MICROSSB: A LIGHTWEIGHT FRAMEWORK FOR ON-LINE DISTRIBUTED APPLICATION BASED ON SOFT SYSTEM BUS

Jian Xiao, Jizhou Sun, Gang Li, Chun Li, Sen Li
Department of Software Engineering, TianJin University, Tian Jin, China

Jingde Cheng
Department of Information and Computer Sciences, Saitama University, Saitama, Japan

Keywords: Persistent Computing, Soft system bus, Software development methodology, Middleware, On-line distributed application.

Abstract: Software development based on Soft System Bus (SSB) is a novel approach to Software Engineering. From the viewpoint of SSB, this paper presents a lightweight framework for developing on-line distributed applications, called MicroSSB. The framework partly implements the core functions of SSB-based system, including communication channel, data-instruction station, message exchange, security check and dynamic component management etc. The paper also proposes a guideline for using MicroSSB. By using MicroSSB, the designers and developers of distributed applications can focus on the core of their product instead of struggling with the low-level distributed programming. As case studies, the paper also shows two real applications based on MicroSSB: an experimental collaborative decision making system for air traffic flow control and a marine emergency commanding system.

1 INTRODUCTION

Modern society is more and more dependent on various on-line distributed systems such as air/train traffic control systems, emergency commanding systems, various collaborative systems and process control systems etc, and therefore how to design, develop and maintain these large-scale on-line distributed systems has become a very important issue in modern software engineering. Most modern on-line distributed applications have some essential requirements:

ER1: high availability and reliability
ER2: robust transmission mechanism
ER3: unified but flexible message exchange
ER4: loose component coupling
ER5: scalability for both small and large scale

Though many large distributed systems are developed by putting some traditional middleware solutions together, the main problem is that there is not a unified methodology on how to use traditional middlewares together, in other words, most traditional middlewares cannot severely satisfy the above requirements and simple assembly of the middlewares could not ensure persistent availability.

On the other hand, software development based on Soft System Bus (SSB) is a novel approach to Software Engineering, and it provides a new methodology for design, development and maintenance of distributed systems (J. Cheng, 2005). A system built using this methodology is called Soft System Bus Based System (SSBBS).

SSB-based system (Fig.1) consists of a number of components and one or more SSBs. The components are connected to the SSB. An SSB is a communication channel used to provide hardware and platform independent middleware support to the components. It conveys the data/instructions from component to component, provides language independent unified interface to the components and preserves the data/instructions if the destination component is not connected to the SSB.

There are two types of components in an SSBBS: one or more general purpose permanent Control Components (CCs) and some application specific Functional Components (FCs). Based on runtime information, the CCs record, measure, monitor and control the FCs in some way. On the other hand, the

Xiao J., Sun J., Li G., Li C., Li S. and Cheng J..
MICROSSB: A LIGHTWEIGHT FRAMEWORK FOR ON-LINE DISTRIBUTED APPLICATION BASED ON SOFT SYSTEM BUS.
DOI: 10.5220/0003421502110217
In *Proceedings of the 6th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2011), pages 211-217
ISBN: 978-989-8425-57-7
Copyright © 2011 SCITEPRESS (Science and Technology Publications, Lda.)

FCs, which provide functionalities to the application, are developed by the application developers. In an SSBBS, any two components are not allowed to communicate directly. They must use the unified interface of the SSB to interact with each other (J. Cheng, 2006).

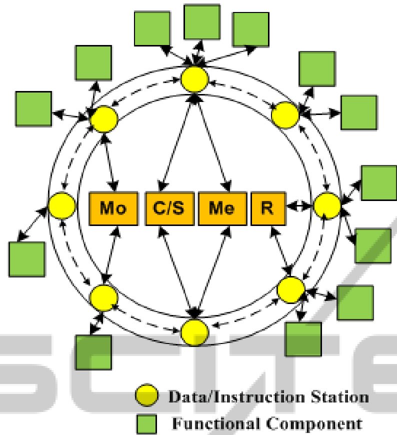

Figure 1: Circular SSB structure.

