Mechanical_Design 设计说明书

项目特色：

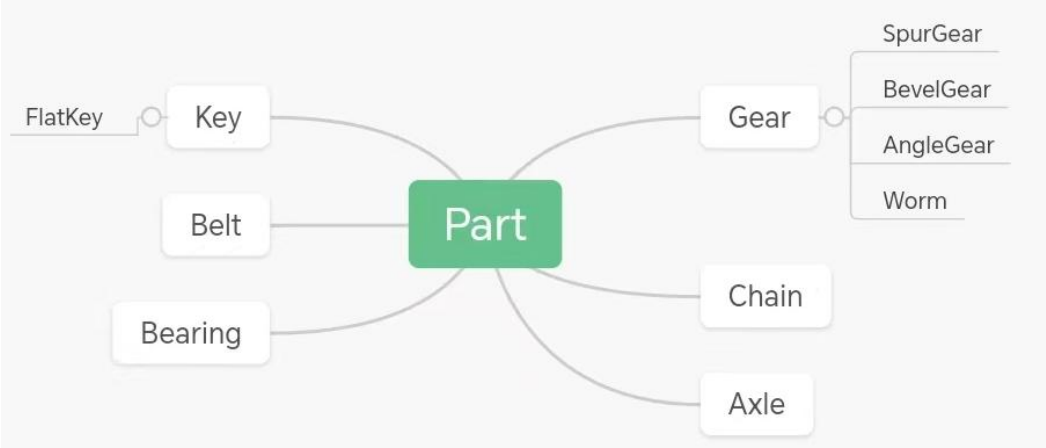
1 采用智能指针控制内存分配，同时避免内存泄露
2 将零件与设计分开，分别实现零件类与设计类
3 利用动态绑定，使得一个容器中能包含多种设计
4 将设计与交互过程分开，使得代码更加简洁，结构紧凑
5 采用继承和模块化实现，实现大量代码复用
6 函数命名统一，注释和代码规范。例如：setXXX（）函数设置对应变量，getXXX（）函数获取对应变量
7 采用多层代码封装，精简单一函数中的代码，便于后续拓展与调试
8 利用宏编译，控制输出
9 采用了文件系统以输出完整的设计结果

设计的总体框架：

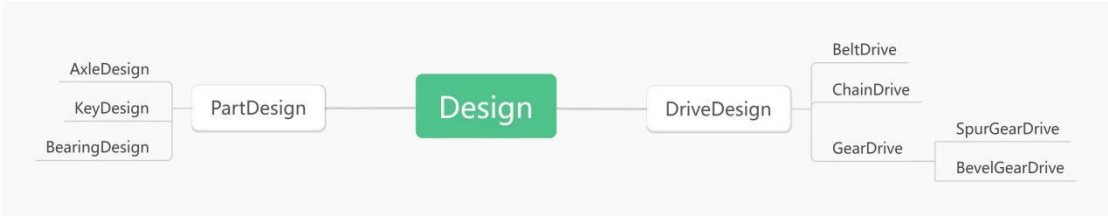
Mechanical_Design 类为最外层类，用于调度；
Interaction 类负责交互；
Design_Vec 类负责设计操作；
具体的设计是 Design 的子类实现的；
对应的 Design 类中包含对应的零件类 Part 对象；

继承体系的设计：

（类的具体命名会有些差别，只实现了部分类，参看具体实现）
Part 类：



Design 类：



具体的实现：

Mechanical_Design 类：

实现总体调度，实现如下：

```
class Mechanical_Design {
public:
    Mechanical_Design() {
        vec = std::make_shared<Design_Vec>();
        Ite = std::make_shared<Interaction>();
        startDesign(std::cout);    //开始设计
    };
    ~Mechanical_Design() {};
private:
    shared_ptr<Design_Vec> vec;                    //设计表对象
    shared_ptr<Interaction> Ite;                   //交互对象
    Outputter output;
    //每个零件处理自己的最终设计过程，Mechanical_Design 只提供公共接口
    void startDesign(std::ostream& os);            //开始设计
    void startDriveDesign(std::ostream& os);       //开始传动设计
    void startPartDesign(std::ostream& os);        //开始零件设计
    void showDesignInfo(std::ostream& os);         //显示设计信息
```

