# 系统建模

仿真模型选取有刷直流电机。

## 直流电机的基本结构

直流电机可概括地分为静止和转动两大部分。静止部分称为定子；转动部分称为转子。定、 转子之间由空气隙分开。

## 直流电机的基本工作原理

在A、B电刷上接入直流电源U，则该电机模型为电动机模型。转子线圈中流过电流时，受电磁力的作用而产生的电磁转矩可用下式表示：



式中，T为电磁转矩(N·m）；为电枢电流（A)；为与电机结构有关的常数，称为转矩常数，。

## 直流电机的数学模型

直流电动机中，有电机转速变化的机械过程和电量变化的电磁过程这两个过程,按照实际情，在此模型中，我们认为它们同时发生。据此，从电枢电压的平衡方程和转矩平衡方程出发，建立其状态方程及传递函数关系。

电枢电压平衡方程为： 

转矩平衡方程为： 

其中

 电枢电压(V)

 电枢反电势 (V)

 电枢电流 (A)

 电枢电感(H)

 电枢电阻(Q)

 每极磁通(Wb)

 转子转速(r／min)

、 直流电机电势常数和转矩常数，Ce =PN／(60a)、Cm = PN／(2πa)，其中P为电机极对数，N为电枢总导体数，a 为并联支路数

M、 电磁转矩和负载转矩 (N•m)

J 转速惯量(N•m•s•min／r)

由



得到



利用拉氏变换得到





其中

 电枢回路放大倍数

 电枢回路电磁时间常数(s)

令

 机电时间常数(s)

建立精确模型时要考虑到机电(Tm)和电磁(Ta)的过渡过程，反映在传递函数中，即它形成了一个有反馈的闭环二阶系统，由此系统可得电机精确模型的转速与电枢电压的传递函数:

 （1）

## 直流电机控制器设计（最小拍控制器）

令









代入（1）即可得：



对上式进行z变换，并加上零阶保持器，采样时间设为得到离散化的模拟对象：



化为zpk的形式：

 (2)

设计一个最小拍控制器，由上式可见只有一个不稳定的，按来设计，因此得到

 (3)