Our ultimate purpose is to build the general purpose part of SSBBSs, i.e., the SSB and CCs which is collectively called SSB package. The SSB package will be used to build large-scale long-lived distributed systems, and the application developers' responsibility is to develop the FCs and to attach them to the SSB (J. Cheng, 2007). Till now no SSB has been fully implemented, in this paper we will present a micro implementation of SSB package, called MicroSSB, which can provide core functions of SSBBSs with some limitations, as well as our experience in bring SSB approach to practice of one real application.

2 MICROSSB ARCHITECTURE

MicroSSB (Fig.2) provides a simple pragmatic approach for using SSB model, and it is designed to meet the general requirements of several different classes of on-line distributed application. Fig.2 shows the architecture of MicroSSB, all the critical control components and functional components are connected by a linear communication channel, and resident in several SSBNodes, a common run-time environment. There are two types of SSBNode, FSN (functional component node) and CSN (control component node), and the only difference between FSN and CSN is that the CSNs are invisible and inaccessible to any end-user.

SSBNode is the container of control components or functional components, and it controls the life-cycle of components. Only through SSBNode can the components connect to data/instruction stations for sending and receiving messages. There are four main parts within an SSBNode:

Communication Layer: provides common functions of message transmission and several configurable policies to support various transmission requirements.

Message Layer: defines a uniform format of message (data and instructions) based on xml, and provides a flexible mechanism for message processing and permission check.

Node Control Component: each SSBNode has only one node control component, designed for performing some management task, e.g., initializing the run time environment of this node and restarting dead functional components etc.

Functional Components: application-specific, and they may be redundant for high availability. Functional components within one SSBNode share the same run time environment and resources.

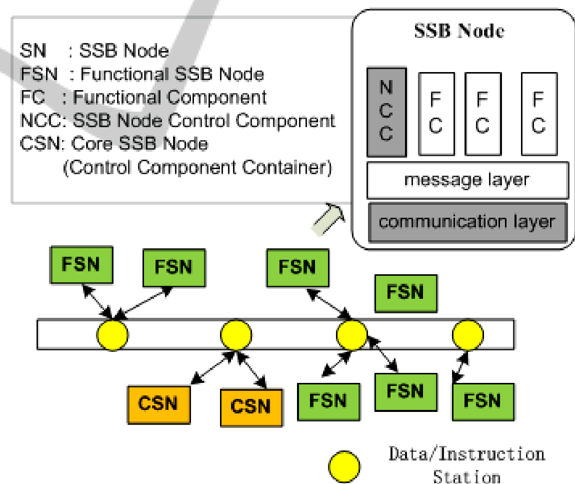

Figure 2: MicroSSB linear bidirectional structure.

2.1 Communication Layer

The implementation of communication channel is based on ActiveMQ (Apache, 2008), which has similar functions with SSB's data/instruction station. ActiveMQ's broker provides a series of excellent mechanisms for message preservation, re-transportation, security, failover, disaster recovery etc. Fig.3 shows the structure of communication layer. Data/instruction station's core part is MQ broker, which provides a fast and reliable transactional message preservation solution by

persisting messages in a transactional journal or a database. Multiple brokers can work in cluster mode that means when one station fails, another station could quickly takeover all the SSB nodes connected to the failure one. Transport channel can support several protocols and communication libraries, and furthermore developers can create their own interceptor for marshalling/unmarshalling messages in order to meet various messaging requirements. Message consumers and producers are used to receive and send messages; a listener is assigned to fetch incoming messages and forward them to the message layer (Fig.4).

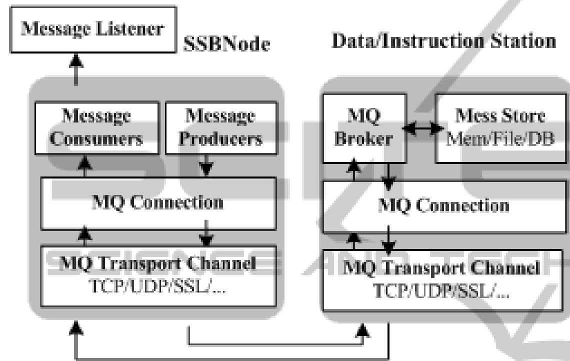

Figure 3: Communication layer structure.

