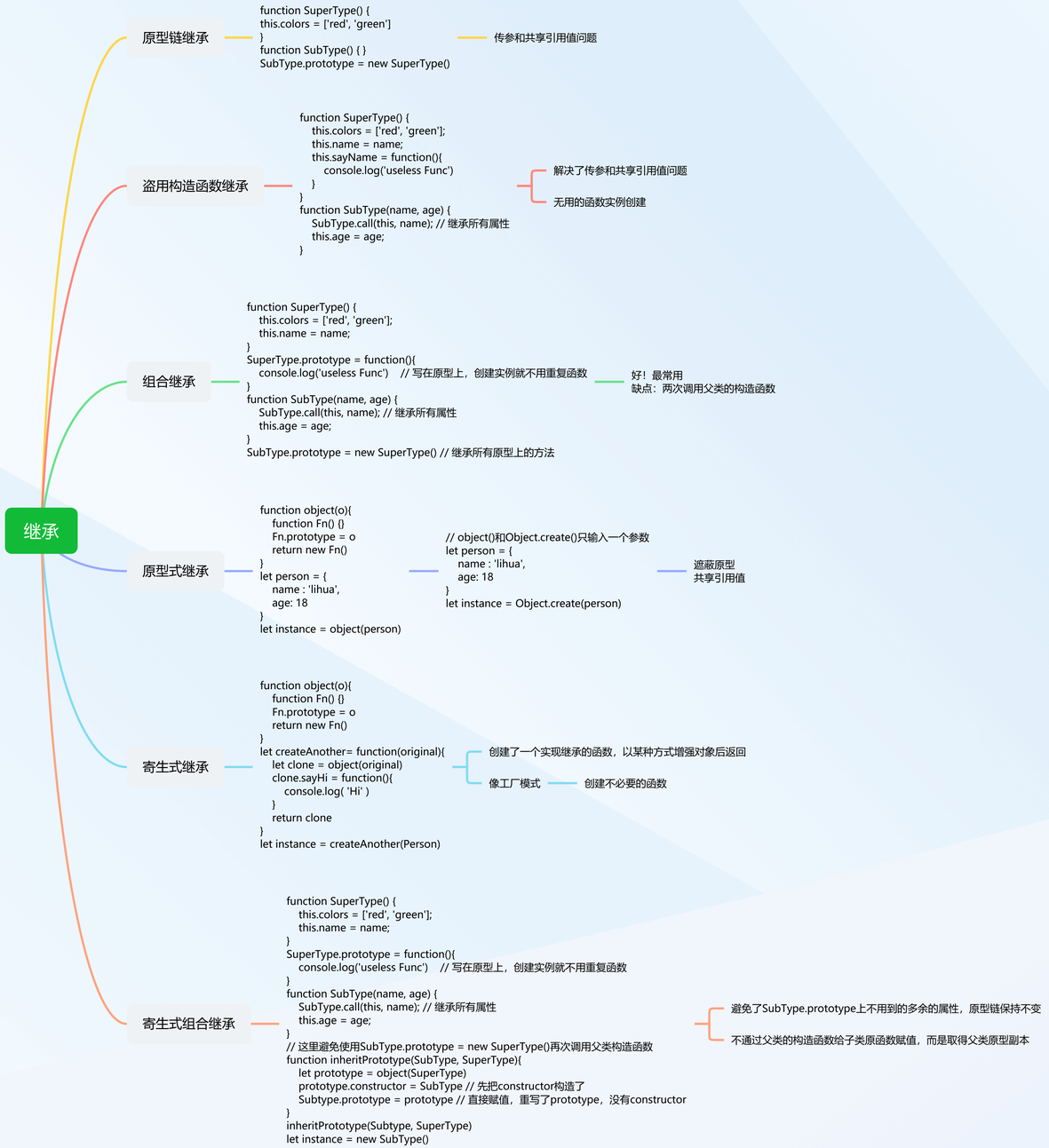
继承
原型链继承
function SuperType() {
this.colors = ['red', 'green']
}
function SubType() { }
SubType.prototype = new SuperType()
传参和共享引用值问题
盗用构造函数继承
function SuperType() {
this.colors = ['red', 'green'];
this.name = name;
this.sayName = function(){
console.log('useless Func')
}
}
function SubType(name, age) {
SubType.call(this, name); // 继承所有属性
this.age = age;
}
解决了传参和共享引用值问题
无用的函数实例创建
组合继承
function SuperType() {
this.colors = ['red', 'green'];
this.name = name;
}
SuperType.prototype = function(){
console.log('useless Func') // 写在原型上，创建实例就不用重复函数
}
function SubType(name, age) {
SubType.call(this, name); // 继承所有属性
this.age = age;
}
SubType.prototype = new SuperType() // 继承所有原型上的方法
好！最常用
缺点：两次调用父类的构造函数
原型式继承
function object(o){
function Fn() {}
Fn.prototype = o
return new Fn()
}
let person = {
name : 'lihua',
age: 18
}
let instance = object(person)
// object()和Object.create()只输入一个参数
let person = {
name : 'lihua',
age: 18
}
let instance = Object.create(person)
遮蔽原型
共享引用值
寄生式继承
function object(o){
function Fn() {}
Fn.prototype = o
return new Fn()
}
let createAnother= function(original){
let clone = object(original)
clone.sayHi = function(){
console.log('Hi')
}
return clone
}
let instance = createAnother(Person)
创建了一个实现继承的函数，以某种方式增强对象后返回
像工厂模式
创建不必要的函数
寄生式组合继承
function SuperType() {
this.colors = ['red', 'green'];
this.name = name;
}
SuperType.prototype = function(){
console.log('useless Func') // 写在原型上，创建实例就不用重复函数
}
function SubType(name, age) {
SubType.call(this, name); // 继承所有属性
this.age = age;
}
// 这里避免使用SubType.prototype = new SuperType()再次调用父类构造函数
function inheritPrototype(SubType, SuperType){
let prototype = object(SuperType)
prototype.constructor = SubType // 先把constructor构造了
Subtype.prototype = prototype // 直接赋值，重写了prototype，没有constructor
}
inheritPrototype(Subtype, SuperType)
let instance = new SubType()
避免了SubType.prototype上不用到的多余的属性，原型链保持不变
不通过父类的构造函数给子类原函数赋值，而是取得父类原型副本