};
以 showDesignInfo 函数的具体实现为例：

```
void Mechanical_Design::showDesignInfo(std::ostream& os) {
    Ite->InteractionOfInfo(os);
    char ch4 = Ite->getShow();
    string str = "design";
    int tmp;
    while (ch4 != 'Q') {
        switch (ch4) {
        case 'A':
            vec->showTotalDesign(true);           //详细信息
            break;
        case 'B':
            vec->showTotalDesign(false);      //粗略信息
            break;
        case 'C':                                 //单个设计信息
            if (vec->getSize() == 0) {
                os << "当前无设计" << std::endl;
            }
            else {
                os << "请输入设计序号 0-" << vec->getSize()-1<< " :" << std::endl;
                std::cin >> tmp;
                vec->showDesign(tmp);
            }
            break;
        case 'D':
            os << "删除所有设计" << std::endl;
            vec->deleteTotalDesign();
            break;
        case 'E':
            if (vec->getSize() == 0) {
                os << "当前无设计" << std::endl;
            }
            else {
                os << "请输入设计序号 0-" << vec->getSize() -1<< " :" <<
std::endl;
                std::cin >> tmp;
                vec->deleteDesign(tmp);
            }
            break;
        case 'F':
            os << "请输入要保存的文件名：" << std::endl;
            std::cin >> str;
            output(str + ".txt", *vec);       //保存设计信息
```

```
            break;
        default:
            break;
        }
        Ite->InteractionOfInfo(os);
        ch4 = Ite->getShow();
    }
}
```

先是 `Ite` 对象显示交互界面，然后获取交互后的结果进行循环判断。`vec` 对象实现显示不同信息的操作。其它几个函数的实现类似。
注意到此处的形参是 `std::ostream&os`。这样做格式规范一点。而且不需改动代码，即可将结果输出到文件（用 `ofstream&` 对象）。
同时智能指针的使用，方便了内存处理，不用担心内存泄漏的问题。

Interaction 类：

交互类，接受输入。交互与实际操作分开的设计，很方便拓展代码，格式上也很美观。

```
class Interaction {
public:
    Interaction() {}
    ~Interaction() {}

    void setIni(std::ostream& os);                //设置初始化界面
    void InteractionOfTotal(std::ostream& os);        //总设计界面的交互
    void InteractionOfDriveDesign(std::ostream& os);//传动设计界面的交互
    void InteractionOfPartDesign(std::ostream& os);  //零件设计界面的交互
    void InteractionOfGearDrive(std::ostream& os);   //齿轮传动设计界面
    void InteractionOfKey(std::ostream& os);      //键设计界面
    void InteractionOfAxle(std::ostream& os);     //轴设计界面
    void InteractionOfBearing(std::ostream& os); //轴承的强度设计
    void InteractionOfChainDrive(std::ostream& os);  //链传动
    void InteractionOfBeltDrive(std::ostream& os);   //带传动
    void InteractionOfInfo(std::ostream& os);     //显示设计信息
    //其他部分的设计界面...
    char getGearDrive() const { return chGearDrive; }
    char getAxle() const { return chAxle; }
    char getTotal() const { return chTotal; }
    char getDrive() const { return chDrive; }
    char getPart() const { return chPart; }
    char getShow()const { return chShow; }
    char getBearing()const { return chBearing; }
```

```
    void setTotal(char ch) { chTotal = ch;}
    void setDrive(char ch) { chDrive = ch; }
    void setPart(char ch) { chPart = ch; }
    void setGearDrive(char ch) { chGearDrive = ch; }
    void setKey(char ch) { chKey = ch; }
    void setShow(char ch) { chShow = ch; }
    void setBearing(char ch) { chBearing = ch; }

private:
    char chTotal;                                   //总体选项
    char chDrive;                                   //传动选项
    char chPart;                                    //零件选项
    //拓展用选项
    char chGearDrive;                               //齿轮传动
    char chKey;                                     //键的设计
    char chAxle;                                    //轴的设计
    char chBearing;                                 //轴承的设计
    char chShow;                                    //设计信息
};
```