The communication layer is high configurable, to establish a communication channel quickly, the developers almost needn't do any program coding except some configurations. For example, using the following configuration fragment, some communication channels can be automatically established when SSBNode starts.

```
<channel>
   <id>MicroSSB_CSN01</id>
   <station>
     tcp://localhost:61616
   </station>
   <publisher
     name="csn01_publisher"
     topic="instruction_topic" />
   <producer
     name="csn01_producer" queue="" />
   <consumer
     name="fsn_consumer"
     queue="csn01_queue"
     dispatcher="default"
     newThread="true"
     filter="SSB_Me_Info"/>
</channel>
```

The above configuration fragment comes from the configuration file of a CSN, and at the beginning, it defines a unique id for the SSBNode, and the `station` tag indicates the connection URL to the SSB station (MQ broker) which the SSBNode will connect to. The follow content defines a message publisher for publishing general instruction or data, and a message producer for sending message to any one SSBNode. Finally a message consumer is assigned for receiving the incoming messages, the `newThread` attribute indicates that a new thread will be created and dedicated to the consumer in order to improve the efficiency of message processing, and further more a filter is applied to insure that only the messages of system measure could be picked up for further processing.

2.2 Message Layer

From the view of distributed application, a message (data/instruction) represents an operation issued from message producer to message consumer(s), and the component which receives the message should know how to process it. From the view of software design, a message should include some basic attributes such as sender id, sending time, receiver id, receiving time, message id/name and body content etc. In high security environment some real-time check mechanisms must be provided to forbid illegal accesses before further processing. MicroSSB's message is defined in xml format.

```
<microSSB-message>
   <header>
     <id>the serial num</id>
     <name>
      unique name for this type message
     </name>
     <source>
      the source SSBNode
     </source>
     <destination>
       the destination SSBNode
     </destination>
     <replyTo>
       only for request-response model
     </replyTo>
     <sendTime>timestamp</sendTime>
     <recvTime>timestamp</recvTime>
   </header>
   <data>
      application-specific data
   </data>
</microSSB-message>
```

It is worth noting that though the message processing is scheduled and invoked by message layer but only the component (message consumer) can know when and how to process it, thus here needs an extensible mechanism for developers to

customize and implement the various application-specific messages and its processors. MicroSSB defines a message processor mapping mechanism for automatic message processing and security checking. Message processor contains a fragment of codes which can be invoked to deal with the corresponding messages. In order to meet the various needs of different applications, message layer uses various processor invocation methods and processors chain to support high extensibility. Fig.4 shows the structure of message layer and the main message processing flow:

1) Message layer registers message listeners in communication layer
2) When the message listener detects an incoming message it passes the message to a message dispatcher immediately
3) The dispatcher passes the message to an assigned security checker
   3.1) If the message is legal, it will be forward to the process controller and then a series of processors will be invoked to deal with the message in a predefined order.
   3.2) If the message is illegal, the dispatcher simply discards it.

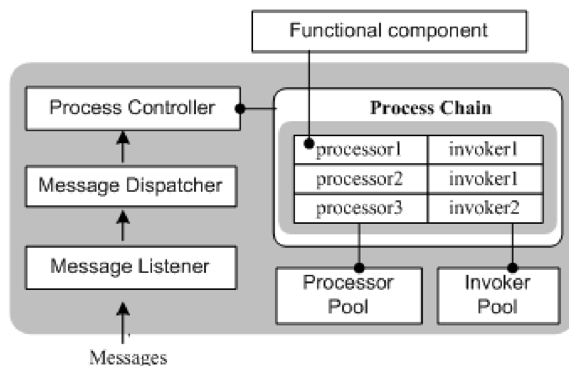

Figure 4: Message layer structure.

Processor Invoker is used to control when and how to invoke a message processor into running. Consider the following scenarios: in GUI applications it is a good practice to allocate a separate or a background thread to do time-consuming calculations asynchronously in order to keep the GUI system responsive to the end users. The invokers give the developers more flexibility in customizing message processor invocation policies and developing their own invokers.

Processor chain is useful when the same message need to be processed by a series of processors in a predefined flow. For example, a message which contains instruction from the central control component may need to be forward to all the functional components in an SSBNode and be processed sequentially by the processors of each functional component. Generally these processors assigned to process the same message may not know each other at all, and in a different deployment environment some processors may be removed from or added into the processor chain. The process controller works like a mini work flow engine, manage all the processor chains in the SSBNode and control all processing flows.