设控制器输入、输出分别为  、，模拟对象输入输出为 、

由（2）（3）可以得到模拟对象和控制器的离散差分方程：



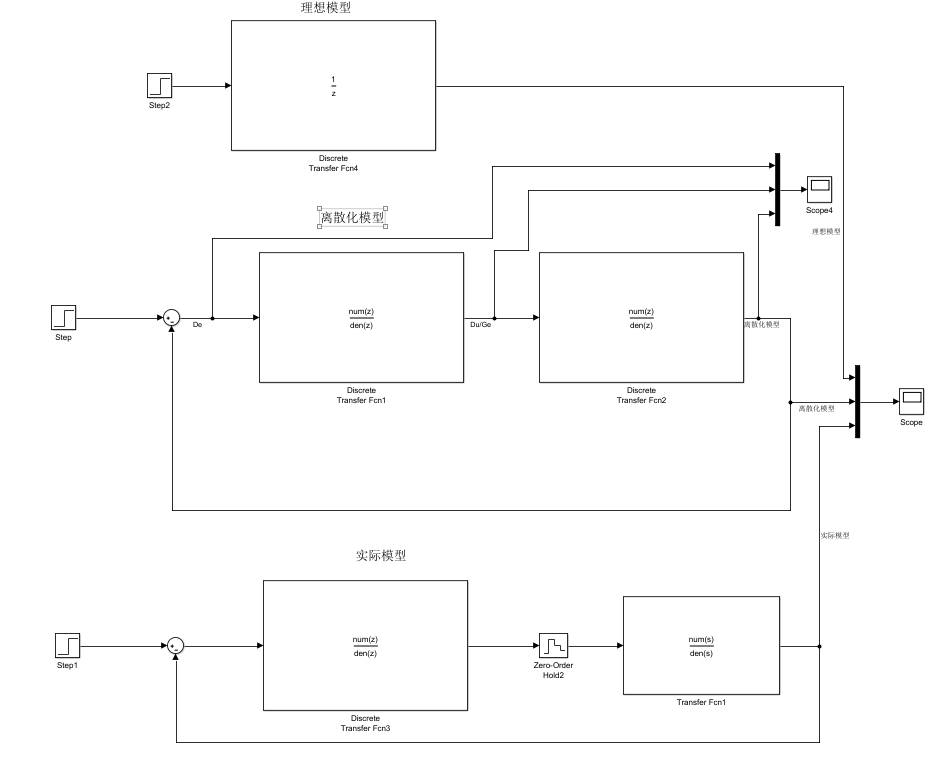


# 模型及控制器仿真验证

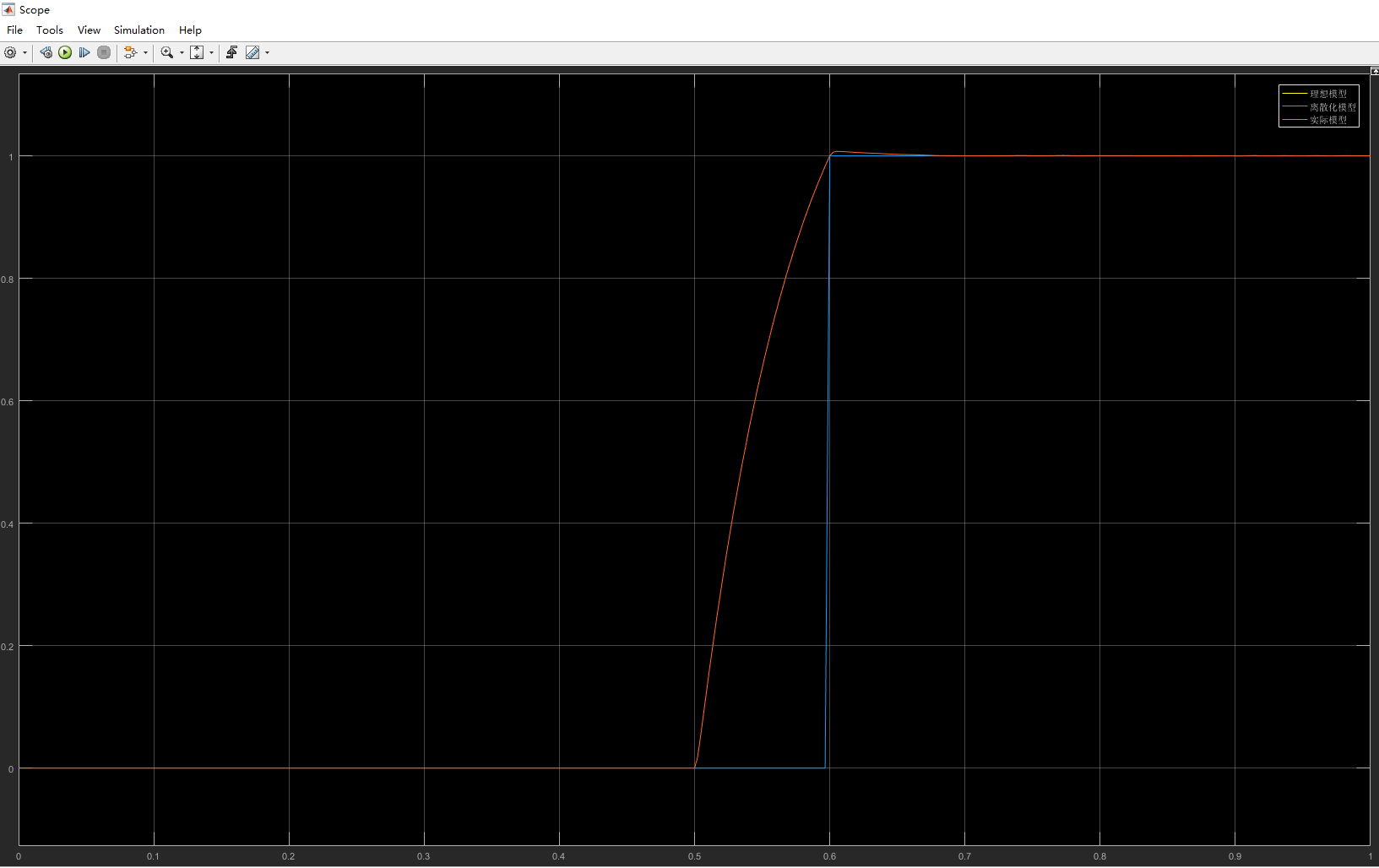
## Simulink仿真

构造模型并进行离散化后，为了方便先到simulink进行仿真验证结果是否正确。

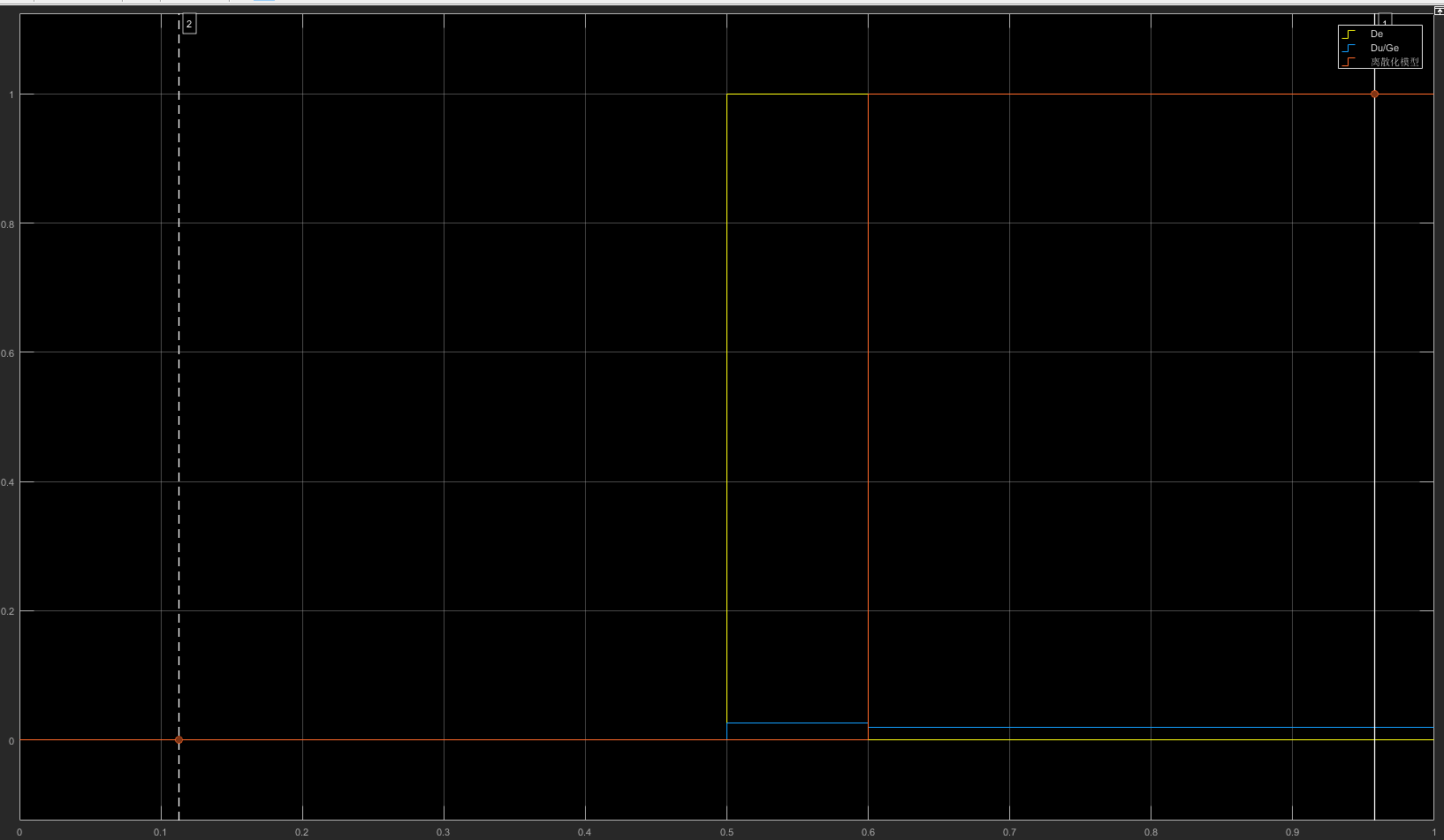
根据系统框图构造simulink仿真模型如下图



在0.5s加入阶跃输入，Ts=0.1s，得到如下电机输出波形，可以看到理想模型跟离散化模型输出是完全重叠一致的，而实际模型在时域上是缓慢上升的。但三者都是在第二拍就能达到稳定了。可见仿真结果是完全正确的，控制器设计成功。



为方便计算仿真，对控制器输入输出及模拟对象输入输出进行仿真得到：



## Matlab脚本文件计算仿真

Simulink仿真结果正确以后，再对系统进行实际计算仿真，这一步同样是借助Matlab仿真。

给定一个R=1的稳定阶跃输入，代入

 (4)

 (5)

由于在实际模型中，控制器的输入实际上使用的是上一拍的对象输出，因此有：

 (6)

 (7)

