个观察点，如果节点事件发生了，需要分析当前节点和上级节点中是否存在需要参与观察的Watcher对象，如果存在需要立即进行回调。考虑到回调的执行线程结构，要求回调线程为内部独立的运行线程，不能使用主动调用方法的运行线程。