合肥學院
HEFEI UNIVERSITY

Distributed Computing

Homework 4: RMI-based Chat

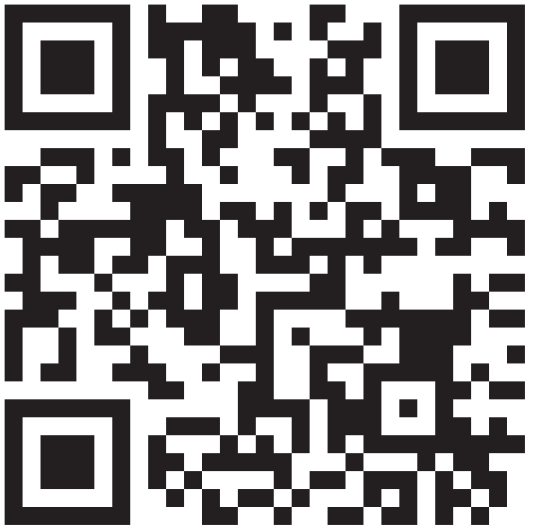
Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://www.it-weise.de

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

Outline



Overview

- Use `Java` RMI [1–3]
- Implement a RMI server for chat

Events and Model-View-Controler

- Many UIs are designed according to the Model-View-Controller (or Model-Delegate) pattern [4–7]

Events and Model-View-Controler

- Many UIs are designed according to the Model-View-Controller (or Model-Delegate) pattern [4–7]
- Some underlying process or model manages a state

Events and Model-View-Controler

- Many UIs are designed according to the Model-View-Controller (or Model-Delegate) pattern [4–7]
- Some underlying process or model manages a state
- Views display the state of the model

Events and Model-View-Controler

- Many UIs are designed according to the Model-View-Controller (or Model-Delegate) pattern [4–7]
- Some underlying process or model manages a state
- Views display the state of the model
- Views are updated by receiving events

Events and Model-View-Controler

- Many UIs are designed according to the Model-View-Controller (or Model-Delegate) pattern [4–7]
- Some underlying process or model manages a state
- Views display the state of the model
- Views are updated by receiving events
- Events usually encapsulated in objects

Events and Model-View-Controler

- Many UIs are designed according to the Model-View-Controller (or Model-Delegate) pattern [4–7]
- Some underlying process or model manages a state
- Views display the state of the model
- Views are updated by receiving events
- Events usually encapsulated in objects
- Views implement certain *callback* interfaces, i.e., have methods that can process events

Events and Model-View-Controler

- Many UIs are designed according to the Model-View-Controller (or Model-Delegate) pattern [4–7]
- Some underlying process or model manages a state
- Views display the state of the model
- Views are updated by receiving events
- Events usually encapsulated in objects
- Views implement certain *callback* interfaces, i.e., have methods that can process events
- Java is full of this, just google for `ActionListener`, `MouseListener`, `TableModelListener`, `VetoableChangeListener`, ...

RMI Callbacks

- RMI is a technology for building distributed applications

RMI Callbacks

- RMI is a technology for building distributed applications
- A server object is registered in a naming service (`java.rmi.registry.Registry`)

RMI Callbacks

- RMI is a technology for building distributed applications
- A server object is registered in a naming service (`java.rmi.registry.Registry`)
- Object functionality is provided via interfaces

RMI Callbacks

- RMI is a technology for building distributed applications
- A server object is registered in a naming service (`java.rmi.registry.Registry`)
- Object functionality is provided via interfaces
- The server side implements the functionality with a 'real' class

RMI Callbacks

- RMI is a technology for building distributed applications
- A server object is registered in a naming service (`java.rmi.registry.Registry`)
- Object functionality is provided via interfaces
- The server side implements the functionality with a 'real' class
- The client side automatically creates a proxy instance of the object that forwards all calls to the server

RMI Callbacks

- RMI is a technology for building distributed applications
- A server object is registered in a naming service (`java.rmi.registry.Registry`)
- Object functionality is provided via interfaces
- The server side implements the functionality with a 'real' class
- The client side automatically creates a proxy instance of the object that forwards all calls to the server
- In Java RMI, we can also create *callback* objects