代入(4)(5)(6)(7)，R(k)取阶跃输入，得到系统输出值Gu为

Gu

0

0

0.9999

1.0016

1.0004

1.0001

1.0000

1.0000

Du

0

0

0.0260

0.0192

0.0192

0.0192

0.0192

0.0192

可见系统在第二拍即可达到稳定，跟设计的预想输出效果一致。Matlab计算仿真结果正确。

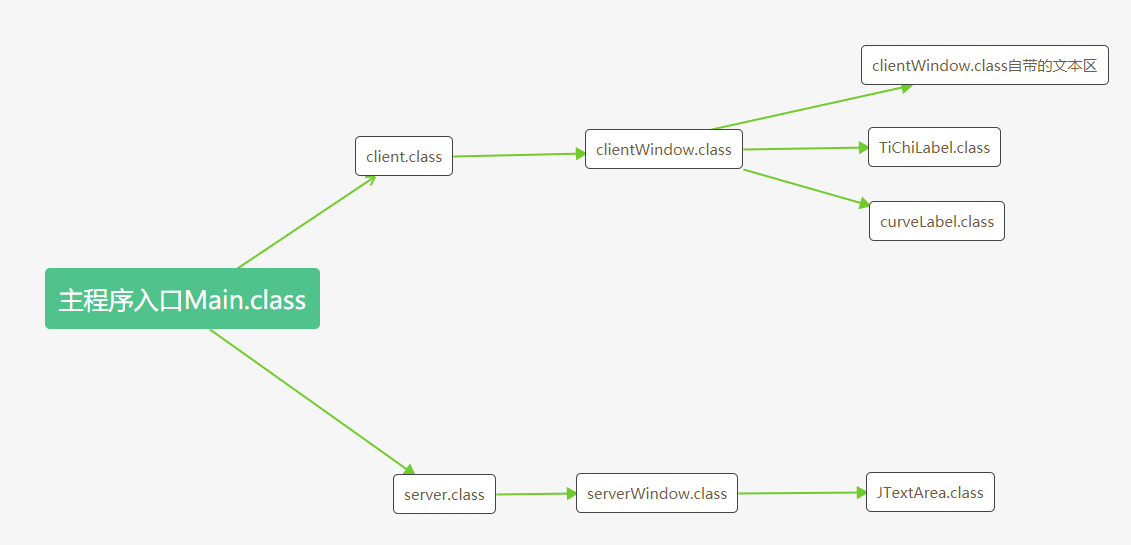
# 编写程序

## 程序总体框架

程序有一个入口程序，入口程序可以通过server.class启动服务器，也可以通过client.class启动一个新模拟对象。

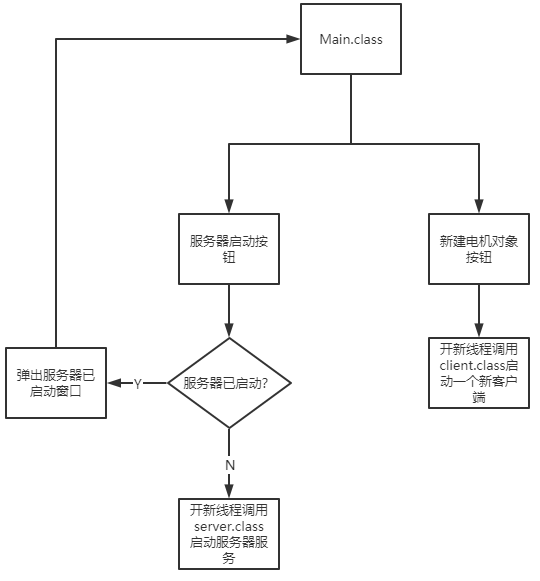
server.class启动时，先新建一个serverWindow.class对象来作为服务器显示窗口，然后会监听端口

client.class启动时，新建一个clientWindow.class对象来作为客户端窗口，然后尝试建立socket连接。ClientWindow.class使用了TiChiLabel.class来维护一个转动的太极图案以显示动画效果，还使用curveLabel.class来维护模拟对象的输入输出的曲线图显示。