交互类没有什么特别之处，有一个检测输入是否合理的循环。
项目命名规范和统一在这里得到很具体的体现。

Design_Vec 类：

框架内实现具体设计操作的类，定义了一个枚举变量 Design_Vector，为 addDesign 的调用作了一级封装。
类似的调用如下：
addDesign(Design_Vector::AxleDesign);
在 Mechanical_Design 中编写相应的代码就比较方便了。
因为项目中设计类使用了继承，利用动态绑定，就可以在一个 vector 容器中添加不同类型的设计。

```
class Design_Vec {
public:
    Design_Vec() = default;    //默认构造函数
    Design_Vec(const std::vector<shared_ptr<Design>> &vec):VecPtr(vec){}
    Design_Vec(const Design_Vec& rhs):VecPtr(rhs.VecPtr){}
    Design_Vec& operator=(const Design_Vec& rhs) {    //重载赋值运算符
        VecPtr = rhs.VecPtr;
        return *this;
    }
    ~Design_Vec() { std::cout << "完成所有设计!" << std::endl;}

    int getSize()const { return VecPtr.size(); } //返回设计对象数目
```

```
    //注意这里调用设计操作时，引入了 ostream& 类型的形参
    //方便后续将设计结果打印到文件中
    void addDesign(int type,std::ostream&os = std::cout);            //添加
一个设计对象
    void deleteDesign(int pos,std::ostream&os = std::cout);          //删除
一个设计对象
    void showDesign(int pos,std::ostream&os = std::cout);            //显示
一个设计对象
    void showTotalDesign(bool b,std::ostream&os = std::cout);        //显示
所有设计对象，详细与否
    void deleteTotalDesign(std::ostream&os = std::cout);
    //删除所有设计对象

    enum DesignVector { SpurGearDrive, BevelGearDrive,
BeltDrive,ChainDrive,KeyDesign, AxleDesign,
        DeepBearingDesign,AngleBearingDesign };
    //定义枚举值，用作用域运算符区分
private:
    std::vector<shared_ptr<Design>> VecPtr;          //使用包含智能指针的容器

};
```

实际添加，删除对象都是利用 vector 的。

Outputter 类：

实现结果输出，这里重载了运算符“()”

```
class Outputter {
public:
    Outputter(){};
    ~Outputter() {};
    void operator()(string str, Design_Vec& vec) {
        setOutput(str, vec);                        //重载运算符(),使该类成为可调
用对象
}

private:
    void setOutput(string str, Design_Vec& vec);
};
```

总体框架中的实现难度都比较小，最重要的是这个框架的建立极大地方便了后续其他设计功能的添加与处理。因为整个项目中框架和具体设计类之间没有任何耦合。设计变动，不会让框架也变动。

Design 类：

作为抽象基类，定义了四个子类必须要重新实现的虚函数：

```
class Design {
public:
    Design()noexcept{}
    virtual ~Design() {};
    virtual void setDesign(bool b) = 0;                    //虚函数，开始设计
    virtual void showDesignInfo(ostream& os) = 0;     //显示设计信息

    int getNum() const { return Num; }                     //返回设计编号
    string getName() const { return Name; }                //获取设计名称

    void setName(string str) { Name = str; }          //修改设计名称
    void setNum(int i) { Num = i; }                        //设定设计编号
    void setLevel(int val) { Level = val; }                //设定设计的精度等级

    double Angle_To_Radian(double angle) { return angle / 180 * M_PI; }; //
角度换算
protected:
    virtual void setDefault() = 0;                    //默认参数设计
    virtual void setUserChoice(std::ostream& os) = 0;//按用户选定的参数来设计

    time_t timer = 0;
    int Num = 1;                                           //设计编号
    int Level = 7;                                         //精度等级
    string Name = "设计";                                  //设计名称
};
```