RMI Callbacks

- RMI is a technology for building distributed applications
- A server object is registered in a naming service (`java.rmi.registry.Registry`)
- Object functionality is provided via interfaces
- The server side implements the functionality with a 'real' class
- The client side automatically creates a proxy instance of the object that forwards all calls to the server
- In Java RMI, we can also create *callback* objects
- These inherit from `java.rmi.server.UnicastRemoteObject` and implement a callback interface derived from `java.rmi.Remote`

RMI Callbacks

- RMI is a technology for building distributed applications
- A server object is registered in a naming service (`java.rmi.registry.Registry`)
- Object functionality is provided via interfaces
- The server side implements the functionality with a 'real' class
- The client side automatically creates a proxy instance of the object that forwards all calls to the server
- In Java RMI, we can also create *callback* objects
- These inherit from `java.rmi.server.UnicastRemoteObject` and implement a callback interface derived from `java.rmi.Remote`
- They can be passed to an RMI server without needing to be registered in a registry

RMI Callbacks

- RMI is a technology for building distributed applications
- A server object is registered in a naming service (`java.rmi.registry.Registry`)
- Object functionality is provided via interfaces
- The server side implements the functionality with a 'real' class
- The client side automatically creates a proxy instance of the object that forwards all calls to the server
- In Java RMI, we can also create *callback* objects
- These inherit from `java.rmi.server.UnicastRemoteObject` and implement a callback interface derived from `java.rmi.Remote`
- They can be passed to an RMI server without needing to be registered in a registry
- $\longrightarrow$ We can have *distributed* callbacks

RMI Callbacks

- RMI is a technology for building distributed applications
- A server object is registered in a naming service (`java.rmi.registry.Registry`)
- Object functionality is provided via interfaces
- The server side implements the functionality with a 'real' class
- The client side automatically creates a proxy instance of the object that forwards all calls to the server
- In Java RMI, we can also create *callback* objects
- These inherit from `java.rmi.server.UnicastRemoteObject` and implement a callback interface derived from `java.rmi.Remote`
- They can be passed to an RMI server without needing to be registered in a registry
- $\longrightarrow$ We can have *distributed* callbacks, and therefore: distributed model-view-controler application structures

Callbacks for Chat!

- With this, we can implement a chat system!

Callbacks for Chat!

- With this, we can implement a chat system!
- Server

Callbacks for Chat!

- With this, we can implement a chat system!
- Server:
  - central instance distributing chat events

Callbacks for Chat!

- With this, we can implement a chat system!
- Server:
  - central instance distributing chat events
  - provides methods to login, log out, and to send messages

Callbacks for Chat!

- With this, we can implement a chat system!
- Server:
  - central instance distributing chat events
  - provides methods to login, log out, and to send messages
  - maintains a list of logged in clients

Callbacks for Chat!

- With this, we can implement a chat system!
- Server:
  - central instance distributing chat events
  - provides methods to login, log out, and to send messages
  - maintains a list of logged in clients
- Client

Callbacks for Chat!

- With this, we can implement a chat system!
- Server:
  - central instance distributing chat events
  - provides methods to login, log out, and to send messages
  - maintains a list of logged in clients
- Client
  - uses server to log in, log out, and to send message

Callbacks for Chat!

- With this, we can implement a chat system!
- Server:
  - central instance distributing chat events
  - provides methods to login, log out, and to send messages
  - maintains a list of logged in clients and corresponding callback interfaces
- Client
  - uses server to log in, log out, and to send message
  - registeres a callback interface at log in

Callbacks for Chat!

- With this, we can implement a chat system!
- Server:
  - central instance distributing chat events
  - provides methods to login, log out, and to send messages
  - maintains a list of logged in clients and corresponding callback interfaces
- Client
  - uses server to log in, log out, and to send message
  - registeres a callback interface at log in
  - client receives events from server and updates ui

Chat Program

- For this homework, the following things have been done

Chat Program

- For this homework, the following things have been done:
  - The functionality of a chat server has been defined in interface `IChatServer`

Chat Program

- For this homework, the following things have been done:
  - The functionality of a chat server has been defined in interface `IChatServer`
  - The client callback interface has been defined in `IChatClient`

Chat Program

- For this homework, the following things have been done:
  - The functionality of a chat server has been defined in interface `IChatServer`
  - The client callback interface has been defined in `IChatClient`
  - A chat client GUI that can connect to a chat server implementation has fully been implemented in `ChatClient`