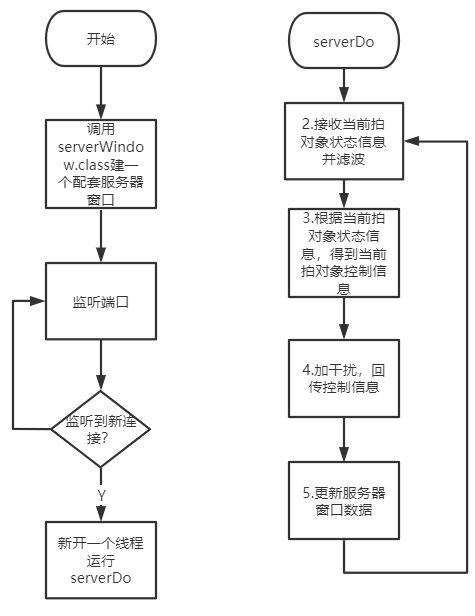


## 程序各部分功能框图

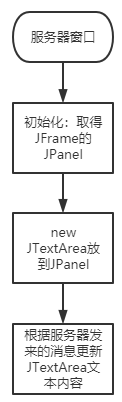
### 主程序功能



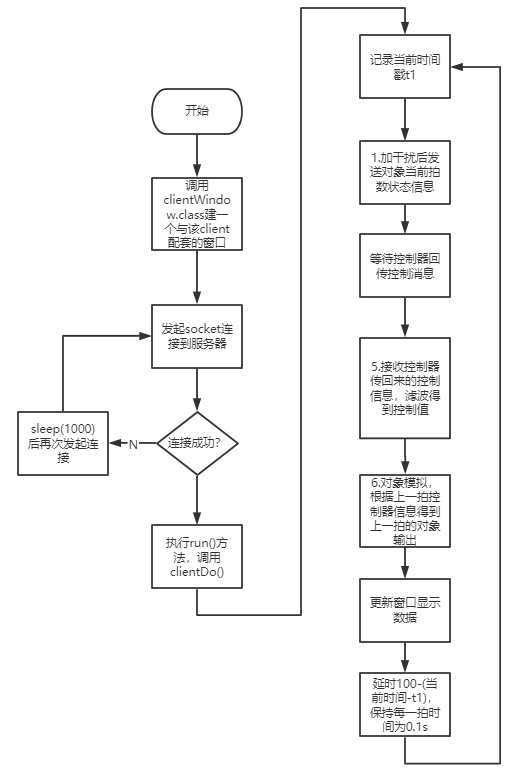
### 服务端功能

****

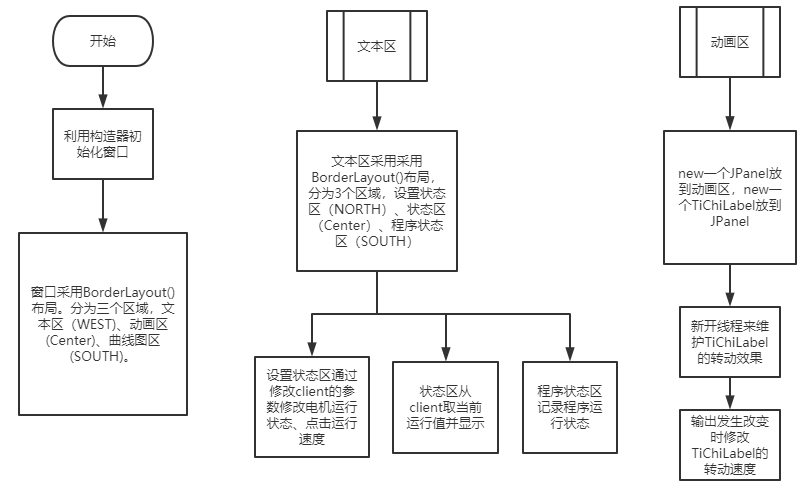
### 服务器窗口功能

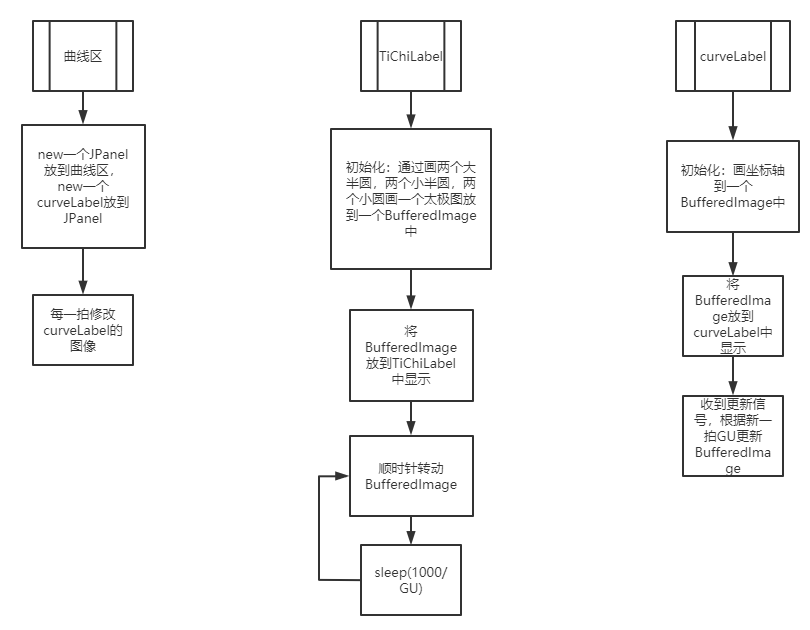
****

### 客户端功能

****

### 客户端窗口功能

****

****

## 底层技术及关键代码

本程序用到的主要技术有：连续控制系统离散化、最小拍控制器设计、z传递函数转化为差分方程、socket通信、swing图形显示、多线程技术、异常处理技术。

### Socket通信

socket通信主要步骤是：服务器监听，客户端发起连接，建立连接，客户端发出消息，服务器接收消息，服务器处理消息，服务器回传消息，客户端接收消息。

**服务器监听：**

1. **while** (**true**) {
2. Socket socket = serverSocket.accept(); //监听端口，如果没有连接则堵塞
3. **new** Thread(**new** serverDo(socket, objNum, serverWindow)).start();   //收到一个新连接后就开启一个新线程去处理
4. objNum++;
5. }

**客户端发起连接：**

首先会尝试发起连接，捕捉到连接异常时会调用reconnect延迟1s后再次发起连接请求

1. **try** {
2. clientWindow.setTextProgramer("正在连接服务器\n");
3. **return** **new** Socket(host, port);
4. } **catch** (IOException e) {
5. **return** reConnect(host, port);
6. }

**客户端发送消息：**

发送的消息是一个对象，因此用objectOutputStream来发送，用objectInputStream来接收。

1. //1.发送对象当前拍数状态信息
2. //加干扰，发送3次信号
3. mM.setGUNow(mM.getGUNow() + **new** Random().nextFloat());
4. objectOutputStream.writeObject(mM);
5. mM.setGUNow(mM.getGUNow() + **new** Random().nextFloat());
6. objectOutputStream.writeObject(mM);
7. mM.setGUNow(mM.getGUNow() + **new** Random().nextFloat());
8. objectOutputStream.writeObject(mM);
9. objectOutputStream.flush();

**服务端接收消息：**

1. //2.接收当前拍对象状态信息
2. //滤波，整理好数据再放回到mM
3. mM = (motorModule) objectInputStream.readObject();
4. GU += mM.getGUNow();
5. mM = (motorModule) objectInputStream.readObject();
6. GU += mM.getGUNow();
7. mM = (motorModule) objectInputStream.readObject();
8. GU += mM.getGUNow();
9. GU /= 3;
10. mM.setGUNow(GU);

### swing图形显示

swing是用于生成可视化图形页面的类，该类下有JFrame作为窗口，提供JPanel用来盛放下级组件。

服务器只用了简单的JTextArea来显示当前服务器的流状态。

客户端窗口使用BorderLayout管理布局，分为三个区域，文本区（WEST)、动画区(Center)、曲线图区(SOUTH)。

**文本区：**

文本区采用采用BorderLayout()布局，分为3个区域，设置状态区（NORTH）、状态区（Center）、程序状态区（SOUTH）。设置状态区放开始停止按钮，修改按钮并增加相对应的按钮事件监听器。状态区显示当前拍数信息。程序状态区显示当前程序状态。这部分代码非常多，所以不展示出来了。

**动画区**

程序方法为，调用继承JLabel，并实现Runnable接口的 TiChiLabel对象来显示一个转动的太极图，将该TiChiLabel放到动画区

1. //动画显示区
2. JPanel cartoon = **new** JPanel();
3. cartoon.setBackground(Color.GRAY);
4. tiChiLabel = **new** TiChiLabel();
5. tiChiLabel.setPreferredSize(**new** Dimension(300,300));
6. cartoon.add(tiChiLabel);
7. contentPane.add(cartoon);
8. **new** Thread(tiChiLabel).start();

**曲线图区：**

1. //曲线显示区
2. **this**.curveLabel = **new** curveLabel();
3. JPanel curvePanel = **new** JPanel();
4. curvePanel.setPreferredSize(**new** Dimension(0,200));
5. curvePanel.add(curveLabel);
6. contentPane.add(curvePanel,SOUTH);