Run Time Security Check can be easily implemented in the MicroSSB's message driven architecture. Similar with the message processor mapping, there is a mapping list for messages and security checkers, and the checkers is scheduled by message layer before message processors invoked. Developers also can easily implement application-specific security checkers.

As shown in the following configuration fragment, message processing mechanisms and flows can be configured easily.

```
<messages>
    <message
        name="LoginResponse"
        processor="LoginProcessor"
        invoker="synInvoker"/>
</messages>
<processors>
    <processor
        name="LoginProcessor"
        class="microssb.LoginProcessor>
        <chain
          name="success"
          forward="ClientInitProcessor/>
        <chain
          name="failure"
          forward="ErrorProcessor"/>
     </processor>
</processors>
```

For performance reasons, the processor pool and invoker pool are used for reducing the overload of frequently creating and destroying objects and the security check result can also be cached to avoid some performance lost by frequent security check.

2.3 Component Design

According to the architecture of MicroSSB, the components in one SSBNode share common running environment, communication layer and message layer. MicroSSB defines a unified interface for all the central control components and functional components, including most common operations for dynamic component management, e.g., loading,

initializing, reactivating, suspending, stopping and unloading component.

The central control components can be regarded as the "heart" and/or "brain" of the system, and should never die, that is persistently continuous functioning. All the functional components are measured, recorded, monitored and controlled by the central control components. There are three major control components in MicroSSB (Fig.1, Fig.2):

1) System monitor: collects and display the run time status data, and can be regarded as a simple combination of the central measurer (Mo) and the central monitor (Me).
2) Component manager: manages the whole life cycle of all functional components, partially implements the controller/scheduler(C/S).
3) Data manager: partially implements the central recorder(R), provides the common interfaces for caching and storing data on run time. It plays a critical role in failover and disaster recovery.

The functional components are application-specific, and according to SSB's architecture they can be redundant, and each SSBNode can run same or different components set. Because most common functions are provided by MicroSSB, the design and development of functional components become very simple, and developers just need to implement core functions of their product. Message transmitting, processing and security check rules can be easily defined in configuration files and automatically scheduled by MicroSSB.

3 MICROSSB APPLICATONS

3.1 Development Guideline and Flow

MicroSSB can be regarded as a general-purpose package of SSB model for on-line distributed applications, and in order to construct a complete SSB-based system, the system designers and developers just need to follow some customizing steps and add functional components to it. Suggested steps as follows:

1) Divide functional components
2) Define message processing flows
3) Define content of the messages and their operations (processors)
4) Select/develop proper processor invokers
5) Select/develop proper security checkers
6) Analyze the communication requirement for all the different types of message, and assign proper message producers and consumers
7) Program and write configuration file
8) Assemble relative components into one or more SSBNodes
9) Deploy all the SSBNodes

3.2 Case Studies

A usage of on-line distributed system in air traffic control is to facilitate remote users (traffic flow control centers, airports, airline operation centers etc.) collaboratively to adjust the fight plans (Kan Chang et al, 2001) through multiple interaction methods, such as text, graphic, audio etc. Fig.5 shows the architecture of the collaborative air traffic flow management system (CAFM). Note that it is just an auxiliary system for air traffic control, not for real time flight controlling and scheduling.

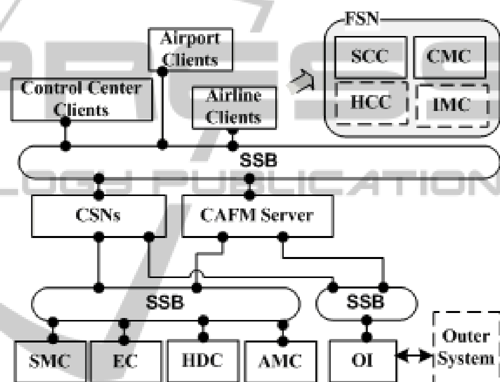

Figure 5: The architecture of CAFM.