SetDesign 函数有个 bool 形参，用来控制默认设计和用户设计。
ShowDesignInfo 中有个 ostream&的形参，Design_Vec 类中，将该形参换成 ofstream&的，就能实现输出结果到文件中了。

Drive_Design 和 Part_Design 类：

这两个设计类都只定义了几个新变量。
仍旧是抽象类。

Part 类：

零件类，定义了一些零件属性。`class Part {`

```
public:
    Part(string mat,string hard):Material(mat),Hardness(hard){}
    virtual ~Part() {};                                    //虚函数
    virtual void showInfo(ostream& os) = 0;                //纯虚函数，输出零件
具体信息

    string getMaterial() const { return Material; }       //获取材料
    string getHardness() const { return  Hardness; } //获取硬度
    int getLevel() const { return Level; }                 //获取精度等级

    void setMaterial(string str) { Material = str; } //设置精度等级
    void setHardness(string str) { Hardness = str; }
    void setLevel(int val) { Level = val; }

protected:
    string Material;                                       //材料
    string Hardness;                                       //硬度
    int Level = 7;                                         //精度等级
};
```

接下来以齿轮设计类-Gear_Drive 作为重点来介绍。
这里会略去一些不重要的函数与变量，齿轮设计类是整个项目最复杂，实现最漂亮的一个类。参考这个类，很容易就能实现其他设计类。
直到具体设计类中，才出现 shared_ptr<Part>对象 part。
考虑到具体设计，非常依赖零件属性，继承来的 part 难堪重任。

Gear_Drive 类：

略去了不重要的变量和函数。
总设计流程分两大部分：
Bend_Fatigue_Design 和 Contact_Fatigue_Design;
因为设计一般设计多个齿轮，故定义了一个用于插值法的静态变量，节约内存。

```
class Gear_Drive : public Drive_Design {
public:
    Gear_Drive(shared_ptr<Gear> p1 = nullptr, shared_ptr<Gear> p2 =
nullptr)noexcept :
        Drive_Design(), part1(p1), part2(p2) {
    }
    virtual ~Gear_Drive() {}; //又是一个抽象基类
protected:
    shared_ptr<Gear> part1;        //两个零件对象
```

```
    shared_ptr<Gear> part2;
        virtual void Bend_Fatigue_Design();                                    //
弯曲和接触疲劳强度设计
    virtual void Contact_Fatigue_Design();

protected:
    virtual void setTrialDiameter();                                  //获取试算
直径，按接触强度,为虚函数
    virtual void setTrialModulus(double OF1,double OF2,double z1,double p =
0);  //获取试算模数，按弯曲强度
    virtual void setGear(double z1);                                  //设置齿轮
的参数
    //virtual void setDefault() = 0;                                  //默认参数
设计
    void setUserChoice(std::ostream&os) override;                     //按用户选
定的参数来设计

    virtual void setTriE(double a, double p = 0);                     //三个E常数
的处理,为虚函数
    virtual void sete(int z1, int z2, double a, double p = 0);        //设置
重合度,为虚函数
    virtual void setYe() { Ye = 0.25 + 0.75 / e; };                   //设置
重合度系数,也为虚函数
    void setOH(double OH1, double OH2);                               //设置
接触疲劳极限
    void setOF(double& OF1, double& OF2);                             //设置弯曲
疲劳极限,用引用
    void setFourK(bool b, double v1);                                 //四个K常数
    void setKHbandKFb(double val, double val2) { KHb = val, KFb = val2; }
    void setmt();                                                     //调整模数,
按默认规则圆整后的值可能出错
    void setKHaandKFa(double d);                                      //设置KHa
和KFa参数
    void setTwoY(double& YFa1, double& YSa1,double z1);               //自动
利用插值法计算YFa, YSa
    //setUserChoice函数的具体构成
    void setBasicParameter(std::ostream& os);                         //基本参数,
即不需查表的参数
    void setLifeParameter(std::ostream& os);                          //寿命系数
设定
    void setKHbParameter(std::ostream& os);                           //齿向
载荷分配系数,疲劳对应的可以直接查表
    //使用常量表达式和静态全局变量，优化内存
    constexpr static double TwoY[][2] = ...
```

```
};
```
下面结合具体实现来介绍
```
void Gear_Drive::Bend_Fatigue_Design() {
    //从零件获取所需信息
    int z1 = part1->getZ();
    double OF1 = part1->getOF(), OF2 = part2->getOF();

    setYe();                                    //计算重合度系数
    setOF(OF1,OF2);                             //设置弯曲疲劳极限
    setTrialModulus(OF1, OF2, z1);              //计算弯曲对应的模数

    double d1 = mt * z1,tmp = KHb,tmp1;             //分度圆半径
    double h = (2 * HA + C) * mt,tmp2;              //计算宽高比
    tmp2 = q * d1 / h;
    //输出部分信息
    std::cout << "宽高比为：" << tmp2 << " " << "接触疲劳的齿向载荷分配系数 KHb 为：
" << tmp << std::endl;
    std::cout << "请查表获得弯曲疲劳的齿向载荷分配系数 KFb:" << std::endl;
    std::cin >> tmp1;
    setKHbandKFb(tmp, tmp1);

    setKHaandKFa(d1);                           //计算圆周速度
    setFourK(0,1.08);                           //四个接触疲劳 K 常数的确定
    std::cout << "按弯曲疲劳强度设计：" << std::endl;
    setmt();                                    //模数的调整
}
```
这个函数就是多层代码封装的一个很好的例子。
每一个具体操作过程都对应了相应的函数。使得整个函数结构紧凑，一目了然。项目中利用多层封装，规避许多代码量巨多的函数的出现。同时代码修改和调试都方便多了。
Contact_Fatigue_Design 函数的实现类似。
一些设计细节在代码注释中都有很好的体现。继承和模块化的采用，使得两个齿轮实际类中新增的代码都不多。