### 多线程技术

程序启用了多个线程，服务器占据一个线程，服务器建立一个新连接会占据一个线程，每新开一个客户端会新占一个线程，每个客户端对应的太极转动效果图也会占一个新线程。

用到了多线程的对象都实现了Runnable接口，以client为例：

1. //实现Runnable接口的客户端类
2. **public** **class** client **implements** Runnable {
3. @Override
4. **public** **void** run() {
5. }
6. }
7. //新开线程来运行客户端
8. **new** Thread(**new** client()).start();

### 异常处理技术

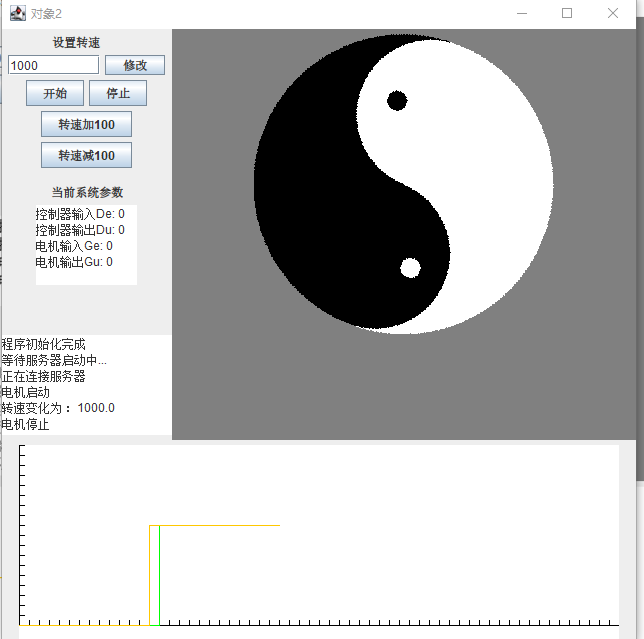
程序多处用到异常处理机制，例如客户端发起socket连接失败的异常会被捕获，然后重新发起连接

1. **try** {
2. clientWindow.setTextProgramer("正在连接服务器\n");
3. **return** **new** Socket(host, port);
4. } **catch** (IOException e) {
5. **return** reConnect(host, port);
6. }

# 代码效果及测试

## 程序基本效果展示

基本功能为转速可修改，电机启停控制，曲线图显示，电机输出动画效果（报告加不了动画展示不出来）。曲线图中，黄色是给定输入值，绿色是电机转速输出值，可以看到电机转速输出是在第二拍达到稳定的，跟simulink仿真结果完全一致。基本功能测试成功。