The main flows of analysis and design of the system are as follows:

1) Functional components analysis: the system consists of three groups of functional components, i.e., user interface group (client), server group and outer interface group.
   1.1) Client group consists of session client component (SCC), collaborative modification component (CMC), historical case client component (HCC), and instant message component (IMC), and the last two (dashed line box) are optional. Each client has the same set of functional components, but the users may have different permission list, e.g., only the sponsor of a session can activate, suspend or stop the session (A session means the whole process of one collaborative task).
   1.2) Server group consists of a central server component (CAFM Server) to coordinate all the other functional components, session management components (SMC) to deal with current active session data, evaluation

component (EC) to evaluate the decision result, historical data component (HDC) to manage all the historical cases and account manage component (AMC) to deal with users account data.

1.3) Outer interface group consists of just one component (OI) to communicate with outer system for fetching the original fight plans and publishing the final adjusted plans.

2) The main processing flow analysis
   2.1) The sponsor activate a session
   2.2) All the users collaborative modify flight plans through sending and receiving flight plan updating messages
   2.3) The sponsor submit the final modifications
   2.4) The server calculating the evaluation result
   2.5) If evaluation is good, the sponsor publish the decision and close the session
   2.6) If evaluation is not good, repeat 2.2-2.4 until obtain a good result
3) Messages definition:
   Session message:
   create/activate/suspend/stop/delete
   Flight plan message:
   modify/synchronize/submit/evaluate
   Others:
   user online/offline,
   audio chat start/end, audio stream etc.
4) Message processing analysis: For all the messages except evaluation of flight plans can be processed in a short time, so the synchronous message process invoker is proper. While evaluation calculation is a time-consuming task, so an asynchronous message process invoker is proper.
5) Communication channels analysis: All the messages except audio stream can be transmitted in pure text (xml) format, thus for instant message component (IMC), a stream-oriented channel should be established temporarily between two SSBNodes when an audio chat is starting. And for all of the other components within same SSBNode, one shared producer plus one shared consumer is enough for sending and receiving text messages.
6) Implementation: Both the MicroSSB and the system are implemented by Java technology.

Deployment analysis: in order to provide high security, we use three separate SSBs rather than only one SSB to connect client group, server group and outer interface group.

There also is a marine emergency commanding system (MECS) based on MicroSSB and the goal of the system is to aid relevant departments to process accidents or rescue collaboratively on the sea. What the biggest different from the above system is that there are various functional clients and services, most of which must run continuously and even more make real-time responses, such as real time data collectors, situation monitor, alarm manager, command center etc. Fig.6 shows the architecture of the system.

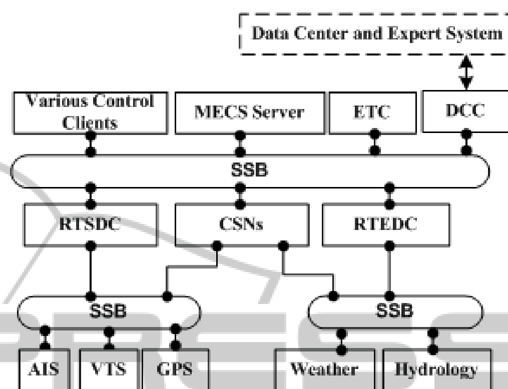

Figure 6: The architecture of MECS.

SCIENCE AND TECHNOLOGY PUBLICATIONS

The system consists of six groups of functional components, i.e.

1) Real time ship data component (RTSDC), collecting the ship information from various sources, such as AIS (automatic identification system), VTS (vessel traffic services) and GPS.
2) Real time environment data component (RTEDC), collecting weather and hydrology information.
3) Various control clients for receiving alarms, situation monitor, and resource schedule etc.
4) data center component (DCC) for integrating knowledge base and expert system
5) Server component for controlling the whole flow of accident processing or rescue.
6) Electronic training component (ETC)

4 RELATED WORKS

There are some excellent remote control infrastructures for distributed application monitoring and management, such as Plush (Jeannie Albrecht et al, 2007), group communication libraries for high available message transmission over large-scale network, such as JGroups (Bela Ban, 2007) and dynamic module frameworks for adding, removing and replacing any part of a system in runtime, such as OSGi (Andre L.C. Tavares, Marco Tulio Valente, 2008).

Selim et al.(2006) presented a fully comparative

study between the SSB and different types of traditional middlewares, e.g., Request/Response, Message Oriented, Publish/Subscribe middlewares, etc. and his paper shows that although existing middlewares have some characteristics that are common in an SSB too, they lack some features which are unique and essential for an SSB, especially in component coupling, dynamic connectivity, data preservation, availability and reliability, and unified interface etc.