Chat Program

- For this homework, the following things have been done:
  - The functionality of a chat server has been defined in interface `IChatServer`
  - The client callback interface has been defined in `IChatClient`
  - A chat client GUI that can connect to a chat server implementation has fully been implemented in `ChatClient`
  - An event class for chat events has been defined in `ChatEvent`

Chat Program

- For this homework, the following things have been done:
  - The functionality of a chat server has been defined in interface `IChatServer`
  - The client callback interface has been defined in `IChatClient`
  - A chat client GUI that can connect to a chat server implementation has fully been implemented in `ChatClient`
  - An event class for chat events has been defined in `ChatEvent`
  - A skeleton class for the chat server, but no functionality is provided in `ChatServer`

Chat Program

- For this homework, the following things have been done:
  - The functionality of a chat server has been defined in interface `IChatServer`
  - The client callback interface has been defined in `IChatClient`
  - A chat client GUI that can connect to a chat server implementation has fully been implemented in `ChatClient`
  - An event class for chat events has been defined in `ChatEvent`
  - A skeleton class for the chat server, but no functionality is provided in `ChatServer`
- What remains to be done is

Chat Program

- For this homework, the following things have been done:
  - The functionality of a chat server has been defined in interface `IChatServer`
  - The client callback interface has been defined in `IChatClient`
  - A chat client GUI that can connect to a chat server implementation has fully been implemented in `ChatClient`
  - An event class for chat events has been defined in `ChatEvent`
  - A skeleton class for the chat server, but no functionality is provided in `ChatServer`
- What remains to be done is:
  - Fill the chat server `ChatServer` with life!

Chat Program

- For this homework, the following things have been done:
  - The functionality of a chat server has been defined in interface `IChatServer`
  - The client callback interface has been defined in `IChatClient`
  - A chat client GUI that can connect to a chat server implementation has fully been implemented in `ChatClient`
  - An event class for chat events has been defined in `ChatEvent`
  - A skeleton class for the chat server, but no functionality is provided in `ChatServer`
- What remains to be done is:
  - Fill the chat server `ChatServer` with life!
  - Follow the comments in `ChatServer` and `IChatServer`!

Chat Program

- For this homework, the following things have been done:
  - The functionality of a chat server has been defined in interface `IChatServer`
  - The client callback interface has been defined in `IChatClient`
  - A chat client GUI that can connect to a chat server implementation has fully been implemented in `ChatClient`
  - An event class for chat events has been defined in `ChatEvent`
  - A skeleton class for the chat server, but no functionality is provided in `ChatServer`
- What remains to be done is:
  - Fill the chat server `ChatServer` with life!
  - Follow the comments in `ChatServer` and `IChatServer`!
  - Start the server

Chat Program

- For this homework, the following things have been done:
  - The functionality of a chat server has been defined in interface `IChatServer`
  - The client callback interface has been defined in `IChatClient`
  - A chat client GUI that can connect to a chat server implementation has fully been implemented in `ChatClient`
  - An event class for chat events has been defined in `ChatEvent`
  - A skeleton class for the chat server, but no functionality is provided in `ChatServer`
- What remains to be done is:
  - Fill the chat server `ChatServer` with life!
  - Follow the comments in `ChatServer` and `IChatServer`!
  - Start the server, connect some clients to the server

Chat Program

- For this homework, the following things have been done:
  - The functionality of a chat server has been defined in interface `IChatServer`
  - The client callback interface has been defined in `IChatClient`
  - A chat client GUI that can connect to a chat server implementation has fully been implemented in `ChatClient`
  - An event class for chat events has been defined in `ChatEvent`
  - A skeleton class for the chat server, but no functionality is provided in `ChatServer`
- What remains to be done is:
  - Fill the chat server `ChatServer` with life!
  - Follow the comments in `ChatServer` and `IChatServer`!
  - Start the server, connect some clients to the server, chat...