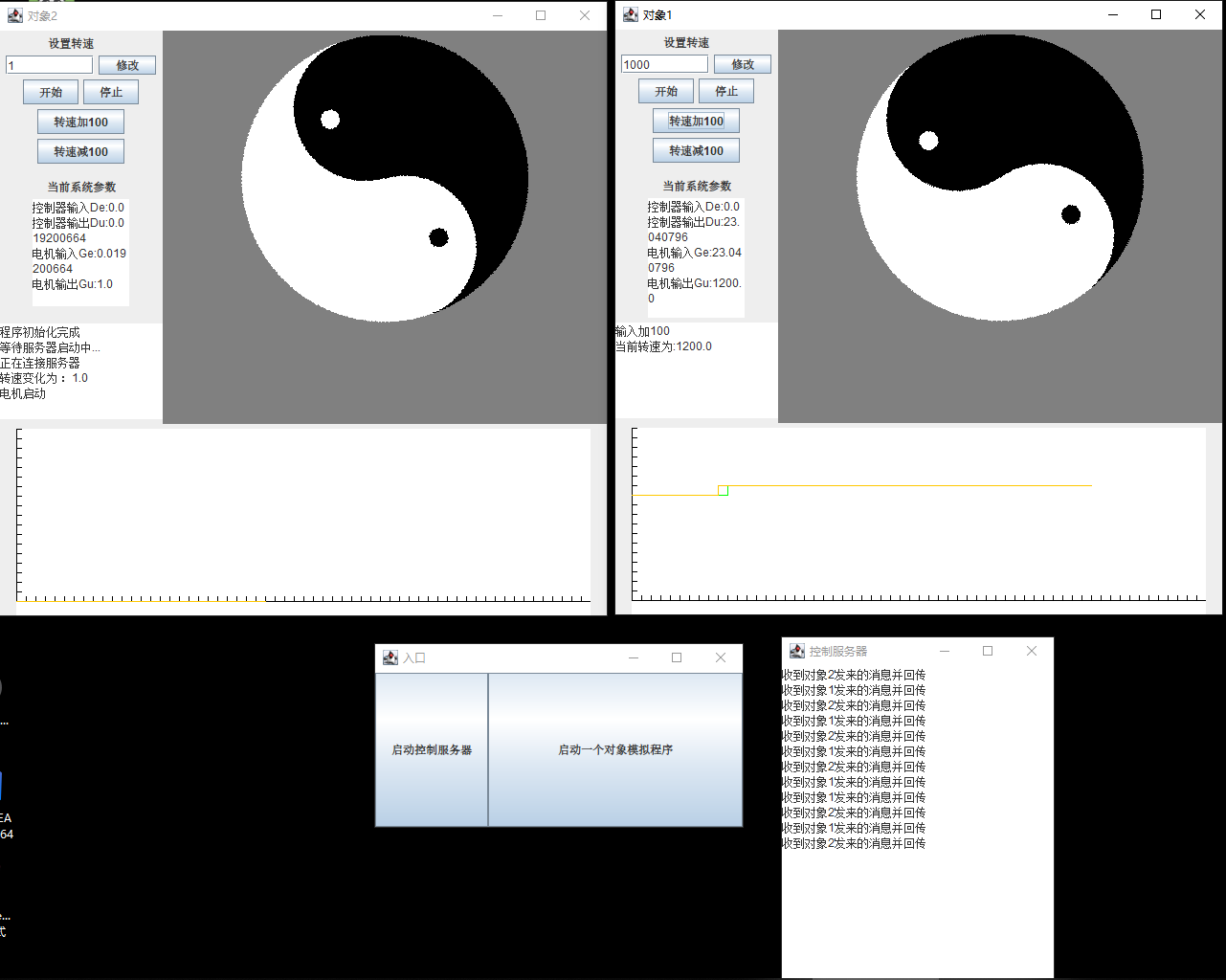


## 模块化实现，代码复用效果测试

修改对象1转速为1000，然后按两次装速+100，使得给定转速为1200，给对象2设定转速为1。

该程序在电脑性能允许的范围下，可以开启无限多个模拟控制程序，并通过socket通信使用同一个服务器进行控制，实现了代码的良好复用。

模块化实现及代码复用测试成功。



**只能开启一个服务器**

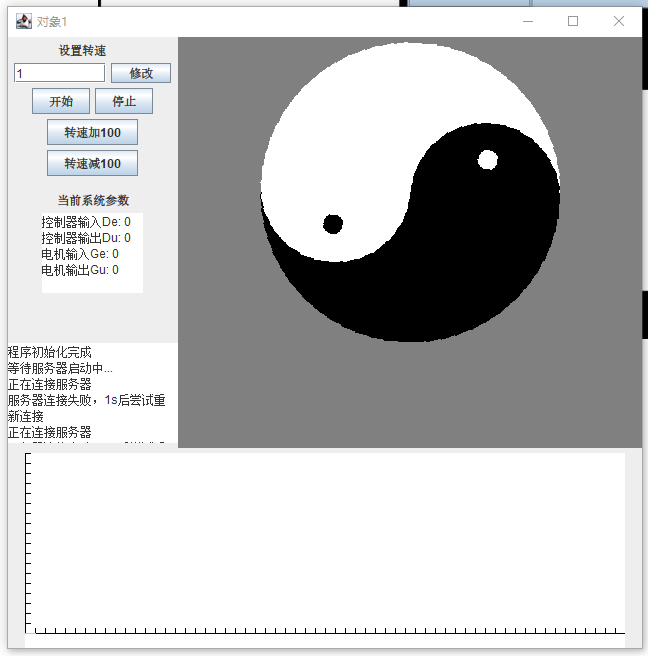
已经启动服务器的情况下再次按下控制服务器启动按钮，会出现提示对话框，并不响应新建服务器的请求。



## socket通信功能

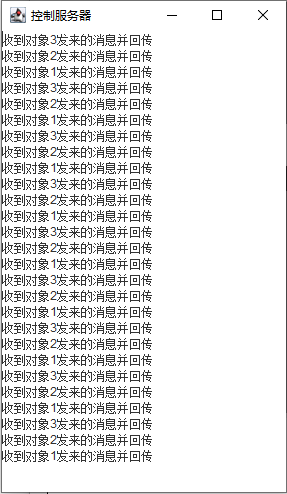
**客户端自动重连测试**

观察程序状态显示区可以看到，在没有开启服务器的情况下，开启客户端，客户端得不到有效连接会自动延迟1s后进行重连



**控制服务器Socket通信测试**

由下图可以看到一个控制服务器可以通过多线程和Socket通信实现控制服务器复用功能。

****

## 滤波功能

客户端信息发送时加上随机干扰

1. //1.发送对象当前拍数状态信息
2. //加干扰，发送3次信号
3. mM.setGUNow(mM.getGUNow() + **new** Random().nextFloat() - 0.5f);
4. objectOutputStream.writeObject(mM);
5. mM.setGUNow(mM.getGUNow() + **new** Random().nextFloat() - 0.5f);
6. objectOutputStream.writeObject(mM);
7. mM.setGUNow(mM.getGUNow() + **new** Random().nextFloat() - 0.5f);
8. objectOutputStream.writeObject(mM);
9. objectOutputStream.flush();