Gear_Drive 类仍然不是一个完整的类。缺了 setDefault,setDesign 和 showDesignInfo 三个函数的实现。下面给出 Spur_Gear_Drive 类中对应的实现：
```
void Spur_Gear_Drive::setDefault() {  //默认参数设计
    setT(P / n * 9550000);
    setu(3.2);
    setq(1);
    setSH(1.0);
    setSF(1.4);
    setKHN(0.88, 0.91);                     //寿命系数与安全系数
```

```
    setKFN(0.85, 0.88);                     //设置寿命系数
    //setKHaandKFa();                     //这一步必须在算出分度圆半径后进行
    setKA();
    setKHbandKFb(1.320, 1.276);
}

void Spur_Gear_Drive::setDesign(bool b) {
    Name = "直齿圆柱齿轮传动";              //设计名称
    if (b)
        setDefault();
    else
        setUserChoice(std::cout);       //用户手动操作
    Contact_Fatigue_Design();           //接触疲劳强度设计
    Bend_Fatigue_Design();              //弯曲疲劳强度设计
    setGear(dt / mt);                   //调整齿数
}

void Spur_Gear_Drive::showDesignInfo(ostream&os) {
    os << "直齿圆柱齿轮传动的主要设计参数如下:\n" << std::endl;
    if (part1)
        part1->showInfo(os);
    if (part2)
        part2->showInfo(os);
    time(&timer);
    os << "齿轮副的中心距：  " << a << "mm" << std::endl;
    os << "设计编号：" << Num << std::endl;
    os << "当前时间：" << ctime(&timer) << std::endl;
}
```

项目总结

这个项目难度不大，只是稍微有一点耗时间。
项目很好地体现了 C++中的面向对象思想，还有一些代码技巧。

2024/7/26　于福安市