Chat Program

- For this homework, the following things have been done:
  - The functionality of a chat server has been defined in interface `IChatServer`
  - The client callback interface has been defined in `IChatClient`
  - A chat client GUI that can connect to a chat server implementation has fully been implemented in `ChatClient`
  - An event class for chat events has been defined in `ChatEvent`
  - A skeleton class for the chat server, but no functionality is provided in `ChatServer`
- What remains to be done is:
  - Fill the chat server `ChatServer` with life!
  - Follow the comments in `ChatServer` and `IChatServer`!
  - Start the server, connect some clients to the server, chat...
- Send me your complete Eclipse project folder packed as zip archive with name `hw04_[your_student_id].zip` (where `[your_student_id]` is replaced with your student id)

谢谢

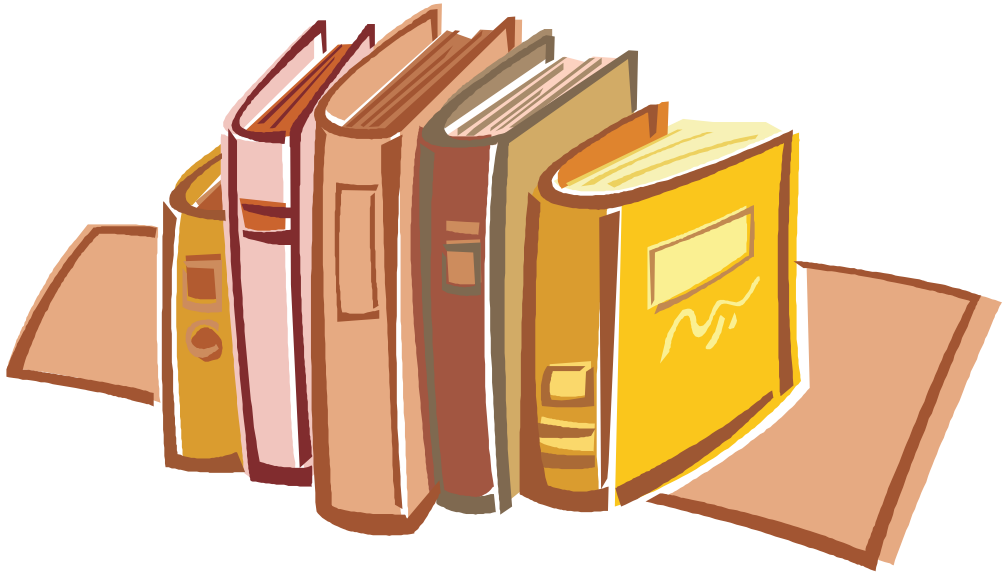
Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://www.it-weise.de

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog

Bibliography



Bibliography I

1. Ann Wollrath, Roger Riggs, and Jim Waldo. A distributed object model for the java system. In Douglas C. Schmidt and Doug Lea, editors, *Proceedings of the USENIX 1996 Conference on Object-Oriented Technologies (COOTS)*, Toronto, ON, Canada, 1996. URL `http://pdos.csail.mit.edu/6.824/papers/waldo-rmi.pdf`.
2. William Grosso. *Java RMI*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2011. ISBN 1449315356 and 9781449315351. URL `http://books.google.de/books?id=TeK5uL2dWwQC`.
3. Josef Stepisnik. *Distributed Object-Oriented Architectures: Sockets, Java RMI and CORBA*. Hamburg, Germany: Diplomica Verlag GmbH, 2007. ISBN 3836650339 and 9783836650335. URL `http://books.google.de/books?id=qNGTzYdJt18C`.
4. William Crawford and Jonathan Kaplan. *J2EE Design Patterns*. Patterns of the Real World. Sebastopol, CA, USA: O'Reilly Media, Inc., 2003. ISBN 0596004273 and 9780596004279. URL `http://books.google.de/books?id=x-7_W0P9KGsC`.
5. Barbara Purchase and Steven Holzner. *Design Patterns for Dummies*. For Dummies Computers Series. New York, NY, USA: John Wiley & Sons Ltd., 2006. ISBN 0471798541 and 9780471798545. URL `http://books.google.de/books?id=6rZbzSbKsQMC`.
6. Robert T. Monroe, Andrew Kompanek, Ralph Melton, and David Garlan. Architectural styles, design patterns, and objects. *IEEE Software Magazine*, 14(1):43–52, January–February 1997. doi: 10.1109/52.566427. URL `http://www.cs.cmu.edu/~able/publications/ObjPatternsArch-ieee97/`.
7. Frank Buschmann, Regine Meunier, Hand Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. New York, NY, USA: John Wiley & Sons Ltd., August 8, 1996. ISBN 047-1958-697 and 978-0471958697. URL `http://books.google.de/books?id=0kUFZDuqvmEC`.