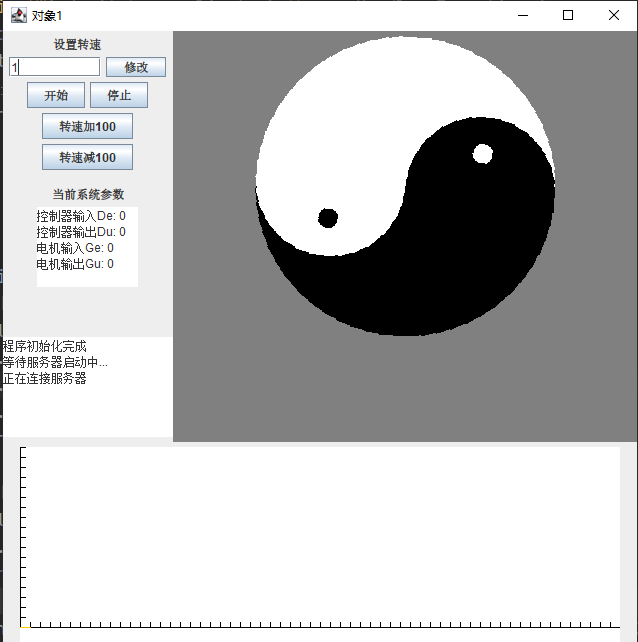
服务端接收信息后滤波

1. //2.接收当前拍对象状态信息
2. //滤波，整理好数据再放回到mM
3. mM = (motorModule) objectInputStream.readObject();
4. GU += mM.getGUNow();
5. mM = (motorModule) objectInputStream.readObject();
6. GU += mM.getGUNow();
7. mM = (motorModule) objectInputStream.readObject();
8. GU += mM.getGUNow();
9. GU /= 3;
10. mM.setGUNow(GU);

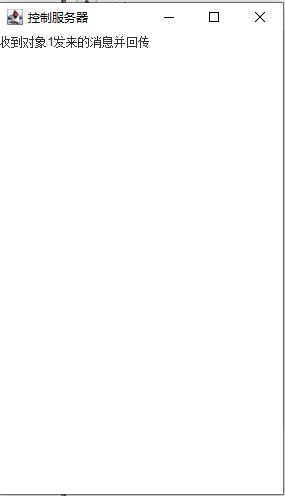
服务端发送控制信号和客户端接收后滤波同样方法实现，不再赘述。

## 图形化人机交互页面

人机页面采用SWING实现，分为客户端和服务器两种。客户端页面采用BorderLayout布局管理，分为文本区（西）、动画区（中心）、曲线图区（南），文本区也采用BorderLayout布局管理，又可细分为系统状态设置区（北）、系统状态显示区（中心）、程序状态区（南）



服务器的页面实现较为简单，不再赘述



# 关键问题及解决方案

**1. 系统的输入输出差分方程实现**

一开始我按照连续域上的做法去实现这个离散系统的时候，控制器的输入输出，对象的输入输出都是使用当前拍数。结果系统不稳定，仿真结果发散。后来我通过simulink的仿真观察控制器的输入输出，对象的输入输出，发现控制器的输入使用的是上一拍的对象输出和给定输入的误差值，而不是当前拍数的。对象的输入也是使用控制器上一拍的输出。后来按照这个方案进行仿真结果就正确了。

这体现了离散域和连续域系统的区别，连续域上的输入输出都是实时的，但在离散域上，一拍只能完成一个动作，因此系统的输入是要使用上一拍的结果的。

**2. 如何设计图形化交互页面，选取什么方案来实现图形化交互页面**

由于之前没有学习过图形化页面的内容，所以我对这方面不是很熟悉。为了完成这次作业的任务要求，我去查了一下资料，发现Java官方给出的图形化方案主要是AWT和SWING两种。其中SWING是完全由JAVA实现的，因此具有平台无关性，即是它的实现效果是跟操作系统无关的，无论是在Windows还是在Linux下它的显示效果都是一样的。因此我选择了SWING的方案，然后找一些Demo和教学来学习swing，遇到什么问题再去看相关的具体解决方案。

**3. 如何实现代码的复用？**

为了实现代码的复用，在设计之初我就制定了编写代码的原则：要实现复用的代码块全部使用非静态。静态的内容是属于类持有的，这很不利于代码的复用。而非静态代码是被当前实例化对象私有的，因此只要采用非静态的编写代码原则，最后的结果就能很容易的实现代码的复用。

**4. socket通信发送完消息后不结束，InputStream会阻塞**

一开始写socket的Demo的时候会发现阻塞的问题，以客户端发送消息为例，客户端发送消息完，但socket仍在等待客户端的输入，没有结束标志就会产生堵塞。为解决这个问题，一个简单的方案是发送完后使用flush()将缓冲区内容全部输出即可结束输入。但在实际代码中，我使用的是ObjectInputStream，这种方法并不会产生阻塞。

# 心得体会

**计算机控制系统的心得**

通过这样一个完整的建模，离散，设计控制器，代码模拟控制器及对象过程，，我对计算机控制系统的内涵有了更加深刻的认识，基本上一个完整的流程我心里都有了一定的了解。

**程序编写的心得**

由于Java我也是最近才开始学习的，这次刚好碰到这个需要编写程序的任务，因此我就尝试着去用Java来实现这次的任务。这是我学Java以来写的第一个项目，在完成了这个项目之后，我发现我对Java的语法知识也更加熟悉了，对程序设计的一些常见的思路也懂了一点，可以说这个作业对我的启发很大，给了我一个很好的练习机会。

**反思**

在完成作业后再看下自己的程序，发现我的程序还是有些不足的地方。首先就是代码有一些乱，并且写完之后没有好好地整理一下代码。我觉得在编写代码这方面还是要学习一些规范，这样就有可能写出结构比较简洁的代码。

另一方面，我发现我的程序功能还是有一些不足的地方，例如客户端窗口行满后重新刷写时，最后写入的内容不会显示到第一行，修改这个还得花费一点时间。另外一方面就是在关闭一个客户端窗口之后，应该释放所有有关于这个客户端的资源，包括client线程，太极显示图线程，服务器socket线程，这里我的代码关闭了client线程、太极显示图线程，但服务器持有的socket线程还没有释放掉，这里还需要花时间完成。

# 附录

本次编程环境为： IDEA 2020.1 + openjdk - 14