The Open Services Gateway Initiative (OSGi) is a framework that supports the implementation of component-based, service-oriented applications in Java. The framework manages the life-cycle of modules and provides means to publish and search for services. Moreover, it supports the dynamic install and uninstall of bundles like an SSBBS. R-OSGi extended the centralized, industry-standard OSGi specification to support distributed module management (J. S. Rellermeyer et al, 2007) and further DR-OSGi enhanced distributed component application with the ability to continue executing in the presence of network volatility (Young-Woo Kwon et al, 2009). However, SSBBS can also provide fault tolerance to the components, and it can ensure the components continuously functioning through preserving the program states of the running components attached to it, but OSGi is just simply to restart the failure one without restore its states. Furthermore, SSBBS provides a unified mechanism to measure, monitor and control the Functional Components automatically with minimum manual intervention.

5 CONCLUSIONS

We have presented MicroSSB as a lightweight framework of SSB package, and MicroSSB allows application developers to build on-line distributed applications using SSB methodology, and it provides most basic functions of SSB, including communication channel, data-instruction station, message exchange, security check and dynamic component management etc. We also proposed a design and development flow for using MicroSSB.

As a case study, we have presented the whole process of developing an experimental collaborative decision making system for air traffic flow control based on MicroSSB step by step, and our experience has shown that by using it the application developers can focus on the core of their products and just develop the functional components and attach them to the SSB, thus with minimum effort a distributed system can be built.

However, at present the Soft System Bus technology is not completely developed, and the current MicroSSB, as one step towards bringing SSB approach into practice, is just a prototype implementation only including some basic features of SSBBS, especially lacking a real sense of Control Components group of SSB, thus at current stage it is not suitable for some critical systems and very large-scale applications.

REFERENCES

J. Cheng, 2005. ‘Connecting Components with Soft System Buses: A New Methodology for Design, Development, and Maintenance of Reconfigurable, Ubiquitous, and Persistent Reactive Systems’, *Proceedings of the 19th International Conference on Advanced Information Networking and Applications,* vol. 1, pp. 667-672.

J. Cheng, 2006. ‘Persistent Computing Systems as Continuously Available, Reliable, and Secure Systems’, *Proceedings of the First International Conference on Availability, Reliability and Security*, pp. 631-638.

J. Cheng, 2007. ‘Persistent Computing Systems Based on Soft System Buses as an Infrastructure of Ubiquitous Computing and Intelligence’*, Journal of Ubiquitous Computing and Intelligence, vol. 1*, no. 1, pp. 35-41.

M. R. Selim, T. Endo, Y. Goto, and J. Cheng, 2006. ‘A Comparative Study between Soft System Bus and Traditional Middlewares’*, LNCS Vol.4278*, pp. 1264-1273

Jeannie Albrecht, Ryan Braud, Darren Dao, Nikolay Topilski, Christopher Tuttle, Alex C. Snoeren, Amin Vahdat, 2007. ‘Remote control: distributed application configuration, management, and visualization with Plush’. *Proceedings of the 21st conference on Large Installation System Administration Conference*, pp.1-19.

Andre L.C. Tavares, Marco Tulio Valente, 2008. ‘A Gentle Introduction to OSGi’. ACM SIGSOFT Software Engineering Notes, vol.33, no.5.

J. S. Rellermeyer, G. Alonso, and T. Roscoe. 2007. ‘R-OSGi: Distributed applications through software modularization’. *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware*, pp.1-20.

Young-Woo Kwon, Eli Tilevich, Taweesup Apiwattanapong, 2009. ‘DR-OSGi: Hardening Distributed Components with Network Volatility Resiliency’. *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware, LNCS vol.5896* pp.373-392

Kan Chang, Ken Howard, Rick Oiesen, Lara Shisler, Midori Tanino, and Michael C. Wambsganss, 2001. ‘Enhancements to the FAA Ground-Delay Program Under Collaborative Decision Making’. Interfaces, vol.31, no.1, pp.57-76

Apache, ActiveMQ, 2008), <http://activemq.apache.org>

Bela Ban, JGroups - A Toolkit for Reliable Multicast Communication (2007), http://www.jgroups.org/javagroupsnew/docs/index.htmL